JAVA FOR THE STUDY OF EVOLUTION

VOLUME XIV

THE HAND II

José del Carmen Rodríguez Santamaría
The EvolJava Community

Contents

1 Deformable shapes **1**
- 1.1 Deforming the mesh of a cube 1
- 1.2 Changing the texture of a cube 10
- 1.3 For more than fun 10
- 1.4 The simulation of forces 63
- 1.5 Conclusion 67

2 A random walk of mutations **69**
- 2.1 Disfigurement of the sphere. 69
- 2.2 Conclusion 69

3 Parametric mutations **71**
- 3.1 Animated hand. 72
- 3.2 Principal axis theory of the enlargement of shape. 72
- 3.3 Conclusion 212

4 Battling mutation **215**
- 4.1 Is recombination a weapon against mutation? 215
- 4.2 More on JavaFX: trees 221
- 4.3 Homemade binary trees 244
- 4.4 Pleiotropy in action 274
- 4.5 Conclusion 441

5 On the evolution of biological shape **443**

6 Answers to exercises **445**
- 6.1 Biblio 1012

Preface

The series *Java for the Study of Evolution* is directed to scientists that want to manage a serious but not excessively expensive tool to study evolution by direct experimentation under perfectly controlled conditions. These requirements cannot be met in nature but only in simulations and mathematical models. In consequence, the series has three main purposes:

1. To endow the community of **researchers in biology and evolution** with *high level programming*, enabling an accurate study of models and simulations of the most diverse nature.

2. To clearly show how this tool is used to study the fundamental questions of evolution.

3. To suggest that the study of Java could be *very fruitful* for **undergraduates** in biological sciences even more than calculus alone.

This is the 14th volume: We learn how to deform meshes. This enables a straightforward proof that mutation is extremely powerful to erode the form of shapes. Next, we bring forth various proposals to explain the evolutionary stability of biological shapes: natural selection, recombination and pleitropy. In this regard, we propose and simulate a theory of biological shape that results naturally from cell division.

Introduction

Our overall goal is to learn here how to make evolvable meshes to next face the problem of how to design stagnation as an emergent feature. Thus, chapter I is dedicated to learn how to deform meshes. This enables to program mutations, chapter II and III. The lesson is that mutation is extremely powerful to erode shapes. Hence, a question arises: how is biological shape stabilized against mutation? In chapter IV we bring forth various solutions: natural selection, recombination and pleitropy. In this regard, we propose and simulate a very natural theory about the formation of biological shape that arises naturally from cell reproduction.

We will find here various programs that classify as graphics-intensive. So, to diminish the load of the CPU you can manage the parameters of each program and control the refreshing time of the display. Or, eventually, you might desire to get a graphics card but, before doing that, verify the list of supported cards by JavaFX (Oracle, [6], 2015). Additionally, most programs work as separate independent units so, programs can be divided in two or more projects: too many programs in the same project would cause a waste of memory during compilation time. In that case, all projects must have a copy of the accompanying packages, tools and ejvol14p.

Bogotá, Colombia, *José Rodríguez*
November 2015

Chapter 1

Deformable shapes

`Shapes that can be changed`

1 ***Purpose.*** *We already know how to make 3D shapes in JavaFX with the help of meshes. Our task here is to learn how to provide them with the possibility of being deformed. In that way, we will enable evolution.*

1.1 Deforming the mesh of a cube

Let us make an animation in which a cube is enlarged over time but not because the camera is moved closer to it but because the cube itself is modified.

2 ***The following is the code for a cube, as a Java Object, that gets larger over time.***

```
/*
    Program N2 EvoBrick
    Shows an animation in which
    a  brick gets enlarged.
    You must copy the following images
    to the same package where this file is:
        UnfoldedBrick.png
        Beijing.png
        Camping.JPG
    These files are in the project of volume 13.
 */

package evobrick;
```

```
import java.util.Random;
import javafx.animation.Animation;
import javafx.animation.AnimationTimer;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.DepthTest;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.MeshView;
import javafx.scene.shape.TriangleMesh;
import javafx.scene.transform.Rotate;
import javafx.scene.transform.Translate;
import javafx.stage.Stage;
import javafx.util.Duration;


public class EvoBrick extends Application
{

    private Timeline animation;


private void init(Stage primaryStage)
{
    Group root = new Group();
    root.setDepthTest(DepthTest.ENABLE);
    primaryStage.setResizable(false);
    primaryStage.setScene(new Scene(root, 1800, 850, true));
    primaryStage.getScene().setCamera(new PerspectiveCamera());
    root.getTransforms().addAll
    (
```

```
        new Translate(400 / 2, 150 / 2),
        new Rotate(180, Rotate.X_AXIS)
    );
    root.getChildren().add(create3dContent());
}


//Manager of bricks
private Node create3dContent()
{

    //Central brick
    Brick b = new Brick(30,30,30, "UnfoldedBrick.png");
    b.rx.setAngle(45);
    b.ry.setAngle(45);
    //Brick at t he right
    Brick b2 = new Brick(30,20,30, "Beijing.png");
    b2.setTranslateX(100);
    b2.rx.setAngle(45);
    b2.ry.setAngle(45);
    //Brick at the left
    Brick b3 = new Brick(40,100,20, "Camping.JPG");
    b3.setTranslateX(-100);
    b3.rx.setAngle(45);
    b3.ry.setAngle(-45);


    //Bricks rotate
    animation = new Timeline();
    animation.getKeyFrames().addAll
    (
        new KeyFrame//Initial states at initial time
        (
            Duration.ZERO,
            new KeyValue(b.ry.angleProperty(), 0d),
            new KeyValue(b2.rx.angleProperty(), 0d),
            new KeyValue(b3.rz.angleProperty(), 0d)
        ),
        new KeyFrame//Final states at final time
        (
```

```
            Duration.seconds(5),
            new KeyValue(b.ry.angleProperty(), 360d),
            new KeyValue(b2.rx.angleProperty(), 360d),
            new KeyValue(b3.rz.angleProperty(), 360d)
        )
    );
    animation.setCycleCount(Animation.INDEFINITE);


    return new Group(b,b2,b3);
}

public void play()
{
    animation.play();
}

@Override public void stop()
{
    animation.pause();
}


private class Brick extends Group
{
    //The mesh is declared and instantiated
    TriangleMesh mesh = new TriangleMesh();
    Random r = new Random();


final Rotate rx = new Rotate(0,Rotate.X_AXIS);
final Rotate ry = new Rotate(0,Rotate.Y_AXIS);
final Rotate rz = new Rotate(0,Rotate.Z_AXIS);
public  Brick(float a, float b, float c, String image)
  {
    getTransforms().addAll(rz, ry, rx);
    if ((a * b * c == 0) || (a < 0) || (b < 0) ||(c< 0))
    {
      return;
    }
```

```
//Halves of measures.
//Type double is not accepted,
//instead float or int are allowed
float ha = (float) (a / 2);
float hb = b / 2;
float hc = c / 2f; //f stands for type float (optional)



//Declaration of the points of the mesh.
//Each point has 3 coordinates.
//There is neither a tag for  points
//or a special separator in within points!
mesh.getPoints().addAll
    (
        ha, hb, hc, //point 0
        ha, hb, -hc, //point 1
        ha, -hb, -hc, //point 2
        ha, -hb, hc, //point 3
        -ha, hb, hc, //point 4
        -ha, hb, -hc, //point 5
        -ha, -hb, -hc, //point 6
        -ha, -hb, hc //point 7
    );

//The brick will be covered by wallpaper,
//which is marked with points.
//Each point of the wallpaper has two coordinates.
//Coordinates go from 0 to 1 in both dimensions.
//So, wallpaper might be a rectangular image
//but it is seen here  as a square.

//Coordinates for the marking points of wallpaper.
//These points serve to form triangles
//that are next used to cover triangles of the mesh.
float x0 = 0f;
float x1 = 1f / 3f;//this means x1 = 1/3 but with decimals
```

```
float x2 = 2f / 3f;
float x3 = 3f / 3f;

float y0 = 0f;
float y1 = 1f / 4f;
float y2 = 2f / 4f;
float y3 = 3f / 4f;
float y4 = 1f;


/*
float y0 = 1f;
float y1 = 3f / 4f;
float y2 = 2f / 4f;
float y3 = 1f / 4f;
float y4 = 0f;
*/


//Points in wallpaper are distingushed by the order
//as they appear in the following list.
//Say, point 0 has coordinates (x1,y0)
//point 3 has coordinates (x1,y1) and so on.
//Order is the only separator of points.
mesh.getTexCoords().addAll(
  x1, y0, //Point 0 of wallpaper
  x2, y0,
  x1, y1,
  x2, y1,//Point 3 of wallpaper
  x0, y2,
  x1, y2,
  x2, y2,
  x3, y2,
  x0, y3,
  x1, y3,
  x2, y3,
  x3, y3,
  x1,y4,
  x2,y4//Point 13 of wallpaper
```

```
);

/* List of triangles:

//We have 12 triangles, two for each face of the box.
//Each triangle is defined by a triade of points:
5,6,4, //Rear
7,4,6,
7,6,3,//Left
2,3,6,
4,7,0,//Upper
3,0,7,
5,4,1, //Right
0,1,4,
0,3,1, //Front
2,1,3,
1,2,5, //Bottom
6,5,2


*/

//The official form as  the list of triangles
//must be declared for a mesh in JavaFX
//demands additional information in regard
//as the form as the covering wallpaper
//must be glued to the brick. To that aim,
//after listing a vertex of the brick,
//a vertex of the wallpaper is also attached.
mesh.getFaces().addAll(
//Rear face with vertexes 5, 6, 4
//must be covered with triangle
//13, 12, 10 of the wallpaper,
//in the cited order.
5, 13, 6, 12, 4, 10, //Rear
7, 9, 4, 10, 6, 12,
7, 9, 6, 8, 3, 5, //Left
2, 4, 3, 5, 6, 8,
4, 10, 7, 9, 0, 6, //Upper
3, 5, 0, 6, 7, 9,
```

```
5, 11, 4, 10, 1, 7,//Right
0, 6, 1, 7, 4, 10,
0, 6, 3, 5, 1, 3, //Front
2, 2, 1, 3, 3, 5,
1, 3, 2, 2, 5, 1,//Bottom
6, 0, 5, 1, 2, 2
);



//The following grouping renders the
//triangles of same face
//with one normal:
//the sun is reflected homogeneously
//along their two triangles.
int smoothingGroups[]
        = { 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5};


/*
//This grouping renders triangles
//of the same face of the brick
//with slightly different normals:
//the sun is reflected inhomogeneously.
int smoothingGroups[]
        = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};
*/

mesh.getFaceSmoothingGroups().addAll(smoothingGroups);

MeshView brick = new MeshView(mesh);
brick.setMesh(mesh);

//Wallpaper is gotten
Image wallpaper = new Image(getClass()
        .getResourceAsStream(image));
//Declaration of material for painting the brick
PhongMaterial material = new PhongMaterial();
//Sun
```

```
    material.setSpecularColor(Color.WHEAT);
    //Wallpaper is declared as covering material
    material.setDiffuseMap(wallpaper);
    //Brick is covered with wallpaper
    brick.setMaterial(material);

    //The mesh changes over time
    AnimationTimer timer = new AnimationTimer()
        {
            @Override
            public void handle(long l)
            {
                changePoints();
            }
        };
        timer.start();

    //Booking
    final Group parent = new Group(brick);
    getChildren().addAll(parent);
  }



//The points of the mesh get transformed.
//To that aim, the coordinates of points are read,
//modified and then reinserted into the mesh
private void changePoints()
{
    int n = mesh.getPoints().size();
    float [] pointsFloat = new float[n];
    float factor = 1.001f;
    //Reading and modifying.
    //Modification = global enlargement
    for(int i = 0; i < n; i++)
        pointsFloat[i]    = factor*mesh.getPoints().get(i);
    //Reinsertion
    mesh.getPoints().setAll(pointsFloat, 0, n);
}
```

```
}//End of class Brick



@Override public void start(Stage primaryStage)
        throws Exception
{
    init(primaryStage);
    primaryStage.show();
    play();
}


public static void main(String[] args)
{
    launch(args);
}
}//End of main class N2 EvoBrick
```

3 ***Exercise.*** *Run the previous program and play with the code.*

4 ***Exercise.*** *Compose an animation in which the cube is enlarged only along the x-coordinate.* Answer

1.2 Changing the texture of a cube

We already know that to deform the mesh of a cube one must retrieve the points of the mesh, change them and store again them in their place. Following this very same idea one can change its texture.

5 ***Exercise.*** *Compose an animation of a cube in which the texture softly evolves over time from covering a quarter of the image to its full size.* Answer

1.3 For more than fun

Our purpose in this section is to present the simplest model of a planet to explain the origin of seasons, which are of paramount biological importance. Our intention is to show the temperature of a place by its color.

6 ***The Earth as a spherical mesh with texture.***

The following construction is of universal application: to design the mesh of a shape we take a rectangle in the plane and stretch or squeeze or bend it appropriately to fit into the shape.

In technical words and for this case, a rectangle of height π and width 2π is transformed into a sphere by means of spherical coordinates. The coordinates of the plane R^2 are (u,v) and those of the space R^3 are (x,y,z). The spherical transformation goes from the rectangle to the 3D space and produces a sphere of radius ρ :

$x = \rho sinvcosu$
$y = \rho sinvsinu$
$z = \rho cosv$

We say that u is the horizontal angle that runs from 0, along the positive $X-AXIS$, to 2π to make a complete tour. On the other hand, v is the vertical angle that runs from 0, along the positive $Z-AXIS$, to π to run for half a complete tour. If the radius ρ is a constant, then we get the shell of the sphere and to get the filled sphere, the radius must vary from 0 to its maximal allowed value. In our case, we need the shell, so the radius is constant.

It happens that the exposed mathematics is not enough to determine the mesh. Thus, a lot of work in situ with trial and error must be done to fill in the details. The code follows:

```
/*
    Program N6 Planet.
    Planet Earth is modelled as
    a spherical mesh with texture.
 */


package Programs;

//Files in the hosting project
import Tools.Xform;
import Tools.Cameras;
import Tools.Axes;
//Files in JavaFX
```

```
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.control.MenuBar;
import javafx.scene.image.Image;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.CullFace;
import javafx.scene.shape.DrawMode;
import javafx.scene.shape.MeshView;
import javafx.scene.shape.TriangleMesh;
import javafx.scene.transform.Rotate;
import javafx.stage.Stage;



public class Planet extends Application
{


    final Group root = new Group();
    //Citation from outside file
    final Xform planet = new Xform();
    final Xform guidesXform = new Xform();
    //Citation from outside planet
    Axes axes = new Axes();
    private static boolean print = false;

@Override
public void start(Stage primaryStage)
{
    primaryStage.setTitle("Planet Earth");
    //The whole drawing  is created
    //as a JavaFX space-like object
    final Group parent = new Group(create3dContent());

    //Window of the scene
```

```
    final Scene scene = new Scene(root, 900, 900, true);
    //Background
    scene.setFill(Color.BLACK);
    //Citation from an outside file
    Cameras camera = new Cameras();
    camera.buildCamera();
    camera.handleMouse(scene);
    primaryStage.setScene(scene);
    primaryStage.show();

    axes.buildAxes();
    axes.handleKeyboard(scene);

    scene.setCamera(camera.camera);
    root.getChildren().addAll(parent, axes.axesXform);

}

//Manager of graphic
private Node create3dContent()
{
    //Shape
    Graphic b = new Graphic();
    return new Group(b);
}


//A mesh is coverted into a meshView.
private class Graphic extends Group
{
final Rotate rx = new Rotate(0,Rotate.X_AXIS);
final Rotate ry = new Rotate(0,Rotate.Y_AXIS);
final Rotate rz = new Rotate(0,Rotate.Z_AXIS);

//The ambient of the graphic is a box
//of measures a, b, and c.
private  Graphic()
  {
    //Planet Earth as a mesh with texture
    Xform planet = MeshSphere(300f,10, Color.GREEN);
```

```
    //Illumination
    PointLight pointLight;
    pointLight = new PointLight(Color.ANTIQUEWHITE);
    pointLight.setTranslateX(0);
    pointLight.setTranslateY(0);
    pointLight.setTranslateZ(0);
    //Booking
    getChildren().addAll(planet, pointLight  );
  }
}//End of class Graphic



//The mesh of a sphere.
//Pi has  n subdivisions
 Xform MeshSphere(float rho,
        int nSubdivisions,
        Color color)
{
    TriangleMesh mesh = new TriangleMesh();

    float v = 0;

    float deltaAngle = (float) Math.PI /nSubdivisions;

    System.out.println(deltaAngle);
    //Faces are built.
    //Two triangles for each rectangle.
    int n = nSubdivisions;
    int m = 2*n;
    int sign = 1;
    //Points of mesh and of texture are built
    for (int j = 0; j <= n ; j++)
    {
        float u = 0;
        for (int i = 0; i <= m ; i++)
        {
```

```
            //A minus is inserted to get the correct orientation
            //of resultant triangles
            float x
                = (float) (rho * Math.sin(v) * Math.cos(-u));
            float y
                = (float) (rho * Math.sin(v) * Math.sin( -u));
            float z
                = (float) (rho * Math.cos(v));


            mesh.getPoints().addAll(x,y,z);
            //Texture coordinates
            float tx = i* 1.0f/m;
            float ty = j* 1.0f/n;
            //Booking
            mesh.getTexCoords().addAll(tx, ty);

            if(print)
                System.out.println(
                    x + "  "  + y + "  "  + z + "  " +
                    tx + "  "  + ty + "  " );

            u  = u + deltaAngle ;

        }
        v =  v + deltaAngle ;
    }


    //There are nm tiny rectangles.
    //Each rectangle has 2 triangles

    for (int j = 0; j < n; j++)
    {
        for (int i = 0; i < m; i++)
        {
            //Points of rectangle of the mesh
            int p00 = j * (m+1) + i;
            int p01 = p00 + 1;
            int p10 = p00 + (m+1);
```

```
            int p11 = p10 + 1;
            //Points of rectangle of texture
            int tc00 = j * (m+1) + i;
            int tc01 = tc00 + 1;
            int tc10 = tc00 + (m+1);
            int tc11 = tc10 + 1;

            if ( j < n-1)
            {
                //First triangle
                mesh.getFaces().addAll(
                        p00, tc00, p10, tc10, p11, tc11);
                //second triangle
                mesh.getFaces().addAll(
                        p11, tc11, p01, tc01, p00, tc00);
            }
            else
            //second triangle alone
            mesh.getFaces().addAll(
                    p11, tc11, p01, tc01, p00, tc00);
        }
    }


     //Wallpaper is gotten
    //Mapa Mundi clip art, shared by Mirian 06-13-2013
    //http://www.clker.com/clipart-mapa-mundi.html
    Image wallpaper = new Image(getClass()
            .getResourceAsStream("PlanetEarth.png"));
    //Declaration of material for painting the pyramid
    PhongMaterial material = new PhongMaterial();
    //Wallpaper is declared as covering material
    material.setDiffuseMap(wallpaper);
    //material.setDiffuseColor(Color.ALICEBLUE);
    MeshView  meshView = new MeshView(mesh);
    meshView.setMaterial(material);
    //meshView.setDrawMode(DrawMode.LINE);
    meshView.setDrawMode(DrawMode.FILL);
    meshView.setCullFace(CullFace.BACK);
```

```
        final Xform meshViewXform = new Xform();
        meshViewXform.getChildren().add(meshView);


        return meshViewXform;
    }



    public static void main(String[] args)
    {
        launch(args);
    }

    }//End of main class  N6 Planet
```

7 ***Exercise.*** *Run the previous program and play with the code.*

We have a static planet. Let it move.

8 ***Exercise.*** *Endow the planet with rotation and translation. Hint: Program L44 Jumps5 of Vol XII.* Answer

9 ***Exercise.*** *Make your planet into an enlarging ellipsoid.* Answer

Let us pass now to solve the task of measuring and showing the degree of illumination of some points in the Earth to show local temperature. We plan to show local temperature by superimposing a colored square: hot will be shown by red, warm by green and cold by blue. To that aim, we will need to learn to modify the wallpaper that serves us to draw the Earth. Apart from looking to following programs, the Reader is invited to learn from the source (Hommel [2], 2013).

10 ***An image pixel by pixel.*** *The next code allows to draw an image pixel by pixel.*

```
/*
 * Program N10 PixelLab
 * A JavaFX application to built an image
```

```
 * pixel by pixel.
 */
package pixellab;


import java.util.Random;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.image.PixelWriter;
import javafx.scene.image.WritableImage;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;


public class PixelLab extends Application
{

@Override
public void start(Stage primaryStage)
{
    Image image = randomPixels(300,400);
    ImageView imageView = new ImageView();
    imageView.setImage(image);
    StackPane root = new StackPane();
    root.getChildren().add(imageView);
    Scene scene = new Scene(root, 300, 250);
    primaryStage.setTitle("Image is built pixel by pixel");
    primaryStage.setScene(scene);
    primaryStage.show();
}


//An image is built pixel by pixel
private Image randomPixels(int width, int height)
{
    WritableImage img = new WritableImage(width, height);
    PixelWriter   pw  = img.getPixelWriter();
```

```
    Random        rnd = new Random();
    for (int y = 0 ; y < height ; y++)
    {
        for (int x = 0 ; x < width ; x++)
        {
            //Pixel manipulation
            pw.setColor(x, y, Color.rgb(rnd.nextInt(255),
                                        rnd.nextInt(255),
                                        rnd.nextInt(255)));
        }
    }
return img;
}


public static void main(String[] args)
{
    launch(args);
}

}//End of program N10 PixelLab
```

11 ***Exercise.*** *Run the previous program and play with the code.*

12 ***Exercise.*** *Develop a program that draws an image pixel by pixel but with a random color of the red palette.* Answer

13 ***The next code shows how to modify an image by superimposing a colorful rectangle:***

```
/*
 * Program N13 Pixellab3
 * Modification of a writable image.
 */
package pixellab;

import java.util.Random;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.Group;
```

```
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.image.ImageView;
import javafx.scene.image.PixelWriter;
import javafx.scene.image.WritableImage;
import javafx.scene.paint.Color;
import javafx.stage.Stage;


public class Pixellab3  extends Application
{
    int width = 400;
    int height = 400;

@Override
public void start(Stage primaryStage)
{
    //A green rect is prepared
    Canvas canvas = new Canvas(width, height);
    GraphicsContext gc = canvas.getGraphicsContext2D();
    gc.setFill(Color.GREEN);
    gc.fillRect(0, 0, width, height);

    //Rect is copied to a writableImage
    WritableImage wImage = canvas.snapshot(null, null);
    //Modification
    wImage = modifyImage(wImage);
    // Display image on screen
    ImageView imageView = new ImageView();
    imageView.setImage(wImage);
    Group root = new Group();
    root.getChildren().add(imageView);
    primaryStage.setScene(new Scene(root));
    primaryStage.show();
}



//A red rect is inserted at the middle of wImage
```

```
private WritableImage modifyImage(WritableImage wImage)
{
    PixelWriter pw = wImage.getPixelWriter();
    Random        rnd = new Random();
    //A small reddish rect is inserted
    for (int y = 0 ; y < 50 ; y++)
    {
        for (int x = 0 ; x < 50 ; x++)
        {
            //Pixel manipulation: red, green, blue.
            //Level of each color moves from 0 to 225.
            //Red tonality
            pw.setColor(150 + x, 200 + y, Color.rgb(225,
                                          rnd.nextInt(255),
                                          rnd.nextInt(255)));
        }
    }
    return wImage;
}


public static void main(String[] args)
{
    launch(args);
}

}//End of main class N13 Pixellab3
```

14 ***Exercise.*** *Run the previous program and play with the code.*

15 ***A false attempt to make a planet with thermometers is shown by the next code.*** *The cause of the failure is that JavaFX does not provide the user with the coordinates of moving nodes and for this reason it is impossible to know where the error is.*

```
/*
    Program N15 Planet4.
    Planet Earth is modelled as
    a spherical mesh with texture.
```

```
    It can rotate about itself and
    translate around a sun in an elliptic orbit.
    Some thermometers were installed with the task
    of showing
    temperature in three colors:
    blue (cold), green (template), red (hot);
    The strategy: try to get local values from
    JavaFX 3D.
    To the best effort of the Author, the strategy failed.
 */


package Programs;
//Files in the hosting project
import Tools.Axes;
import Tools.Cameras;
import Tools.Xform;


//Files in JavaFX
import javafx.animation.Animation;
import static javafx.animation.Animation.INDEFINITE;
import javafx.animation.AnimationTimer;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.SequentialTransition;
import javafx.animation.Timeline;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.image.PixelReader;
import javafx.scene.image.PixelWriter;
import javafx.scene.image.WritableImage;
import javafx.scene.paint.Color;
```

```
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.CullFace;
import javafx.scene.shape.DrawMode;
import javafx.scene.shape.MeshView;
import javafx.scene.shape.Sphere;
import javafx.scene.shape.TriangleMesh;
import javafx.stage.Stage;
import javafx.util.Duration;


public class Planet4 extends Application
{

    private static Timeline animation;
    //The texture of the mesh changes over time
    //private static AnimationTimer timer;
    private final Group root = new Group();
    Cameras camera = new Cameras();
    Axes axes = new Axes();




    //Angular velocity
    private static final double omega = 0.001;
    //Paramters of the ellipse
    //The radius of the mother circular orbit

    private static final double a = 200;
    private static final double b = 100;
    //Time is discrete
    private static Integer t = 0;

    //The radius of the sun
    private static final Integer rSun = 10;
    //Parameters of the planet
    private static float radiusPlanet = 30f;
    private static int nSubdivisions = 12;
    private static int  initialTranslateX = 100;
```

```
    private static  Image wallpaper;
    // Create WritableImage
    private static  WritableImage wImage;
    private static int w;
    private static int h;

    private static final int[] coloredCell = new int[1000];

    private static final int threshold1 = 20;
    private static final int threshold2 = 45;

    private static final boolean print = false;
    private static final boolean printDrawSeasonsXY = false;


@Override
public void start(Stage primaryStage)
{


    primaryStage.setTitle("Planet Earth 3D");

    //Wallpaper is gotten
    //Mapa Mundi clip art, shared by Mirian 06-13-2013
    //http://www.clker.com/clipart-mapa-mundi.html
    wallpaper = new Image(getClass()
            .getResourceAsStream("PlanetEarth.png"));

    w = (int) wallpaper.getWidth();
     h = (int) wallpaper.getHeight();

     System.out.println("Image Width: " + w);
        System.out.println("Image Height: " + h);

    wImage = wI(wallpaper);

    for(int i = 0; i < 1000; i++)
        coloredCell[i] = -1; //no color
```

```
    //The whole drawing  is created
    //as a JavaFX space-like object
    final Group parent = new Group(create3dContent());

    //Window of the scene
    final Scene scene = new Scene(root, 900, 900, true);
    //Background
    scene.setFill(Color.BLACK);

    camera.buildCamera();
    camera.handleMouse(scene);
    primaryStage.setScene(scene);
    primaryStage.show();
    axes.buildAxes();
    axes.handleKeyboard(scene);
    scene.setCamera(camera.camera);
    root.getChildren().addAll(parent, axes.axesXform);
}

//Manager of graphic
private WritableImage wI(Image wallpaper)
{
    ImageView imageView = new ImageView();
    imageView.setImage(wallpaper);

     // Obtain PixelReader
        PixelReader pixelReader = wallpaper.getPixelReader();

        System.out.println("Pixel Format: " +pixelReader.getPixelFormat());


        // Create WritableImage
       wImage
                 = new WritableImage(w,h);
         PixelWriter pixelWriter = wImage.getPixelWriter();

        // Determine the color of each pixel in a specified row
        for(int readY=0;readY<wallpaper.getHeight();readY++){
            for(int readX=0; readX<wallpaper.getWidth();readX++){
```

```
                Color color = pixelReader.getColor(readX,readY);
                if (print)
                {
                System.out.println("\nPixel color at coordinates ("+
                        readX+","+readY+") "
                        +color.toString());
                System.out.println("R = "+color.getRed());
                System.out.println("G = "+color.getGreen());
                System.out.println("B = "+color.getBlue());
                System.out.println("Opacity = "+color.getOpacity());
                System.out.println("Saturation = "+color.getSaturation
                }
                // Now write a brighter color to the PixelWriter.
                color = color.brighter();
                pixelWriter.setColor(readX,readY,color);
            }
        }
       int  ww = (int) wImage.getWidth();
     int hh = (int) wImage.getHeight();

     System.out.println("wImage Width: " + ww);
        System.out.println("wImage Height: " + hh);
    return wImage;
}

//Manager of graphic
private Node create3dContent()
{
    //Shape
    Planet p = new Planet();

    Sphere sun = new Sphere(rSun);
    PhongMaterial material = new PhongMaterial(Color.GOLD);
    sun.setMaterial(material);
    //Initial position
    sun.setTranslateX(0);
    sun.setTranslateY(0);
    sun.setTranslateZ(0);
    sun.setCullFace(CullFace.BACK);
    //Solar system
```

```
    return new Group(p, sun);
}

public void play()
{
    animation.play();
}


@Override public void stop()
{
    animation.pause();
}

//A planet with rotation + translation
private class Planet extends Group
{

private  TriangleMesh mesh;
final Xform meshViewXform = new Xform();


//Constructor
//A mesh is converted into an anmated meshView.
private  Planet()
  {


    //Planet Earth as a meshView with texture
    Xform planet = MeshSphere(radiusPlanet, nSubdivisions);


    animation = new Timeline();
    //Rotation
    animation.getKeyFrames().addAll
    (
        new KeyFrame//Initial states at initial time
        (
            Duration.ZERO,
            //new KeyValue(b.ry.angleProperty(), 0d),
```

```
            //new KeyValue(b.rx.angleProperty(), 0d),
            new KeyValue(planet.rz.angleProperty(), 0d)
        ),
        new KeyFrame//Final states at final time
        (
            Duration.seconds(3),
            //new KeyValue(b.ry.angleProperty(), 360d),
            //new KeyValue(b.rx.angleProperty(), 360d),
            new KeyValue(planet.rz.angleProperty(), 360d)
        )
    );
    animation.setCycleCount(Animation.INDEFINITE);


    animation.play();


    //Translation
    //Duration of each iteration
    Duration duration = Duration.millis(10);

    //Transitions are nested:
    //a translation tT is nested into
    //a sequential transition s
    //that defines the kinematics
    //for one cycle = one year.
    SequentialTransition s = new SequentialTransition();
    s.setCycleCount(INDEFINITE);
    //Iterations for one cycle.
    //6275 found by trial and error to complete a cycle.
    //It must be synchronized with omega
    for(int i = 0; i < 6275; i++)
    {
        //Shifting values for an ellipse
        double byX =  - a*omega
                    * Math.sin(omega* t);
        double byY =  b*omega
                    * Math.cos(omega* t);
        //Translation of the planet
        TranslateTransition tT =
```

```
            new TranslateTransition(duration, planet);
        tT.setByX(byX);
        tT.setByY(byY);
        tT.setCycleCount(1);
        //Nesting of tT into s
        s.getChildren().add(tT);
        //Time updating
        t++;
    }
    s.play();

    //Booking
    getChildren().addAll(planet, axes.axesXform);
  }



//The mesh of a sphere of radius rho
//Pi has  n subdivisions
 private Xform MeshSphere(float rho,
        int nSubdivisions)
{
    mesh = new TriangleMesh();

    float v = 0;

    float deltaAngle = (float) Math.PI /nSubdivisions;

    System.out.println(deltaAngle);

    int n = nSubdivisions;
int m = 2*n;

    //Points of mesh and of texture are built
    for (int j = 0; j <= n ; j++)
    {
        float u = 0;
        for (int i = 0; i <= m ; i++)
        {
```

```
            //if (j == 0) sign = 1; else sign = -1;
            //A minus is inserted to get the correct orientation
            //of resultant triangles
            float x
                = (float) (rho * Math.sin(v) * Math.cos(-u));
            float y
                = (float) (rho * Math.sin(v) * Math.sin( -u));
            float z
                = (float) (rho * Math.cos(v));


            mesh.getPoints().addAll(x,y,z);
            //Texture coordinates
            float tx = i* 1.0f/m;
            float ty = j* 1.0f/n;
            //Booking
            mesh.getTexCoords().addAll(tx, ty);

            if(print)
                System.out.println(
                    x + "  "  + y + "  "  + z + "  " +
                    tx + "  "  + ty + "  " );

            u  = u + deltaAngle ;

        }
        v =  v + deltaAngle ;
    }


    //There are nm tiny rectangles.
    //Each rectangle has 2 triangles

    for (int j = 0; j < n; j++)
    {
        for (int i = 0; i < m; i++)
        {
            //Points of rectangle of the mesh
            int p00 = j * (m+1) + i;
            int p01 = p00 + 1;
```

```
            int p10 = p00 + (m+1);
            int p11 = p10 + 1;
            //Points of rectangle of texture
            int tc00 = j * (m+1) + i;
            int tc01 = tc00 + 1;
            int tc10 = tc00 + (m+1);
            int tc11 = tc10 + 1;

            if ( j < n-1)
            {
                //First triangle
                mesh.getFaces().addAll(
                        p00, tc00, p10, tc10, p11, tc11);
                //second triangle
                mesh.getFaces().addAll(
                        p11, tc11, p01, tc01, p00, tc00);
            }
            else
            //second triangle alone
            mesh.getFaces().addAll(
                    p11, tc11, p01, tc01, p00, tc00);
        }
    }



    //Declaration of material for painting the shape
    PhongMaterial material = new PhongMaterial();
    //Wallpaper is declared as covering material
    material.setDiffuseMap(wImage);
    //material.setDiffuseColor(Color.ALICEBLUE);
    MeshView  meshView = new MeshView(mesh);
    meshView.setMaterial(material);
    //meshView.setDrawMode(DrawMode.LINE);
    meshView.setDrawMode(DrawMode.FILL);
    meshView.setCullFace(CullFace.BACK);
```

```
    meshViewXform.getChildren().add(meshView);
    //Initial position
    meshViewXform.setTranslate(100, 0, 0);
    //Initial inclination in degrees
    meshViewXform.rx.setAngle(0);
    meshViewXform.ry.setAngle(23.4);
    meshViewXform.rz.setAngle(0);



    //The mesh changes over time
    AnimationTimer timer = new AnimationTimer()
    {
        @Override
        public void handle(long l)
        {
            wallpaper = drawSeasons();

        }
    };

    timer.start();
    return meshViewXform;
}



//
 private Image drawSeasons()
{

    //float [] pointsFloat = new float[n];
    Point3D centerOfEarth= new Point3D(
                meshViewXform.getTranslateX() + initialTranslateX,
                meshViewXform.getTranslateY(),
                meshViewXform.getTranslateZ()
            );

    if (printDrawSeasonsXY)
    System.out.println("Earth.X = "
```

```
            + centerOfEarth.getX()
  +  "Earth.Y = "
            + centerOfEarth.getY()
                );



   int n = nSubdivisions;
   int m = 2*n;

  double Flux;
   int counter = 0;


int counterOfColoredCells = 0;
PixelWriter pw = wImage.getPixelWriter();


   //Points of mesh and ??
   for (int j = 0; j < n ; j = j + 3)
   {

       for (int i = 0; i < m ; i = i+6)
       {

           counter =  j*(m+1) + i;
           float x =  mesh.getPoints().get(counter);
           float y =  mesh.getPoints().get(counter+1);
           float z =  mesh.getPoints().get(counter+2);

           // Point3D p = new Point3D(x + initialTranslateX, y, z);
           Point3D p = new Point3D(x, y, z);
           Point3D radiusVector = p.subtract(centerOfEarth);

           double distanceToSun = p.distance(0,0,0);

           double cosinus = radiusVector.dotProduct(p)
                   / ( radiusVector.magnitude()* p.magnitude() );
           Flux = 100000* cosinus/(p.magnitude() * p.magnitude());
```

```
            if ( ( j == 6) & (i == 6))
                System.out.println(" cosinus = " + cosinus
                        + "  Flux = " + Flux
                + "distanceToSun = " + distanceToSun
                + "Magnitude = " + p.magnitude());


            int tx = (i* w/m);
            int ty = (j* h/n);

            int xx;
            int yy;

            int oldColor =  coloredCell[counterOfColoredCells];
            int newColor;
            if ( Flux> threshold2 ) newColor = 0; //red
            else if ( Flux> threshold1 ) newColor = 1; //green
            else newColor = 2; //blue

            if ( newColor == oldColor);
            else


    //A small colored rect is inserted at (xx,yy)
    //on wImage
    for (int l = 0 ; l < 20 ; l++)
    {
        for (int k = 0 ; k < 20 ; k++)
        {

            xx = tx + k;
            yy = ty + l;
            if (xx > w) xx =  w;
            if (yy > h) yy =  h;


            //Pixel manipulation: rgb = red, green, blue.
            //Level of each color moves from 0 to 225.
            int red;
```

```
            int green;
            int blue;

            if ( Flux > threshold2 )//rec is red
            {
                red = 225;
                green = 0;
                blue = 0;
                coloredCell[counterOfColoredCells] = 0;
            }
            else if  ( Flux > threshold1 )//rec is green
            {
                red = 0;
                green = 225;
                blue = 0;
                coloredCell[counterOfColoredCells] = 1;
            }

            else //rec is blue
                {
                    red = 0;
                    green = 0;
                    blue = 225;
                    coloredCell[counterOfColoredCells] = 2;
                }

            pw.setColor(xx, yy , Color.rgb(red, green, blue));
        }
    }


    counterOfColoredCells++;


            }

      }

    ImageView imageView = new ImageView();
```

```
    // Display image on screen
    imageView.setImage(wImage);

    return wImage;
}

}//End of inner class Planet



public static void main(String[] args)
{

    launch(args);
}

}//End of main class N15 Planet4
```

16 ***Exercise.*** *Run the previous program and play with the code. Make some experiment to prove that the code is a flaw.* Answer

17 ***Deciding when a point gets illuminated.***

As shown by the following figure, there is simple procedure to determine when a point must be illuminated and when not.

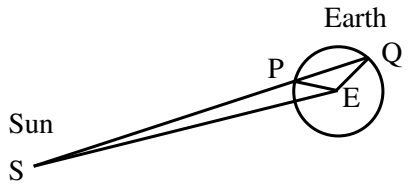


Figure 1.0. The center of the sun is at S, that of the earth at E. A ray departing from the sun crosses the surface of the Earth at two points, P and Q. The first is illuminated, the second not. How are we going to program this discrimination? It suffices to take into account that the angle SEP is acute while angle SEQ is obtuse.

How are we going to discriminate which point along a ray departing from the Sun must be illuminated? Since we work with vectors, we measure the angle in within vector SP and EP, which must be obtuse with a negative cosine while the angle in within SQ and EQ must be acute with a positive cosine. Moreover, we suppose that the flux of light at P is proportional to the absolute value of the cosine, so if the ray of light falls perpendicular to the surface, the angle is zero, the absolute value of cosine +1 and the flux is maximal. Instead of taking absolute values, we add a minus sign.

We are ready now to pass to the implementation of thermometers but with our own hands while javaFX works from in within a second plane.

18 ***Let us do it by ourselves.*** *If JavaFX refuses to provide coordinates, we must build our own system for tracking them. A delicate problem arises in regard with synchronization of our coordinates and JavaFX animations. To solve this problem, we make our own system and enslave JavaFX animations to it. We have already done this in Vol XII when working with worms and we found that this functions nicely. We incorporate some Matrix algebra, taken from Vol IV, where we learned the fundamental concepts behind 3D graphics. We also adapt a Timer class from the literature (Pollastrini [7], 2014). The code follows:*

```
/*
    Program N18 Planet5.
    Planet Earth is modelled as
    a spherical mesh with texture.
    It can rotate about itself and
    translate around a sun in an elliptic orbit.
    Some thermometers were installed with the task
    of showing
    temperature in three colors:
    blue (cold), green (template), red (hot);
    The strategy: do it by yourself. In this case,
    a 3d matrix of points is defined
    to which JavaFX objects are enslaved.
 */


package Programs;
//Files in the hosting project Programs
import Tools.Axes;
import Tools.Cameras;
```

```
import Tools.Xform;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.Executors;
import java.util.concurrent.ThreadFactory;

//Files in JavaFX

import javafx.animation.SequentialTransition;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.concurrent.ScheduledService;
import javafx.concurrent.Task;
import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.Node;
//import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.image.PixelReader;
import javafx.scene.image.PixelWriter;
import javafx.scene.image.WritableImage;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.CullFace;
import javafx.scene.shape.DrawMode;
import javafx.scene.shape.MeshView;
import javafx.scene.shape.Sphere;
import javafx.scene.shape.TriangleMesh;
import javafx.stage.Stage;
import javafx.util.Duration;


public class Planet5 extends Application
{
    private final Group root = new Group();
    private final Cameras camera = new Cameras();
    //Axes : ctrl + A
```

```
    private final Axes axes = new Axes();

    //Planet Earth
    private static Xform planetXform;
    //Angular velocity of the earth around its Z axis
    private static final double spinVelocity = 0.2;
    //Angular velocity of the Sun-earth vector
    private static final double omega = 0.05;
    //Translating movement can be turned off
    private static final boolean translation = true;
    //Paramters of the ellipse
    //The radius of the mother circular orbit

    //Parameters of the elliptical orbit.
    private static final double a = 20;
    private static final double b = 10;


    //The radius of the sun
    private static final Integer rSun = 10;
    //Parameters of the planet
    private static final float radiusPlanet = 30f;

    //Mesh of  planet
    private  static TriangleMesh mesh;
    //Number of vertical subdivisions
    private static final int nSubdivisions = 12;

    //Linear array with coordinates of points
    //of Earth centered at the origin and with
    //rotations around the Z axis
    private static float[] rotPoints;
    //The center of the Earth with its initial translate
    private static Point3D centerOfEarth
            =  new Point3D(100, 0, 0);
    //Obliquite of the Earth in degrees
    private static final double  obliquite = 23.4;

    //Scheduler
    private final EarthTimer timer = new EarthTimer();
```

```
//Drawing for the mesh
private static  Image wallpaper;
// Create WritableImage
private static  WritableImage wImage;
//Horizontal and vertical measures
private static int w;
private static int h;
//colors of thermometers
private static final int[] colorOfCell = new int[1000];

//Thermometers have three levels:
//cool = blue; green = medium; hot = red.
//Threshold from cool to warm:
private static final int threshold1 = 0;
//Threshold from   warm to hot:
private static final int threshold2 = 10;

private static final boolean printToWritableImage = false;

private static final boolean printUpdatePoints = false;
private static final boolean printMeshSphere = false;

private static double basisRot[][] = new double[3][3];
private static double matrixRotz[][] = new double[3][3];

//Points of a centered copy of the Earth with
//rotation around the Z axis
private final List<Point3D> Points = new ArrayList<>();
//Waited time since last iteration
private static double deltaTime;
//Local time
private static double time;
private static final Xform meshViewXform = new Xform();
```

```
@Override
public void start(Stage primaryStage)
{

    primaryStage.setTitle("Planet Earth 3D");

    //Wallpaper is gotten
    //Mapa Mundi clip art, shared by Mirian 06-13-2013
    //http://www.clker.com/clipart-mapa-mundi.html
    wallpaper = new Image(getClass()
            .getResourceAsStream("PlanetEarth.png"));

    w = (int) wallpaper.getWidth();
    h = (int) wallpaper.getHeight();

    System.out.println("Image Width: " + w);
    System.out.println("Image Height: " + h);
    //Wallpaper is made into a modifiable image
    wImage = toWritableImage(wallpaper);

    for(int i = 0; i < 1000; i++)
        colorOfCell[i] = -1; //no color


    //The whole drawing  is created
    //as a JavaFX space-like object
    final Group solarSystem = new Group(create3dContent());
    Xform solarSytemXform = new Xform();
    solarSytemXform.getChildren().add(solarSystem);
    solarSytemXform.setScale(1);

    //Window of the scene
    final Scene scene = new Scene(root, 900, 900, true);
    //Background
    scene.setFill(Color.BLACK);

    camera.buildCamera();
    camera.handleMouse(scene);
    primaryStage.setScene(scene);
    primaryStage.show();
```

```
    axes.buildAxes();
    axes.handleKeyboard(scene);
    scene.setCamera(camera.camera);
    root.getChildren().addAll(solarSytemXform
            , axes.axesXform);
    timer.start();
}

//Image  is made into a modifiable, writable image
private WritableImage toWritableImage(Image wallpaper)
{
    ImageView imageView = new ImageView();
    imageView.setImage(wallpaper);

    // Obtain PixelReader
    PixelReader pixelReader = wallpaper.getPixelReader();

    System.out.println("Pixel Format: "
            + pixelReader.getPixelFormat());


    // Create WritableImage
    wImage = new WritableImage(w,h);
     PixelWriter pixelWriter = wImage.getPixelWriter();

    // Determine the color of each pixel in a specified row
    for(int readY=0;readY<wallpaper.getHeight();readY++){
        for(int readX=0; readX<wallpaper.getWidth();readX++){
            Color color = pixelReader.getColor(readX,readY);
            if (printToWritableImage)
            {
            System.out.println(
                    "\nPixel color at coordinates ("+
                    readX+","+readY+") "
                    +color.toString());
            System.out.println("R = "+color.getRed());
            System.out.println("G = "+color.getGreen());
            System.out.println("B = "+color.getBlue());
            System.out.println("Opacity = "
                    + color.getOpacity());
```

```
            System.out.println("Saturation = "
                    + color.getSaturation());
            }
            // Now write a brighter color to the PixelWriter.
            color = color.brighter();
            pixelWriter.setColor(readX,readY,color);
        }
    }
    int  ww = (int) wImage.getWidth();
    int hh = (int) wImage.getHeight();

    System.out.println("wImage Width: " + ww);
       System.out.println("wImage Height: " + hh);
    return wImage;
}


//========== Matrix algebra=============

private static void printMatrix(double m[][], String title)
{
    System.out.println("This is the matrix " + title);
    for(int i=0;i<3;i++)
        System.out.println(m[i][0] + " " + m[i][1]+ " "
                                              + m[i][2]);
    System.out.println();
}

//The transpose of a matrix
  private static  double[][] matrixTranspose(double m[][])
  {
    double[][] helpM = new double[3][3];
    for(int i=0; i<3; i++)
        for(int j=0; j<3; j++)
            helpM[i][j] =m[j][i];
    return helpM;
  }

  //The multiplication of two matrixes
  private static  double[][] matrixMult(double p[][],
```

```
                                          double q[][])
{
  double[][] helpM = new double[3][3];

  for(int i=0; i<3; i++)
      for(int j=0; j<3; j++)
          helpM[i][j] =0;

  for(int i=0; i<3; i++)
      for(int j=0; j<3; j++)
          for(int k=0; k<3; k++)
              helpM[i][j] =helpM[i][j] + p[i][k]*q[k][j];
 return helpM;
}

//The multiplication of a matrix times a vector
private static  double[] matrixTimesVector(double m[][],
                                           double v[])
{
  double[]  helpV = new double[3];
  for(int i=0; i<3; i++)
      helpV[i] =0;
  for(int i=0; i<3; i++)
      for(int j=0; j<3; j++)
          helpV[i]  =helpV[i]  + m[i][j]*v[j];
  return helpV;
}



//To find a matrix of a rotation, one must find first
//an orthogonal basis with 3 vectors
//The third vector is the axis of rotation
private static  double[] thirdVector(double[] v)
{
  double a = v[0];
  double b = v[1];
  double c = v[2];
  double norm = Math.sqrt(a*a + b*b + c*c);
  //System.out.println("norm = " + norm);
```

```
    double[] helpV = new double[3];
    helpV[0] = a/norm;
    helpV[1]= b/norm;
    helpV[2] = c/norm;
    /*System.out.println("Third vector " + helpV[0]+
                                    helpV[1]+helpV[2]);*/
    return helpV;
  }

  //The second vector is perpendicular to the first
  private static  double[] secondVector(double[] v)
  {
    double[] helpV = new double[3];
    double   x,y,z;
    double a = v[0];
    double b = v[1];
    double c = v[2];
    x=b;
    y = -a;
    z=0;
    if (b==0)
       {
            x=c;
            z = -a;
            y=0;
       }
    if (a==0)
       {
            y=c;
            z = -b;
            x=0;
       }
    double norm = Math.sqrt(x*x + y*y + z*z);

    helpV[0] = x/norm;
    helpV[1]  = y/norm;
    helpV[2]  = z/norm;
    return helpV;
  }
```

```
 //The first vector is the crux product of the other 2.
 private static  double[] firstVector(double[] v,
                                      double[] w)
 {
   double a,b,c;
   double[] helpV = new double[3];
   a=  (v[1]*w[2] -w[1]*v[2]);
   b= - (v[0]*w[2]-w[0]*v[2] );
   c=  (v[0]*w[1]  -w[0]*v[1] );
   double norm = Math.sqrt(a*a + b*b + c*c);
   helpV[0] = a/norm;
   helpV[1]  = b/norm;
   helpV[2]  = c/norm;
   return helpV;

 }


//An axis (a,b,c) generates a basis B
 private static  double[][] basisB(double v[])
 {
   double[][]  helpM = new double[3][3];
   double[]   helpT, helpS, helpF;

   helpT = thirdVector(v);
   helpS = secondVector(v);
   helpF = firstVector(helpT,helpS);
   for(int i=0; i<3;i++)
   {
    helpM[i][0] = helpF[i];
    helpM[i][1] = helpS[i];
    helpM[i][2] = helpT[i];
   }

   return helpM;
 }
```

```
//A rotation is defined by an axis v that crosses
//through 0,
//and a counterclockwise angle theta of rotation
private static  double[][] mRotation(double v[],
                                     double theta)
{
    matrixRotz= zRot(theta);
    basisRot = basisB(v);
    //printMatrix(matrixRotz, "Rotz");
    //printMatrix(basisRot, "BasisRot");
    double[][]  helpM = new double[3][3];
    helpM = matrixMult(matrixRotz,
                               matrixTranspose(basisRot));
    helpM = matrixMult(basisRot,helpM);

    //printMatrix(helpM, "matrixRot");
    //Test: helpM*helpM^T must be the identity
    /*helpM = matrixMult(helpM,matrixTranspose(helpM));
    printMatrix(helpM, "identity =R*R^T ");*/
    return helpM;
}


//Rotation about the z-axis.
private static  double[][] zRot(double theta)
{
    double[][]  mRotz = new double[3][3];
    mRotz[0][0] = Math.cos(theta);
    mRotz[0][1] = Math.sin(theta);
    mRotz[0][2] = 0;
    mRotz[1][0] = -Math.sin(theta);
    mRotz[1][1] = Math.cos(theta);
    mRotz[1][2] = 0;
    mRotz[2][0] = 0;
    mRotz[2][1] = 0;
    mRotz[2][2] = 1;
    return mRotz;
}

//Rotation about the y-axis.
```

```
private static  double[][] yRot(double theta)
 {
    double[] yAxis = {0,1,0};
    double[][]  mRoty = mRotation(yAxis,  theta);
    return mRoty;
 }



  //================================

//Manager of graphic
private Node create3dContent()
{
    //Shape
    Planet planet = new Planet();
    Sphere sun = new Sphere(rSun);
    PhongMaterial material = new PhongMaterial(Color.GOLD);
    sun.setMaterial(material);
    //Initial position
    sun.setTranslateX(0);
    sun.setTranslateY(0);
    sun.setTranslateZ(0);
    sun.setCullFace(CullFace.BACK);

    /*
    //Illumination by the sun
    //This MALFUNCTIONS.
    PointLight pointLight;
    pointLight = new PointLight(Color.GOLD);
    pointLight.setTranslateX(0);
    pointLight.setTranslateY(0);
    pointLight.setTranslateZ(0);
    Group solarSystem = new Group(planet,sun, pointLight);
    */


    //Booking
    Group solarSystem = new Group(planet,sun);
    return solarSystem;
```

```
}


//A planet with rotation + translation
private class Planet extends Group
{


//Constructor
//A mesh is converted into an animated meshView.
private  Planet()
{
    //Planet Earth as a meshView with texture
    planetXform = MeshSphere(radiusPlanet, nSubdivisions);
    //Booking
    getChildren().addAll(planetXform, axes.axesXform);
}



//The mesh of a sphere of radius rho
//Pi has  n subdivisions
 private Xform MeshSphere(float rho,
        int nSubdivisions)
{
    mesh = new TriangleMesh();

    float v = 0;
    //Angle of a subdivision
    float deltaAngle = (float) Math.PI /nSubdivisions;

    System.out.println("Delta alngle = " + deltaAngle);
    //Number of vertical subdivisions
    int n = nSubdivisions;
    //Number of horizontal subdivisions
    int m = 2*n;


    //Points of mesh and of texture and faces are built
    //All math is made in double type.
```

```
//Vertical counter
for (int j = 0; j <= n ; j++)
{
    float u = 0;
    //Horizontal counter
    for (int i = 0; i <= m ; i++)
    {
        //From the plane to 3D space, spherical coordinates
        //A minus is inserted to get the correct orientation
        //of resultant triangles
        double x
            = rho * Math.sin(v) * Math.cos(-u);
        double y
            = rho * Math.sin(v) * Math.sin( -u);
        double z
            =  rho * Math.cos(v);
        //A point is generated
        Point3D point = new Point3D(x, y, z);
        //List is populated
        Points.add(point);

        //Preparing type compatibility
        float xf = (float) x;
        float yf = (float) y;
        float zf = (float) z;
        //Mesh is populated as a
        //float carbon copy of Points
        mesh.getPoints().addAll(xf,yf,zf);

        //Texture coordinates
        float tx = i* 1.0f/m;
        float ty = j* 1.0f/n;
        //Booking
        mesh.getTexCoords().addAll(tx, ty);

        if(printMeshSphere)
            System.out.println(
                x + "  "  + y + "  "  + z + "  " +
                tx + "  "  + ty + "  " );
```

```
            u  = u + deltaAngle ;

        }
        v =  v + deltaAngle ;
    }


    //There are nm tiny rectangles.
    //Each rectangle has 2 triangles

    for (int j = 0; j < n; j++)
    {
        for (int i = 0; i < m; i++)
        {
            //Points of rectangle of the mesh
            int p00 = j * (m+1) + i;
            int p01 = p00 + 1;
            int p10 = p00 + (m+1);
            int p11 = p10 + 1;
            //Points of rectangle of texture
            int tc00 = j * (m+1) + i;
            int tc01 = tc00 + 1;
            int tc10 = tc00 + (m+1);
            int tc11 = tc10 + 1;

            if ( j < n-1)
            {
                //First triangle
                mesh.getFaces().addAll(
                        p00, tc00, p10, tc10, p11, tc11);
                //second triangle
                mesh.getFaces().addAll(
                        p11, tc11, p01, tc01, p00, tc00);
            }
            else
            //second triangle alone
            mesh.getFaces().addAll(
                    p11, tc11, p01, tc01, p00, tc00);
        }
    }
```

```
    //Declaration of material for painting the shape
    PhongMaterial material = new PhongMaterial();
    //Wallpaper is declared as covering material
    material.setDiffuseMap(wImage);
    //material.setDiffuseColor(Color.ALICEBLUE);
    MeshView  meshView = new MeshView(mesh);
    meshView.setMaterial(material);
    //meshView.setDrawMode(DrawMode.LINE);
    meshView.setDrawMode(DrawMode.FILL);
    meshView.setCullFace(CullFace.BACK);

     //Initial position
    meshViewXform.setTranslate(100, 0, 0);


    meshViewXform.getChildren().add(meshView);


    return meshViewXform;
}



}//End of inner class Planet


//A point3D is printed to console
private void point3DtoString(Point3D point, String name)
{
    System.out.println(name);
    System.out.println(" X = " + point.getX());
    System.out.println(" Y = " + point.getY());
    System.out.println(" Z = " + point.getZ());
    System.out.println(" Magnitude = " + point.magnitude());

}
```

```
//Thermometers are drawn ad updated
private Image drawThermeters()
{

    int n = nSubdivisions;
    int m = 2*n;
    double Flux;
    int counter;
    int indexOfCell = 0;
    PixelWriter pw = wImage.getPixelWriter();


    //point3DtoString(centerOfEarth, "centerOfEarth");

    /*
    //To carry out a test, change loops below by these
    for (int j = 6; j < n ; j = j + 300)
    {

        for (int i = 6; i < m ; i = i + 600)
        {
    */

    //Points of mesh are selectively
    //visited to put there a colored rect
    for (int j = 3; j < n ; j = j + 3)
    {

        for (int i = 6; i < m ; i = i + 6)
        {

            counter =  j*(m+1) + i;
            //Coordinates of points of Earth without translation
            double x =  rotPoints[3*counter];
            double y =  rotPoints[3*counter+1];
            double z =  rotPoints[3*counter+2];
            //To compensate for correct orientation.
            //Found by analyzing printed coordinates
            //of tracer point
```

```
y = -y;


//Point in the surface  of Earth as a vector
//that starts at its centre
Point3D radiusVector = new Point3D(x, y, z);
//Point referred to sun coordinates
Point3D p = centerOfEarth.add(radiusVector);
//Report to console
point3DtoString(p, "Marked point" + i + "  " +j);


//Cosine of the angle in within Sun - Point
//radio vector and center of Earth-Point radio vector
double cosine = p.dotProduct(radiusVector)
    / (  p.magnitude()
        * radiusVector.magnitude() );
//Theory: flux is directly proportional to cosine
//and inversely to distance to sun squared
Flux = -1000000* cosine
    /(p.magnitude() * p.magnitude());


//Texture coordinates of scanned points
//where a little rect will be drawn
int tx = (i* w/m);
int ty =  (j* h/n);

//Texture coordinates of pixelated point
int xx;
int yy;
//Color of rectanular cell
int oldColor =  colorOfCell[indexOfCell];
int newColor = 0; //blue
if ( Flux> threshold1 ) newColor = 1; //green
if ( Flux> threshold2 ) newColor = 2; //red
//Color definition according to received flux
if ( newColor == oldColor); //do nothing
else
```

```
//A small colored rect of measures 20 x 20
//is drawn pixel by pixel at (tx,ty) over wImage
for (int l = 0 ; l < 20 ; l++)
{
    for (int k = 0 ; k < 20 ; k++)
    {

        xx = tx + k;
        yy = ty + l;
        if (xx > w) xx =  w;
        if (yy > h) yy =  h;

        //Flux trifurcation in three ranges.
        //Pixel manipulation: rgb = red, green, blue.
        //Level of each color moves from 0 to 225.
        //Initial color = 0 = blue;
        int red = 0;
        int green = 0;
        int blue = 225;
        colorOfCell[indexOfCell] = 0;

        if  ( Flux > threshold1 )//green
        {
            red = 0;
            green = 225;
            blue = 0;
            colorOfCell[indexOfCell] = 1;
        }

        if ( Flux > threshold2 )//red
        {
            red = 225;
            green = 0;
            blue = 0;
            colorOfCell[indexOfCell] = 2;
        }

        //Pixel coloring
        pw.setColor(xx, yy , Color.rgb(red, green, blue));
```

```
        }
    }


        //Tracer point    is printed
        if ( ( j == 6) & (i == 6))
            System.out.println(
                " Color = " + colorOfCell[indexOfCell]
                + "  Flux = " + Flux
                + "  DistanceToSun = " + p.magnitude());
        indexOfCell++;
    }

    }


    ImageView imageView = new ImageView();
    //Display image on screen
    imageView.setImage(wImage);

    return wImage;
}//End of  drawThermometers();


/**
 * ****************************************************************
 * EarthTimer
 * It is a  timer class for updating the mesh of the Earth-
 * Adapted from ClothTimer in ClothMesh.java in   FXyzLib, IV/2015.
 * http://birdasaur.github.io/FXyz/
 * @author Jason  Pollastrini aka jdub1581
 ******************************************************************
 */
/**
 * Timer to handle Earth updates
 */
private class EarthTimer extends ScheduledService<Void>
{

    private final long ONE_NANO = 1000000000L;
```

```
    private final double ONE_NANO_INV = 1f / 1000000000L;

    private long startTime, previousTime;

    private final double fixedDeltaTime = 0.16;
    private int leftOverDeltaTime, timeStepAmt;

    private final NanoThreadFactory tf;
    private boolean paused;

public EarthTimer()
{
    super();

    this.setPeriod(Duration.millis(50));

    this.tf = new NanoThreadFactory();
    this.setExecutor(Executors.newSingleThreadExecutor(tf));
}



//JavaFX thread for  rotation + translation of the Earth
private void updateMesh()
{

    mesh.getPoints().setAll(rotPoints);

    //Duration of each iteration
    Duration duration = Duration.millis(1);

    //Transitions are nested:
    //a translation tT is nested into
    //a sequential transition s
    //that defines the kinematics
    //for one cycle = one year.
    SequentialTransition s = new SequentialTransition();
    s.setCycleCount(1);

    if (printUpdatePoints)
```

```
        System.out.println("time = " + time
                + "  deltaTime = " + deltaTime);


    //Shifting values for an ellipse
    double byX =   - a*omega
                * Math.sin(omega* time)*deltaTime;
    double byY =  b*omega
            * Math.cos(omega* time)*deltaTime;
    double byZ = 0;
    //Translation of the planet
    TranslateTransition tT =
     new TranslateTransition(duration, planetXform);
    tT.setByX(byX );
    tT.setByY(byY);
    tT.setByZ(byZ);
    tT.setCycleCount(1);
    //Nesting of tT into s
    s.getChildren().add(tT);

    if (translation)
    {
        s.play();
        centerOfEarth = centerOfEarth.add(byX, byY, byZ);
    }

    wallpaper = drawThermeters();

}



public void updateUI()
{
    updateMesh();
}

/**
* @return elapsed time as a double
*/
```

```
public double getTimeAsSeconds() {
    return getTime() * ONE_NANO_INV;
}

/**
*
* @return elapsed time as a long
*/
public long getTime() {
    return System.nanoTime() - startTime;
}

/**
*
* @return one nano second
*/
private long getOneSecondAsNano() {
    return ONE_NANO;
}

/**
*
* @return deltaTime
*/
public double getDeltaTime() {
    return deltaTime;
}

/**
*
* @return updates Timers clock values
*/
private void updateTimer()
{
    deltaTime
        = (getTime() - previousTime) * (10.0f / ONE_NANO);
    previousTime = getTime();
    timeStepAmt
        = (int) ((deltaTime + leftOverDeltaTime)
            / fixedDeltaTime);
```

```
    timeStepAmt  = Math.min(timeStepAmt, 5);
    leftOverDeltaTime
        = (int) (deltaTime - (timeStepAmt * fixedDeltaTime));
}

//This task makes the Earth rotate about the Z axis.
//Result is saved to Points, which is recycled here.
//Obliquity (initial rotation aound Y axis) is added.
//Result is saved to rotPoints which next will be
//added to translation to be   used by JavaFX
//to update points of the mesh.
@Override
protected Task<Void> createTask() {
return new Task<Void>() {
    @Override
    protected Void call() throws Exception {
        updateTimer();

        rotPoints = new float[Points.size()*3];

        for(int i = 0; i < Points.size(); i++)
        {
            double x = Points.get(i).getX();
            double y = Points.get(i).getY();
            double z = Points.get(i).getZ();
            double[] w = {x,y,z};
            time = getTimeAsSeconds();
            deltaTime = getDeltaTime();

            double angle = deltaTime * spinVelocity;

            //Delta rotation of point w around Z axis
            w = matrixTimesVector ( zRot( angle), w );

            Point3D wp = new Point3D(w[0], w[1], w[2]);
            //Updating
            Points.set(i, wp);


            //Obliquite = Initial global rotation
```

```
            //= 23.4 degrees around y axis.
            //Obliquite in radians
             double obliquiteRad = obliquite *  Math.PI /180;

            w = matrixTimesVector(yRot(obliquiteRad),w);


            rotPoints[i*3 ] = (float) (w[0] );
            rotPoints[i*3 + 1] = (float) (w[1] );
            rotPoints[i*3 + 2] = (float) (w[2] );
        }


        return null;
    }
};
}

//If task is done, its thread is closed and
//JavaFX can makes its part
@Override
protected void succeeded() {
    super.succeeded();
    updateUI();
}

@Override
protected void failed() {
    getException().printStackTrace(System.err);
}



@Override
protected void cancelled() {
    super.cancelled();
    reset();
}

@Override
```

```
public void start() {
    if (isRunning()) {
        return;
    }
    super.start();
    if (startTime <= 0) {
        startTime = System.nanoTime();
    }

}

protected void pause() {
paused = true;
if (isRunning()) {
    if (cancel()) {
        cancelled();
    }
}
}

@Override
public void reset() {
    super.reset();
    if (!paused) {
        startTime = System.nanoTime();
        previousTime = getTime();
    }
}

@Override
public String toString() {
return "EarthTimer{"
        + "startTime=" + startTime
        + ", previousTime=" + previousTime
        + ", deltaTime=" + deltaTime + ""
        + ", fixedDeltaTime=" + fixedDeltaTime
        + ", leftOverDeltaTime=" + leftOverDeltaTime
        + ", timeStepAmt=" + timeStepAmt + '}';
}
```

```
/*=========================================================

*/
private class NanoThreadFactory implements ThreadFactory {

    public NanoThreadFactory() {
    }

    @Override
    public Thread newThread(Runnable r) {
        Thread t = new Thread(r, "EarthTimerThread");
        t.setDaemon(true);
        return t;
    }

}

}//End EarthTimer=========================================


public static void main(String[] args)
{

    launch(args);
}

}//End of main class N18 Planet5
```

19 ***Exercise.*** *Run the previous program and play with the code. Play enough and make appropriate experiments to agree else disagree with the claim of the Author: the code has been correctly designed.*

1.4 The simulation of forces

Biology is the science of life. And life implies forces everywhere, every now and in many diverse forms. Let us learn how to simulate them in the simplest case, when the force creates acceleration of a point-like massive particle.

20 ***How to integrate forces.***

When we observe a displacement, we get velocity. But if the velocity increases, as when a car starts, we get acceleration. To cause an acceleration we need to use the energy of some fuel. There are some fuels more energetic than others, causing more acceleration after the same time keeping all other conditions alike. Thus, force is defined as the product of mass times acceleration. If the force is given, this is translated into acceleration, which causes an increasing velocity, which ultimately is reflected in the change of positions. We say that we integrate forces when all this is done and we can see how a body moves under the action of forces.

We have assumed that we deal with positive forces but negative forces that cause deceleration also exist, say, those caused by breaks. A friend of mine told me the following story: *I was traveling at relatively high speed and with a heavy load when a cat appeared out of the blue traversing my way right in front of me. I broke but the car began to deviate and so I tried to correct its motion and then it began to suffer strong oscillations in the direction of movement and in one of them I almost shocked against various persons outside the street. Fortunately, I was able to keep the car under control and nothing bad happened.* So, why did the car begin to go astray when he made nothing apart from breaking? The reason is that the breaking crafts were new and the exercised friction forces were unbalanced and so the wheel on one side turned over faster than the other and this caused a change in the direction of movement. But the friction forces were inhomogeneous along the same wheel and so the effect was to go astray to the right in one moment to go to the left at the next one.

There are many methods of integration that vary on the way approximations are done. A very popular one is the Verlet method that is as follows:

Let us consider an object in movement that occupies position x_{n-1} at time t_{n-1}, then passes to x_n at time t_n and then to x_{n+1} at time t_{n+1}. Velocity is the rate of change of displacement in relation to elapsed time. As usual, let us take elapsed time in within measurement as homogeneous and equal to Δt. So, velocity at time t_{n-1} is $v_{n-1} = \frac{x_n - x_{n-1}}{\Delta t}$ while velocity at time t_n is $v_n = \frac{x_{n+1} - x_n}{\Delta t}$. Now, acceleration is the rate of change of velocity with respect to time. Thus, acceleration at time t_n is

$$a_n = \frac{v_n - v_{n-1}}{\Delta t}$$

$$a_n = \frac{\frac{x_{n+1} - x_n}{\Delta t} - \frac{x_n - x_{n-1}}{\Delta t}}{\Delta t} = \frac{\frac{x_{n+1} - x_n - x_n + x_{n-1}}{\Delta t}}{\Delta t}$$

$$a_n = \frac{x_{n+1} - 2x_n + x_{n-1}}{(\Delta t)^2}$$

$$a_n(\Delta t)^2 = x_{n+1} - 2x_n + x_{n-1}$$

Clearing for x_{n+1}, we get:

$$x_{n+1} = 2x_n - x_{n-1} + a_n(\Delta t)^2$$

This is an equation for vectors that are described by 3-tuples. Used discretization imposes linear approximations so, this method creates errors (see Wikipedia $->$ Verlet integration) . So, results lost convincing power when time grows old. A natural remedy is to shorten Δt and to run tests in the field of interest to compare outputs with mathematically known solutions.

21 ***Variable time differences.***

In our simulations we will use a timer to schedule events. The problem is that a timer operates as soon as it is possible but this may depend of many factors that might be outside the application, say, CPU load. So, the previous model is not suitable for our purpose. Let us remade calculations when time differences are variable. Our setting is as follows:

We consider an object in movement that occupies position x_{n-1} at time t_{n-1}, then it passes to x_n at time t_n and then to x_{n+1} at time t_{n+1}. Intervals of time among measurements are $\Delta t_{n-1} = t_n - t_{n-1}$ and $\Delta t_n = t_{n+1} - t_n$. We might say that velocity at time t_{n-1} is $v_{n-1} = \frac{x_n - x_{n-1}}{\Delta t_{n-1}}$ while velocity at time t_n is $v_n = \frac{x_{n+1} - x_n}{\Delta t_n}$. Nevertheless, there is another assignation that has proven to be efficient: velocity at time $t_{n-1} + \Delta t_{n-1}/2$ is $v_{n-1} = \frac{x_n - x_{n-1}}{\Delta t_{n-1}}$ while velocity at time $t_n + \Delta t_n/2$ is $v_n = \frac{x_{n+1} - x_n}{\Delta t_n}$. So, we assigned velocities to the middle point of the time interval.

Now, acceleration is the rate of change of velocity with respect to time. Thus, acceleration at time t_n is

$$a_n = \frac{v_n - v_{n-1}}{\Delta t}$$

but now Δt is given by the interval of time in within middle points so, we get half delta t from the first interval and half delta t from the ensuing one:

$$\Delta t \to \Delta t_n/2 + \Delta t_{n-1}/2 = \frac{\Delta t_n + \Delta t_{n-1}}{2}$$

We get for the acceleration at time t_n:

$$a_n = \frac{\frac{x_{n+1}-x_n}{\Delta t_n} - \frac{x_n - x_{n-1}}{\Delta t_{n-1}}}{\frac{\Delta t_n + \Delta t_{n-1}}{2}}$$

22 *Exercise.* *Prove that* $x_{n+1} = x_n + (x_n - x_{n-1})\frac{\Delta t_n}{\Delta t_{n-1}} + a_n \frac{\Delta t_n + \Delta t_{n-1}}{2}\Delta t_n$. *Visit the comments of Jonathan "lonesock" Dummer (2015 [1]) to see how good is this approximation. Run a test: if in this formula we make* $\Delta t_n = \Delta t_{n-1}$, *do we get the old expression for homogeneous intervals of time? Answer*

23 *Exercise.* *Define an algorithm (there may exit many) that predicts positions when time differences are variable. You must define initial conditions in a suitable form and plug them into a recursion formula. Answer*

24 *Exercise.* *Implement your algorithm on the program that shows planet Earth circulating around the sun. Take into account that the acceleration is created by gravitational forces of central direction:*

$a = f/m$

where the magnitude of the force is

$\frac{kMm}{r^2}$

where k is a constant, M the mass of the sun, m the mass of the Earth and r the distance from the center of the Sun to the center of the Earth. Find by trial and error values for the constants in such a way that resulting video could appear as natural to the beholder (no interest in real astronomy as yet). Answer

25 *Challenge. Personalize your Planet. Say, localize your city and follow its temperature. Add a Moon. Add the effect of seas as a regulator of night temperature... you might pursue to become a climatologist.*

1.5 Conclusion

We have learned how to modify both form and texture of JavaFX 3D shapes. We feel very satisfied and expectant to see how we can use this awesome potential for the study of evolution.

Chapter 2

A random walk of mutations

`Mutations that make nonsense`

26 ***Purpose.*** *We can move every point of a mesh as arbitrarily as desired. We program such moves in tandem to see the output.*

2.1 Disfigurement of the sphere.

To gain insight, we take a sphere and subject it to a random walk of mutations that can affect each one of its points. What will happen? It is certain that a sphere can be deformed in whatever one wants. Nevertheless, nothing interesting will result because the set of shapes that are important for human beings and for biology are too few in comparison with all possible deformities. Now, are we biased by pessimism or are we in the right track? Let us see:

27 ***Exercise.*** *Develop a program to implement a random walk of mutations of the positions of points of the mesh of a sphere. Run your program to agree else reject the intuition of the Author: apart from a disfigurement, nothing will be gotten.* *Answer*

28 ***Exercise.*** *Eventually, randomness could be a great source of originality and creativeness if only it is put in the right track. So, develop the code to give a try to radial random mutations of positions to the points of the mesh of a sphere.* *Answer*

2.2 Conclusion

Purely random modifications of a mesh clearly showed why randomness is despised by the Author as an explanation of form of biological beings: interesting

biological shapes have nothing to do with shapes deformed by random walk of mutations. Nevertheless, randomness could be a great source of originality and creativeness if only it is put in the right track. So, we developed the code to give a try to radial random mutations of positions of the points of the mesh of a sphere. Found result was somehow artistic but it seems to us that it is far from biology.

Chapter 3

Parametric mutations

`Families of mutations controlled by some few parameter`

29 ***Introduction.*** *We witnessed in the previous chapter that a mesh can be mutated point by point. But, beware, this idea is deprived of biological meaning for we will be forced to imagine a separate gen for encoding the position of each one point. Instead, we might imagine that shape in biology is the result of proliferation (Michael and Davies [4], 2015)+ apoptosis (Suzanne and Steller [9], 2013) of cells whose reproduction or destruction is controlled by physic chemical processes, by mechanisms that can control the behavior of thousands of cells as in a neighborhood as in separate regions. These processes can be controlled by some few parameters that in their turn will depend on the operation of some few genes. All this happens during ontogenesis. In short, our postulate reads: biological shape depends on parameters that are dictated by genetic mechanisms whose effects are mediated by physic chemical processes.*

30 ***Purpose.*** *Our interest is to model some simple theories of the arising of biological shape. We will consider that the shape is first done in tiny size and that then it is enlarged. We will synthesize a model of the enlargement of bones that depends on as few parameters as possible. So, we will develop the* ***principal axis theory of the enlargement of shape****, which says that the enlargement of shape is in principle described by 3 parameters that regulate its size along principal axes, just as it happens with an ellipsoid. Next, we will run mutation of resultant parameters to see the creation of variability and, eventually, the erosion of shape and function.*

3.1 Animated hand.

We will produce mutant hands. These are best appreciated through animations that we will make step by step.

31 ***Exercise.*** *Animate the skeleton of the human hand. Prerequisites: 1) The skeleton must be imported using program M112 MlgrmLeftHand4 of Vol XIII. 2) Study the code of Program M29 HandOnlyBoxess12 in Vol XIII that animated a hand made out of boxes. To determine the pivot for each bone, read from the corresponding file a line that begins with* `Bounds(mm)` *. That line contains the coordinates of start and final points of the box associated to and containing the shape. Use them to calculate pivoting points.* Answer

3.2 Principal axis theory of the enlargement of shape.

Instead of looking at bones as very complex shapes, we will look at them as if they were ellipsoids, which are very simple because their geometry depends just on three measures along its axes. Precisely, those measures are the parameters that bellow will be given to mutate and that is why the corresponding theory of the enlargement of shape is called *principal axis theory of shape*.

32 ***PCA = Principal Component Analysis.***

To find the ellipsoid that best fits a set of data we use Principal Component Analysis. We adopt the explanations given by *Wikipedia: Principal component analysis*.

The major axis is such that it accounts for as much of the variability in the data as possible. Finding the largest variance amounts to find the largest eigenvector of the covariance matrix, which is symmetric. Once the first axis is found, the corresponding variance is subtracted and a second axis may be found and so on. If some axis of the ellipse is small, then the variance along that axis is also small, and so it can be omitted.

The concrete procedure is as follows:

1. To find the axes of the ellipsoid, we must first subtract the mean of each variable from the dataset to center the data around the origin.

2. Compute the covariance matrix of the data, and calculate the eigenvalues and corresponding eigenvectors of this covariance matrix.

3. Eigenvectors are sorted in descending order and corresponding eigenvectors are found. They define the axes of the ellipsoid.

4. The proportion of the variance that each eigenvector represents can be calculated by dividing the eigenvalue corresponding to that eigenvector by the sum of all eigenvalues. This is not necessary in our case.

5. To express a point in the basis determined by PCA, one forms the matrix of eigenvectors M, one vector per column, and uses the identity $y = M^T x$, where x is the vector in normal coordinates, y the vector in the eigenbasis and M^T the transpose of M that is equal to its inverse.

33 ***PCA for Java.***

Most possible, there is somewhere a specialized library for PCA for Java. Nevertheless, we prefer to work with a library with usual matrix functions, such as eigenvector decomposition. To that aim, we use *JAMA : A Java Matrix Package*. We can download the corresponding *.jar* file from (JAMA [3], 2015).

Paste the file to a suitable directory and add its path to a Java Project: under the *Projects' tab*, right click upon the name of the Project. Select *Properties*. Expand *Libraries*. Select *Add Jar/Folder*. Browse for *Jama-1.0.3.jar* and add it. Give *OK* everywhere. Next, run the following program that comes with the documentation:

```
/*
Program N33 MagicSquareExample.
Test for the use of
Jama = a Java matrix package.
Example from the documentation with slight modifications.
http://math.nist.gov/javanumerics/jama/
 */
package Programs;


import Jama.*;
import java.util.Date;

/** Example of use of Matrix Class, featuring magic squares. **/

public class MagicSquareExample {

   /** Generate magic square test matrix. **/

public static Matrix magic (int n) {
```

```
double[][] M = new double[n][n];

// Odd order

if ((n % 2) == 1) {
   int a = (n+1)/2;
   int b = (n+1);
   for (int j = 0; j < n; j++) {
      for (int i = 0; i < n; i++) {
         M[i][j] = n*((i+j+a) % n) + ((i+2*j+b) % n) + 1;
      }
   }

// Doubly Even Order

} else if ((n % 4) == 0) {
   for (int j = 0; j < n; j++) {
      for (int i = 0; i < n; i++) {
         if (((i+1)/2)%2 == ((j+1)/2)%2) {
            M[i][j] = n*n-n*i-j;
         } else {
            M[i][j] = n*i+j+1;
         }
      }
   }

// Singly Even Order

} else {
   int p = n/2;
   int k = (n-2)/4;
   Matrix A = magic(p);
   for (int j = 0; j < p; j++) {
      for (int i = 0; i < p; i++) {
         double aij = A.get(i,j);
         M[i][j] = aij;
         M[i][j+p] = aij + 2*p*p;
         M[i+p][j] = aij + 3*p*p;
         M[i+p][j+p] = aij + p*p;
      }
```

```
      }
      for (int i = 0; i < p; i++) {
         for (int j = 0; j < k; j++) {
            double t = M[i][j];
            M[i][j] = M[i+p][j]; M[i+p][j] = t;
         }
         for (int j = n-k+1; j < n; j++) {
            double t = M[i][j];
            M[i][j] = M[i+p][j]; M[i+p][j] = t;
         }
      }
      double t = M[k][0];
      M[k][0] = M[k+p][0]; M[k+p][0] = t;
      t = M[k][k];
      M[k][k] = M[k+p][k]; M[k+p][k] = t;
   }
   return new Matrix(M);
}

/** Shorten spelling of print. **/

private static void print (String s) {
   System.out.print(s);
}

/** Format double with Fw.d. **/

public static String fixedWidthDoubletoString
        (double x, int w, int d) {
   java.text.DecimalFormat fmt
           = new java.text.DecimalFormat();
   fmt.setMaximumFractionDigits(d);
   fmt.setMinimumFractionDigits(d);
   fmt.setGroupingUsed(false);
   String s = fmt.format(x);
   while (s.length() < w) {
      s = " " + s;
   }
   return s;
}
```

```
/** Format integer with Iw. **/

public static String fixedWidthIntegertoString
        (int n, int w) {
   String s = Integer.toString(n);
   while (s.length() < w) {
      s = " " + s;
   }
   return s;
}


public static void main (String argv[]) {

/*
 | Tests LU, QR, SVD and symmetric Eig decompositions.
 |
 |   n       = order of magic square.
 |   trace   = diagonal sum,
                  should be the magic sum, (n^3 + n)/2.
 |   max_eig = maximum eigenvalue of (A + A')/2,
                  should equal trace.
 |   rank    = linear algebraic rank,
 |             should equal n if n is odd,
                 be less than n if n is even.
 |   cond    = L_2 condition number,
                  ratio of singular values.
 |   lu_res  = test of LU factorization,
                  norm1(L*U-A(p,:))/(n*eps).
 |   qr_res  = test of QR factorization,
                  norm1(Q*R-A)/(n*eps).
 */

   print("\n    Test of Matrix Class, using magic squares.\n");
   print("    See MagicSquareExample.main() "
           + "for an explanation.\n");
   print("\n      n     trace       max_eig   rank      "
           + "  cond      lu_res      qr_res\n\n");
```

```
Date start_time = new Date();
double eps = Math.pow(2.0,-52.0);
for (int n = 3; n <= 32; n++) {
   print(fixedWidthIntegertoString(n,7));

   Matrix M = magic(n);

   int t = (int) M.trace();
   print(fixedWidthIntegertoString(t,10));

   EigenvalueDecomposition E =
      new EigenvalueDecomposition
         (M.plus(M.transpose()).times(0.5));
   double[] d = E.getRealEigenvalues();
   print(fixedWidthDoubletoString(d[n-1],14,3));

   int r = M.rank();
   print(fixedWidthIntegertoString(r,7));

   double c = M.cond();
   print(c < 1/eps ? fixedWidthDoubletoString(c,12,3) :
      "         Inf");

   LUDecomposition LU = new LUDecomposition(M);
   Matrix L = LU.getL();
   Matrix U = LU.getU();
   int[] p = LU.getPivot();
   Matrix R = L.times(U).minus(M.getMatrix(p,0,n-1));
   double res = R.norm1()/(n*eps);
   print(fixedWidthDoubletoString(res,12,3));

   QRDecomposition QR = new QRDecomposition(M);
   Matrix Q = QR.getQ();
   R = QR.getR();
   R = Q.times(R).minus(M);
   res = R.norm1()/(n*eps);
   print(fixedWidthDoubletoString(res,12,3));

   print("\n");
}
```

```
   Date stop_time = new Date();
   double etime = (stop_time.getTime()
           - start_time.getTime())/1000.;
   print("\nElapsed Time = " +
      fixedWidthDoubletoString(etime,12,3) + " seconds\n");
   print("Adios\n");

   //Solve a linear system with solution (1,1,1)
 double[][] array = {{1.,2.,3.},{4.,5.,6.},{7.,8.,10.}};
 Matrix A = new Matrix(array);
 //Matrix b = Matrix.random(3,1);
 double[][] bArray = {{6},{15},{25}};
Matrix b = new Matrix(bArray);
 Matrix x = A.solve(b);
 Matrix Residual = A.times(x).minus(b);
 double rnorm = Residual.normInf();
 print("\nErrorr = " + rnorm);
 print("\nx = " + x.get(0, 0));
 print("\ny = " + x.get(1, 0));
 print("\nz = " + x.get(2, 0));


 System.out.println("\n");

 //Diagonal matrix {3,4,5}
 System.out.println("Diagonal matrix {3,4,5}");
 double[][] array2 = {{3.,0.,0.},{0.,4.,0.},{0.,0.,5.}};
 Matrix A2 = new Matrix(array2);

 EigenvalueDecomposition E2 =
         new EigenvalueDecomposition
            (A2);

 double[] d2 = E2.getRealEigenvalues();




 System.out.println();
    for(int i = 0; i < 3; i++)
```

```
        System.out.println("Eigenvalue " + i + " : " + d2[i]);
    double[][] EV = E2.getV().getArray();
    for(int i = 0; i < 3; i++)
    {
        System.out.println("Eigenvector " + i + " : " );
        for (int j = 0; j <  3; j++)
        {
            System.out.println(EV[i][j]);
        }
        System.out.println();
    }

}
}//End of main class N33 MagicSquareExample
```

34 ***Exercise.*** *Run the previous program and play with the code.*

PCA allows us to align an orthogonal triad of axes with the natural directions of a bone. So, we expect the principal axes associated with the largest eigenvalue to align along the 'vertical' direction of the bone. Now, we would like to visualize that triad. To that aim, we will rotate appropriately the usual XYZ triad to fit the desired new one. The needed technology was taught to us by Euler:

35 ***The Euler Technology.*** *To transform the usual XYZ triad into another one, xyz, that is also orthonormal, we calculate the inverse transformation that rotates xyz to get XYZ (Euler Angles, Wikipedia [5], 2015).*

Our problem is to rotate xyz triad, the orthonormalized PCA basis, to fit the usual XYZ triad.

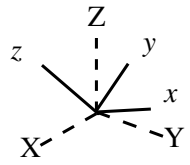


Figure 3.0. Triad xyz must be rotated to fit the triad XYZ.

To that aim, we first rotate the xyz triad around the z axis until the x axis becomes aligned with the XY plane. We need to calculate the angle of rotation γ:

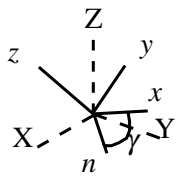


Figure 3.1. Triad xyz is rotated around z until x gets aligned with plane XY. Rotated x axis becomes n. The angle of rotation is γ.

Let $X=(1,0,0)$, $Y=(0,1,0)$ and $Z=(0,0,1)$. By the same token, $x=(a,b,c)$, $y=(d,e,f)$, $z=(g,h,i)$. Axis x becomes $n=(n_1,n_2,0)$ when aligned with the XY plane. Because n and z are orthogonal one to another:

$n\cdot z=0=n_1g+n_2h$

$n_1=-n_2\frac{h}{g}$

But n has norm one:

$n_1^2+n_2^2=1$,

${n_2}^2=1-{n_1}^2$,

$n_2=\sqrt{1-n_1^2}$

Replacing

$n_1=-\sqrt{1-n_1^2}\frac{h}{g}$

$n_1^2=(1-n_1^2)\frac{h^2}{g^2}=\frac{h^2}{g^2}-n_1^2\frac{h^2}{g^2}$

$n_1^2(1+\frac{h^2}{g^2})=\frac{h^2}{g^2}$

$n_1=\pm\frac{\frac{h}{g}}{\sqrt{1+\frac{h^2}{g^2}}}$

$n_2=-n_1\frac{g}{h}=-\pm\frac{\frac{h}{g}}{\sqrt{1+\frac{h^2}{g^2}}}\frac{g}{h}=-\pm\frac{1}{\sqrt{1+\frac{h^2}{g^2}}}$

So,

$n\cdot x=||n||||x||cos\gamma=cos\gamma$

$\gamma=Acos(n\cdot x)$

Now we add a rotation around n that transforms z into Z. This will also move y into the plane XY because xyz is an orthogonal triad.

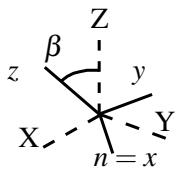


Figure 3.2. Triad nyz is rotated around n until z gets aligned with Z. The angle of rotation is β.

The axis of rotation is n and the angle results from
$z \cdot Z = ||z||||Z||cos\beta = cos\beta$
$\beta = Acos(z \cdot Z) = Acos(i)$

At last, we add a rotation around Z that transforms n into X. Additionally, y will get into Y.

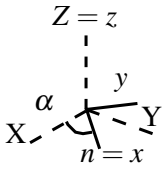


Figure 3.3. Triad nyZ is rotated around Z until n gets aligned with X. The angle of rotation is α.

The axis of rotation is Z and the angle results from
$n \cdot X = ||n||||X||cos\alpha = cos\alpha$
$\alpha = Acos(n \cdot X) = Acos(n_1)$
In conclusion:
Let the triad XYZ be $X = (1,0,0)$, $Y = (0,1,0)$ and $Z = (0,0,1)$ and the triad xyz be $x = (a,b,c)$, $y = (d,e,f)$, $z = (g,h,i)$. Axis x becomes $n = (n_1,n_2,0)$ when aligned with the XY plane. The coordinates of n are:

$$n_1 = \pm \frac{\frac{h}{g}}{\sqrt{1+\frac{h^2}{g^2}}}$$

$$n_2 = - \pm \frac{1}{\sqrt{1+\frac{h^2}{g^2}}}$$

To rotate triad xyz to fit the usual triad XYZ:

1. Triad xyz is rotated around z until x gets aligned with plane XY. Rotated x axis becomes n. The angle of rotation is $\gamma = Acos(n \cdot x)$.
2. Triad nyz is rotated around n until z gets aligned with Z. The angle of rotation is $\beta = Acos(z \cdot Z) = Acos(i)$.
3. Triad nyZ is rotated around Z until n gets aligned with X. The angle of rotation is $\alpha = Acos(n \cdot X) = Acos(n_1)$.

To rotate triad XYZ to fit the usual triad xyz:

1. Triad XYZ is rotated around Z until X gets aligned with n. The angle of rotation is $\alpha = -Acos(n \cdot X) = Acos(n_1)$.
2. Triad nYZ is rotated around n until Z gets aligned with z. The angle of rotation is $\beta = -Acos(z \cdot Z) = -Acos(i)$.
3. Triad nYZ is rotated around z until n gets aligned with x. The angle of rotation is $\gamma = -Acos(n \cdot x)$.

36 *PCA of a bone under Xforms.* *The following program tests the compatibility of Xforms with PCA. Our attempt is a failure.*

```
/*
 * Program N36 PCAofHandBones
   This program tests the compatibility of Xforms
   with the principal component analysis of a single bone.
   The angles for the Xforms are   the Euler rotations.
   This is a failure.

   Tested only with some  files,
    particularly  with those of the BodyParts3D project.
    http://lifesciencedb.jp/bp3d/
    Those files are licensed:

    The license for this database is specified in the
    Creative Commons Attribution-Share Alike 2.1 Japan.
    If you use data from this database, please be sure
    attribute this database as follows: "BodyParts3D, (c)
    The Database Center for Life Science licensed under
    CC Attribution-Share Alike 2.1 Japan".
```

```
    http://dbguidehive.biosciencedbc.jp/en/bodyparts3d/lic.html
 */
package Programs;
import Jama.*;

//Files in the hosting project Programs
import Tools.Axes;
import Tools.Cameras;
import Tools.Xform;
//import Tools.XformWithPivot;



import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.event.ActionEvent;
import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.TextArea;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.MeshView;;
import javafx.scene.shape.TriangleMesh;
import static javafx.scene.transform.Rotate.Y_AXIS;
import static javafx.scene.transform.Rotate.Z_AXIS;
```

```
import javafx.stage.FileChooser;
import javafx.stage.Stage;



public class PCAofHandBones extends Application
{
    //Visibility of the mesh
    //might depend on the correct choice
    //of this factor. Move it in within 1 and 500.
    private static final double scaleFactor = 4;
    //Printing for debugging
    final static boolean printReadFile = false;
    private static final String title
            =  "Importer of meshes in format .obj";

    //Tool to read files
    BufferedReader br ;
    private static  String textOfFile = "";

    private static int counterOfFaces = 0;
    private static int counterOfVertexes = 0;
    private static int counterOfNormals = 0;

    private static int[] destFaces ;


    //*****************

    final Group root = new Group();


    private static final Cameras camera = new Cameras();


    final Xform cameraXform = new Xform();
    final Xform cameraXform2 = new Xform();
    final Xform cameraXform3 = new Xform();

    //Axes : ctrl + A
```

```
    private static final Axes axes = new Axes();


    final double cameraDistance = 1450;//450
    final Xform world3DXform = new Xform();
    final Xform bonesXform = new Xform();



    boolean timelinePlaying = false;
    double ONE_FRAME = 1.0/24.0;
    double DELTA_MULTIPLIER = 200.0;
    double CONTROL_MULTIPLIER = 0.1;
    double SHIFT_MULTIPLIER = 0.1;
    double ALT_MULTIPLIER = 0.5;

    double mousePosX;
    double mousePosY;
    double mouseOldX;
    double mouseOldY;
    double mouseDeltaX;
    double mouseDeltaY;




   //Menubar declaration and initialization
    MenuBar menuBar = new MenuBar();

private static double basisRot[][] = new double[3][3];
    private static double matrixRotz[][] = new double[3][3];



@Override
public void start(Stage primaryStage)
{
    System.out.println("start");

    Scene scene = new Scene(root, 1200, 1000, true);
```

```
/*
   Multiple file loader
*/

//Content of files will be shown in textArea
final TextArea textArea = new TextArea();
textArea.prefWidth(400);
textArea.setWrapText(true);
textArea.setText("");
//Container  for textArea.
ScrollPane scrollPane = new ScrollPane();
scrollPane.getStyleClass().add("noborder-scroll-pane");
scrollPane.setContent(textArea);
scrollPane.setFitToWidth(true);
scrollPane.setPrefWidth(400);
scrollPane.setPrefHeight(180);
VBox vBox = new VBox();
vBox.visibleProperty().setValue(false);
vBox.getChildren().addAll(scrollPane);
root.getChildren().addAll(vBox);


// --- Menu files
Menu filesMenu = new Menu("Files");
//Menu to load multiple files
MenuItem loadFiles
        = new MenuItem("Load + process multiple files ");
//MenuItem added to menu
filesMenu.getItems().addAll(loadFiles);
menuBar.getMenus().add(filesMenu);
root.getChildren().add(menuBar);

//Listener
loadFiles.setOnAction((ActionEvent t) ->
{
    FileChooser fileChooser = new FileChooser();
```

```
        //Name of file with extension .obj
        FileChooser.ExtensionFilter extFilter
                = new FileChooser
                        .ExtensionFilter("TXT files (*.obj)"
                                , "*.obj");
        fileChooser.getExtensionFilters().add(extFilter);
        //Files will be kept in a list
        List<File> files = new FileChooser()
                .showOpenMultipleDialog(primaryStage);
        //Each file is processed
        files.stream().forEach((File file) -> { try
        {
            //Text of file is read and mesh is extracted
            textOfFile = readFile(file);
            //Text copied to textArea
            textArea.appendText(textOfFile);

            } catch (IOException ex)
            {
                Logger.getLogger(PCAofHandBones
                        .class.getName())
                        .log(Level.SEVERE, null, ex);
            }
        });

    //To remedy a not understood bug
    //( = menuBar is added to the graphic )
    root.getChildren().remove(menuBar);

    vBox.visibleProperty().set(true);
    });

    //====== End of File loader

    primaryStage.setTitle(title
            + ". Drag the mouse to rotate. "
        + " Ctrl + A = Axis visibility"
            );
    scene.setFill(Color.AQUAMARINE);
    primaryStage.setScene(scene);
```

```
    primaryStage.show();
    axes.buildAxes();
    axes.handleKeyboard(scene);
    root.getChildren().addAll(axes.axesXform);
    camera.handleMouse(scene);
    camera.handleMouse(scene);


    //Hide textArea
    Button hideTextArea = new Button();
    hideTextArea.setText("View graphic");
    //Listener
    hideTextArea.setOnAction((ActionEvent t) ->
    {
        vBox.visibleProperty().set(false);
        camera.buildCamera();
        /*
        bonesXform.setScale(scaleFactor);
        //Centering: found by trial and error.
        double xMax = bonesXform.getBoundsInLocal().getMaxX();
        double yMax = bonesXform.getBoundsInLocal().getMaxY();
        double zMax = bonesXform.getBoundsInLocal().getMaxZ();
        double xMin = bonesXform.getBoundsInLocal().getMinX();
        double yMin = bonesXform.getBoundsInLocal().getMinY();
        double zMin = bonesXform.getBoundsInLocal().getMinZ();
        double x = scaleFactor *  0.82 * (xMax + xMin) ;
        double y = scaleFactor * 0.78 * (yMax + yMin);
        double z = scaleFactor * 0.9 * (zMax + zMin) ;
        bonesXform.setTranslate(-x, -y, -z);
        */
        world3DXform.getChildren().add(bonesXform);
        root.getChildren().add(world3DXform);
        scene.setCamera(camera.camera);
    });

    vBox.getChildren().addAll(hideTextArea);
    primaryStage.show();
}
```

```
//File is read and its content is returned as a string.
//The associated mesh is recovered
//and correct orientation fo faces is provided.
//The mesh is a global variable.


private String readFile(File file)
        throws FileNotFoundException, IOException
{
    //Name of file is printed
    String name = file.getName();
    System.out.println(name);
    //Buffer to keep the content of file
    //StringBuilder stringBuffer = new StringBuilder();
    StringBuilder stringBuffer  = new StringBuilder();
    //Tool to read file
    BufferedReader bufferedReader = null;

    //A mesh is declared and instantiated
    TriangleMesh mesh  = new TriangleMesh();
    //The mesh has no predefined texture
    mesh.getTexCoords().addAll(0,0);
    //In the .obj format, counting begins with 1
    //but in Java, we begin with 0.
    //So, an additional inocuos point is added.
    mesh.getPoints().addAll(0,0,0);

    if (printReadFile)
        System.out.println("File content: ");
//File is read
try
{
//
bufferedReader = new BufferedReader(new FileReader(file));

String text;
```

```
while( (text = bufferedReader.readLine()) != null)
{
    //Text for the textArea
    stringBuffer.append(text);

    //Vertexes are read
    if (((text.length() > 0 ) && text.charAt(0) == 'v')
            && text.charAt(1) == ' ')
    {
        if (printReadFile)
            System.out.println("Vertex  = "
                    + counterOfVertexes + " = "+text);
        counterOfVertexes++;
        text = text.substring(2);
        int j = 0;
        //Three vertexes are read
        for(int i = 0; i < 3; i++)
        {
            double d ;
            String s = "";
            Boolean go = true;

            while (go)
            {
                char thisChar = text.charAt(j);
                s = s + thisChar;
                char nextChar = ' ';
                //++j means add one to j
                //and then evaluate expression with <
                if (++j < text.length())
                 nextChar = text.charAt(j);
                if ((nextChar == ' ') ||
                        (j >= text.length()))
                        go = false;

            }
            //string to double
            d = Double.parseDouble(s);
```

```
            float df = (float) d;
            if (printReadFile)
                System.out.println("Float number = " + df);
            mesh.getPoints().addAll(df);

        }
    }


    //Normal vectors are counted and displayed
    if ((text.length() > 0 ) && (text.charAt(0) == 'v')
            && (text.charAt(1) == 'n'))
    {

        if (printReadFile)
            System.out.println("Normal " + counterOfNormals
                + " = " + text);
        counterOfNormals++;
    }


    //Face is read, one face per line.
    //A face has 3 or more vertexes. These are extracted.
    //Additional information is ignored
    if ( (text.length() > 0 ) && (text.charAt(0) == 'f'))
    {

        if (printReadFile)
            System.out.println("Face  = " + counterOfFaces
                + " = "+text);
        counterOfFaces++;
        text = text.substring(2);

        int vertex[] = new int[4];
        int j = 0;

        Boolean isNotBlank ;
        //Three vertexes are extracted
        for(int i = 0; i < 3; i++)
        {
```

```
double d ;
String s = "";
Boolean go1 = true;


while (go1)
{
    if (j >= text.length()-2)
        break;

    char thisChar = text.charAt(j);
    s = s + thisChar;
    /*
    System.out.println(" j = "
            + j + "    s = " + s);
            */
    char nextChar = ' ';
    j++;
    if (j < text.length())
     nextChar = text.charAt(j);
    char nextNextChar;
    isNotBlank = true;
    if (nextChar == '/' )
    {
        go1 = false;
        //go until the following blank
        while ((isNotBlank)
                && (j < text.length()-1))
        {
            j++;
            nextNextChar = text.charAt(j);
            if (nextNextChar == ' ' )
                isNotBlank = false;

        }
    }

}
```

```
    d = Double.parseDouble(s);
    int v = (int) d;
    vertex[i] = v;
    if (printReadFile)
        System.out.println("Int number = " + v);
}//End of cycle for 3 vertexes

//A face with 3 vertexes is added to mesh
mesh.getFaces().addAll(vertex[0], 0, vertex[1], 0,
    vertex[2], 0);

if (printReadFile)
{
    for(int i = 0; i < 3; i++)
     System.out.print(vertex[i] + " . ");
    System.out.println();
}

//Is there a four vertex in the face?
if   ( (text.length() - j )> 3)
{
    String s = "";
    Boolean go1 = true;
    while (go1)
    {


        char thisChar = text.charAt(j);
        s = s + thisChar;
        j++;
        char nextChar = ' ';
        if (j < text.length())
         nextChar = text.charAt(j);
        if (nextChar == '/' ) go1 = false;
    }
    if (printReadFile)
        System.out.println("4th = " + s);
    double d = Double.parseDouble(s);
    int v = (int) d;
    vertex[3] = v;
```

```
            //A polyface with 4 vertex, say, a,b,c,d,
            //can be covered by 2 triangles:
            //Triangle 1: a,b,c (Already added to mesh)
            //and triangle 2: a,c,d.
            //Triangle 2 with 3 vertexes is added to mesh
            mesh.getFaces().addAll(vertex[0], 0, vertex[2],
                    0, vertex[3], 0);
        }
    }//End of face cycle


        //Bounds of containing box are read and displayed
    if ( (text.length() > 0 )
            && (text.contains("# Bounds(mm):")))
    {
        System.out.println(text);

        String a = "# Bounds(mm): (";
        int m = a.length();
        text = text.substring(m);
        text = text.replace(")-(", ",");
        text = text.replace(")", ",");
        //System.out.println(text);

        int j = 0;


        //Six vertexes are extracted
        for(int i = 0; i < 6; i++)
        {

            double d ;
            String s = "";
            Boolean go1 = true;


            while (go1)
            {
                if (j >= text.length()-2)
```

```
                    break;

                char thisChar = text.charAt(j);
                s = s + thisChar;
                /*
                System.out.println(" j = "
                        + j + "    s = " + s);
                */
                char nextChar = ' ';
                j++;
                if (j < text.length())
                 nextChar = text.charAt(j);
                if (nextChar == ',' )
                {
                    go1 = false;
                   if ( j < text.length() ) j++;
                }

            }

            d = Double.parseDouble(s);


           if (printReadFile)
                System.out.println("Coordinate = " + i + "= " + d);
        }//End of cycle for 3 vertexes
    }

}

} //End of  try.
    //Exception management
    catch (FileNotFoundException ex)
    {
        Logger.getLogger(PCAofHandBones
        .class.getName()).log(Level.SEVERE, null, ex);
    }
    catch (IOException ex)
    {
        Logger.getLogger(PCAofHandBones
```

```
            .class.getName()).log(Level.SEVERE, null, ex);
        }
        finally
        {
            try
            {
                bufferedReader.close();
            } catch (IOException ex)
            {
                Logger.getLogger(PCAofHandBones
                .class.getName()).log(Level.SEVERE, null, ex);
            }
        }

        if (printReadFile)
        {
            System.out.println("End of file content. ");
            //Other indicators are reported
            report(mesh);
        }

       //mesh = centerMesh(mesh);
        //======MeshWork=======


        PCA(mesh);

        MeshView boneMesh =
                new MeshView(mesh);

        PhongMaterial goldenMaterial
                = new PhongMaterial(Color.GOLDENROD);
        boneMesh.setMaterial(goldenMaterial);
        Group boneGroup = new Group( boneMesh);
        //Mesh is added to bonesXform
        bonesXform.getChildren().add(boneGroup);


        return stringBuffer.toString();
    }
```

```
//Indicators of retrieved data are printed
private void report(TriangleMesh mesh)
{
    float[] destPoints = new float[mesh.getPoints().size()];
    mesh.getPoints().toArray(destPoints);
    System.out.println("POINTS");
    for(int j = 0; j <  destPoints.length/3; j++)
    {
        System.out.print(j + "   " );
        for(int k = 0; k < 3; k++)
        {
            System.out.print( destPoints[j*3 + k] + "   " );
        }
        System.out.println();
    }

    destFaces = new int[mesh.getFaces().size()];
    //Faces go to a matrix
    mesh.getFaces().toArray(destFaces);
    System.out.println("FACES WITH TEXTURE COORDINATES");
    for(int j = 0; j <  destFaces.length/6; j++)
    {
        System.out.print(j + "   " );
        for(int k = 0; k < 6; k++)
        {
            System.out.print( destFaces[j*6 + k] + "   " );
        }
        System.out.println();
    }

    System.out.println("counterOfPoints = "
            + counterOfVertexes );
    System.out.println("counterOfFaces = "
            + counterOfFaces );
    System.out.println("counterOfNormals = "
```

```
            + counterOfNormals);
    System.out.println("mesh.getPoints().size() = "
            + mesh.getPoints().size() );
    System.out.println("mesh.getFaces().size() = "
            + mesh.getFaces().size() );

    System.out.println("getPointElementSize = "
            + mesh.getPointElementSize() );
    System.out.println("getfaceElementSize = "
            + mesh.getFaceElementSize() );

}

private static void print (String s)
{
   System.out.print(s);
}
public static String fixedWidthDoubletoString
        (double x, int w, int d) {
   java.text.DecimalFormat fmt
           = new java.text.DecimalFormat();
   fmt.setMaximumFractionDigits(d);
   fmt.setMinimumFractionDigits(d);
   fmt.setGroupingUsed(false);
   String s = fmt.format(x);
   while (s.length() < w) {
      s = " " + s;
   }
   return s;
}

//Principal component analysis of a mesh
private void PCA(TriangleMesh mesh)
{
    //Point are declared as a matrix of coordinates
    int n = mesh.getPoints().size();
        float nPoints = n;
    nPoints = nPoints/3;
    float[][] CoordinatesFloat = new float[(int) nPoints][3];
```

```
/*
//Test: adapted from
//Stat Trek: Variance-Covariance Matrix
//http://stattrek.com/matrix-algebra/covariance-matrix.aspx

float[] data = {90,  60,  90,
                90,  90,  30,
                60,  60,  60,
                60,  60,  90,
                30,  30,  30 };


    n = 15;
    nPoints = 5;
int counter = 0;
for(int i = 0; i < n; i = i+3)
{
    //Reading
    float x =  data[i];
    float y =  data[i+1] ;
    float z =  data[i+2] ;
    CoordinatesFloat[counter][0] = x;
    CoordinatesFloat[counter][1] = y;
    CoordinatesFloat[counter][2] = z;
    counter++;
}

//Mean is found
float[] mean = new float[3];

for(int i = 0; i < 3; i++)
{
    mean[i] = 0;
    for (int j = 0; j <  nPoints; j++)
        mean[i] = mean[i] + CoordinatesFloat[j][i];
    mean[i] = mean[i]/(nPoints);
}

System.out.println("meanX = " + mean[0]);
```

```
System.out.println("meanY = " + mean[1]);
System.out.println("meanZ = " + mean[2]);



//Calculate covariance matrix
double[][] Cov = new double[3][3];
for(int i = 0; i < 3; i++)
{
    for (int j = 0; j <  3; j++)
    {
        Cov[i][j] = 0;
        for (int k = 0; k <  nPoints; k++)
            Cov[i][j] = Cov[i][j] +
                    (CoordinatesFloat[k][i] - mean[i])
                    * (CoordinatesFloat[k][j] - mean[j]);

        Cov[i][j] = Cov[i][j]/nPoints;

    }

}

printMatrix(Cov, "Cov");

//Answer: cov matrix must be:
//504  360  180
//360  360  0
//180  0  720
*/

//=============

int counter = 0;
for(int i = 0; i < n; i = i+3)
{
    //Reading
    float x =  mesh.getPoints().get(i);
    float y =  mesh.getPoints().get(i+1);
```

```
        float z =  mesh.getPoints().get(i+2);
        CoordinatesFloat[counter][0] = x;
        CoordinatesFloat[counter][1] = y;
        CoordinatesFloat[counter][2] = z;
        counter++;
    }

    //Mean is found
    float[] mean = new float[3];

    for(int i = 0; i < 3; i++)
    {
        mean[i] = 0;
        for (int j = 0; j <  nPoints; j++)
            mean[i] = mean[i] + CoordinatesFloat[j][i];
        mean[i] = mean[i]/(nPoints);
    }

    System.out.println("meanX = " + mean[0]);
    System.out.println("meanY = " + mean[1]);
    System.out.println("meanZ = " + mean[2]);




    //Calculate covariance matrix
    double[][] Cov = new double[3][3];
    for(int i = 0; i < 3; i++)
    {
        for (int j = 0; j <  3; j++)
        {
            Cov[i][j] = 0;
            for (int k = 0; k <  nPoints; k++)
                Cov[i][j] = Cov[i][j] +
                        (CoordinatesFloat[k][i] - mean[i])
                        * (CoordinatesFloat[k][j] - mean[j]);

            Cov[i][j] = Cov[i][j]/nPoints;

        }
```

```
}

printMatrix(Cov, "Cov");

//=========

//Eigenvalues of Cov matrix are found
Matrix M = new Matrix(Cov);


EigenvalueDecomposition E =
     new EigenvalueDecomposition(M);

double[] d = E.getRealEigenvalues();

for(int i = 0; i < 3; i++)
    System.out.println("Eigenvalue " + i + " : " + d[i]);
double[][] EV = E.getV().getArray();
for(int i = 0; i < 3; i++)
{
    System.out.println("Eigenvector " + i + " : " );
    for (int j = 0; j <  3; j++)
    {
        System.out.println(EV[i][j]);
    }
    System.out.println();
}

//Test: eigenvectors of a real symmetric matrix
//are orthogonal one to another: dot product must be zero
System.out.println("Orthogonality test : matrix I is expected");
double[][] Ort = new double[3][3];
for(int i = 0; i < 3; i++)
    {

        for (int j = 0; j <  3; j++)
        {

            //sum = sum + EV[i][j]*EV[i+1][j];
```

```
                Ort[i][j] = dotProduct(EV[i], EV[j]);

            }
            //System.out.println(sum);

        }

    printMatrix(Ort, "Ort test");

    //Determinant must be 1.
    System.out.println("Orientation test: "
            + "+1 is expected. Det E = " + E.getV().det() + " \n");



    //Bone is translated
    bonesXform.setTranslate(-mean[0], -mean[1], -mean[2]);
    axes.axesGroup.setVisible(true);



    //Implementation of Euler technology

    //Coordinates of xVector : first eigenvector
    double a = EV[0][0];
    double b = EV[0][1];
    double c = EV[0][2];

    double[] xVector = {a,b,c};

    //Coordinates of zVector: third eigenvector
    double g = EV[2][0];
    double h = EV[2][1];
    double i = EV[2][2];

    //double[] zVector = {g,h,i};

    //Sign found by trial and error
    double n1 =  (h/g)/Math.sqrt(1 +  h*h/(g*g));
    double n2 =  1.0/Math.sqrt(1 +  h*h/(g*g));
```

```
    Point3D nPoint3D = new Point3D(n1,n2,0);

    double[] nVector = {n1,n2,0};
    //Test: norm^2 of nVector must be 1
    double norm2 = n1*n1 + n2*n2;
    System.out.println("n1 = " +  n1);
    System.out.println("n2 = " +  n2);
    System.out.println("Test: Norm^2 of nVector must be 1= "
            +  norm2);
    //Conversion factor from radians to degrees
    double cf = 180.0 /Math.PI;
    double gamma = cf * Math.acos(dotProduct(nVector, xVector));
    double beta = cf* Math.acos(i);
    double alpha = cf * Math.acos(n1);

    System.out.println("Alpha = " + alpha);
    System.out.println("Beta = " + beta);
    System.out.println("Gamma = " + gamma);


    axes.axesXform.setRotationAxis(Z_AXIS);
    axes.axesXform.setRotate(-alpha );

    axes.axesXform.setRotationAxis(nPoint3D);
    axes.axesXform.setRotate(beta );

    axes.axesXform.setRotationAxis(Y_AXIS);
    axes.axesXform.setRotate(-gamma );


}

//============== Matrix algebra ==============


private static void printMatrix(double m[][], String title)
{
    System.out.println("This is the matrix " + title);
    for(int i=0;i<3;i++)
```

```
        System.out.println(m[i][0] + " " + m[i][1]+ " "
                                                + m[i][2]);
    System.out.println();
}


  //To find a matrix of a rotation, one must find first
  //an orthogonal basis with 3 vectors
  //The third vector is the axis of rotation
  private static  double[] thirdVector(double[] v)
  {
    double a = v[0];
    double b = v[1];
    double c = v[2];
    double norm = Math.sqrt(a*a + b*b + c*c);
    //System.out.println("norm = " + norm);
    double[] helpV = new double[3];
    helpV[0] = a/norm;
    helpV[1]= b/norm;
    helpV[2] = c/norm;
    /*System.out.println("Third vector " + helpV[0]+
                                  helpV[1]+helpV[2]);*/
    return helpV;
  }

  //The second vector is perpendicular to the first
  private static  double[] secondVector(double[] v)
  {
    double[] helpV = new double[3];
    double   x,y,z;
    double a = v[0];
    double b = v[1];
    double c = v[2];
    x=b;
    y = -a;
    z=0;
    if (b==0)
       {
            x=c;
            z = -a;
```

```
            y=0;
        }
    if (a==0)
        {
            y=c;
            z = -b;
            x=0;
        }
    double norm = Math.sqrt(x*x + y*y + z*z);

    helpV[0] = x/norm;
    helpV[1]  = y/norm;
    helpV[2]  = z/norm;
    return helpV;
}

//Normalized crux product of vectors v and w
private static  double[] cruxProduct(double[] v,
                                     double[] w)
{
    double a,b,c;
    double[] helpV = new double[3];
    a=  (v[1]*w[2] -w[1]*v[2]);
    b= - (v[0]*w[2]-w[0]*v[2] );
    c=  (v[0]*w[1]  -w[0]*v[1] );
    double norm = Math.sqrt(a*a + b*b + c*c);
    helpV[0] = a/norm;
    helpV[1]  = b/norm;
    helpV[2]  = c/norm;
    return helpV;

}

//Normalized crux product of vectors v and w
private static  double dotProduct(double[] v,
                                  double[] w)
{
    return v[0]*w[0] + v[1]*w[1] + v[2]*w[2];

}
```

```
 //An axis (a,b,c) generates a basis B
  private static  double[][] basisB(double v[])
  {
    double[][]  helpM = new double[3][3];
    double[]   helpT, helpS, helpF;

    helpT = thirdVector(v);
    helpS = secondVector(v);
    //The first vector is the crux product of the other 2.
    helpF = cruxProduct(helpT,helpS);
    for(int i=0; i<3;i++)
    {
     helpM[i][0] = helpF[i];
     helpM[i][1] = helpS[i];
     helpM[i][2] = helpT[i];
    }

    return helpM;
  }

public static void main(String[] args)
{

    launch(args);
}

}//End of main class N36 PCAofHandBones
```

37 ***Exercise.*** *Run the previous program and play with the code. Play enough to agree else disagree with the Author: Xforms and Euler's rotations are incompatible.*

38 ***PCA of a bone with Mesh transformations.*** *The following program tests the compatibility of JavaFX 3D meshes with PCA. After too much suffering with false starts, the Author remembered that the first point of a mesh is artificially filled*

in zeros because the .obj format begins counting with 1 while Java begins with 0. Ensuing corrections were made and at last the program produced drawings as expected yet not to perfect agreement with what the brains expects. We use a matrix of change of basis to align the bone with ordinary axes.

```
/*
    Program N38 PCAofHandBones2
    This program tests the compatibility of JavaFX 3D meshes
    with the principal component analysis of a single bone.

    Tested only with some  files,
    particularly  with those of the BodyParts3D project.
    http://lifesciencedb.jp/bp3d/
    Those files are licensed:

    The license for this database is specified in the
    Creative Commons Attribution-Share Alike 2.1 Japan.
    If you use data from this database, please be sure
    attribute this database as follows: "BodyParts3D, (c)
    The Database Center for Life Science licensed under
    CC Attribution-Share Alike 2.1 Japan".
    http://dbguidehive.biosciencedbc.jp/en/bodyparts3d/lic.html
 */
package Programs;
import Jama.*;



//Files in the hosting project Programs
import Tools.Axes;
import Tools.Cameras;
import Tools.Xform;
//import Tools.XformWithPivot;



import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
```

```
import java.io.IOException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.event.ActionEvent;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.TextArea;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.MeshView;;
import javafx.scene.shape.TriangleMesh;
import javafx.stage.FileChooser;
import javafx.stage.Stage;



public class PCAofHandBones2 extends Application
{
    //Visibility of the mesh
    //might depend on the correct choice
    //of this factor. Move it in within 1 and 500.
    private static final float scaleFactor = 15;
    //Printing for debugging
    final static boolean printReadFile = false;
    private static final String title
            =  "Importer of meshes in format .obj";

    //Tool to read files
    BufferedReader br ;
    private static  String textOfFile = "";
```

```
private static int counterOfFaces = 0;
private static int counterOfVertexes = 0;
private static int counterOfNormals = 0;

private static int[] destFaces ;


//*****************

final Group root = new Group();


private static final Cameras camera = new Cameras();


final Xform cameraXform = new Xform();
final Xform cameraXform2 = new Xform();
final Xform cameraXform3 = new Xform();

//Axes : ctrl + A
private static final Axes axes = new Axes();


final double cameraDistance = 1450;//450
final Xform world3DXform = new Xform();
final Xform bonesXform = new Xform();



boolean timelinePlaying = false;
double ONE_FRAME = 1.0/24.0;
double DELTA_MULTIPLIER = 200.0;
double CONTROL_MULTIPLIER = 0.1;
double SHIFT_MULTIPLIER = 0.1;
double ALT_MULTIPLIER = 0.5;

double mousePosX;
double mousePosY;
double mouseOldX;
double mouseOldY;
```

```
    double mouseDeltaX;
    double mouseDeltaY;




   //Menubar declaration and initialization
    MenuBar menuBar = new MenuBar();

private static double basisRot[][] = new double[3][3];
    private static double matrixRotz[][] = new double[3][3];



@Override
public void start(Stage primaryStage)
{
    System.out.println("start");

    Scene scene = new Scene(root, 1200, 1000, true);



    /*
       Multiple file loader
    */

    //Content of files will be shown in textArea
    final TextArea textArea = new TextArea();
    textArea.prefWidth(400);
    textArea.setWrapText(true);
    textArea.setText("");
    //Container  for textArea.
    ScrollPane scrollPane = new ScrollPane();
    scrollPane.getStyleClass().add("noborder-scroll-pane");
    scrollPane.setContent(textArea);
    scrollPane.setFitToWidth(true);
    scrollPane.setPrefWidth(400);
    scrollPane.setPrefHeight(180);
    VBox vBox = new VBox();
```

```
vBox.visibleProperty().setValue(false);
vBox.getChildren().addAll(scrollPane);
root.getChildren().addAll(vBox);


// --- Menu files
Menu filesMenu = new Menu("Files");
//Menu to load multiple files
MenuItem loadFiles
        = new MenuItem("Load + process multiple files ");
//MenuItem added to menu
filesMenu.getItems().addAll(loadFiles);
menuBar.getMenus().add(filesMenu);
root.getChildren().add(menuBar);

//Listener
loadFiles.setOnAction((ActionEvent t) ->
{
    FileChooser fileChooser = new FileChooser();

    //Name of file with extension .obj
    FileChooser.ExtensionFilter extFilter
            = new FileChooser
                    .ExtensionFilter("TXT files (*.obj)"
                            , "*.obj");
    fileChooser.getExtensionFilters().add(extFilter);
    //Files will be kept in a list
    List<File> files = new FileChooser()
            .showOpenMultipleDialog(primaryStage);
    //Each file is processed
    files.stream().forEach((File file) -> { try
    {
        //Text of file is read and mesh is extracted
        textOfFile = readFile(file);
        //Text copied to textArea
        textArea.appendText(textOfFile);

        } catch (IOException ex)
        {
            Logger.getLogger(PCAofHandBones
```

```
                    .class.getName())
                    .log(Level.SEVERE, null, ex);
        }
    });

//To remedy a not understood bug
//( = menuBar is removed from the graphic )
root.getChildren().remove(menuBar);

vBox.visibleProperty().set(true);
});

//====== End of File loader

primaryStage.setTitle(title
        + ". Drag the mouse to rotate. "
    + " Ctrl + A = Axis visibility"
        );
scene.setFill(Color.AQUAMARINE);
primaryStage.setScene(scene);
primaryStage.show();
axes.buildAxes();
axes.handleKeyboard(scene);
root.getChildren().addAll(axes.axesXform);
camera.handleMouse(scene);
camera.handleMouse(scene);


//Hide textArea
Button hideTextArea = new Button();
hideTextArea.setText("View graphic");
//Listener
hideTextArea.setOnAction((ActionEvent t) ->
{
    vBox.visibleProperty().set(false);
    camera.buildCamera();

    world3DXform.getChildren().add(bonesXform);
    root.getChildren().add(world3DXform);
    scene.setCamera(camera.camera);
```

```
    });

    vBox.getChildren().addAll(hideTextArea);
    primaryStage.show();
}



//File is read and its content is returned as a string.
//The associated mesh is recovered
//and correct orientation fo faces is provided.
//The mesh is a global variable.


private String readFile(File file)
        throws FileNotFoundException, IOException
{
    //Name of file is printed
    String name = file.getName();
    System.out.println(name);
    //Buffer to keep the content of file
    //StringBuilder stringBuffer = new StringBuilder();
    StringBuilder stringBuffer  = new StringBuilder();
    //Tool to read file
    BufferedReader bufferedReader = null;

    //A mesh is declared and instantiated
    TriangleMesh mesh  = new TriangleMesh();
    //The mesh has no predefined texture
    mesh.getTexCoords().addAll(0,0);
    //In the .obj format, counting begins with 1
    //but in Java, we begin with 0.
    //So, an additional inocuos point is added.
    mesh.getPoints().addAll(0,0,0);

    if (printReadFile)
        System.out.println("File content: ");
//File is read
try
```

```
{
//
bufferedReader = new BufferedReader(new FileReader(file));

String text;




while( (text = bufferedReader.readLine()) != null)
{
    //Text for the textArea
    stringBuffer.append(text);

    //Vertexes are read
    if (((text.length() > 0 ) && text.charAt(0) == 'v')
            && text.charAt(1) == ' ')
    {
        if (printReadFile)
            System.out.println("Vertex  = "
                    + counterOfVertexes + " = "+text);
        counterOfVertexes++;
        text = text.substring(2);
        int j = 0;
        //Three vertexes are read
        for(int i = 0; i < 3; i++)
        {
            double d ;
            String s = "";
            Boolean go = true;

            while (go)
            {
                char thisChar = text.charAt(j);
                s = s + thisChar;
                char nextChar = ' ';
                //++j means add one to j
                //and then evaluate expression with <
                if (++j < text.length())
                 nextChar = text.charAt(j);
```

```
                if ((nextChar == ' ') ||
                        (j >= text.length()))
                        go = false;

            }
            //string to double
            d = Double.parseDouble(s);
            float df = (float) d;
            if (printReadFile)
                System.out.println("Float number = " + df);
            mesh.getPoints().addAll(df);

        }
    }


    //Normal vectors are counted and displayed
    if ((text.length() > 0 ) && (text.charAt(0) == 'v')
            && (text.charAt(1) == 'n'))
    {

        if (printReadFile)
            System.out.println("Normal " + counterOfNormals
                + " = " + text);
        counterOfNormals++;
    }


    //Face is read, one face per line.
    //A face has 3 or more vertexes. These are extracted.
    //Additional information is ignored
    if ( (text.length() > 0 ) && (text.charAt(0) == 'f'))
    {

        if (printReadFile)
            System.out.println("Face  = " + counterOfFaces
                + " = "+text);
        counterOfFaces++;
        text = text.substring(2);
```

```
int vertex[] = new int[4];
int j = 0;

Boolean isNotBlank ;
//Three vertexes are extracted
for(int i = 0; i < 3; i++)
{

    double d ;
    String s = "";
    Boolean go1 = true;


    while (go1)
    {
        if (j >= text.length()-2)
            break;

        char thisChar = text.charAt(j);
        s = s + thisChar;
        /*
        System.out.println(" j = "
                + j + "    s = " + s);
                */
        char nextChar = ' ';
        j++;
        if (j < text.length())
         nextChar = text.charAt(j);
        char nextNextChar;
        isNotBlank = true;
        if (nextChar == '/' )
        {
            go1 = false;
            //go until the following blank
            while ((isNotBlank)
                    && (j < text.length()-1))
            {
                j++;
                nextNextChar = text.charAt(j);
                if (nextNextChar == ' ' )
```

```
                        isNotBlank = false;

                }
            }

        }

        d = Double.parseDouble(s);
        int v = (int) d;
        vertex[i] = v;
        if (printReadFile)
            System.out.println("Int number = " + v);
    }//End of cycle for 3 vertexes

    //A face with 3 vertexes is added to mesh
    mesh.getFaces().addAll(vertex[0], 0, vertex[1], 0,
        vertex[2], 0);

    if (printReadFile)
    {
        for(int i = 0; i < 3; i++)
         System.out.print(vertex[i] + " . ");
        System.out.println();
    }

    //Is there a four vertex in the face?
    if   ( (text.length() - j )> 3)
    {
        String s = "";
        Boolean go1 = true;
        while (go1)
        {


            char thisChar = text.charAt(j);
            s = s + thisChar;
            j++;
            char nextChar = ' ';
            if (j < text.length())
             nextChar = text.charAt(j);
```

```
            if (nextChar == '/' ) go1 = false;
        }
        if (printReadFile)
            System.out.println("4th = " + s);
        double d = Double.parseDouble(s);
        int v = (int) d;
        vertex[3] = v;

        //A polyface with 4 vertex, say, a,b,c,d,
        //can be covered by 2 triangles:
        //Triangle 1: a,b,c (Already added to mesh)
        //and triangle 2: a,c,d.
        //Triangle 2 with 3 vertexes is added to mesh
        mesh.getFaces().addAll(vertex[0], 0, vertex[2],
                0, vertex[3], 0);
    }
}//End of face cycle


    //Bounds of containing box are read and displayed
if ( (text.length() > 0 )
        && (text.contains("# Bounds(mm):")))
{
    System.out.println(text);

    String a = "# Bounds(mm): (";
    int m = a.length();
    text = text.substring(m);
    text = text.replace(")-(", ",");
    text = text.replace(")", ",");
    //System.out.println(text);

    int j = 0;


    //Six vertexes are extracted
    for(int i = 0; i < 6; i++)
    {

        double d ;
```

```
            String s = "";
            Boolean go1 = true;


            while (go1)
            {
                if (j >= text.length()-2)
                    break;

                char thisChar = text.charAt(j);
                s = s + thisChar;
                /*
                System.out.println(" j = "
                        + j + "    s = " + s);
                */
                char nextChar = ' ';
                j++;
                if (j < text.length())
                 nextChar = text.charAt(j);
                if (nextChar == ',' )
                {
                    go1 = false;
                   if ( j < text.length() ) j++;
                }

            }

            d = Double.parseDouble(s);


           if (printReadFile)
                System.out.println("Coordinate = " + i + "= " + d);
        }//End of cycle for 3 vertexes
    }

}

} //End of  try.
    //Exception management
    catch (FileNotFoundException ex)
```

```
    {
        Logger.getLogger(PCAofHandBones
        .class.getName()).log(Level.SEVERE, null, ex);
    }
    catch (IOException ex)
    {
        Logger.getLogger(PCAofHandBones
        .class.getName()).log(Level.SEVERE, null, ex);
    }
    finally
    {
        try
        {
            bufferedReader.close();
        } catch (IOException ex)
        {
            Logger.getLogger(PCAofHandBones
            .class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    if (printReadFile)
    {
        System.out.println("End of file content. ");
        //Other indicators are reported
        report(mesh);
    }

   //mesh = centerMesh(mesh);
    //======MeshWork=======


    PCA(mesh);

    MeshView boneMesh =
            new MeshView(mesh);

    PhongMaterial goldenMaterial
            = new PhongMaterial(Color.GOLDENROD);
    boneMesh.setMaterial(goldenMaterial);
```

```
    Group boneGroup = new Group( boneMesh);
    //Mesh is added to bonesXform
    bonesXform.getChildren().add(boneGroup);


    return stringBuffer.toString();
}
```

```
//Indicators of retrieved data are printed
private void report(TriangleMesh mesh)
{
    float[] destPoints = new float[mesh.getPoints().size()];
    mesh.getPoints().toArray(destPoints);
    System.out.println("POINTS");
    for(int j = 0; j <  destPoints.length/3; j++)
    {
        System.out.print(j + "   " );
        for(int k = 0; k < 3; k++)
        {
            System.out.print( destPoints[j*3 + k] + "   " );
        }
        System.out.println();
    }

    destFaces = new int[mesh.getFaces().size()];
    //Faces go to a matrix
    mesh.getFaces().toArray(destFaces);
    System.out.println("FACES WITH TEXTURE COORDINATES");
    for(int j = 0; j <  destFaces.length/6; j++)
    {
        System.out.print(j + "   " );
        for(int k = 0; k < 6; k++)
        {
```

```
            System.out.print( destFaces[j*6 + k] + "   " );
        }
        System.out.println();
    }

    System.out.println("counterOfPoints = "
            + counterOfVertexes );
    System.out.println("counterOfFaces = "
            + counterOfFaces );
    System.out.println("counterOfNormals = "
            + counterOfNormals);
    System.out.println("mesh.getPoints().size() = "
            + mesh.getPoints().size() );
    System.out.println("mesh.getFaces().size() = "
            + mesh.getFaces().size() );

    System.out.println("getPointElementSize = "
            + mesh.getPointElementSize() );
    System.out.println("getfaceElementSize = "
            + mesh.getFaceElementSize() );

}



public static String fixedWidthDoubletoString
        (double x, int w, int d) {
   java.text.DecimalFormat fmt
           = new java.text.DecimalFormat();
   fmt.setMaximumFractionDigits(d);
   fmt.setMinimumFractionDigits(d);
   fmt.setGroupingUsed(false);
   String s = fmt.format(x);
   while (s.length() < w) {
      s = " " + s;
   }
   return s;
}
```

```
//Principal component analysis of a mesh
private void PCA(TriangleMesh mesh)
{
    //Point are declared as a matrix of coordinates
    int n = mesh.getPoints().size();
        float nPoints = n;
    nPoints = nPoints/3;
    float[][] CoordinatesFloat = new float[(int) nPoints][3];


    /*
    //Test: adapted from
    //Stat Trek: Variance-Covariance Matrix
    //http://stattrek.com/matrix-algebra/covariance-matrix.aspx

    float[] data = {90,  60,  90,
                    90,  90,  30,
                    60,  60,  60,
                    60,  60,  90,
                    30,  30,  30 };


        n = 15;
        nPoints = 5;
    int counter = 0;
    for(int i = 0; i < n; i = i+3)
    {
        //Reading
        float x =  data[i];
        float y =  data[i+1] ;
        float z =  data[i+2] ;
        CoordinatesFloat[counter][0] = x;
        CoordinatesFloat[counter][1] = y;
        CoordinatesFloat[counter][2] = z;
        counter++;
    }

    //Mean is found
    float[] mean = new float[3];
```

```
for(int i = 0; i < 3; i++)
{
    mean[i] = 0;
    for (int j = 0; j <  nPoints; j++)
        mean[i] = mean[i] + CoordinatesFloat[j][i];
    mean[i] = mean[i]/(nPoints);
}

System.out.println("meanX = " + mean[0]);
System.out.println("meanY = " + mean[1]);
System.out.println("meanZ = " + mean[2]);




//Calculate covariance matrix
double[][] Cov = new double[3][3];
for(int i = 0; i < 3; i++)
{
    for (int j = 0; j <  3; j++)
    {
        Cov[i][j] = 0;
        for (int k = 0; k <  nPoints; k++)
            Cov[i][j] = Cov[i][j] +
                    (CoordinatesFloat[k][i] - mean[i])
                    * (CoordinatesFloat[k][j] - mean[j]);

        Cov[i][j] = Cov[i][j]/nPoints;

    }

}

printMatrix(Cov, "Cov");

//Answer: cov matrix must be:
//504  360  180
//360  360  0
//180  0  720
*/
```

```
//=============

int counter = 0;
for(int i = 0; i < n; i = i+3)
{
    //Reading
    float x =  mesh.getPoints().get(i);
    float y =  mesh.getPoints().get(i+1);
    float z =  mesh.getPoints().get(i+2);
    CoordinatesFloat[counter][0] = x;
    CoordinatesFloat[counter][1] = y;
    CoordinatesFloat[counter][2] = z;
    counter++;
    //Warning: the zeroth point is artificial,
    //filled with zeros.
    if (i == 0)
        System.out.println("Zeroth point: x = " + x + "   y =   "
                + y + "   z =  "  + z);
}

//Mean is found. It is vector with 3 coordinates
float[] mean = new float[3];

for(int i = 0; i < 3; i++)
{
    mean[i] = 0;
    //Zeroth point does not count
    for (int j = 1; j <  nPoints; j++)
        mean[i] = mean[i] + CoordinatesFloat[j][i];
    mean[i] = mean[i]/(nPoints -1);
}

System.out.println("meanX = " + mean[0]);
System.out.println("meanY = " + mean[1]);
System.out.println("meanZ = " + mean[2]);
```

```
//Calculate covariance matrix
double[][] Cov = new double[3][3];
for(int i = 0; i < 3; i++)
{
    for (int j = 0; j <  3; j++)
    {
        Cov[i][j] = 0;
        for (int k = 1; k <  nPoints; k++)
            Cov[i][j] = Cov[i][j] +
                    (CoordinatesFloat[k][i] - mean[i])
                    * (CoordinatesFloat[k][j] - mean[j]);

        Cov[i][j] = Cov[i][j]/(nPoints-1);

    }

}

printMatrix(Cov, "Cov");

//=========

//Eigenvalues of Cov matrix are found
Matrix M = new Matrix(Cov);
EigenvalueDecomposition E =
     new EigenvalueDecomposition(M);
double[] d = E.getRealEigenvalues();

for(int i = 0; i < 3; i++)
    System.out.println("Eigenvalue " + i + " : " + d[i]);
double[][] EV = E.getV().getArray();
for(int i = 0; i < 3; i++)
{
    System.out.println("Eigenvector " + i + " : " );
    for (int j = 0; j <  3; j++)
    {
        System.out.println(EV[i][j]);
    }
    System.out.println();
}
```

```
//Test: eigenvectors of a real symmetric matrix
//are orthogonal one to another: dot product must be zero
System.out.println("Orthogonality test : matrix I is expected");
double[][] Ort = new double[3][3];
for(int i = 0; i < 3; i++)
{

    for (int j = 0; j <  3; j++)
    {

        //sum = sum + EV[i][j]*EV[i+1][j];
        Ort[i][j] = dotProduct(EV[i], EV[j]);

    }
    //System.out.println(sum);

}

printMatrix(Ort, "Ort test");

//Determinant must be 1.
System.out.println("Orientation test: "
        + "+1 is expected. Det E = " + E.getV().det() + " \n");


 //Mesh is centered
float [] pointsFloat = new float[n];
for(int i = 0; i < n; i = i+3)
{
    //Reading
    float x =  mesh.getPoints().get(i);
    float y =  mesh.getPoints().get(i+1);
    float z =  mesh.getPoints().get(i+2);

    //Mean is subtracted
    pointsFloat[i]    =  scaleFactor*(x - mean[0]);
    pointsFloat[i+1]  =  15*(y - mean[1]);
    pointsFloat[i+2]  =  15*(z - mean[2]);
}
```

```
    //Resetting of points of mesh
    mesh.getPoints().setAll(pointsFloat, 0, n);

    axes.axesGroup.setVisible(true);



    //1D array for recording
    float[]rotPoints = new float[n];
    //Mesh is aligned with the new basis
    //given by the PCA eigenvetors
    for(int i = 0; i < n; i = i+3)
    {
         //Reading
        float x =  mesh.getPoints().get(i);
        float y =  mesh.getPoints().get(i+1);
        float z =  mesh.getPoints().get(i+2);
        //Three coordinates define a point
        double[] w = {x,y,z};
        //Coordinates of point w are transformed
        //to the new basis
        w = matrixTimesVector(E.getV()
                .transpose().getArray(),w);

        //Point is added to the  1-dimensional array
        rotPoints[i ] = (float) (w[0] );
        rotPoints[i + 1] = (float) (w[1] );
        rotPoints[i + 2] = (float) (w[2] );
    }
    //Points are reset
    mesh.getPoints().setAll(rotPoints, 0, n);

}

//============== Matrix algebra ==============


private static void printMatrix(double m[][], String title)
{
```

```
    System.out.println("This is the matrix " + title);
    for(int i=0;i<3;i++)
        System.out.println(m[i][0] + " " + m[i][1]+ " "
                                                + m[i][2]);
    System.out.println();
}



  //The multiplication of a matrix times a vector
  private static  double[] matrixTimesVector(double m[][],
                                             double v[])
  {
    double[]  helpV = new double[3];
    for(int i=0; i<3; i++)
        helpV[i] =0;
    for(int i=0; i<3; i++)
        for(int j=0; j<3; j++)
            helpV[i]  =helpV[i]  + m[i][j]*v[j];
    return helpV;
  }



  //To find a matrix of a rotation, one must find first
  //an orthogonal basis with 3 vectors
  //The third vector is the axis of rotation
  private static  double[] thirdVector(double[] v)
  {
    double a = v[0];
    double b = v[1];
    double c = v[2];
    double norm = Math.sqrt(a*a + b*b + c*c);
    //System.out.println("norm = " + norm);
    double[] helpV = new double[3];
    helpV[0] = a/norm;
    helpV[1]= b/norm;
    helpV[2] = c/norm;
    /*System.out.println("Third vector " + helpV[0]+
```

```
                                    helpV[1]+helpV[2]);*/
  return helpV;
}

//The second vector is perpendicular to the first
private static  double[] secondVector(double[] v)
{
  double[] helpV = new double[3];
  double   x,y,z;
  double a = v[0];
  double b = v[1];
  double c = v[2];
  x=b;
  y = -a;
  z=0;
  if (b==0)
     {
          x=c;
          z = -a;
          y=0;
     }
  if (a==0)
     {
          y=c;
          z = -b;
          x=0;
     }
  double norm = Math.sqrt(x*x + y*y + z*z);

  helpV[0] = x/norm;
  helpV[1]  = y/norm;
  helpV[2]  = z/norm;
  return helpV;
}

//Normalized crux product of vectors v and w
private static  double[] cruxProduct(double[] v,
                                     double[] w)
{
  double a,b,c;
```

```
    double[] helpV = new double[3];
    a=  (v[1]*w[2] -w[1]*v[2]);
    b= - (v[0]*w[2]-w[0]*v[2] );
    c=  (v[0]*w[1]  -w[0]*v[1] );
    double norm = Math.sqrt(a*a + b*b + c*c);
    helpV[0] = a/norm;
    helpV[1]  = b/norm;
    helpV[2]  = c/norm;
    return helpV;

  }

  //Normalized crux product of vectors v and w
  private static  double dotProduct(double[] v,
                                    double[] w)
  {
    return v[0]*w[0] + v[1]*w[1] + v[2]*w[2];

  }




 //An axis (a,b,c) generates a basis B
  private static  double[][] basisB(double v[])
  {
    double[][]  helpM = new double[3][3];
    double[]   helpT, helpS, helpF;

    helpT = thirdVector(v);
    helpS = secondVector(v);
    //The first vector is the crux product of the other 2.
    helpF = cruxProduct(helpT,helpS);
    for(int i=0; i<3;i++)
    {
     helpM[i][0] = helpF[i];
     helpM[i][1] = helpS[i];
     helpM[i][2] = helpT[i];
    }
```

```
    return helpM;
  }
```

```
public static void main(String[] args)
{

    launch(args);
}

}//End of main class N38 PCAofHandBones2
```

39 ***Exercise.*** *Run the previous program and play with the code. Can you make a further correction to get a perfect agreement of drawings with what the brain expects?*

40 ***A new try.*** *Since program N36 PCAofHandBones was a failure, it was claimed that Xforms and Euler's rotations were incompatible. But it was forgotten to take into account that the first point of a mesh is artificially filled in zeros. The appropriate corrections were made to see what happens.*

```
/*
    Program N40 PCAofHandBones3
    This program tests the compatibility of Xforms
    with the principal component analysis of a single bone.
    The angles for the Xforms are   the Euler rotations.
    The bug in program N36 PCAofHandBones that forgot
    that the first point was artificially filled in zeros
    was corrected but
    the Author was unable to verify the expected.

    Tested only with some  files,
    particularly  with those of the BodyParts3D project.
```

```
    http://lifesciencedb.jp/bp3d/
    Those files are licensed:

    The license for this database is specified in the
    Creative Commons Attribution-Share Alike 2.1 Japan.
    If you use data from this database, please be sure
    attribute this database as follows: "BodyParts3D, (c)
    The Database Center for Life Science licensed under
    CC Attribution-Share Alike 2.1 Japan".
    http://dbguidehive.biosciencedbc.jp/en/bodyparts3d/lic.html
 */
package Programs;
import Jama.*;

//Files in the hosting project Programs
import Tools.Axes;
import Tools.Cameras;
import Tools.Xform;
//import Tools.XformWithPivot;



import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.event.ActionEvent;
import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
```

```
import javafx.scene.control.ScrollPane;
import javafx.scene.control.TextArea;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.MeshView;;
import javafx.scene.shape.TriangleMesh;
import static javafx.scene.transform.Rotate.Y_AXIS;
import static javafx.scene.transform.Rotate.Z_AXIS;
import javafx.stage.FileChooser;
import javafx.stage.Stage;



public class PCAofHandBones3 extends Application
{
    //Visibility of the mesh
    //might depend on the correct choice
    //of this factor. Move it in within 1 and 500.
    private static final double scaleFactor = 4;
    //Printing for debugging
    final static boolean printReadFile = false;
    private static final String title
            =  "Importer of meshes in format .obj";

    //Tool to read files
    BufferedReader br ;
    private static  String textOfFile = "";

    private static int counterOfFaces = 0;
    private static int counterOfVertexes = 0;
    private static int counterOfNormals = 0;

    private static int[] destFaces ;


    //*****************

    final Group root = new Group();
```

```
    private static final Cameras camera = new Cameras();


    final Xform cameraXform = new Xform();
    final Xform cameraXform2 = new Xform();
    final Xform cameraXform3 = new Xform();

    //Axes : ctrl + A
    private static final Axes axes = new Axes();


    final double cameraDistance = 1450;//450
    final Xform world3DXform = new Xform();
    final Xform bonesXform = new Xform();



    boolean timelinePlaying = false;
    double ONE_FRAME = 1.0/24.0;
    double DELTA_MULTIPLIER = 200.0;
    double CONTROL_MULTIPLIER = 0.1;
    double SHIFT_MULTIPLIER = 0.1;
    double ALT_MULTIPLIER = 0.5;

    double mousePosX;
    double mousePosY;
    double mouseOldX;
    double mouseOldY;
    double mouseDeltaX;
    double mouseDeltaY;




   //Menubar declaration and initialization
    MenuBar menuBar = new MenuBar();

private static double basisRot[][] = new double[3][3];
    private static double matrixRotz[][] = new double[3][3];
```

```
@Override
public void start(Stage primaryStage)
{
    System.out.println("start");

    Scene scene = new Scene(root, 1200, 1000, true);



    /*
       Multiple file loader
    */

    //Content of files will be shown in textArea
    final TextArea textArea = new TextArea();
    textArea.prefWidth(400);
    textArea.setWrapText(true);
    textArea.setText("");
    //Container  for textArea.
    ScrollPane scrollPane = new ScrollPane();
    scrollPane.getStyleClass().add("noborder-scroll-pane");
    scrollPane.setContent(textArea);
    scrollPane.setFitToWidth(true);
    scrollPane.setPrefWidth(400);
    scrollPane.setPrefHeight(180);
    VBox vBox = new VBox();
    vBox.visibleProperty().setValue(false);
    vBox.getChildren().addAll(scrollPane);
    root.getChildren().addAll(vBox);


    // --- Menu files
    Menu filesMenu = new Menu("Files");
    //Menu to load multiple files
    MenuItem loadFiles
            = new MenuItem("Load + process multiple files ");
    //MenuItem added to menu
```

```
filesMenu.getItems().addAll(loadFiles);
menuBar.getMenus().add(filesMenu);
root.getChildren().add(menuBar);

//Listener
loadFiles.setOnAction((ActionEvent t) ->
{
    FileChooser fileChooser = new FileChooser();

    //Name of file with extension .obj
    FileChooser.ExtensionFilter extFilter
            = new FileChooser
                    .ExtensionFilter("TXT files (*.obj)"
                            , "*.obj");
    fileChooser.getExtensionFilters().add(extFilter);
    //Files will be kept in a list
    List<File> files = new FileChooser()
            .showOpenMultipleDialog(primaryStage);
    //Each file is processed
    files.stream().forEach((File file) -> { try
    {
        //Text of file is read and mesh is extracted
        textOfFile = readFile(file);
        //Text copied to textArea
        textArea.appendText(textOfFile);

        } catch (IOException ex)
        {
            Logger.getLogger(PCAofHandBones
                    .class.getName())
                    .log(Level.SEVERE, null, ex);
        }
    });

//To remedy a not understood bug
//( = menuBar is added to the graphic )
root.getChildren().remove(menuBar);

vBox.visibleProperty().set(true);
});
```

```
//====== End of File loader

primaryStage.setTitle(title
        + ". Drag the mouse to rotate. "
    + " Ctrl + A = Axis visibility"
        );
scene.setFill(Color.AQUAMARINE);
primaryStage.setScene(scene);
primaryStage.show();
axes.buildAxes();
axes.handleKeyboard(scene);
root.getChildren().addAll(axes.axesXform);
camera.handleMouse(scene);
camera.handleMouse(scene);


//Hide textArea
Button hideTextArea = new Button();
hideTextArea.setText("View graphic");
//Listener
hideTextArea.setOnAction((ActionEvent t) ->
{
    vBox.visibleProperty().set(false);
    camera.buildCamera();
    /*
    bonesXform.setScale(scaleFactor);
    //Centering: found by trial and error.
    double xMax = bonesXform.getBoundsInLocal().getMaxX();
    double yMax = bonesXform.getBoundsInLocal().getMaxY();
    double zMax = bonesXform.getBoundsInLocal().getMaxZ();
    double xMin = bonesXform.getBoundsInLocal().getMinX();
    double yMin = bonesXform.getBoundsInLocal().getMinY();
    double zMin = bonesXform.getBoundsInLocal().getMinZ();
    double x = scaleFactor *  0.82 * (xMax + xMin) ;
    double y = scaleFactor * 0.78 * (yMax + yMin);
    double z = scaleFactor * 0.9 * (zMax + zMin) ;
    bonesXform.setTranslate(-x, -y, -z);
    */
    world3DXform.getChildren().add(bonesXform);
```

```
        root.getChildren().add(world3DXform);
        scene.setCamera(camera.camera);
    });

    vBox.getChildren().addAll(hideTextArea);
    primaryStage.show();
}



//File is read and its content is returned as a string.
//The associated mesh is recovered
//and correct orientation fo faces is provided.
//The mesh is a global variable.


private String readFile(File file)
        throws FileNotFoundException, IOException
{
    //Name of file is printed
    String name = file.getName();
    System.out.println(name);
    //Buffer to keep the content of file
    //StringBuilder stringBuffer = new StringBuilder();
    StringBuilder stringBuffer  = new StringBuilder();
    //Tool to read file
    BufferedReader bufferedReader = null;

    //A mesh is declared and instantiated
    TriangleMesh mesh  = new TriangleMesh();
    //The mesh has no predefined texture
    mesh.getTexCoords().addAll(0,0);
    //In the .obj format, counting begins with 1
    //but in Java, we begin with 0.
    //So, an additional inocuos point is added.
    mesh.getPoints().addAll(0,0,0);

    if (printReadFile)
        System.out.println("File content: ");
```

```
//File is read
try
{
//
bufferedReader = new BufferedReader(new FileReader(file));

String text;




while( (text = bufferedReader.readLine()) != null)
{
    //Text for the textArea
    stringBuffer.append(text);

    //Vertexes are read
    if (((text.length() > 0 ) && text.charAt(0) == 'v')
            && text.charAt(1) == ' ')
    {
        if (printReadFile)
            System.out.println("Vertex  = "
                    + counterOfVertexes + " = "+text);
        counterOfVertexes++;
        text = text.substring(2);
        int j = 0;
        //Three vertexes are read
        for(int i = 0; i < 3; i++)
        {
            double d ;
            String s = "";
            Boolean go = true;

            while (go)
            {
                char thisChar = text.charAt(j);
                s = s + thisChar;
                char nextChar = ' ';
                //++j means add one to j
                //and then evaluate expression with <
```

```
                if (++j < text.length())
                 nextChar = text.charAt(j);
                if ((nextChar == ' ') ||
                        (j >= text.length()))
                        go = false;

            }
            //string to double
            d = Double.parseDouble(s);
            float df = (float) d;
            if (printReadFile)
                System.out.println("Float number = " + df);
            mesh.getPoints().addAll(df);

        }
    }


    //Normal vectors are counted and displayed
    if ((text.length() > 0 ) && (text.charAt(0) == 'v')
            && (text.charAt(1) == 'n'))
    {

        if (printReadFile)
            System.out.println("Normal " + counterOfNormals
                + " = " + text);
        counterOfNormals++;
    }


    //Face is read, one face per line.
    //A face has 3 or more vertexes. These are extracted.
    //Additional information is ignored
    if ( (text.length() > 0 ) && (text.charAt(0) == 'f'))
    {

        if (printReadFile)
            System.out.println("Face  = " + counterOfFaces
                + " = "+text);
        counterOfFaces++;
```

```
text = text.substring(2);

int vertex[] = new int[4];
int j = 0;

Boolean isNotBlank ;
//Three vertexes are extracted
for(int i = 0; i < 3; i++)
{

    double d ;
    String s = "";
    Boolean go1 = true;


    while (go1)
    {
        if (j >= text.length()-2)
            break;

        char thisChar = text.charAt(j);
        s = s + thisChar;
        /*
        System.out.println(" j = "
                + j + "    s = " + s);
                */
        char nextChar = ' ';
        j++;
        if (j < text.length())
         nextChar = text.charAt(j);
        char nextNextChar;
        isNotBlank = true;
        if (nextChar == '/' )
        {
            go1 = false;
            //go until the following blank
            while ((isNotBlank)
                    && (j < text.length()-1))
            {
                j++;
```

```
                    nextNextChar = text.charAt(j);
                    if (nextNextChar == ' ' )
                        isNotBlank = false;

                }
            }

        }

        d = Double.parseDouble(s);
        int v = (int) d;
        vertex[i] = v;
        if (printReadFile)
            System.out.println("Int number = " + v);
    }//End of cycle for 3 vertexes

    //A face with 3 vertexes is added to mesh
    mesh.getFaces().addAll(vertex[0], 0, vertex[1], 0,
        vertex[2], 0);

    if (printReadFile)
    {
        for(int i = 0; i < 3; i++)
         System.out.print(vertex[i] + " . ");
        System.out.println();
    }

    //Is there a four vertex in the face?
    if   ( (text.length() - j )> 3)
    {
        String s = "";
        Boolean go1 = true;
        while (go1)
        {


            char thisChar = text.charAt(j);
            s = s + thisChar;
            j++;
            char nextChar = ' ';
```

```
                if (j < text.length())
                 nextChar = text.charAt(j);
                if (nextChar == '/' ) go1 = false;
            }
            if (printReadFile)
                System.out.println("4th = " + s);
            double d = Double.parseDouble(s);
            int v = (int) d;
            vertex[3] = v;

            //A polyface with 4 vertex, say, a,b,c,d,
            //can be covered by 2 triangles:
            //Triangle 1: a,b,c (Already added to mesh)
            //and triangle 2: a,c,d.
            //Triangle 2 with 3 vertexes is added to mesh
            mesh.getFaces().addAll(vertex[0], 0, vertex[2],
                    0, vertex[3], 0);
        }
    }//End of face cycle


        //Bounds of containing box are read and displayed
    if ( (text.length() > 0 )
            && (text.contains("# Bounds(mm):")))
    {
        System.out.println(text);

        String a = "# Bounds(mm): (";
        int m = a.length();
        text = text.substring(m);
        text = text.replace(")-(", ",");
        text = text.replace(")", ",");
        //System.out.println(text);

        int j = 0;


        //Six vertexes are extracted
        for(int i = 0; i < 6; i++)
        {
```

```
            double d ;
            String s = "";
            Boolean go1 = true;


            while (go1)
            {
                if (j >= text.length()-2)
                    break;

                char thisChar = text.charAt(j);
                s = s + thisChar;
                /*
                System.out.println(" j = "
                        + j + "    s = " + s);
                */
                char nextChar = ' ';
                j++;
                if (j < text.length())
                 nextChar = text.charAt(j);
                if (nextChar == ',' )
                {
                    go1 = false;
                   if ( j < text.length() ) j++;
                }

            }

            d = Double.parseDouble(s);


           if (printReadFile)
                System.out.println("Coordinate = " + i + "= " + d);
        }//End of cycle for 3 vertexes
    }

}

} //End of  try.
```

```
    //Exception management
    catch (FileNotFoundException ex)
    {
        Logger.getLogger(PCAofHandBones
        .class.getName()).log(Level.SEVERE, null, ex);
    }
    catch (IOException ex)
    {
        Logger.getLogger(PCAofHandBones
        .class.getName()).log(Level.SEVERE, null, ex);
    }
    finally
    {
        try
        {
            bufferedReader.close();
        } catch (IOException ex)
        {
            Logger.getLogger(PCAofHandBones
            .class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    if (printReadFile)
    {
        System.out.println("End of file content. ");
        //Other indicators are reported
        report(mesh);
    }

   //mesh = centerMesh(mesh);
    //======MeshWork=======


    PCA(mesh);

    MeshView boneMesh =
            new MeshView(mesh);

    PhongMaterial goldenMaterial
```

```
            = new PhongMaterial(Color.GOLDENROD);
    boneMesh.setMaterial(goldenMaterial);
    Group boneGroup = new Group( boneMesh);
    //Mesh is added to bonesXform
    bonesXform.getChildren().add(boneGroup);


    return stringBuffer.toString();
}




//Indicators of retrieved data are printed
private void report(TriangleMesh mesh)
{
    float[] destPoints = new float[mesh.getPoints().size()];
    mesh.getPoints().toArray(destPoints);
    System.out.println("POINTS");
    for(int j = 0; j <  destPoints.length/3; j++)
    {
        System.out.print(j + "   " );
        for(int k = 0; k < 3; k++)
        {
            System.out.print( destPoints[j*3 + k] + "   " );
        }
        System.out.println();
    }

    destFaces = new int[mesh.getFaces().size()];
    //Faces go to a matrix
    mesh.getFaces().toArray(destFaces);
    System.out.println("FACES WITH TEXTURE COORDINATES");
    for(int j = 0; j <  destFaces.length/6; j++)
    {
        System.out.print(j + "   " );
        for(int k = 0; k < 6; k++)
        {
            System.out.print( destFaces[j*6 + k] + "   " );
```

```
        }
        System.out.println();
    }

    System.out.println("counterOfPoints = "
            + counterOfVertexes );
    System.out.println("counterOfFaces = "
            + counterOfFaces );
    System.out.println("counterOfNormals = "
            + counterOfNormals);
    System.out.println("mesh.getPoints().size() = "
            + mesh.getPoints().size() );
    System.out.println("mesh.getFaces().size() = "
            + mesh.getFaces().size() );

    System.out.println("getPointElementSize = "
            + mesh.getPointElementSize() );
    System.out.println("getfaceElementSize = "
            + mesh.getFaceElementSize() );

}

private static void print (String s)
{
   System.out.print(s);
}
public static String fixedWidthDoubletoString
        (double x, int w, int d) {
   java.text.DecimalFormat fmt
           = new java.text.DecimalFormat();
   fmt.setMaximumFractionDigits(d);
   fmt.setMinimumFractionDigits(d);
   fmt.setGroupingUsed(false);
   String s = fmt.format(x);
   while (s.length() < w) {
      s = " " + s;
   }
   return s;
}
```

```
//Principal component analysis of a mesh
private void PCA(TriangleMesh mesh)
{
    //Point are declared as a matrix of coordinates
    int n = mesh.getPoints().size();
        float nPoints = n;
    nPoints = nPoints/3;
    float[][] CoordinatesFloat = new float[(int) nPoints][3];


    /*
    //Test: adapted from
    //Stat Trek: Variance-Covariance Matrix
    //http://stattrek.com/matrix-algebra/covariance-matrix.aspx

    float[] data = {90,  60,  90,
                    90,  90,  30,
                    60,  60,  60,
                    60,  60,  90,
                    30,  30,  30 };


        n = 15;
        nPoints = 5;
    int counter = 0;
    for(int i = 0; i < n; i = i+3)
    {
        //Reading
        float x =  data[i];
        float y =  data[i+1] ;
        float z =  data[i+2] ;
        CoordinatesFloat[counter][0] = x;
        CoordinatesFloat[counter][1] = y;
        CoordinatesFloat[counter][2] = z;
        counter++;
    }

    //Mean is found
    float[] mean = new float[3];
```

```
for(int i = 0; i < 3; i++)
{
    mean[i] = 0;
    for (int j = 0; j <  nPoints; j++)
        mean[i] = mean[i] + CoordinatesFloat[j][i];
    mean[i] = mean[i]/(nPoints);
}

System.out.println("meanX = " + mean[0]);
System.out.println("meanY = " + mean[1]);
System.out.println("meanZ = " + mean[2]);




//Calculate covariance matrix
double[][] Cov = new double[3][3];
for(int i = 0; i < 3; i++)
{
    for (int j = 0; j <  3; j++)
    {
        Cov[i][j] = 0;
        for (int k = 0; k <  nPoints; k++)
            Cov[i][j] = Cov[i][j] +
                    (CoordinatesFloat[k][i] - mean[i])
                    * (CoordinatesFloat[k][j] - mean[j]);

        Cov[i][j] = Cov[i][j]/nPoints;

    }

}

printMatrix(Cov, "Cov");

//Answer: cov matrix must be:
//504  360  180
//360  360  0
//180  0  720
*/
```

```
//=============

int counter = 0;
for(int i = 0; i < n; i = i+3)
{
    //Reading
    float x =  mesh.getPoints().get(i);
    float y =  mesh.getPoints().get(i+1);
    float z =  mesh.getPoints().get(i+2);
    CoordinatesFloat[counter][0] = x;
    CoordinatesFloat[counter][1] = y;
    CoordinatesFloat[counter][2] = z;
    counter++;
    //Warning: the zeroth point is artificial,
    //filled with zeros.
    if (i == 0)
        System.out.println("Zeroth point: x = " + x + "   y =   "
                + y + "   z =  "  + z);
}

//Mean is found. It is vector with 3 coordinates
float[] mean = new float[3];

for(int i = 0; i < 3; i++)
{
    mean[i] = 0;
    //Zeroth point does not count
    for (int j = 1; j <  nPoints; j++)
        mean[i] = mean[i] + CoordinatesFloat[j][i];
    mean[i] = mean[i]/(nPoints -1);
}

System.out.println("meanX = " + mean[0]);
System.out.println("meanY = " + mean[1]);
System.out.println("meanZ = " + mean[2]);
```

```
//Calculate covariance matrix
double[][] Cov = new double[3][3];
for(int i = 0; i < 3; i++)
{
    for (int j = 0; j <  3; j++)
    {
        Cov[i][j] = 0;
        for (int k = 1; k <  nPoints; k++)
            Cov[i][j] = Cov[i][j] +
                    (CoordinatesFloat[k][i] - mean[i])
                    * (CoordinatesFloat[k][j] - mean[j]);

        Cov[i][j] = Cov[i][j]/(nPoints-1);

    }

}

printMatrix(Cov, "Cov");

//=========

//Eigenvalues of Cov matrix are found
Matrix M = new Matrix(Cov);
EigenvalueDecomposition E =
     new EigenvalueDecomposition(M);
double[] d = E.getRealEigenvalues();

for(int i = 0; i < 3; i++)
    System.out.println("Eigenvalue " + i + " : " + d[i]);
double[][] EV = E.getV().getArray();
for(int i = 0; i < 3; i++)
{
    System.out.println("Eigenvector " + i + " : " );
    for (int j = 0; j <  3; j++)
    {
        System.out.println(EV[i][j]);
    }
    System.out.println();
}
```

```
//Test: eigenvectors of a real symmetric matrix
//are orthogonal one to another: dot product must be zero
System.out.println("Orthogonality test : matrix I is expected");
double[][] Ort = new double[3][3];
for(int i = 0; i < 3; i++)
{

    for (int j = 0; j <  3; j++)
    {

        //sum = sum + EV[i][j]*EV[i+1][j];
        Ort[i][j] = dotProduct(EV[i], EV[j]);

    }
    //System.out.println(sum);

}

printMatrix(Ort, "Ort test");

//Determinant must be 1.
System.out.println("Orientation test: "
        + "+1 is expected. Det E = " + E.getV().det() + " \n");



//Bone is centered
bonesXform.setTranslate(-mean[0], -mean[1], -mean[2]);
//Axes are made visible
axes.axesGroup.setVisible(true);



//Implementation of Euler technology

//Coordinates of xVector : first eigenvector
double a = EV[0][0];
double b = EV[0][1];
double c = EV[0][2];
```

```
double[] xVector = {a,b,c};

//Coordinates of zVector: third eigenvector
double g = EV[2][0];
double h = EV[2][1];
double i = EV[2][2];

//double[] zVector = {g,h,i};


double n1 =  (h/g)/Math.sqrt(1 +  h*h/(g*g));
double n2 =  1.0/Math.sqrt(1 +  h*h/(g*g));

Point3D nPoint3D = new Point3D(n1,n2,0);

double[] nVector = {n1,n2,0};
//Test: norm^2 of nVector must be 1
double norm2 = n1*n1 + n2*n2;
System.out.println("n1 = " +  n1);
System.out.println("n2 = " +  n2);
System.out.println("Test: Norm^2 of nVector must be 1= "
        +  norm2);
//Conversion factor from radians to degrees
double cf = 180.0 /Math.PI;
double gamma = cf * Math.acos(dotProduct(nVector, xVector));
double beta = cf* Math.acos(i);
double alpha = cf * Math.acos(n1);

System.out.println("Alpha = " + alpha);
System.out.println("Beta = " + beta);
System.out.println("Gamma = " + gamma);


//Tinkering  with signs
alpha = alpha;
beta = -beta;
gamma = -gamma;

//Axes are aligned with the bone
```

```
    axes.axesXform.setRotationAxis(Z_AXIS);
    axes.axesXform.setRotate(alpha );

    axes.axesXform.setRotationAxis(nPoint3D);
    axes.axesXform.setRotate(beta );

    axes.axesXform.setRotationAxis(Y_AXIS);
    axes.axesXform.setRotate(gamma );
}

//============== Matrix algebra ==============


private static void printMatrix(double m[][], String title)
{
    System.out.println("This is the matrix " + title);
    for(int i=0;i<3;i++)
        System.out.println(m[i][0] + " " + m[i][1]+ " "
                                              + m[i][2]);
    System.out.println();
}


  //To find a matrix of a rotation, one must find first
  //an orthogonal basis with 3 vectors
  //The third vector is the axis of rotation
  private static  double[] thirdVector(double[] v)
  {
    double a = v[0];
    double b = v[1];
    double c = v[2];
    double norm = Math.sqrt(a*a + b*b + c*c);
    //System.out.println("norm = " + norm);
    double[] helpV = new double[3];
    helpV[0] = a/norm;
    helpV[1]= b/norm;
    helpV[2] = c/norm;
    /*System.out.println("Third vector " + helpV[0]+
                                  helpV[1]+helpV[2]);*/
    return helpV;
```

```
}

//The second vector is perpendicular to the first
private static  double[] secondVector(double[] v)
{
  double[] helpV = new double[3];
  double   x,y,z;
  double a = v[0];
  double b = v[1];
  double c = v[2];
  x=b;
  y = -a;
  z=0;
  if (b==0)
     {
          x=c;
          z = -a;
          y=0;
     }
  if (a==0)
     {
          y=c;
          z = -b;
          x=0;
     }
  double norm = Math.sqrt(x*x + y*y + z*z);

  helpV[0] = x/norm;
  helpV[1]  = y/norm;
  helpV[2]  = z/norm;
  return helpV;
}

//Normalized crux product of vectors v and w
private static  double[] cruxProduct(double[] v,
                                     double[] w)
{
  double a,b,c;
  double[] helpV = new double[3];
  a=  (v[1]*w[2] -w[1]*v[2]);
```

```
    b= - (v[0]*w[2]-w[0]*v[2] );
    c=  (v[0]*w[1]  -w[0]*v[1] );
    double norm = Math.sqrt(a*a + b*b + c*c);
    helpV[0] = a/norm;
    helpV[1]  = b/norm;
    helpV[2]  = c/norm;
    return helpV;

  }

  //Normalized crux product of vectors v and w
  private static  double dotProduct(double[] v,
                                    double[] w)
  {
    return v[0]*w[0] + v[1]*w[1] + v[2]*w[2];

  }




//An axis (a,b,c) generates a basis B
 private static  double[][] basisB(double v[])
 {
    double[][]  helpM = new double[3][3];
    double[]   helpT, helpS, helpF;

    helpT = thirdVector(v);
    helpS = secondVector(v);
    //The first vector is the crux product of the other 2.
    helpF = cruxProduct(helpT,helpS);
    for(int i=0; i<3;i++)
    {
     helpM[i][0] = helpF[i];
     helpM[i][1] = helpS[i];
     helpM[i][2] = helpT[i];
    }

    return helpM;
```

```
  }

public static void main(String[] args)
{

    launch(args);
}

}//End of main class N40 PCAofHandBones3
```

41 ***Exercise.*** *Run the previous program and play with the code.*

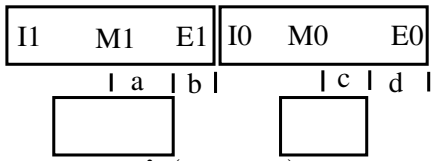
42 ***Exercise.*** *Implement PCA mutations of a bone. This type of mutations are defined as scale changes along anyone principal axis. These mutations represent change without change of morphological pattern.* Answer

43 ***Exercise.*** *The previous program shows that PCA mutations can produce monstrosities that are not seen in nature. Implement cylindrical mutations to see how natural they look.* Answer

44 ***Exercise.*** *Implement PCA mutations over the bones of the hand separately and show the mutant hand. Animate it to see the effect of mutations. Redact your conclusion: Do PCA mutations have a place in nature?* Answer

45 ***Exercise.*** *Implement small cylindrical mutations over the bones of the hand separately and show the mutant hand. Animate it to see the effect of mutations. Redact your conclusion: Do small cylindrical mutations have a place in nature?* Answer

46 ***A bit of professionalism.*** *Let us make an effort to add a professional flavor to the previous application: mutant bones must be connected appropriately without voids or overlapping. Our try is based on the following conceptualization about the geometric relation of bones and their mutant shapes:*



$$a = \lambda_1(M1 - E1)$$
$$b = (1-\lambda_1)(M1 - E1)$$
$$c = \lambda_0(M0 - E0)$$
$$d = (1-\lambda_0)(M0 - E0)$$

Figure 3.4. A scaling application that produces a shrinking is shown. The upper row shows two bones in their original size. Bellow they are shrunk, the first by factor λ_1, the second by factor λ_0. A translation is needed to tie the ends of bones that became separated. Needed translation adds to $b+d$.

The code follows:

```
/*
    Program N46 Cylindricalmutations2
    This program shows how to accommodate
    the phalanges of the  index finger
    after  small cylindrical mutations.
    A cylindrical mutation over a cylinder
    may enlarge its radius or its height.
    PCA (Principal Component Analysis) allows to fit
    the best cylinder over a bone,
    which is then  mutated cylindrically.

    Tested only with some  files,
    particularly  with those of the BodyParts3D project.
    http://lifesciencedb.jp/bp3d/
    Those files are licensed:

    The license for this database is specified in the
    Creative Commons Attribution-Share Alike 2.1 Japan.
    If you use data from this database, please be sure
    attribute this database as follows: "BodyParts3D, (c)
    The Database Center for Life Science licensed under
    CC Attribution-Share Alike 2.1 Japan".
    http://dbguidehive.biosciencedbc.jp/en/bodyparts3d/lic.html
```

```
 */

package Programs;

//Files in the hosting project Programs
import Jama.EigenvalueDecomposition;
import Jama.Matrix;
import Tools.Axes;
import Tools.Cameras;
import Tools.Xform;
import Tools.XformWithPivot;


import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.List;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.binding.NumberBinding;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.Menu;
```

```
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.Slider;
import javafx.scene.control.TextArea;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.MeshView;;
import javafx.scene.shape.TriangleMesh;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import javafx.util.Duration;



public class CylindricalMutations2 extends Application
{
    //Visibility of the mesh
    //might depend on the correct choice
    //of this factor. Move it in within 1 and 500.
    private static final float scaleFactor = 5;

    //Scale factors along width, height and length
    static private float scaleFactorW = 1;
    static private float scaleFactorH = 1;
    static private float scaleFactorL = 1;

    //Printing for debugging
    final static boolean printReadFile = false;
    private static final String title
            =  "Importer of meshes in format ";

    //Tool to read files
    BufferedReader br ;
    private static  String textOfFile = "";
```

```
private static int counterOfFaces = 0;
private static int counterOfVertexes = 0;
private static int counterOfNormals = 0;

private static int[] destFaces ;

private static TriangleMesh[] bonesMeshesArray
        = new TriangleMesh[100];
//Array of Xforms
private static Xform[] bonesXformsArray
        = new Xform[100];
private static MeshView[] bonesViewArray
        = new MeshView[100];
private static final String[] names1  = new String[100];
private static final String[] names2  = new String[100];
//Group of Xforms
private static Xform bonesXforms  = new Xform();
//Counter
private static int counterOfBones = 0;

//*****************

final static Group root = new Group();

final static Group axisGroup = new Group();


final static Xform cameraXform = new Xform();
final static Xform cameraXform2 = new Xform();
final static Xform cameraXform3 = new Xform();

final static double cameraDistance = 0; //1450;//450
final static Xform world3DXform = new Xform();
final static Xform bonesXform = new Xform();

//Axes : ctrl + A
private static final Axes axes = new Axes();
private static final Cameras camera = new Cameras();

boolean timelinePlaying = false;
```

```
    double ONE_FRAME = 1.0/24.0;
    double DELTA_MULTIPLIER = 200.0;
    double CONTROL_MULTIPLIER = 0.1;
    double SHIFT_MULTIPLIER = 0.1;
    double ALT_MULTIPLIER = 0.5;

    double mousePosX;
    double mousePosY;
    double mouseOldX;
    double mouseOldY;
    double mouseDeltaX;
    double mouseDeltaY;




   //Menubar declaration and initialization
    MenuBar menuBar = new MenuBar();


//This is the outer pane to place tabs
private static BorderPane outerPane = new BorderPane();
private static Pane paneHand = new Pane();

//Slider for zooming
    //private  static Slider zSlider;
    //Position of graphic along the z-axis,
    //normal to the screen.
    //It is a double property because
    //it will get involved into a binding
    //private  static final DoubleProperty zPos
    //        = new SimpleDoubleProperty(200);

    //Slider for angle of rotation of fingers
    private  static Slider closeHandSlider;
    //To close the hand,
    //each phalange suffers a rotation.
    //It is a double property because
    //it will get involved into a binding
    private  static final DoubleProperty closeHandProperty
```

```
        = new SimpleDoubleProperty(0);

//Slider for opening fingers thanks to
//their transversal rotation
private  static Slider openFingersSlider;
//To open the fingers,
//each phalange suffers a transversal rotation.
//It is a double property because
//it will get involved into a binding
private  static final DoubleProperty openFingersProperty
        = new SimpleDoubleProperty(0);

//Slider for opening/closing the thumb
private  static Slider closeThumbSlider;
//To close the thumb we need a rotation.
//It is a double property because
//it will get involved into a binding
private  static final DoubleProperty closeThumbProperty
        = new SimpleDoubleProperty(0);

//Slider for rotating the thumb to form the grip
private  static Slider rotateThumbSlider;
//To form the grip we need a rotation of the thumb.
//It is a double property because
//it will get involved into a binding
private  static final DoubleProperty rotateThumbProperty
        = new SimpleDoubleProperty(0);


//Nested definition of   nodes is implemented
//to have, say,  the palm dragging the fingers,
//and a phalanx dragging the rest of its the finger.


//Each finger generates various natural groups:
//Group with the extreme phalanx of finger0
//Symbolic initialization
private static final XformWithPivot finger0Ph2Group
        = new XformWithPivot();
//Group with the two extreme phalanges
```

```
private static final XformWithPivot finger0Ph21Group
        = new XformWithPivot();
//Group with all three phalanges. It is the finger
private static final XformWithPivot finger0Ph210Group
        = new XformWithPivot();
//Finger1
private static final XformWithPivot finger1Ph2Group
        = new XformWithPivot();
private static final XformWithPivot finger1Ph21Group
        = new XformWithPivot();
private static final XformWithPivot finger1Ph210Group
        = new XformWithPivot();
//Finger2
private static final XformWithPivot finger2Ph2Group
        = new XformWithPivot();
private static final XformWithPivot finger2Ph21Group
        = new XformWithPivot();
private static final XformWithPivot finger2Ph210Group
        = new XformWithPivot();
//Finger3
private static final XformWithPivot finger3Ph2Group
        = new XformWithPivot();
private static final XformWithPivot finger3Ph21Group
        = new XformWithPivot();
private static final XformWithPivot finger3Ph210Group
        = new XformWithPivot();
//Thumb
private static final XformWithPivot thumbPh2Group
        = new XformWithPivot();
private static final XformWithPivot thumbPh21Group
        = new XformWithPivot();
private static final XformWithPivot thumbPh210Group
        = new XformWithPivot();

 private static final XformWithPivot wristGroup
        = new XformWithPivot();
//Group with wrist and fingers
private static final XformWithPivot palmGroup
        = new XformWithPivot();
```

```
    private static final Timeline animation
            = new Timeline();

    private static final double[][] coordBoxes
            = new double[100][6];

    private static BoneDescription[] bones
            = new BoneDescription[30];



    Random r = new Random();

@Override
public void start(Stage primaryStage)
{
    System.out.println("start");

    Scene scene = new Scene(root, 1200, 1000, true);

    paneHand.setPrefSize(500, 500);

    outerPane.setOpacity(0);
    //outerPane.setLayoutX(700);
    //outerPane.setLayoutY(700);

    /*
       Multiple file loader
    */

    //Content of files will be shown in textArea
    final TextArea textArea = new TextArea();
    textArea.prefWidth(400);
    textArea.setWrapText(true);
    textArea.setText("");
    //Container  for textArea.
    ScrollPane scrollPane = new ScrollPane();
    scrollPane.getStyleClass().add("noborder-scroll-pane");
    scrollPane.setContent(textArea);
    scrollPane.setFitToWidth(true);
```

```
scrollPane.setPrefWidth(400);
scrollPane.setPrefHeight(180);
VBox vBox = new VBox();
vBox.visibleProperty().setValue(false);
vBox.getChildren().addAll(scrollPane);
root.getChildren().addAll(vBox);


// --- Menu files
Menu filesMenu = new Menu("Files");
//Menu to load multiple files
MenuItem loadFiles
        = new MenuItem("Load + process multiple files ");
//MenuItem added to menu
filesMenu.getItems().addAll(loadFiles);
menuBar.getMenus().add(filesMenu);
root.getChildren().add(menuBar);

//Listener
loadFiles.setOnAction((ActionEvent t) ->
{
    FileChooser fileChooser = new FileChooser();

    //Name of file with extension "
    FileChooser.ExtensionFilter extFilter
            = new FileChooser
                    .ExtensionFilter("TXT files (*.obj)"
                            , "*.obj");
    fileChooser.getExtensionFilters().add(extFilter);

    //Files will be kept in a list
    List<File> files = new FileChooser()
            .showOpenMultipleDialog(primaryStage);



        //Delete possible repetitions
        //files = deleteRepetitions(files);
    //counterOfBones = 0;
```

```
for(int j = 0; j < files.size(); j++)
{
    File file = files.get(j);
    String name = file.getName();
    names1[j] = name;
}
counterOfBones = 0;
//Each file is processed
for(int j = 0; j < files.size(); j++)
{

    String name = names1[j];
    boolean repeated = false;
    for (int i = 0; i <  j; i++)
    {
        if (name.equals(names1[i]))
        {
            repeated = true;

            //System.out.println("Repeated" + name);
        }
    }

    if (repeated);
    else
    try
    {
        File file = files.get(j);
        //Text of file is read and mesh is extracted
        textOfFile = readFile(file);
        //Text copied to textArea
        textArea.appendText(textOfFile);

    } catch (IOException ex)
        {
            Logger.getLogger(AnimatedHandSkeleton
                    .class.getName())
                    .log(Level.SEVERE, null, ex);
```

```
            }



    }

    defineGroups();



//To remedy a not understood bug
//( = menuBar is added to the graphic )
root.getChildren().remove(menuBar);

vBox.visibleProperty().set(true);
});

//====== End of File loader

primaryStage.setTitle(title
        + ". Drag the mouse to rotate. "
    + " Ctrl + A = Axis visibility"
        );
scene.setFill(Color.BLUE);
primaryStage.setScene(scene);
primaryStage.show();
axes.buildAxes();
axes.handleKeyboard(scene);
//camera.handleMouse(scene);
camera.handleMouse(paneHand);

//Hide textArea and show graphic
Button hideTextArea = new Button();
hideTextArea.setText("View graphic");
//Listener
hideTextArea.setOnAction((ActionEvent t) ->
{
    vBox.visibleProperty().set(false);
    camera.buildCamera();
```

```
        world3DXform.getChildren().add(bonesXform);


        //world3DXform.setRx(180);
        //world3DXform.setRy(90);
        world3DXform.setScale(scaleFactor);
        world3DXform.setTranslate(-3500, -1400, 0);

        paneHand.getChildren().add(world3DXform);
        root.getChildren().add(paneHand);
        scene.setCamera(camera.camera);
    });

    vBox.getChildren().addAll(hideTextArea);



    //This slider   controls the closing of the hand.
    closeHandSlider = new Slider();
    closeHandSlider.setMax(90);
    closeHandSlider.setMin(0);
    closeHandSlider.setValue(10);

    //To close the hand, an angle of rotation is enabled.
    //Bind angleProperty to the reading of the slider
    closeHandSlider.valueProperty()
            .bindBidirectional(closeHandProperty);


    //This slider controls the openning of fingers.
    openFingersSlider = new Slider();
    openFingersSlider.setMax(15);
    openFingersSlider.setMin(0);
    openFingersSlider.setValue(10);

    //To open fingers, an angle of rotation is enabled.
    //Bind openFingersProperty to the reading of the slider
    openFingersSlider.valueProperty()
```

```
        .bindBidirectional(openFingersProperty);


//Slider to open  the thumb
closeThumbSlider = new Slider();
closeThumbSlider.setMax(55);
closeThumbSlider.setMin(0);
closeThumbSlider.setValue(-0);
closeThumbSlider.adjustValue(closeThumbSlider.getValue());

//To open the thumb, an angle of rotation is enabled.
//Bind closeThumbProperty to the reading of the slider
closeThumbSlider.valueProperty()
        .bindBidirectional(closeThumbProperty);


//Slider to rotate  the thumb
rotateThumbSlider = new Slider();
rotateThumbSlider.setMax(50);
rotateThumbSlider.setMin(0);
rotateThumbSlider.setValue(0);
rotateThumbSlider
        .adjustValue(rotateThumbSlider.getValue());

//To rotate the thumb, an angle of rotation is enabled.
//Bind closeThumbProperty to the reading of the slider
rotateThumbSlider.valueProperty()
        .bindBidirectional(rotateThumbProperty);



//Other JavaFX controls, labels and containers
GridPane gridPane = new GridPane();
gridPane.setVgap(4);
gridPane.setHgap(10);
gridPane.setPadding(new Insets(5, 5, 5, 5));
//Column 0, Row   0
gridPane.add(new Label("Zoom: "), 0, 0);
//gridPane.add(zSlider, 1, 0);
gridPane.add(new Label("Close hand: "), 2, 0);
```

```
    gridPane.add(closeHandSlider, 3, 0);
    gridPane.add(new Label("Open fingers: "), 0, 1);
    gridPane.add(openFingersSlider, 1, 1);
    gridPane.add(new Label("Close thumb: "), 0, 2);
    gridPane.add(closeThumbSlider, 1, 2);
    gridPane.add(new Label("Rotate thumb: "), 2, 2);
    gridPane.add(rotateThumbSlider, 3,2);



    SubScene subScene
            = new SubScene(gridPane, 880, 200);
    subScene.setLayoutX(10);
    subScene.setLayoutY(800);
    paneHand.getChildren().add(subScene);









    primaryStage.show();
}




//File is read and its content is returned as a string.
//The associated mesh is recovered
//and correct orientation of faces is provided.
```

```
//The mesh is a global variable.
private  String readFile(File file)
        throws FileNotFoundException, IOException
{
    //Name of file is printed
    String name = file.getName();
    //if (printReadFile)
        System.out.println(name);
    names2[counterOfBones] = name;
    BoneDescription b = new BoneDescription();
    b.name = name;
    bones[counterOfBones] = b;

    //Buffer to keep the content of file
    //StringBuilder stringBuffer = new StringBuilder();
    StringBuilder stringBuffer  = new StringBuilder();
    //Tool to read file
    BufferedReader bufferedReader = null;

    //A mesh is declared and instantiated
    TriangleMesh mesh  = new TriangleMesh();
    //The mesh has no predefined texture
    mesh.getTexCoords().addAll(0,0);
    //In the " format, counting begins with 1
    //but in Java, we begin with 0.
    //So, an additional inocuos point is added.
    mesh.getPoints().addAll(0,0,0);

    if (printReadFile)
        System.out.println("File content: ");
//File is read
try
{
//
bufferedReader = new BufferedReader(new FileReader(file));

String text;
```

```
while( (text = bufferedReader.readLine()) != null)
{
    //Text for the textArea
    stringBuffer.append(text);
    //Print each line
    //System.out.println(text);

    //Vertexes are read
    if (((text.length() > 0 ) && text.charAt(0) == 'v')
            && text.charAt(1) == ' ')
    {
        if (printReadFile)
            System.out.println("Vertex  = "
                    + counterOfVertexes + " = "+text);
        counterOfVertexes++;
        text = text.substring(2);
        int j = 0;
        //Three vertexes are read
        for(int i = 0; i < 3; i++)
        {
            double d ;
            String s = "";
            Boolean go = true;

            while (go)
            {
                char thisChar = text.charAt(j);
                s = s + thisChar;
                char nextChar = ' ';
                //++j means add one to j
                //and then evaluate expression with <
                if (++j < text.length())
                 nextChar = text.charAt(j);
                if ((nextChar == ' ') ||
                        (j >= text.length()))
                        go = false;

            }
            //string to double
```

```
            d = Double.parseDouble(s);
            float df = (float) d;
            if (printReadFile)
                System.out.println("Float number = " + df);
            mesh.getPoints().addAll(df);

        }
    }


    //Normal vectors are counted and displayed
    if ((text.length() > 0 ) && (text.charAt(0) == 'v')
            && (text.charAt(1) == 'n'))
    {

        if (printReadFile)
            System.out.println("Normal " + counterOfNormals
                + " = " + text);
        counterOfNormals++;
    }


    //Face is read, one face per line.
    //A face has 3 or more vertexes. These are extracted.
    //Additional information is ignored
    if ( (text.length() > 0 ) && (text.charAt(0) == 'f'))
    {

        if (printReadFile)
            System.out.println("Face  = " + counterOfFaces
                + " = "+text);
        counterOfFaces++;
        text = text.substring(2);

        int vertex[] = new int[4];
        int j = 0;

        Boolean isNotBlank ;
        //Three vertexes are extracted
        for(int i = 0; i < 3; i++)
```

```
{

    double d ;
    String s = "";
    Boolean go1 = true;


    while (go1)
    {
        if (j >= text.length()-2)
            break;

        char thisChar = text.charAt(j);
        s = s + thisChar;
        /*
        System.out.println(" j = "
                + j + "    s = " + s);
                */
        char nextChar = ' ';
        j++;
        if (j < text.length())
         nextChar = text.charAt(j);
        char nextNextChar = ' ';
        isNotBlank = true;
        if (nextChar == '/' )
        {
            go1 = false;
            //go until the following blank
            while ((isNotBlank)
                    && (j < text.length()-1))
            {
                j++;
                nextNextChar = text.charAt(j);
                if (nextNextChar == ' ' )
                    isNotBlank = false;

            }
        }

    }
```

```
    d = Double.parseDouble(s);
    int v = (int) d;
    vertex[i] = v;
    if (printReadFile)
        System.out.println("Int number = " + v);
}//End of cycle for 3 vertexes

//A face with 3 vertexes is added to mesh
mesh.getFaces().addAll(vertex[0], 0, vertex[1], 0,
    vertex[2], 0);

if (printReadFile)
{
    for(int i = 0; i < 3; i++)
     System.out.print(vertex[i] + " . ");
    System.out.println();
}

//Is there a four vertex in the face?
if   ( (text.length() - j )> 3)
{
    String s = "";
    Boolean go1 = true;
    while (go1)
    {


        char thisChar = text.charAt(j);
        s = s + thisChar;
        j++;
        char nextChar = ' ';
        if (j < text.length())
         nextChar = text.charAt(j);
        if (nextChar == '/' ) go1 = false;
    }
    if (printReadFile)
        System.out.println("4th = " + s);
    double d = Double.parseDouble(s);
    int v = (int) d;
```

```
            vertex[3] = v;

            //A polyface with 4 vertex, say, a,b,c,d,
            //can be covered by 2 triangles:
            //Triangle 1: a,b,c (Already added to mesh)
            //and triangle 2: a,c,d.
            //Triangle 2 with 3 vertexes is added to mesh
            mesh.getFaces().addAll(vertex[0], 0, vertex[2],
                    0, vertex[3], 0);
        }
    }//End of face cycle

    //Bounds of containing box are read and displayed
    if ( (text.length() > 0 )
            && (text.contains("# Bounds(mm):")))
    {
        //System.out.println(text);
        System.out.print("Bounds =   ");
        String a = "# Bounds(mm): (";
        int m = a.length();
        text = text.substring(m);
        text = text.replace(")-(",   ",   ");
        text = text.replace(")",   ",   ");
        System.out.println(text);

        int j = 0;


        //Six vertexes are extracted
        for(int i = 0; i < 6; i++)
        {

            double d ;
            String s = "";
            Boolean go1 = true;


            while (go1)
            {
                if (j >= text.length()-2)
```

```
                    break;

                char thisChar = text.charAt(j);
                s = s + thisChar;
                /*
                System.out.println(" j = "
                        + j + "    s = " + s);
                */
                char nextChar = ' ';
                j++;
                if (j < text.length())
                 nextChar = text.charAt(j);
                if (nextChar == ',' )
                {
                    go1 = false;
                   if ( j < text.length() ) j++;
                }

            }

            d = Double.parseDouble(s);

            coordBoxes[counterOfBones][i] = d;
           if (printReadFile)
                System.out.println("Coordinate = "
                        + i + "= " + d);
        }//End of cycle for 3 vertexes
    }

}

} //End of  try.
    //Exception management
    catch (FileNotFoundException ex)
    {
        Logger.getLogger(AnimatedHandSkeleton
        .class.getName()).log(Level.SEVERE, null, ex);
    }
    catch (IOException ex)
    {
```

```
    Logger.getLogger(AnimatedHandSkeleton
    .class.getName()).log(Level.SEVERE, null, ex);
}
finally
{
    try
    {
        bufferedReader.close();
    } catch (IOException ex)
    {
        Logger.getLogger(AnimatedHandSkeleton
        .class.getName()).log(Level.SEVERE, null, ex);
    }
}



if (printReadFile)
{
    System.out.println("End of file content. ");
    //Other indicators are reported
    report(mesh);
}


//======MeshWork=======

PCA(mesh);

MeshView boneMesh =
        new MeshView(mesh);

PhongMaterial goldenMaterial
        = new PhongMaterial(Color.GOLDENROD);
boneMesh.setMaterial(goldenMaterial);
Group boneGroup = new Group(boneMesh);
bonesViewArray[counterOfBones] = boneMesh;

Xform boneXform = new Xform();
boneXform.getChildren().add(boneGroup);
```

```
    bonesXformsArray[counterOfBones] = boneXform;
    bonesXform.getChildren().add(boneXform);
    counterOfBones++;

    return stringBuffer.toString();
}


//Indicators of retrieved data are printed
private void report(TriangleMesh mesh)
{
    float[] destPoints = new float[mesh.getPoints().size()];
    mesh.getPoints().toArray(destPoints);
    System.out.println("POINTS");
    for(int j = 0; j <  destPoints.length/3; j++)
    {
        System.out.print(j + "   " );
        for(int k = 0; k < 3; k++)
        {
            System.out.print( destPoints[j*3 + k] + "   " );
        }
        System.out.println();
    }

    destFaces = new int[mesh.getFaces().size()];
    //Faces go to a matrix
    mesh.getFaces().toArray(destFaces);
    System.out.println("FACES WITH TEXTURE COORDINATES");
    for(int j = 0; j <  destFaces.length/6; j++)
    {
        System.out.print(j + "   " );
        for(int k = 0; k < 6; k++)
        {
            System.out.print( destFaces[j*6 + k] + "   " );
        }
        System.out.println();
    }

    System.out.println("counterOfPoints = "
            + counterOfVertexes );
```

```
    System.out.println("counterOfFaces = "
            + counterOfFaces );
    System.out.println("counterOfNormals = "
            + counterOfNormals);
    System.out.println("mesh.getPoints().size() = "
            + mesh.getPoints().size() );
    System.out.println("mesh.getFaces().size() = "
            + mesh.getFaces().size() );

    System.out.println("getPointElementSize = "
            + mesh.getPointElementSize() );
    System.out.println("getfaceElementSize = "
            + mesh.getFaceElementSize() );

}



public static String fixedWidthDoubletoString
        (double x, int w, int d) {
   java.text.DecimalFormat fmt
           = new java.text.DecimalFormat();
   fmt.setMaximumFractionDigits(d);
   fmt.setMinimumFractionDigits(d);
   fmt.setGroupingUsed(false);
   String s = fmt.format(x);
   while (s.length() < w) {
      s = " " + s;
   }
   return s;
}

//Principal component analysis of a mesh
private void PCA(TriangleMesh mesh)
{
    //Point are declared as a matrix of coordinates
    int n = mesh.getPoints().size();
        float nPoints = n;
    nPoints = nPoints/3;
    float[][] CoordinatesFloat = new float[(int) nPoints][3];
```

```
/*
//Test: adapted from
//Stat Trek: Variance-Covariance Matrix
//http://stattrek.com/matrix-algebra/covariance-matrix.aspx

float[] data = {90,  60,  90,
                90,  90,  30,
                60,  60,  60,
                60,  60,  90,
                30,  30,  30 };


    n = 15;
    nPoints = 5;
int counter = 0;
for(int i = 0; i < n; i = i+3)
{
    //Reading
    float x =  data[i];
    float y =  data[i+1] ;
    float z =  data[i+2] ;
    CoordinatesFloat[counter][0] = x;
    CoordinatesFloat[counter][1] = y;
    CoordinatesFloat[counter][2] = z;
    counter++;
}

//Mean is found
float[] mean = new float[3];

for(int i = 0; i < 3; i++)
{
    mean[i] = 0;
    for (int j = 0; j <  nPoints; j++)
        mean[i] = mean[i] + CoordinatesFloat[j][i];
    mean[i] = mean[i]/(nPoints);
}
```

```
System.out.println("meanX = " + mean[0]);
System.out.println("meanY = " + mean[1]);
System.out.println("meanZ = " + mean[2]);



//Calculate covariance matrix
double[][] Cov = new double[3][3];
for(int i = 0; i < 3; i++)
{
    for (int j = 0; j <  3; j++)
    {
        Cov[i][j] = 0;
        for (int k = 0; k <  nPoints; k++)
            Cov[i][j] = Cov[i][j] +
                    (CoordinatesFloat[k][i] - mean[i])
                    * (CoordinatesFloat[k][j] - mean[j]);

        Cov[i][j] = Cov[i][j]/nPoints;

    }

}

printMatrix(Cov, "Cov");

//Answer: cov matrix must be:
//504  360  180
//360  360  0
//180  0  720
*/

//=============

int counter = 0;
for(int i = 0; i < n; i = i+3)
{
    //Reading
    float x =  mesh.getPoints().get(i);
```

```
        float y =  mesh.getPoints().get(i+1);
        float z =  mesh.getPoints().get(i+2);
        CoordinatesFloat[counter][0] = x;
        CoordinatesFloat[counter][1] = y;
        CoordinatesFloat[counter][2] = z;
        counter++;
        //Warning: the zeroth point is artificial,
        //filled with zeros.
        if (i == 0)
            System.out.println("Zeroth point: x = " + x
                    + "   y =   " + y + "   z =  "  + z);
    }

    //Mean is found. It is vector with 3 coordinates
    float[] mean = new float[3];

    for(int i = 0; i < 3; i++)
    {
        mean[i] = 0;
        //Zeroth point does not count
        for (int j = 1; j <  nPoints; j++)
            mean[i] = mean[i] + CoordinatesFloat[j][i];
        mean[i] = mean[i]/(nPoints -1);
    }

    System.out.println("meanX = " + mean[0]);
    System.out.println("meanY = " + mean[1]);
    System.out.println("meanZ = " + mean[2]);
    Point3D meanPoint = new Point3D(mean[0], mean[1], mean[2]);
    bones[counterOfBones].mean = meanPoint;



    //Calculate covariance matrix
    double[][] Cov = new double[3][3];
    for(int i = 0; i < 3; i++)
    {
        for (int j = 0; j <  3; j++)
        {
            Cov[i][j] = 0;
```

```
            for (int k = 1; k <  nPoints; k++)
                Cov[i][j] = Cov[i][j] +
                        (CoordinatesFloat[k][i] - mean[i])
                        * (CoordinatesFloat[k][j] - mean[j]);

            Cov[i][j] = Cov[i][j]/(nPoints-1);

        }

    }

    printMatrix(Cov, "Cov");

    //=========

    //Eigenvalues of Cov matrix are found
    Matrix M = new Matrix(Cov);
    EigenvalueDecomposition E =
         new EigenvalueDecomposition(M);
    double[] d = E.getRealEigenvalues();

    for(int i = 0; i < 3; i++)
        System.out.println("Eigenvalue " + i + " : " + d[i]);
    double[][] EV = E.getV().getArray();
    for(int i = 0; i < 3; i++)
    {
        System.out.println("Eigenvector " + i + " : " );
        for (int j = 0; j <  3; j++)
        {
            System.out.println(EV[i][j]);
        }
        System.out.println();
    }

    //Test: eigenvectors of a real symmetric matrix
    //are orthogonal one to another: dot product must be zero
    System.out.println("Orthogonality test "
            + ": matrix I is expected");
    double[][] Ort = new double[3][3];
    for(int i = 0; i < 3; i++)
```

```
    {

        for (int j = 0; j <  3; j++)
        {

            //sum = sum + EV[i][j]*EV[i+1][j];
            Ort[i][j] = dotProduct(EV[i], EV[j]);

        }
        //System.out.println(sum);

    }

    printMatrix(Ort, "Ort test");

    //Determinant must be 1.
    System.out.println("Orientation test: "
            + "+1 is expected. Det E = "
            + E.getV().det() + " \n");


     //Mesh is centered
    float [] pointsFloat = new float[n];
    for(int i = 0; i < n; i = i+3)
    {
        //Reading
        float x =  mesh.getPoints().get(i);
        float y =  mesh.getPoints().get(i+1);
        float z =  mesh.getPoints().get(i+2);

        //Mesh is centered
        pointsFloat[i]    =  (x - mean[0]);
        pointsFloat[i+1]  =  (y - mean[1]);
        pointsFloat[i+2]  =  (z - mean[2]);
    }
    //Resetting of points of mesh
    mesh.getPoints().setAll(pointsFloat, 0, n);

    axes.axesGroup.setVisible(true);
```

```
//Scale factors are composed
Point3D P =  cylindricalScaleFactors();
//Memorization
bones[counterOfBones].scaleFactors = P;

/*
//Test: slight magnification
scaleFactorW = 1.1f;
scaleFactorH = 1.1f;
scaleFactorL = 1.1f;
*/

//Scale factors as a diagonal matrix
double[][] scaleMatrix = {{scaleFactorW, 0,0},
                          {0, scaleFactorH, 0},
                          {0,0, scaleFactorL}};



//1D array for recording
float[]newPoints = new float[n];
//Mesh is aligned with the new basis
//given by the PCA eigenvetors
for(int i = 0; i < n; i = i+3)
{
     //Reading
    float x =  mesh.getPoints().get(i);
    float y =  mesh.getPoints().get(i+1);
    float z =  mesh.getPoints().get(i+2);
    //Three coordinates define a point
    double[] w = {x,y,z};
    //Coordinates of point w are transformed
    //to the new basis
    w = matrixTimesVector(E.getV()
            .transpose().getArray(),w);
    //PCA scaling is applied
    w = matrixTimesVector(scaleMatrix, w);
    //Coordinates of point w are transformed
    //to the old, natural basis
    w = matrixTimesVector(E.getV().getArray(),w);
```

```
        //Initial center is restored.
        //Point is added to the  1-dimensional array
        newPoints[i ] = (float) ( w[0] + mean[0]);
        newPoints[i + 1] = (float) ( w[1]  + mean[1] );
        newPoints[i + 2] = (float) ( w[2] + mean[2]);
    }
    //Points are reset
    mesh.getPoints().setAll(newPoints, 0, n);

}


//Returns a random float number
//uniformly distributed in the interval
//(1-epsilon, 1 + epsilon)
private float randomFactor(float epsilon)
{
    float s = 2 * epsilon * r.nextFloat();
    s = 1- epsilon + s;
    return s;
}


//Return random values for scale factors.
//Factors can be greater than or less than 1
private Point3D cylindricalScaleFactors()
{
    float epsilon = 0.5f;
    scaleFactorW = randomFactor(epsilon);
    scaleFactorH =  scaleFactorW;
    scaleFactorL = randomFactor(epsilon);
    return new Point3D(scaleFactorW, scaleFactorH,
            scaleFactorL);
}


//Returns th pivot of bone i:
//the middle x-value, the middle y-value and
//the minimal z-value
```

```
private static Point3D pivot(int i)
{
    double x = (coordBoxes[i][0] + coordBoxes[i][3])/2;
    double y = (coordBoxes[i][1] + coordBoxes[i][4])/2;
    double z = coordBoxes[i][5];
    Point3D p = new Point3D(x,y,z);
    return p;
}


private static Point3D initial(int i)
{
    double x = (coordBoxes[i][0] + coordBoxes[i][3])/2;
    double y = (coordBoxes[i][1] + coordBoxes[i][4])/2;
    double z = coordBoxes[i][5];
    Point3D p = new Point3D(x,y,z);
    return p;
}

private static Point3D end(int i)
{
    double x = (coordBoxes[i][0] + coordBoxes[i][3])/2;
    double y = (coordBoxes[i][1] + coordBoxes[i][4])/2;
    double z = coordBoxes[i][2];
    Point3D p = new Point3D(x,y,z);
    return p;
}

private static void    defineGroups()
{
    //Pivots of groups
    Point3D finger0Ph2GroupPivot = new Point3D(0,0,0);
    Point3D finger0Ph21GroupPivot  = new Point3D(0,0,0);
    Point3D finger0Ph210GroupPivot = new Point3D(0,0,0);
    Point3D finger1Ph2GroupPivot = new Point3D(0,0,0);
    Point3D finger1Ph21GroupPivot = new Point3D(0,0,0);
    Point3D finger1Ph210GroupPivot = new Point3D(0,0,0);
    Point3D finger2Ph2GroupPivot = new Point3D(0,0,0);
    Point3D finger2Ph21GroupPivot = new Point3D(0,0,0);
    Point3D finger2Ph210GroupPivot = new Point3D(0,0,0);
```

```
Point3D finger3Ph2GroupPivot = new Point3D(0,0,0);
Point3D finger3Ph21GroupPivot = new Point3D(0,0,0);
Point3D finger3Ph210GroupPivot = new Point3D(0,0,0);
Point3D thumbPh2GroupPivot = new Point3D(0,0,0);
Point3D thumbPh21GroupPivot = new Point3D(0,0,0);
Point3D thumbPh210GroupPivot = new Point3D(0,0,0);
Point3D wristGroupPivot = new Point3D(0,0,0);



//Natural groups are instantiated

for(int i = 0; i < counterOfBones; i++)
{
    String name = names2[i];
    System.out.println("i = " + i + "   " +name);


    //Finger 0 = little and its subgroups
    if (name.contains("little"))
    {
        if (name.contains("Distal"))
        {
            finger0Ph2Group
                .getChildren().add(bonesViewArray[i]);
            finger0Ph2GroupPivot = pivot(i);
            bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "Middle phalanx of"
                    + " left little finger";
            bones[i].next = "";

        }
        if (name.contains("Middle"))
        {
            finger0Ph21Group
                .getChildren()
                .addAll(finger0Ph2Group
                        , bonesViewArray[i]);
```

```
            finger0Ph21GroupPivot = pivot(i);
            bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "Proximal phalanx "
                    + "of left little finger";
            bones[i].next = "Distal phalanx of "
                    + "left little finger";

        }
        if (name.contains("Proximal"))
        {
            finger0Ph210Group
                .getChildren()
                .addAll(finger0Ph21Group
                        , bonesViewArray[i]);
            finger0Ph210GroupPivot = pivot(i);
            bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "Left first "
                    + "metacarpal bone";
            bones[i].next = "Middle phalanx "
                    + "of left little finger";
        }
    }

    //Finger 1 = ring and its subgroups
    else if (name.contains("ring"))
    {
        if (name.contains("Distal"))
        {
            finger1Ph2Group
                .getChildren().add(bonesViewArray[i]);
            finger1Ph2GroupPivot = pivot(i);
            bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "Middle phalanx "
                    + "of left ring finger";
            bones[i].next = "";
        }
        if (name.contains("Middle"))
```

```
        {
            finger1Ph21Group
                .getChildren()
                .addAll(finger1Ph2Group, bonesViewArray[i]);
            finger1Ph21GroupPivot = pivot(i);
            bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "Proximal phalanx "
                    + "of left ring finger";
            bones[i].next = "Distal phalanx "
                    + "of left ring finger";
        }
        if (name.contains("Proximal"))
        {
            finger1Ph210Group
                .getChildren()
                .addAll(finger1Ph21Group, bonesViewArray[i]);
            finger1Ph210GroupPivot = pivot(i);
            bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "Left second "
                    + "metacarpal bone";
            bones[i].next = "Middle phalanx "
                    + "of left ring finger";
        }
    }

    //Finger 2 = middle and its subgroups
    else if (name.contains("middle"))
    {
        if (name.contains("Distal"))
        {
            finger2Ph2Group
            .getChildren().add(bonesViewArray[i]);
            finger2Ph2GroupPivot = pivot(i);
            bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "Middle phalanx "
                    + "of left middle finger";
            bones[i].next = "";
```

```
        }
        if (name.contains("Middle"))
        {
            finger2Ph21Group
                .getChildren()
                .addAll(finger2Ph2Group, bonesViewArray[i]);
            finger2Ph21GroupPivot = pivot(i);
            bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "Proximal phalanx "
                    + "of left middle finger";
            bones[i].next = "Distal phalanx "
                    + "of left middle finger";
        }
        if (name.contains("Proximal"))
        {
            finger2Ph210Group
                .getChildren()
                .addAll(finger2Ph21Group, bonesViewArray[i]);
            finger2Ph210GroupPivot = pivot(i);
            bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "Left third "
                    + "metacarpal bone";
            bones[i].next = "Middle phalanx "
                    + "of left middle finger";
        }
    }

    //Finger 3 = index and its subgroups
    else if (name.contains("index"))
    {
        if (name.contains("Distal"))
        {
            finger3Ph2Group
                .getChildren().add(bonesViewArray[i]);
            finger3Ph2GroupPivot = pivot(i);
            bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "Middle phalanx "
```

```
                    + "of left index finger";
                bones[i].next = "";
            }
            if (name.contains("Middle"))
            {
                finger3Ph21Group
                    .getChildren()
                    .addAll(finger3Ph2Group, bonesViewArray[i]);
                finger3Ph21GroupPivot = pivot(i);
                bones[i].start = initial(i);
                bones[i].end = end(i);
                bones[i].previous = "Proximal phalanx "
                        + "of left index finger";
                bones[i].next = "Distal phalanx "
                        + "of left middle finger";
            }
            if (name.contains("Proximal"))
            {
                finger3Ph210Group
                    .getChildren()
                    .addAll(finger3Ph21Group
                            , bonesViewArray[i]);
                finger3Ph210GroupPivot = pivot(i);
                bones[i].start = initial(i);
                bones[i].end = end(i);
                bones[i].previous = "Left fourth "
                        + "metacarpal bone";
                bones[i].next = "Middle phalanx "
                        + "of left index finger";
            }
        }

        //Thumb and its subgroups
        else if (name.contains("thumb"))
        {
            if (name.contains("Distal"))
            {
                thumbPh2Group
                    .getChildren().add(bonesViewArray[i]);
                thumbPh2GroupPivot = pivot(i);
```

```
                bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "Proximal phalanx "
                    + "of left thumb";
            bones[i].next = "";
        }
        if (name.contains("Proximal"))
        {
            thumbPh21Group
                .getChildren()
                .addAll(thumbPh2Group
                        , bonesViewArray[i]);
            thumbPh21GroupPivot = pivot(i);
            bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "";
            bones[i].next = "Distal phalanx "
                    + "of left thumb";
        }
    }

    //Thumb and its subgroups
    else if (name.contains("first metacarpal"))
    {
        thumbPh210Group
            .getChildren()
            .addAll(thumbPh21Group, bonesViewArray[i]);
        //thumbPh210GroupPivot = pivot(i);
        bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "";
            bones[i].next = "";
    }

    else if (name.contains("second metacarpal"))
    {
        wristGroup.getChildren().add(bonesViewArray[i]);
        bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "";
```

```
                bones[i].next = "Proximal phalanx "
                        + "of left index finger";
        }

        //Thumb and its subgroups
        else if (name.contains("trapezium"))
        {
           thumbPh210GroupPivot = pivot(i);
           wristGroup.getChildren().add(bonesViewArray[i]);
           bones[i].start = initial(i);
                bones[i].end = end(i);
                bones[i].previous = "";
                bones[i].next = "";
        }
        else

        {
            wristGroup.getChildren().add(bonesViewArray[i]);
            if (name.contains("lunate"))
                wristGroupPivot = pivot(i);
            bones[i].start = initial(i);
                bones[i].end = end(i);
                bones[i].previous = "";
                bones[i].next = "";
        }
}


    //Fingers are attached to the palm

        palmGroup.getChildren().addAll(wristGroup
            , finger0Ph210Group
            , finger1Ph210Group
            , finger2Ph210Group
            , finger3Ph210Group
            , thumbPh210Group);

        world3DXform.getChildren().addAll(palmGroup);

    Point3D I0 = new Point3D(0,0,0);
```

```
Point3D I1 = new Point3D(0,0,0);
Point3D I2 = new Point3D(0,0,0);
Point3D I3 = new Point3D(0,0,0);
Point3D M0 = new Point3D(0,0,0);
Point3D M1 = new Point3D(0,0,0);
Point3D M2 = new Point3D(0,0,0);
Point3D M3 = new Point3D(0,0,0);
Point3D E0 = new Point3D(0,0,0);
Point3D E1 = new Point3D(0,0,0);
Point3D E2 = new Point3D(0,0,0);
Point3D E3 = new Point3D(0,0,0);
Point3D S0 = new Point3D(0,0,0);
Point3D S1 = new Point3D(0,0,0);
Point3D S2 = new Point3D(0,0,0);
Point3D S3 = new Point3D(0,0,0);


//Bone groups get reaccommodated
for(int i = 0; i < counterOfBones ; i++)
{
    if (bones[i].name.contains("index"))
    {
        if (bones[i].name.contains("Distal"))
        {
           I0 = bones[i].start;
           M0 = bones[i].mean;
           E0 = bones[i].end;
           S0 = bones[i].scaleFactors;
        }

        if (bones[i].name.contains("Middle"))
        {
           I1 = bones[i].start;
           M1 = bones[i].mean;
           E1 = bones[i].end;
           S1 = bones[i].scaleFactors;
        }

        if (bones[i].name.contains("Proximal"))
        {
```

```
            I2 = bones[i].start;
            M2 = bones[i].mean;
            E2 = bones[i].end;
            S2 = bones[i].scaleFactors;
         }

         if (bones[i].name.contains("Left second "
                + "metacarpal bone"))
         {
            I3 = bones[i].start;
            M3 = bones[i].mean;
            E3 = bones[i].end;
            S2 = bones[i].scaleFactors;
         }
      }
   }
```

```
double tz2 = (1-S0.getZ()) * ((I0.subtract(M0)).getZ())
            + (1-S1.getZ() )* ((M1.subtract(E1)).getZ()) ;
finger3Ph2Group.setTranslateZ(tz2);
finger3Ph2GroupPivot = new Point3D(finger3Ph2GroupPivot.getX()
        ,finger3Ph2GroupPivot.getY(), finger3Ph2GroupPivot.getZ()-

System.out.println(" E0.toString()" + E0.toString());
System.out.println(" M0.toString()" + M0.toString());
System.out.println(" I0.toString()" + I0.toString());
System.out.println(" S0.toString()" + S0.toString());
System.out.println(" E1.toString()" + E1.toString());
System.out.println(" M1.toString()" + M1.toString());
System.out.println(" I1.toString()" + I1.toString());
System.out.println(" S1.toString()" + S1.toString());
System.out.println(" E2.toString()" + E2.toString());
System.out.println(" M2.toString()" + M2.toString());
System.out.println(" I2.toString()" + I2.toString());
System.out.println(" S2.toString()" + S2.toString());

double tz21 = (1-S1.getZ()) * ((I1.subtract(M1)).getZ())
```

```
            + (1-S2.getZ() )* ((M2.subtract(E2)).getZ()) ;
finger3Ph21Group.setTranslateZ(tz21);
finger3Ph21GroupPivot = new Point3D(finger3Ph21GroupPivot.getX()
        ,finger3Ph21GroupPivot.getY(), finger3Ph21GroupPivot.getZ()-tz21);

double tz210 = (1-S2.getZ()) * ((I2.subtract(M2)).getZ())
            + (1-S3.getZ() )* ((M3.subtract(E3)).getZ()) ;
finger3Ph210Group.setTranslateZ(tz210);
finger3Ph210GroupPivot = new Point3D(finger3Ph210GroupPivot.getX()
        ,finger3Ph210GroupPivot.getY(), finger3Ph210GroupPivot.getZ()-tz21


//========OPENING/CL0SING OF THE HAND============

NumberBinding closeHandPropertyHalf
        = closeHandProperty.multiply(0.5);

//The slider that closes the hand extends its
//control to the palm:
//it must rotate around
//the end of the arm that is parallel to the X_AXIS
//This is the pivot of rotation
palmGroup.rwpx.pivotZProperty()
        .setValue(wristGroupPivot.getZ());
palmGroup.rwpx.pivotYProperty()
        .setValue(wristGroupPivot.getY());
//The angle of rotation is controlled by a slider
palmGroup.rwpx.angleProperty()
        .bind(closeHandPropertyHalf);



//Specifications of movement for subgroups of finger 0
finger0Ph2Group.rwpx.pivotZProperty()
        .setValue(finger0Ph2GroupPivot.getZ());
finger0Ph2Group.rwpx.pivotYProperty()
        .setValue(finger0Ph2GroupPivot.getY());
finger0Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);
```

```
finger0Ph21Group.rwpx.pivotZProperty()
        .setValue(finger0Ph21GroupPivot.getZ());
finger0Ph21Group.rwpx.pivotYProperty()
        .setValue(finger0Ph21GroupPivot.getY());
finger0Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger0Ph210Group.rwpx.pivotZProperty()
        .setValue(finger0Ph210GroupPivot.getZ());
finger0Ph210Group.rwpx.pivotYProperty()
        .setValue(finger0Ph210GroupPivot.getY());
finger0Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

//Specifications of movement for subgroups of finger 1
finger1Ph2Group.rwpx.pivotZProperty()
        .setValue(finger1Ph2GroupPivot.getZ());
finger1Ph2Group.rwpx.pivotYProperty()
        .setValue(finger1Ph2GroupPivot.getY());
finger1Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger1Ph21Group.rwpx.pivotZProperty()
        .setValue(finger1Ph21GroupPivot.getZ());
finger1Ph21Group.rwpx.pivotYProperty()
        .setValue(finger1Ph21GroupPivot.getY());
finger1Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger1Ph210Group.rwpx.pivotZProperty()
        .setValue(finger1Ph210GroupPivot.getZ());
finger1Ph210Group.rwpx.pivotYProperty()
        .setValue(finger1Ph210GroupPivot.getY());
finger1Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

//Specifications of movement for subgroups of finger 2
finger2Ph2Group.rwpx.pivotZProperty()
        .setValue(finger2Ph2GroupPivot.getZ());
finger2Ph2Group.rwpx.pivotYProperty()
```

```
            .setValue(finger2Ph2GroupPivot.getY());
    finger2Ph2Group.rwpx.angleProperty()
            .bindBidirectional(closeHandProperty);

    finger2Ph21Group.rwpx.pivotZProperty()
            .setValue(finger2Ph21GroupPivot.getZ());
    finger2Ph21Group.rwpx.pivotYProperty()
            .setValue(finger2Ph21GroupPivot.getY());
    finger2Ph21Group.rwpx.angleProperty()
            .bindBidirectional(closeHandProperty);

    finger2Ph210Group.rwpx.pivotZProperty()
            .setValue(finger2Ph210GroupPivot.getZ());
    finger2Ph210Group.rwpx.pivotYProperty()
            .setValue(finger2Ph210GroupPivot.getY());
    finger2Ph210Group.rwpx.angleProperty()
            .bindBidirectional(closeHandProperty);

    //Specifications of movement for subgroups of finger 3
    finger3Ph2Group.rwpx.pivotZProperty()
            .setValue(finger3Ph2GroupPivot.getZ());
    finger3Ph2Group.rwpx.pivotYProperty()
            .setValue(finger3Ph2GroupPivot.getY());
    finger3Ph2Group.rwpx.angleProperty()
            .bindBidirectional(closeHandProperty);

    finger3Ph21Group.rwpx.pivotZProperty()
            .setValue(finger3Ph21GroupPivot.getZ());
    finger3Ph21Group.rwpx.pivotYProperty()
            .setValue(finger3Ph21GroupPivot.getY());
    finger3Ph21Group.rwpx.angleProperty()
            .bindBidirectional(closeHandProperty);

    finger3Ph210Group.rwpx.pivotZProperty()
            .setValue(finger3Ph210GroupPivot.getZ());
    finger3Ph210Group.rwpx.pivotYProperty()
            .setValue(finger3Ph210GroupPivot.getY());
    finger3Ph210Group.rwpx.angleProperty()
            .bindBidirectional(closeHandProperty);
```

```
//======OPENING/CLOSING OF FINGERS========

//Fingers that are at the border must
//open twice faster as those
//near the center.
NumberBinding openFingersPropertyTwo
        = openFingersProperty.multiply(2);
//Specifications of movement for subgroups of finger 0
finger0Ph210Group.rwpy.pivotXProperty()
        .setValue(finger0Ph210GroupPivot.getX());
finger0Ph210Group.rwpy.pivotYProperty()
        .setValue(finger0Ph210GroupPivot.getY());
finger0Ph210Group.rwpy.pivotZProperty()
        .setValue(finger0Ph210GroupPivot.getZ());
finger0Ph210Group.rwpy.angleProperty()
        .bind(openFingersPropertyTwo);


//Specifications of movement for subgroups of finger 1
finger1Ph210Group.rwpy.pivotXProperty()
        .setValue(finger1Ph210GroupPivot.getX());
finger1Ph210Group.rwpy.pivotYProperty()
        .setValue(finger1Ph210GroupPivot.getY());
finger1Ph210Group.rwpy.pivotZProperty()
        .setValue(finger1Ph210GroupPivot.getZ());

finger1Ph210Group.rwpy.angleProperty()
        .bind(openFingersProperty);

//Finger two remains invariant under opening of fingers

//Opening in the contrary direction of fingers 3 and 4
NumberBinding openFingersPropertyMinus
        = openFingersProperty.multiply(-1);

//Specifications of movement for finger 3
finger3Ph210Group.rwpy.pivotXProperty()
        .setValue(finger3Ph210GroupPivot.getX());
```

```
finger3Ph210Group.rwpy.pivotYProperty()
        .setValue(finger3Ph210GroupPivot.getY());
finger3Ph210Group.rwpy.pivotZProperty()
        .setValue(finger3Ph210GroupPivot.getZ());
finger3Ph210Group.rwpy.angleProperty()
        .bind(openFingersPropertyMinus);

NumberBinding openFingersPropertyMinusTwo
        = openFingersProperty.multiply(-2);



//====== The thumb ========
//Specifications of movement for the thumb


NumberBinding rotateThumbPropertyDown
        = rotateThumbProperty.multiply(-0.4);

NumberBinding rotateThumbPropertyUp
        = rotateThumbProperty.multiply(-2);


NumberBinding closeThumbPropertyMinus
        = closeThumbProperty.multiply(-1);


//Traversal movement
thumbPh210Group.rwpy.pivotXProperty()
        .setValue(thumbPh210GroupPivot.getX());
thumbPh210Group.rwpy.pivotZProperty()
        .setValue(thumbPh210GroupPivot.getZ());
thumbPh210Group.rwpy.angleProperty()
        .bind(closeThumbProperty);

/*
//Rotations of phalanges 2 and 1 to make the grip.
//Rotations around X_AXIS and Y_AXIS at the same time
//Phalanx 2
thumbPh2Group.rwpx.pivotXProperty()
```

```
        .setValue(thumbPh2GroupPivot.getX());
thumbPh2Group.rwpx.pivotYProperty()
        .setValue(thumbPh2GroupPivot.getY());
thumbPh2Group.rwpx.pivotZProperty()
        .setValue(thumbPh2GroupPivot.getZ());
thumbPh2Group.rwpx.angleProperty()
        .bind(rotateThumbPropertyDown);

//Phalanges 2 and 1
thumbPh21Group.rwpx.pivotXProperty()
        .setValue(thumbPh21GroupPivot.getX());
thumbPh21Group.rwpx.pivotYProperty()
        .setValue(thumbPh21GroupPivot.getY());
thumbPh21Group.rwpx.pivotZProperty()
        .setValue(thumbPh21GroupPivot.getZ());
thumbPh21Group.rwpx.angleProperty()
        .bind(rotateThumbPropertyDown);

//Phalanx 2
thumbPh2Group.rwpy.pivotXProperty()
        .setValue(thumbPh2GroupPivot.getX());
thumbPh2Group.rwpy.pivotYProperty()
        .setValue(thumbPh2GroupPivot.getY());
thumbPh2Group.rwpy.pivotZProperty()
        .setValue(thumbPh2GroupPivot.getZ());
thumbPh2Group.rwpy.angleProperty()
        .bind(rotateThumbPropertyDown);
//Phalanges 2 and 1
thumbPh21Group.rwpy.pivotXProperty()
        .setValue(thumbPh21GroupPivot.getX());
thumbPh21Group.rwpy.pivotYProperty()
        .setValue(thumbPh21GroupPivot.getY());
thumbPh21Group.rwpy.pivotZProperty()
        .setValue(thumbPh21GroupPivot.getZ());
thumbPh21Group.rwpy.angleProperty()
        .bind(rotateThumbPropertyDown);
*/

//Rotation of the whole thumb
//to get opposite to other fingers
```

```
    thumbPh210Group.rwpz.pivotXProperty()
            .setValue(thumbPh210GroupPivot.getX());
    thumbPh210Group.rwpz.pivotYProperty()
            .setValue(thumbPh210GroupPivot.getY());
    thumbPh210Group.rwpz.pivotZProperty()
            .setValue(thumbPh210GroupPivot.getZ());
    thumbPh210Group.rwpz.angleProperty()
            .bind(rotateThumbProperty.multiply(-2));



    //===========Animation=========

    double endTime1 = 10000.0; ////1000 = 1 second
    double endTime2 = 2 * endTime1;
    double endTime3 = 3 * endTime1;
    double endTime4 = 4 * endTime1;
    double endTime5 = 5 * endTime1;
    double endTime6 = 6 * endTime1;
    double endTime7 = 7 * endTime1;
    double endTime8 = 8 * endTime1;
    double endTime9 = 9 * endTime1;




    animation.getKeyFrames().addAll
    (

//The hand gets closed
        //closeHandProperty: initial position at initial time
        new KeyFrame(Duration.ZERO,
            //Permanent condition
            new KeyValue(closeThumbProperty, 0)),
        //closeHandProperty: final position at final time
        new KeyFrame(new Duration(endTime1),
            new KeyValue(closeThumbProperty, -50)),

        //The hand gets closed
        //closeHandProperty: initial position at initial time
```

```
new KeyFrame(new Duration(endTime1),
    //Permanent condition
    new KeyValue(closeHandProperty, 0)),
//closeHandProperty: final position at final time
new KeyFrame(new Duration(endTime2),
    new KeyValue(closeHandProperty, -90))
    ,


//The thumb closes the grip
//rotateThumbProperty: initial position at initial time
new KeyFrame(new Duration(endTime2),
    //Permanent condition
    new KeyValue(rotateThumbProperty, 0)),
//rotateThumbProperty: final position at final time
new KeyFrame(new Duration(endTime3),
    new KeyValue(rotateThumbProperty, 37)),

//The thumb gets reopened
//rotateThumbProperty: initial position at initial time
new KeyFrame(new Duration(endTime3),
    //Permanent condition
    new KeyValue(rotateThumbProperty, 37)),
//rotateThumbProperty: final position at final time
new KeyFrame(new Duration(endTime4),
    new KeyValue(rotateThumbProperty, 0)),

//The hand get reopened
//closeHandProperty: initial position at initial time
new KeyFrame(new Duration(endTime4),
    //Permanent condition
    new KeyValue(closeHandProperty, -90)),
//closeHandProperty: final position at final time
new KeyFrame(new Duration(endTime5),
    new KeyValue(closeHandProperty, 0)),

//Fingers get reopened
//openFingersProperty: initial position at initial time
new KeyFrame(new Duration(endTime5),
//Permanent condition
```

```
        new KeyValue(openFingersProperty, 0)),
        //openFingersProperty: final position at final time
        new KeyFrame(new Duration(endTime6),
            new KeyValue(openFingersProperty, 15)),

        //Fingers get closed
        //openFingersProperty: initial position at initial time
        new KeyFrame(new Duration(endTime6),
        //Permanent condition
        new KeyValue(openFingersProperty, 15)),
        //openFingersProperty: final position at final time
        new KeyFrame(new Duration(endTime7),
            new KeyValue(openFingersProperty, 0)),

        //The thumb gets closed
        //closeThumbProperty: initial position at initial time
        new KeyFrame(new Duration(endTime7),
            //Permanent condition
            new KeyValue(closeThumbProperty, -55)),
        //closeThumbProperty: final position at final time
        new KeyFrame(new Duration(endTime8),
            new KeyValue(closeThumbProperty, 0)),

        //The thumb gets reopen
        //closeThumbProperty: initial position at initial time
        new KeyFrame(new Duration(endTime8),
            //Permanent condition
            new KeyValue(closeThumbProperty, 0)),
        //closeThumbProperty: final position at final time
        new KeyFrame(new Duration(endTime9),
            new KeyValue(closeThumbProperty, -25))

    );




animation.play();
```

```
}

private static void printMatrix(double m[][], String title)
{
    System.out.println("This is the matrix " + title);
    for(int i=0;i<3;i++)
        System.out.println(m[i][0] + " " + m[i][1]+ " "
                                              + m[i][2]);
    System.out.println();
}


//The multiplication of a matrix times a vector
  private static  double[] matrixTimesVector(double m[][],
                                             double v[])
  {
    double[]  helpV = new double[3];
    for(int i=0; i<3; i++)
        helpV[i] =0;
    for(int i=0; i<3; i++)
        for(int j=0; j<3; j++)
            helpV[i]  =helpV[i]  + m[i][j]*v[j];
    return helpV;
  }


//Normalized crux product of vectors v and w
  private static  double dotProduct(double[] v,
                                    double[] w)
  {
    return v[0]*w[0] + v[1]*w[1] + v[2]*w[2];

  }


public void play()
{

}
```

```
@Override public void stop()
{

}

static class BoneDescription
{
    String name = "";
    Point3D start ;
    Point3D mean ;
    Point3D end ;
    String previous ;
    String next ;
    Point3D scaleFactors ;

    BoneDescription()
    {
        start = new Point3D(0,0,0);
        mean = new Point3D(0,0,0);
        end = new Point3D(0,0,0);
        previous = "";
        next = "";
        scaleFactors = new Point3D(1f,1f,1f);
    }
}


public static void main(String[] args)
{

    launch(args);
}


}//End of main class N46 Cylindricalmutations2
```

47 ***Exercise.*** *Run and play with the program. Go one step further to accommodate fingers.* Answer

48 ***Exercise.*** *Play enough with N47 Cylindricalmutations3 to realize that its performance is very poor. To reaffirm this idea, run some systematic tests, say, a homogeneous enlargement or a uniform shrinking. Then, make up your mind and compose something useful at least for the phalanges and magnifying mutations.* *Answer*

49 ***Exercise.*** *Previous program N48 Cylindricalmutations4 shows a hand with cylindrical mutations. In spite of a great effort of the Author, achieved performance is very poor. So, play enough with your own guesses to find something better, specially in relation with pivots and axes of rotations. Play enough to agree else disagree with the conclusion of the Author: a very long journey along cycles of mutation and selection should have been left in the fossil record of the hominid hand were evolution the correct explanation of its perfection. Since it is not found, evolution is ruled out immediately.* *Answer*

3.3 Conclusion

The Author lives astonished at the incredible conservation of the pattern of the human hand in spite that all our hands may look a bit different. This observation seems to be incompatible with the idea that mutation is always at work and cannot be stopped anyhow. Now, during ontogenesis, the hand is built in two steps: 1) a tiny hand with its full quality is made and 2) that hand is enlarged to desired size. We have now a problem: how can we model the process of enlargement knowing that it must be somehow encoded and therefore subject to mutation? If we admit that a bone has three natural axes, revealed by PCA (Principal Component Analysis), and that dimensions along them are given to mutate, we discover that mutation rapidly erodes shape. Next, if we consider that a long bone can be modeled as a cylinder so, it admits mutations in two parameters: radius and length. The pattern of the hand seems to be conserved in tiny cylindrical mutations. Nevertheless, when mutation is not that tiny, the pattern is conserved but the function of closing seems strongly distorted. Anyway, accumulation of tiny mutations over time will produce more variability than that that is observed: there are billions of human hands in the world and no matter how different they could be, anyway, they are recognized by everybody as human and not as belonging say, in a chimp or in a monster. Even mutant hands with six fingers fit the human pattern. So, the important question is: what does prevent mutation from eroding form and shape? The natural answer would be to resort to natural selection. Now, by looking at doves at landing, one may suggest that natural selection is important to conserve the morphology of their wings. Nevertheless, the human hand is outside the scope of natural selection: no human mother will reject a son because his or her hands

are not as expected. Therefore, natural selection is not a good candidate to explain why the human hand persists as a pattern along generations. Our purpose for next chapter is to profit on our experience with pleitropy: we saw in Vols X and XI working with trees that applied evolution is not a good tool to solve problems that present a high degree of pleiotropy. So, we plan to study the role of pleiotropy to prevent the eroding effect of mutation over biological shape.

On the other hand, the Author suffered a lot to try to make a good representation of a dynamical mutant hand but achieved quality is very poor. So, the Author feels authorized by his experience to claim that the evolutionary theory is false. In fact, too many crooked hands are expected along the fossil record of the hominid hand. But all hands, including those of lemurs, reptiles and hominids, are a marvelous work of engineering of the most high quality.

Chapter 4

Battling mutation

Trying to defeat mutation

50 ***Introduction and purpose.*** *Simulations that were run in previous chapters allow us to imagine that form and function of the bones of the hand should had been rapidly eroded by mutation because natural selection is inoperative. This originates a puzzle: 9 billions of human hands shall show predicted erosion but -apart from some strange cases- only non harming variability exists. Our purpose for this chapter is to consider some possible answers to this problem. Recombination appears at the first place. Pleitropy is considered next.*

4.1 Is recombination a weapon against mutation?

The natural guess is that recombination is a healing weapon against recessive mutations because when a man with a short leg and a normal woman marry, the offspring is healthy. So, let us make a simulation to make things clearer. We will consider the extreme situation of lethal recessive mutations.

51 ***Recombination vs. recessive lethal mutations.***

Our setup is as follows: a gen can be healthy and encode for a correct function else it has a lethal mutation that makes it lost its meaning and as a consequence the carrier is inviable.

Our purpose is to compose a program to test the postulate that recombination heals recessive lethal mutations but that a price in resources must be paid. In effect, you must guarantee a low mutation rate and therefore you must furnish reproduction with a high quality automatic repairing mechanism of damaged DNA (The

Medical Biochemistry Page [10], 2015) and besides you must increase the population size to diminish the probability of bottlenecks and ensuing extinction of the population because of double recessive mutants. Additionally, double mutants will produce nothing or maybe abortion. To overpass possible extinction, you must waste resources until a viable offspring is found.

Our organisms will be modeled by red circles. A gene is a string that encodes for the measure of its radius. Our genes have thee chars. A viable gene reads 123 or 256 and so on while an inviable one has a mutation that makes it to lost its meaning, say, 1a3, 2h6 and so on. If an embryo has two normal alleles, say 123 and 256, it will develop both genes and produce two superposed red circles that will look like just one, the greatest, i.e., 256. If one allele is normal, say 123, and the other is a nonsensical mutant, say, 2h6, the embryo will look like the red circle encoded by the normal allele, 123. If both genes are nonsensical mutants, say, 1a3 and 2h6, no embryo will be produced.

To implement our program we need to know how to manage inviabilities in Java: it is done through exceptions such as it is shown by the next code that draws a circle and teaches how to manage exceptions produced when a number is expected but a string that encodes for no number is found. The code follows:

52 ***The code for exception handling follows.***

```
/*
    Program N52 Recombination.
    This program shows how to manage exceptions
    of the type NumberFormatException and
    draws a circle.
 */
package Programs;


import java.util.Random;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;
```

```
public class Recombination extends Application
{


    private static final String title
            = "Recombination healing";

    //Individuals are numbered according to an array h x v
    private static final int h = 15;
    private static final int v = 3;

    //Chromosomes
    private static final  String[][] Individual
            = new String[v][h];


    //Number of chars per string
    private static final int numberOfChars = 3;

    // Turn on of the random generator
    private static final Random r  = new Random();


    //Boolean flags for printing

    private static final boolean printDecode = true;



    public void init(Stage primaryStage)
    {
        primaryStage.setTitle(title);
        initialization();
        decode();
    }



    @Override
    public void start(Stage primaryStage)
```

```
    {
        init(primaryStage);
        //Root of tree
        Group root = new Group();
        primaryStage.setResizable(false);
        //Dimensions of the screen
        primaryStage.setScene(new Scene(root, 260, 140));
        //Create a rectangle.
        Circle circle1 =   createCircle();
        //Circle1 is attached to the root
        root.getChildren().addAll(circle1);
        //The scene is shown
        primaryStage.show();


    }

    //A circle ready to be used is built
    private Circle  createCircle()
    {
        //Center at (190,70)
        //Radious = 50
        Circle circle = new Circle(190,70, 50);

//A color is prepared
        Color c = Color.FUCHSIA;
        //The circle is given a color
        circle.setFill(c);

        return circle;
    }


// This method generates random  chars
// that represent digits
private  static  Character  randomDigit()
{
    return  ( char ) ( r.nextInt( 10 ) + 48  );
}
```

```
//This method generates random nonnumeric chars
private  static  Character  randomChar()
{
    return  ( char ) ( r.nextInt( 26 ) + 65  );
}


//Population is initialized.
//All individuals are viable:
//they encode for numbers in range 0..999
private static void initialization( )
{
    System.out.println("Initial population");
    for(int j = 0; j <  v; j++)
    {
        for(int i = 0; i <  h; i++)
        {
            //Initialization in blank.
            Individual[j][i] = "";
            //An individual is assembled char by char
            for(int k = 0; k < numberOfChars; k++)
            {
                // We generate a random numeric char
                Character c =  randomDigit();
                //System.out.println( "char = " + c);
                // Char c is converted into a string s
                String s = "" + c;
                // s is concatenated to Individual[i]
                Individual[j][i] = Individual[j][i] + s;

            }

        }
    }
}

// This method transforms each string into
//a number 3 digits long.
//The incoming string is the genotype.
```

```
//The set with three numbers is an intermediate  phenotype.


//Strings are decoded into int numbers
//if only that is possible. Otherwise,
//an exception is handled
private static void decode()
{
    //Inviable redefinition of individual[0][0]
    Individual[0][0] = "1a1";
    for(int j = 0; j < v; j++)
    {
        for(int i = 0; i < h; i++)
        {
            //Number is retrieved, if that is possible
            try
            {
                int number=Integer.parseInt(Individual[j][i]);

                if (printDecode)
                {
                    System.out.println(i + "   "
                    + j + "   string = "
                        + Individual[j][i]
                        + "    number= " + number);
                }
            }
            //inviable mutants are caught here
            catch(NumberFormatException e1)
            {
               System.out.println(i + "   "
                    + j + "   string = " + Individual[j][i]
                    + "    NO NUMBER " );
               System.out.println("Complaint = "
                       + e1.getMessage());
               System.out.println();
            }
        }

    }
```

```
}


public static void main(String[] args)
{
    launch(args);
}

}//End of application  N52 Recombination
```

53 ***Exercise.*** *Run the previous program and play with the code.*

54 ***Exercise.*** *Compose a program to decide whether or not recombination heals recessive lethal mutations. In affirmative case, inquire about the price to be paid.* *Answer*

55 ***Exercise.*** *In relation with the answer given by the Author to the previous exercise, presented program failed to show how recombination was a healing tool against recessive lethal mutations. Can you see what happens and improve the code?* *Answer*

56 ***Challenge.*** *We have depicted the genetic processes associated with reproduction very simply. Make a fair simulation and explain why nature must be so complex and varied as it actually is.*

57 ***Challenge.*** *We have depicted functions in black or white, i.e., the function is perfect else it is lacked. What about grays?*

Let us pass now to the study of pleiotropy that happens when the change in one item affects the values of others and so one item has various functions.

4.2 More on JavaFX: trees

We plan to investigate the role of pleiotropy as shape keeper against the dissipative role of mutation. On the other hand, we already know that trees and pleitropy are connected. So, let us arm ourselves with some basic knowledge about trees in JavaFX. To that aim, let us follow the course that this theme has been provided by Oracle Documentation (Redko [8], 2013).

58 ***The following code shows us how to build a `JavaFX tree` that consists of a root and some children.***

```
/*
 *  Program N58 TreeViewSample.
    A tree in JavaFX that consists of a root
    and some children.

    Alla Redko, Oracle, 2013
    Using JavaFX UI Controls
    Example 13-1 Creating a Tree View
    https://docs.oracle.com/javafx/2/ui_controls/tree-view.htm
 */

package Programs;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.TreeItem;
import javafx.scene.control.TreeView;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class TreeViewSample extends Application {

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Tree View Sample");

        //Root of tree
        TreeItem<String> rootItem
                        = new TreeItem<> ("Inbox");
        //TreeView can be contracted else expanded
        rootItem.setExpanded(true);
        //Loop to populate the tree with children of the root
        for (int i = 1; i < 6; i++) {
```

```
            //Declaration of children
            TreeItem<String> item
                    = new TreeItem<> ("Message" + i);
            //Children are booked
            rootItem.getChildren().add(item);
        }
        //JavaFX tree rooted on rootItem
        TreeView<String> tree = new TreeView<> (rootItem);
        StackPane root = new StackPane();
        //Tree is booked
        root.getChildren().add(tree);
        primaryStage.setScene(new Scene(root, 300, 250));
        primaryStage.show();
    }
}//End of main class N58 TreeViewSample
```

59 ***Exercise.*** *Run the previous program and play with the code.*

60 ***The following code shows us how to build a JavaFX tree that consists of a root, children and grandchildren.***

```
/*
 *  Program N60 TreeViewSample2
    A tree in JavaFX that consists of a root,
    children and grandchildren.

    Slight modification of:

    Alla Redko, Oracle, 2013
    Using JavaFX UI Controls
    Example 13-2 Creating a Model of
    the Human Resources Tree View.
    The company has departments (children)
    and the departments have employees (grandchildren).
    The application  visualizes human resources data
    of a given company, and enables users to
    add new employees.
    https://docs.oracle.com/javafx/2/ui_controls/tree-view.htm
 */
package Programs;
```

```
import java.util.Arrays;
import java.util.List;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.TreeItem;
import javafx.scene.control.TreeView;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

import javafx.beans.property.SimpleStringProperty;
import javafx.scene.layout.VBox;

public class TreeViewSample2 extends Application {

//My compaby is the root
TreeItem<String> rootNode =
    new TreeItem<>("MyCompany Human Resources");
//The root has departments which have employees
//An employee is defined by a name and his or her department
List<Employee> employees = Arrays.<Employee>asList(
        new Employee("Ethan Williams", "Sales Department"),
        new Employee("Emma Jones", "Sales Department"),
        new Employee("Michael Brown", "Sales Department"),
        new Employee("Anna Black", "Sales Department"),
        new Employee("Rodger York", "Sales Department"),
        new Employee("Susan Collins", "Sales Department"),
        new Employee("Mike Graham", "IT Support"),
        new Employee("Judy Mayer", "IT Support"),
        new Employee("Gregory Smith", "IT Support"),
        new Employee("Jacob Smith", "Accounts Department"),
        new Employee("Isabella Johnson", "Accounts Department"));



public static void main(String[] args) {
    launch(args);
}
```

```
@Override
public void start(Stage stage) {
    rootNode.setExpanded(true);
    for (Employee employee : employees) {
        TreeItem<String> empLeaf
                = new TreeItem<>(employee.getName());
        boolean found = false;
        for (TreeItem<String> depNode
                : rootNode.getChildren()) {
            if (depNode.getValue().contentEquals(employee
                                        .getDepartment())){
                depNode.getChildren().add(empLeaf);
                found = true;
                break;
            }
        }
        if (!found) {
            TreeItem<String> depNode = new TreeItem<>(
                employee.getDepartment()
            );
            rootNode.getChildren().add(depNode);
            depNode.getChildren().add(empLeaf);
        }
    }

    //An extra employee is created and added
    Employee newEmployee
            = new Employee("NEW EMPLOYEE", "IT Support");
    //The instance of class employee gets an instance of TreeItem
    TreeItem<String> empLeaf
            = new TreeItem<>(newEmployee.getName());
    boolean found = false;
    //If newEmployee belongs in a matriculated
    //departement, it is added as grandchildren
        for (TreeItem<String> depNode
                : rootNode.getChildren())
        {
            if (depNode.getValue()
                .contentEquals(newEmployee.getDepartment()))
            {
```

```
                depNode.getChildren().add(empLeaf);
                found = true;
                break;
            }
        }
        //If no departement exists, create one for him
        if (!found)
        {
            TreeItem<String> depNode = new TreeItem<>(
                newEmployee.getDepartment()
            );
            rootNode.getChildren().add(depNode);
            depNode.getChildren().add(empLeaf);
        }


    stage.setTitle("Tree View Sample: expand the tree!");
    //Box to accommodate the tree
    VBox box = new VBox();
    final Scene scene = new Scene(box, 400, 300);
    scene.setFill(Color.LIGHTGRAY);
    //TreeView based at the root
    TreeView<String> treeView = new TreeView<>(rootNode);
    //TreeView is booked
    box.getChildren().add(treeView);
    stage.setScene(scene);
    stage.show();
}

//Get-set class Employee: name + department
public static class Employee {

    private final SimpleStringProperty name;
    private final SimpleStringProperty department;

    private Employee(String name, String department) {
        this.name = new SimpleStringProperty(name);
        this.department
                = new SimpleStringProperty(department);
    }
```

```
    public String getName() {
        return name.get();
    }

    public void setName(String fName) {
        name.set(fName);
    }

    public String getDepartment() {
        return department.get();
    }

    public void setDepartment(String fName) {
        department.set(fName);
    }
}
}//End of Program N60 TreeViewSample2
```

61 ***Exercise.*** *Run the previous program and play with the code.*

62 ***The following code shows us how to build a JavaFX tree that consists of a root, children and grandchildren. It is possible to change the content of a node directly over the interface.***

```
/*
 *  Program N62 TreeViewSample3
    A modifiable tree in JavaFX that consists of a root,
    children and grandchildren.
    The interface is editable.

    Alla Redko, Oracle, 2013
    Using JavaFX UI Controls
    Example 13-3 Implementing a Cell Factory.
    The company has departments (children)
    and the departments have employees (grandchildfren).
    The application  visualizes human resources data
    of a given company, and enables users
    to add new employees and to modify
    employee details directly over the graphic interface.
```

```
    https://docs.oracle.com/javafx/2/ui_controls/tree-view.htm
 */

package Programs;


import java.util.Arrays;
import java.util.List;
import javafx.application.Application;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.TextField;
import javafx.scene.control.TreeCell;
import javafx.scene.control.TreeItem;
import javafx.scene.control.TreeView;
import javafx.scene.image.ImageView;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

import javafx.beans.property.SimpleStringProperty;
import javafx.scene.layout.VBox;

public class TreeViewSample3 extends Application {

private final Node rootIcon =
    new ImageView();

List<Employee> employees = Arrays.<Employee>asList(
        new Employee("Ethan Williams", "Sales Department"),
        new Employee("Emma Jones", "Sales Department"),
        new Employee("Michael Brown", "Sales Department"),
        new Employee("Anna Black", "Sales Department"),
        new Employee("Rodger York", "Sales Department"),
        new Employee("Susan Collins", "Sales Department"),
        new Employee("Mike Graham", "IT Support"),
        new Employee("Judy Mayer", "IT Support"),
        new Employee("Gregory Smith", "IT Support"),
        new Employee("Jacob Smith", "Accounts Department"),
```

```
        new Employee("Isabella Johnson", "Accounts Department"));
TreeItem<String> rootNode =
    new TreeItem<>("MyCompany Human Resources", rootIcon);

public static void main(String[] args) {
    Application.launch(args);
}

@Override
public void start(Stage stage) {
    rootNode.setExpanded(true);
    for (Employee employee : employees) {
        TreeItem<String> empLeaf
                = new TreeItem<>(employee.getName());
        boolean found = false;
        for (TreeItem<String> depNode
                : rootNode.getChildren()) {
            if (depNode.getValue()
                .contentEquals(employee.getDepartment())){
                depNode.getChildren().add(empLeaf);
                found = true;
                break;
            }
        }
        if (!found) {
            TreeItem<String> depNode = new TreeItem<>(
                employee.getDepartment(),
                new ImageView()
            );
            rootNode.getChildren().add(depNode);
            depNode.getChildren().add(empLeaf);
        }
    }

    stage.setTitle("Tree "
            + ": click twice over a field to edit it");
    VBox box = new VBox();
    final Scene scene = new Scene(box, 400, 300);
    scene.setFill(Color.LIGHTGRAY);
```

```
    TreeView<String> treeView = new TreeView<>(rootNode);
    treeView.setEditable(true);
    treeView.setCellFactory((TreeView<String> p)
            -> new TextFieldTreeCellImpl());

    box.getChildren().add(treeView);
    stage.setScene(scene);
    stage.show();
}

//Cell factory implementation:
//Fields can be edited from the GUI
private final class TextFieldTreeCellImpl
                    extends TreeCell<String> {

    private TextField textField;

    public TextFieldTreeCellImpl() {
    }

    @Override
    public void startEdit() {
        super.startEdit();

        if (textField == null) {
            createTextField();
        }
        setText(null);
        setGraphic(textField);
        textField.selectAll();
    }

    @Override
    public void cancelEdit() {
        super.cancelEdit();
        setText( getItem() );
        setGraphic(getTreeItem().getGraphic());
    }

    @Override
```

```
    public void updateItem(String item, boolean empty) {
        super.updateItem(item, empty);

        if (empty) {
            setText(null);
            setGraphic(null);
        } else {
            if (isEditing()) {
                if (textField != null) {
                    textField.setText(getString());
                }
                setText(null);
                setGraphic(textField);
            } else {
                setText(getString());
                setGraphic(getTreeItem().getGraphic());
            }
        }
    }


    private void createTextField() {
        textField = new TextField(getString());
        //Listener of double clicking
        textField.setOnKeyReleased((KeyEvent t) -> {
            if (t.getCode() == KeyCode.ENTER) {
                commitEdit(textField.getText());
            } else if (t.getCode() == KeyCode.ESCAPE) {
                cancelEdit();
            }
        });
    }

    private String getString() {
        return getItem() == null ? "" : getItem();
    }
}


//Get-set class Employee
```

```
public static class Employee {

    private final SimpleStringProperty name;
    private final SimpleStringProperty department;

    private Employee(String name, String department) {
        this.name = new SimpleStringProperty(name);
        this.department
                = new SimpleStringProperty(department);
    }

    public String getName() {
        return name.get();
    }

    public void setName(String fName) {
        name.set(fName);
    }

    public String getDepartment() {
        return department.get();
    }

    public void setDepartment(String fName) {
        department.set(fName);
    }
}
}//Program N62 TreeViewSample3
```

63 ***Exercise.*** *Run the previous program and play with the code.*

64 ***The following code shows us how to build a JavaFX tree that consists of a root, children and grandchildren. It is possible to change the content of a node or to add a new one directly over the interface.***

```
/*
 *  Program N64 TreeViewSample4
    A modifiable tree in JavaFX that consists of a root,
    children and grandchildren.
```

```
    The interface is editable.


    Alla Redko, Oracle, 2013
    Using JavaFX UI Controls
    Example 13-3 Implementing a Cell Factory.
    The company has departments (children)
    and the departments have employees (grandchildfren).
    The application  visualizes human resources data
    of a given company, and enables users
    to add new employees and  modify
    employee details directly over the graphic interface.
    https://docs.oracle.com/javafx/2/ui_controls/tree-view.htm

    Updated by JR, the Evoljava Community, 2015.
 */

package Programs;

import java.util.Arrays;
import java.util.List;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.TextField;
import javafx.scene.control.TreeCell;
import javafx.scene.control.TreeItem;
import javafx.scene.control.TreeView;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

import javafx.beans.property.SimpleStringProperty;
import javafx.event.ActionEvent;
import javafx.scene.control.ContextMenu;
import javafx.scene.control.MenuItem;
import javafx.scene.layout.VBox;

public class TreeViewSample4 extends Application {
```

```
List<Employee> employees = Arrays.<Employee>asList(
        new Employee("Ethan Williams", "Sales Department"),
        new Employee("Emma Jones", "Sales Department"),
        new Employee("Michael Brown", "Sales Department"),
        new Employee("Anna Black", "Sales Department"),
        new Employee("Rodger York", "Sales Department"),
        new Employee("Susan Collins", "Sales Department"),
        new Employee("Mike Graham", "IT Support"),
        new Employee("Judy Mayer", "IT Support"),
        new Employee("Gregory Smith", "IT Support"),
        new Employee("Jacob Smith", "Accounts Department"),
        new Employee("Isabella Johnson", "Accounts Department"));
TreeItem<String> rootNode =
    new TreeItem<>("MyCompany Human Resources");

public static void main(String[] args) {
    Application.launch(args);
}

@Override
public void start(Stage stage) {
    rootNode.setExpanded(true);
    for (Employee employee : employees) {
        TreeItem<String> empLeaf
                = new TreeItem<>(employee.getName());
        boolean found = false;
        for (TreeItem<String> depNode
                : rootNode.getChildren()) {
            if (depNode.getValue()
                .contentEquals(employee.getDepartment())){
                depNode.getChildren().add(empLeaf);
                found = true;
                break;
            }
        }
        if (!found) {
            TreeItem<String> depNode
                = new TreeItem<>(employee.getDepartment());
            rootNode.getChildren().add(depNode);
```

```
            depNode.getChildren().add(empLeaf);
        }
    }

    stage.setTitle("Tree: use double click and right click");
    VBox box = new VBox();
    final Scene scene = new Scene(box, 400, 300);
    scene.setFill(Color.LIGHTGRAY);

    TreeView<String> treeView = new TreeView<>(rootNode);
    treeView.setEditable(true);
    treeView.setCellFactory((TreeView<String> p)
                -> new TextFieldTreeCellImpl());

    box.getChildren().add(treeView);
    stage.setScene(scene);
    stage.show();
}


private final class TextFieldTreeCellImpl
                  extends TreeCell<String> {

    private TextField textField;
    private final ContextMenu addMenu = new ContextMenu();

    public TextFieldTreeCellImpl() {


        MenuItem addMenuItem = new MenuItem("Add Employee");

        addMenu.getItems().add(addMenuItem);
        //Listener of contextMenu
        addMenuItem.setOnAction((ActionEvent t) -> {
            TreeItem<String> newEmployee =
                    new TreeItem<>("Edit New Employee");
            getTreeItem()
                    .getChildren().add(newEmployee);
        });
    }
```

```
@Override
public void startEdit() {
    super.startEdit();

    if (textField == null) {
        createTextField();
    }
    setText(null);
    setGraphic(textField);
    textField.selectAll();
}

@Override
public void cancelEdit() {
    super.cancelEdit();

    setText(getItem());
    setGraphic(getTreeItem().getGraphic());
}

@Override
public void updateItem(String item, boolean empty) {
    super.updateItem(item, empty);

    if (empty) {
        setText(null);
        setGraphic(null);
    } else {
        if (isEditing()) {
            if (textField != null) {
                textField.setText(getString());
            }
            setText(null);
            setGraphic(textField);
        } else {
            setText(getString());
            setGraphic(getTreeItem().getGraphic());
```

```
                if (
                    !getTreeItem().isLeaf()&&getTreeItem()
                            .getParent()!= null
                ){
                    setContextMenu(addMenu);
                }
            }
        }
    }

    private void createTextField() {
        textField = new TextField(getString());
        textField.setOnKeyReleased((KeyEvent t) -> {
            if (t.getCode() == KeyCode.ENTER) {
                commitEdit(textField.getText());
            } else if (t.getCode() == KeyCode.ESCAPE) {
                cancelEdit();
            }
        });

    }

    private String getString() {
        return getItem() == null ? "" : getItem();
    }
}

public static class Employee {

    private final SimpleStringProperty name;
    private final SimpleStringProperty department;

    private Employee(String name, String department) {
        this.name = new SimpleStringProperty(name);
        this.department
                = new SimpleStringProperty(department);
    }

    public String getName() {
        return name.get();
```

```
    }

    public void setName(String fName) {
        name.set(fName);
    }

    public String getDepartment() {
        return department.get();
    }

    public void setDepartment(String fName) {
        department.set(fName);
    }
}
}//End of program N64 TreeViewSample4
```

65 ***Exercise.*** *Run the previous program and play with the code.*

66 ***The following code shows us how to build a JavaFX tree with a CheckBox-TreeCell editor that consists of a root, and its children.***

```
/*
 *  Program N66 TreeViewSample5
    A tree in JavaFX with CheckBoxTreeCell editor
    that consists of a root,  and its children.

    Alla Redko, Oracle, 2013
    Using JavaFX UI Controls
    Example 13-5 Using the CheckBoxTreeCell Class
    Starting JavaFX SDK 2.2, you can use the following
    tree cell editors available in the API:
    CheckBoxTreeCell, ChoiceBoxTreeCell,
    ComboBoxTreeCell, TextFieldTreeCell.
    There classes extend the TreeCell implementation
    to render a particular control inside the cell.
https://docs.oracle.com/javafx/2/ui_controls/tree-view.htm

*/
```

```
package Programs;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.control.cell.CheckBoxTreeCell;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class TreeViewSample5 extends Application {

public static void main(String[] args) {
    launch(args);
}

@Override
public void start(Stage primaryStage) {
    primaryStage.setTitle("Tree View Sample");

    CheckBoxTreeItem<String> rootItem =
        new CheckBoxTreeItem<>("View Source Files");
    rootItem.setExpanded(true);

    final TreeView<String> tree = new TreeView<>(rootItem);
    tree.setEditable(true);

    tree.setCellFactory(CheckBoxTreeCell
                        .<String>forTreeView());
    for (int i = 0; i < 8; i++) {
        final CheckBoxTreeItem<String>
        checkBoxTreeItem =  new CheckBoxTreeItem<>
                                ("Sample" + (i+1));
        rootItem.getChildren()
                .add(checkBoxTreeItem);
    }

    tree.setRoot(rootItem);
    tree.setShowRoot(true);
    //Try this out:
```

```
    rootItem.setIndependent(true);

    StackPane root = new StackPane();
    root.getChildren().add(tree);
    primaryStage.setScene(new Scene(root, 300, 250));
    primaryStage.show();
}
}//End of Program N66 TreeViewSample5
```

67 ***Exercise.*** *Run the previous program and play with the code.*

Let us prepare to work with lengthy programs. To that aim we use a Task.

68 ***The code for a heavy duty tree follows:***

```
/*
 *  Program N68 TreeViewSample6.
    A JavaFX heavy duty tree.

    jewelsea at Stackoverflow, 2013

http://stackoverflow.com/questions/16125311/concurrentmodificationexce
e-in-the-background/16138621#16138621

    Updated by JR of the EvolJava Community 2015

*/

package Programs;

import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.concurrent.Task;
import javafx.concurrent.WorkerStateEvent;
import javafx.scene.Scene;
import javafx.scene.control.TreeItem;
import javafx.scene.control.TreeView;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
```

```
import java.security.SecureRandom;
import java.util.Random;
import static javafx.application.Application.launch;



public class TreeViewSample6 extends Application
{

public static void main(String[] args)
{
    launch(args);
}

@Override
public void start(Stage stage) throws Exception
{
    TreeView<String> treeView
            = new TreeView<>(createNode("root"));

    HBox hBox = new HBox();
    hBox.getChildren().addAll(treeView);
    Scene scene = new Scene(hBox);
    stage.setScene(scene);
    stage.show();
}



Random r = new SecureRandom();



private TreeItem<String> createNode(final String b)
{
    return new TreeItem<String>(b)
    {
        private boolean isLeaf;
        private boolean isFirstTimeChildren = true;
```

```
private boolean isFirstTimeLeaf = true;



@Override
public ObservableList<TreeItem<String>> getChildren()
{
    if (isFirstTimeChildren)
    {
      isFirstTimeChildren = false;
      buildChildren(super.getChildren());
    }
    return super.getChildren();
}



@Override
public boolean isLeaf()
{
    if (isFirstTimeLeaf)
    {
      isFirstTimeLeaf = false;
      //A leaf is terminal node.
      //Recursive construction of tree continues
      //until a leaf is found.
      //A node is defined as leaf
      //with probability 1/8 = (1/2)^3
      isLeaf = r.nextBoolean()
              && r.nextBoolean() && r.nextBoolean();
    }
    return isLeaf;
}



private void buildChildren(
  final ObservableList<TreeItem<String>> children)
{
```

```
final ObservableList<TreeItem<String>> taskChildren
        = FXCollections.observableArrayList();


if (!this.isLeaf())
{
    Task<Integer> task = new Task<Integer>()
    {
        @Override
        protected Integer call() throws Exception
        {
            Integer i;
            int max = r.nextInt(500);
            for (i = 0; i <= max; i++)
            {

                //Original: runs but generates warnings
                /*
                taskChildren.addAll(new TreeItem[]
                  {createNode("#" + r.nextInt())});
                */
                //Modification that generates no warning:
                //String is specified inside TreeItem
                TreeItem<String> c
                        = createNode("#" + r.nextInt())  ;
                taskChildren.add(c);

            }
          return i;
      }
  };


  task.setOnSucceeded(
          (WorkerStateEvent workerStateEvent) ->
  {
      children.setAll(taskChildren);
      System.out.println("State "
              + "of WorkerStateEvent = "
              + workerStateEvent.toString());
```

```
                });
                new Thread(task).start();
            }
        }
    };
}

}//End of program N68 TreeViewSample6
```

69 ***Exercise.*** *Run the previous program and play with the code.*

70 ***Exercise.*** *A* ***binary tree*** *is one in which each node can has at most two children. Develop the code for a random binary tree.* Answer

4.3 Homemade binary trees

Binary trees are simple and could treated simply. In this regard, let us recall the main ideas from Vol X.

71 ***Our encoding for binary trees***

Let us pursue simplicity to its maximal expression. In that regard, we invite the reader to pay attention to the next form of naming the nodes of a binary tree:

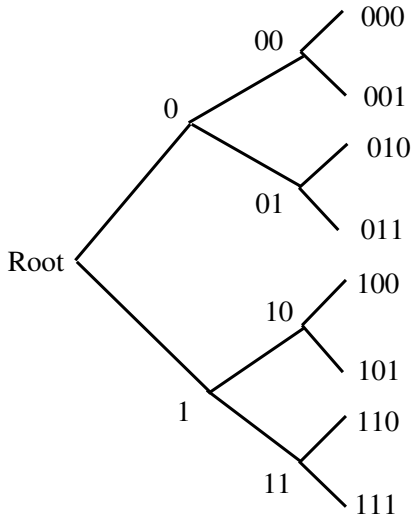


Figure 4.0. Binary numeration of the nodes of a binary tree.

We see that names correspond with binary strings in their full expression, say, 0 is different than 00 and 01 different than 1. From now on, this procedure for numbering the nodes of a binary tree defines their addresses in the tree.

Our notation is transparent: Where are the children of 10? They are in locations 100 and 101. Where is the father of 110? It is in location 11. Where are the brothers of 001? 001 has only one brother that is in position 000.

In order to increase the readability of the code and because 0 and 1 will not be treated as numbers but as chars, we might use - and + instead of 0 and 1.

72 ***The following program reminds us how to draw our trees in Swing. Taken from Vol X.***

```
//Program J76 BinaryBushDrawing6
/*
* The representation of binary trees must be
* as simple as the trees themselves.
* This program deals with the following notation:
* each node has an address given by
* the binary full expansion of its position in the tree,
* beginning from 0 and 1, the sons of the root,
* and ending with 1....1,
* the second son of the last father.
*
* The program receives a complete tree
* in this binary notation
* and outputs a drawing of the tree using
* our own methods.
* The input must be ordered.
*
* The up-down polarity of the drawing is user friendly:
* y -> y - k causes a vertical reflection plus
* a translation.
*
* The present code
* accepts an input with
* incomplete content. So, the complete tree
* is drawn but some nodes may lack
```

```
* corresponding strings.
*
* We rely on a natural interpretation of addresses:
* Address in the tree 0110 means:
* go  up, down, down and up, 4  generations in all.
*
*/

package Programs;


import javax.swing.*;


import java.awt.*;


public class BinaryBushDrawing6 extends JPanel
{
 private static final long serialVersionUID = 1L;
 private JPanel drawingPane6;


 //Vertical  size of the panel
 private static int verticalSize = 800;
 private static int horizontalSize = 800;
 //Scale must be a power of 2
 private static int scale = 16;

 //Relative length of branches of the first U
 private static int horizontalLength = 2;
 private static int verticalLength = 8;

//This is data for a tree in our binary notation:
//Even entries are the addresses in the tree of the nodes,
//odd ones are the names or contents.
private static String[] Input =
{
"0", "a",
"1", "b",
```

```
"00", "aa",
"01", "ab",
//"10", "ba",
"11", "bb",
//"000", "aaa",
//"001", "aab",
"010", "aba",
//"011", "abb",
//"100", "baa",
"101", "bab",
"110", "bba",
"111", "bbb"
};

 private static int InputLength = Input.length;
 //The number of generation is
 //the length of the last address
 private static int inputhLength =
              Input[InputLength-1].length();

 private static int numberOfGenerations = 5;
 //The terminal  leaves define the complete tree
 private static int numberOfLeaves =
   (int) Math.pow(2, numberOfGenerations);
 //Binary paths to leaves
 private static String[] pathsToLeaves =
                    new String[numberOfLeaves];

 //Constructor
 public BinaryBushDrawing6()
 {
  super(new BorderLayout());
  //Set up the drawing area.
  drawingPane6 = new DrawingPane6();
  //Put the drawing area in a scroll pane.
  JScrollPane scroller = new JScrollPane(drawingPane6);
  //scroller.setPreferredSize(new Dimension(700,800));
  add(scroller, BorderLayout.CENTER);
 }
```

```
//The tree is drawn
public class DrawingPane6 extends JPanel
 {
  private static final long serialVersionUID = 1L;


  //Paths to leaves are constructed as binary strings
  private void createPathsToLeaves(int numberOfGenerations)
  {
String s;
for(Integer i = 0; i < numberOfLeaves; i++ )
{
 s  = Integer.toBinaryString(i);
 int l = s.length();
 if (l < numberOfGenerations)
 for(int k = l; k < numberOfGenerations; k++)
 s = "0" + s;
 pathsToLeaves[i] = s;
 System.out.println(pathsToLeaves[i]);
}
  }



 //Drawing machinery
 protected void paintComponent(Graphics g)
  {
    super.paintComponent(g);
    createPathsToLeaves(numberOfGenerations);
    g.setColor(Color.red);
    //Symbolic initialization
    int centerX = 1;
    int centerY = 1;


    //Draw stem
    int x00 =    scale*centerX;
 int y00 =   scale* centerY + 400;
```

```
int x11 =  x00 + scale*horizontalLength;
int y11 =  y00;
  g.drawLine(x00, verticalSize - y00,
      x11,  verticalSize - y11);

  //Horizontal and basic vertical strokes
  int H = scale * horizontalLength;
int V = scale * verticalLength;


  String path = "";
  char direction;

//The tree is drawn:
//A path from the origin of the tree to
//each leaf is drawn
   for(int i = 0; i < numberOfLeaves; i++)
   {
    //Addresses will be used as paths to  be drawn
    path = pathsToLeaves[i];

    //Reinitialization
    Dimension center = new Dimension(x11,y11);

    V = scale * verticalLength;
     //Path is interpreted
       for(int j = 0; j < path.length(); j++)
        {
      direction = path.charAt(j);
      if (direction == '0') //go up
       {
      int x0 =  center.width;
      int y0 =  center.height;
      int x1 =  x0;
      int y1 =  y0 + V;
      //Draw vertical stroke
      g.drawLine(x0, verticalSize -y0, x1,
      verticalSize- y1);
      //Draw horizontal stroke
      g.drawLine(x1, verticalSize - y1,
```

```
        x1 + H,  verticalSize - y1);
      center.height = center.height + V;
       }
      else // if direction == '1'  =  go down
       {
      int x0 =  center.width;
      int y0 =  center.height;
      int x1 =  x0;

      int y1 =  y0 - V;
      //Draw vertical stroke
      g.drawLine(x0, verticalSize - y0,
         x1,  verticalSize - y1);
      //Draw horizontal stroke
      g.drawLine(x1, verticalSize - y1,
         x1 + H,  verticalSize - y1);
      center.height = center.height - V;
       } //End of if + else

      center.width = center.width + H ;

      V = V/2;
        }//End of for j


  }  //End of for: numberOfLeaves


//Strings are drawn:


for(int i = 0; i < InputLength; i = i+2)
 {
  //Addresses will be used as paths to the places
  //where strings must be drawn
  path = Input[i];
  String s = Input[i+1];
  //Reinitialization
  Dimension center = new Dimension(x00,y00);
  Dimension target = center;
```

```
      V = scale * verticalLength;
      //Path is interpreted
      for(int j = 0; j < path.length(); j++)
       {
     direction = path.charAt(j);
     if (direction == '0') //go up
      target.height = target.height + V;
     else // if direction == '1'  =  go down
     target.height = target.height - V;

     target.width = target.width + H ;
     V = V/2;
       }//End of for j
      //String is drawn
     g.drawString(s, target.width + 1,
                    verticalSize - target.height-1);
     }
   }
}//End of   paintComponent



 // Create the GUI and show it.
 private static void createAndShowGUI()
  {
   //Main window.
   JFrame frame = new JFrame("BinaryBushDrawing6");
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   //Content pane where to draw
   JComponent newContentPane6 = new BinaryBushDrawing6();
   frame.setContentPane(newContentPane6);
   frame.setSize(horizontalSize,verticalSize);
   //Display the window.
   frame.setVisible(true);
 }


public static void main(String[] args)
 {
  //Thread technology
```

```
  javax.swing.SwingUtilities.invokeLater(new Runnable()
   {
    public void run()
     {
      createAndShowGUI();

     }
   });
  }
} //End of main class J76 BinaryBushDrawing6
```

73 ***Exercise.*** *Run the program and play with the code.*

74 ***Exercise.*** *Adapt previous program to JavaFX. Answer*

75 ***Exercise.*** *Using our notation, generate and draw a random binary tree. Answer*

76 ***Exercise.*** *Our previous trees are suitable for those phylogenies in which we present the ancestor of species that live now. But besides we need* ***bushes****, which are trees that include the depiction of extinction. Using our notation, compose a code to create and draw bushes. Answer*

77 ***Full bush.*** *Let us learn how to generate a bush in which we list all leaves together with all internal nodes: we refer to this type of tree as* ***fullbush****. The code follows:*

```
//Program N77 BinaryBushJavaFX4
/*
 * A random binary fullbush
 * (with all leaves and all internal nodes).
 * Nodes in the tree have an address. Say,
 * Address   0110 means:
 * go  up, down, down and up, 4  generations at all.
 * Each node has a content that is a string.
 * Information is kept in an array, an address followed
 * by the corresponding content.

 * Example:
```

```
        private static final String[] TreeInfo
            = {
                "0", "a",
                "1", "b",
                "00", "aa",
                "01", "ab",
                //"10", "ba",
                "11", "bb",
                //"000", "aaa",
                //"001", "aab",
                "010", "aba",
                //"011", "abb",
                //"100", "baa",
                "101", "bab",
                "110", "bba",
                "111", "bbb"
            };
 * To generate addresses we generate
 * random integers that are translated to binary strings.

*/
package Programs;


import java.util.Random;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point2D;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.ScrollPane;
import javafx.scene.layout.Priority;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Line;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class BinaryBushJavaFX4 extends Application
{
```

```
    //This is data for a tree in our binary notation:
    //Even entries are the addresses in the tree of the nodes,
    //odd ones are the names or contents.
    private static   String[] TreeInfo;

    private static  int treeInfoLength;

    private static final int numberOfGenerations = 5;
    //The terminal  leaves define the complete tree
    private static final int numberOfLeaves
            = (int) Math.pow(2, numberOfGenerations);
    //Binary paths to leaves
    private static final String[] pathsToLeaves
            = new String[numberOfLeaves];

    //Relative length of branches of the first U
    private static final int horizontalLength = 3;
    private static final int verticalLength = 8;

    //The scene
    private static final Group root = new Group();

    private static final ScrollPane pane = new ScrollPane();
    //Vertical  size of the panel
    private static final int verticalSize =
            (int) (5 * Math.pow(2, numberOfGenerations));
    private static final int horizontalSize = 800;
    //Scale must be a power of 2
    private static final int scale = 16;

    final VBox vb = new VBox();

    private static final boolean print = false;
    private static final Random r = new Random();



//A random binary tree is created:
//addresses and contents
```

```
private static void createTree()
{

    int n = numberOfLeaves;
    TreeInfo = new String[4*n];
    treeInfoLength = TreeInfo.length;

    for(int i = 0; i < 2*n; i= i+2)
    {
       TreeInfo[i] = Integer.toBinaryString(i/2);
       TreeInfo[i+1] = TreeInfo[i];
       TreeInfo[i+1] = TreeInfo[i+1].replace('0', '+');
       TreeInfo[i+1] = TreeInfo[i+1].replace('1', '-');

       System.out.println(i/2 + "   "
              + TreeInfo[i] + "   " + TreeInfo[i+1]);
    }

    for(int i = 2*n; i < 4*n; i= i+2)
    {
        int k = i -2*n;
        String s = Integer.toBinaryString(k/2);
        s = s.replace('0', 'a');
       s = s.replace('1', '0');
       s = s.replace('a', '1');

       TreeInfo[i] = s;
       TreeInfo[i+1] = TreeInfo[i];
       TreeInfo[i+1] = TreeInfo[i+1].replace('0', '+');
       TreeInfo[i+1] = TreeInfo[i+1].replace('1', '-');

       System.out.println(i/2 + "   "
              + TreeInfo[i] + "   " + TreeInfo[i+1]);
    }
}


//All paths to leaves are constructed as binary strings
private static void
        createPathsToLeaves(int numberOfGenerations)
```

```
{
    String s;
    for (Integer i = 0; i < numberOfLeaves; i++)
    {
        s = Integer.toBinaryString(i);
        int l = s.length();
        if (l < numberOfGenerations) {
            for (int k = l; k < numberOfGenerations; k++)
            {
                s = "0" + s;
            }
        }
        pathsToLeaves[i] = s;
        if (print)
            System.out.println(pathsToLeaves[i]);
    }
}

//Drawing machinery
private static void drawTree()
{

    createTree();
    createPathsToLeaves(numberOfGenerations);

    //Symbolic initialization
    int centerX = 1;
    int centerY = 1;

    //Draw stem
    int x00 = scale * centerX;
    int y00 = scale * centerY + numberOfLeaves
            * verticalLength/2;
    int x11 = x00 + scale * horizontalLength;
    int y11 = y00;
    Line l = new Line(x00, verticalSize - y00,
            x11, verticalSize - y11);
    root.getChildren().add(l);

    //Horizontal and basic vertical strokes
```

```
int H = scale * horizontalLength;
int V;

String path;
char direction;

//The tree is drawn:
//A path from the origin of the tree to
//each leaf is drawn
for (int i = 0; i < numberOfLeaves; i++)
{
    //Addresses will be used as paths to  be drawn
    path = pathsToLeaves[i];

    //Reinitialization
    Point2D center = new Point2D(x11, y11);

    V = scale * verticalLength;
    //Path is interpreted
    for (int j = 0; j < path.length(); j++)
    {
        direction = path.charAt(j);
        if (direction == '0') //go up
        {
            int x0 = (int) center.getX();
            int y0 = (int) center.getY();
            int x1 = x0;
            int y1 = y0 + V;
            //Draw vertical stroke
            Line l2 = new Line(x0, verticalSize - y0, x1,
                    verticalSize - y1);
            root.getChildren().add(l2);
            //Draw horizontal stroke
            Line l3 = new Line(x1, verticalSize - y1,
                    x1 + H, verticalSize - y1);
            root.getChildren().add(l3);
            center = new Point2D(center.getX()
                    , center.getY() + V);
            //center.add(new Point2D(0,V));
            if (print)
```

```
                {
                    System.out.println("V = " + V);
                    System.out.println("Center = "
                            + (int) center.getX() + "  "
                            + (int) center.getY());
                }

            } else // if direction == '1'  =  go down
            {
                int x0 = (int) center.getX();
                int y0 = (int) center.getY();
                int x1 = x0;

                int y1 = y0 - V;
                //Draw vertical stroke
                Line l4 = new Line(x0, verticalSize - y0,
                        x1, verticalSize - y1);
                root.getChildren().add(l4);
                //Draw horizontal stroke
                Line l5 = new Line(x1, verticalSize - y1,
                        x1 + H, verticalSize - y1);
                root.getChildren().add(l5);
                center = new Point2D(center.getX()
                        , center.getY() - V);

            } //End of if + else

            center = new Point2D(center.getX() + H
                        , center.getY());

            V = V / 2;
        }//End of for j
    }  //End of for: numberOfLeaves

    //Strings are drawn:
    for (int i = 0; i < treeInfoLength; i = i + 2)
    {
        //Addresses will be used as paths to the places
        //where strings must be drawn
        path = TreeInfo[i];
```

```
String s = TreeInfo[i + 1];
//Reinitialization
Point2D center = new Point2D(x00, y00);
Point2D target = center;
V = scale * verticalLength;
//Path is interpreted
for (int j = 0; j < path.length(); j++) {
    direction = path.charAt(j);
    if (direction == '0') //go up
    {
        target = new Point2D(target.getX()
                , target.getY() + V);
    } else // if direction == '1'  =  go down
    {
        target = new Point2D(target.getX()
                , target.getY() - V);
    }

    target = new Point2D(target.getX() + H
            , target.getY());

    V = V / 2;
}//End of for j
if (print)
    System.out.println("Target = "
            + (int) target.getX() + "  "
        + (int) target.getY());
//String is drawn
Text text = new Text(s);
text.setX(target.getX() + 1);

text.setY(verticalSize - target.getY() - 1);
text.setFill(Color.RED);

root.getChildren().add(text);
pane.setContent(root);
pane.setHbarPolicy(ScrollPane
        .ScrollBarPolicy.ALWAYS);
pane.setVbarPolicy(ScrollPane
        .ScrollBarPolicy.ALWAYS);
```

```
    }
}


@Override
public void start(Stage primaryStage)
{
    VBox box = new VBox();
    Scene scene = new Scene(box
            , horizontalSize, verticalSize);
    primaryStage.setTitle("Binary tree");
    primaryStage.setScene(scene);
    primaryStage.show();
    drawTree();
    box.getChildren().add(pane);
    //Automatic growth of container
    VBox.setVgrow(pane, Priority.ALWAYS);
}


public static void main(String[] args)
{
    launch(args);
}

}//End of Program  N77 BinaryBushJavaFX4
```

78 ***Exercise.*** *Run the program and play with the code.*

79 ***Exercise.*** *To implement extinction over a fullbush we plan to delete from the information of the tree as the address as the corresponding content. Now, this operation is best done over lists. So, in the previous program, change the type of TreInfo from and Array of String to a List of Strings.* Answer

80 ***Exercise.*** *Our previous programs draw trees with all of their leaves. But extinction needs to be able to draw only listed specific branches that may be very short. So, adapt the code to this need: have the drawing method subordinated to the information of the tree.* Answer

81 ***Exercise.*** *Implement extinction over your bushes.* Answer

82 ***Exercise.*** *We know that addresses of nodes in a binary tree can be denoted as binary strings and that these in turn correspond with integer numbers. Prove that all this allows to define a binary bush by a binary string. Compose the corresponding code.* Answer

83 ***Exercise.*** *Run the program presented by the Author as solution to the last exercise. Pay attention to the bookkeeping of addresses that begin with zero and that cannot be recognized as binary numbers.*

84 ***Our official encoding: The last program presented a form as one can deal with addresses that begin with a zero. Here we present another one: our tree is inserted into the branch spanned by node 1 in a tree with one more generation than the given one. The code follows:***

```
/*
 * Program N84 BinaryBushJavaFX9
 * A bush is completely defined by  a binary string.
 * This program generates a bush, encodes it
 * as a binary string that represents
 * the half of a bigger tree,  recalculates and draws the tree
 * associated to that string.
 * One can know at once if the program works by comparing
 * the two drawings: they must coincide.
 *
 * Explanation:
 * Nodes in the tree have an address. Say,
 * Address   0110 means:
 * go  up, down, down and up, 4  generations at all.
 * Each node has a content that is a string.
 * Information is kept in an array, an address followed
 * by the corresponding content.

 * This system of encoding allows to represent
 * the whole set of binary addresses by a binary string

 * Example:

        private static final String[] TreeInfo
            = {
                "0", "a",
```

```
                "1", "b",
                "00", "aa",
                "01", "ab",
                //"10", "ba",
                "11", "bb",
                //"000", "aaa",
                //"001", "aab",
                "010", "aba",
                //"011", "abb",
                //"100", "baa",
                "101", "bab",
                "110", "bba",
                "111", "bbb"
            };
 * Each address is a binary number that is in 1 to 1
 *corresondence with ordered integers beginning from
 * 0 up to the number of nodes.

 * Ancient encoding
 * Integers are naturally numbered.
 * If a node is listed, the correspondent integer also.
 * This allows to define a tree by a binary string:
 * The null string represents the null tree.
 * If node I exists in a tree, then place I in the string
 * is set to 1.

 * New encoding:   we add a "1" as prefix to each address
   (101 becomes 1101 and 001 becomes 1001)
   and translate it to integer.
   The corresponding bit is set to 1.
   Decoding: if bit I has a 1,
   I is translated to a binary string,
   the first "1" is deleted and
   we get the original address.
 *
*/

package Programs;

import java.util.ArrayList;
```

```
import java.util.List;
import java.util.Random;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point2D;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.ScrollPane;
import javafx.scene.layout.Priority;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Line;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class BinaryBushJavaFX9 extends Application
{

    //This is data for a tree in our binary notation:
    //Even entries are the addresses in the tree of the nodes,
    //odd ones are the names or contents.
    private static   List<String> TreeInfo;

    private static  int treeInfoLength;

    private static final int numberOfGenerations = 5;
    //The terminal  leaves define the complete tree
    private static final int numberOfLeaves
            = (int) Math.pow(2, numberOfGenerations);

    //Relative length of branches of the first U
    private static final int horizontalLength = 3;
    private static final int verticalLength = 8;

    //The scene
    private static final Group root = new Group();

    private static final ScrollPane pane = new ScrollPane();
    //Vertical  size of the panel
    private static final int verticalSize =
```

```
            (int) (5 * Math.pow(2, numberOfGenerations));
    private static final int horizontalSize = 800;
    //Scale must be a power of 2
    private static final int scale = 16;

    final VBox vb = new VBox();

    private static final boolean print = false;
    private static final boolean print2 = false;
    private static final boolean print3 = false;
    private static final Random r = new Random();

    //Symbolic initialization
    private static int centerX = 1;
    private static final int centerY = 1;

    //To build the tree, a full tree is constructed
    //and then each node undergoes deletion with
    //this probability
    private static final double probOfExtinction = 0.1;


    private static void printTreeInfo()
{
    System.out.println("TreeInfo");
    int l = TreeInfo.size();
    for(int i = 0; i < l; i = i+2)
    {
        System.out.println(i/2 + "   " + TreeInfo.get(i)
        + "   " + TreeInfo.get(i+1));
    }

}


//A random binary tree is created:
//addresses and contents
private static void createTree()
{
```

```
//int n = r.nextInt(numberOfLeaves);
//Test
int n = numberOfLeaves;
TreeInfo = new  ArrayList<>(2*n);

//Generation of the lower part of the tree
if (print)
    System.out.println("Lower part of full tree");
for(int i = 0; i < 2*n; i= i+2)
{

    TreeInfo.add(Integer.toBinaryString(i/2));
    String s = TreeInfo.get(i).replace('0', '+');
    s = s.replace('1', '-');
    TreeInfo.add(  s);
    if (print)
        System.out.println(i/2 + "   "
        + TreeInfo.get(i) + "   " + TreeInfo.get(i+1));
}
//Generation of the upper part of the tree
if (print)
    System.out.println("Upper part of full tree");
//1 and 0  must not be repeated.
for(int i = 4; i < 2*n; i= i+2)
{

    String s = Integer.toBinaryString(i/2);
    s = s.replace('0', 'a');
    s = s.replace('1', '0');
    s = s.replace('a', '1');
    TreeInfo.add(s);

    s = s.replace('0', '+');
    s = s.replace('1', '-');
    TreeInfo.add(  s);
    if (print)
        System.out.println((2*n + i-4)/2 + "   "
            + TreeInfo.get(2*n + i-4) + "   "
            + TreeInfo.get(2*n + i-4+1));
}
```

```
    extinction();

    treeInfoLength = TreeInfo.size();
}

//If a node listed at place l must be removed,
//the forward subtree is also deleted
private static void removeSubtree(int l)
{
    String address = TreeInfo.get(l);
    int a = address.length();
    int size0 = TreeInfo.size();
    //All nodes are  scanned backwards for removing
    for(int i = size0-2; i > -1;  i = i-2)
    {
        //If a node is 10001,
        //a descendant has address 10001 + something
        String s = TreeInfo.get(i);
        int b = s.length();
        if (b >= a)
        {
            String leftSubstring
                = TreeInfo.get(i).substring(0, a);
            if (leftSubstring.contentEquals(address))
            {
                //Content is removed
                TreeInfo.remove(i+1);
                //Address is removed
                TreeInfo.remove(i);
            }
        }
    }
}


//All listed nodes of the tree are subjected to
//deletion with the same probability
private static void extinction()
{
```

```
    int size0 = TreeInfo.size();
    //All nodes are scanned backwards for removing
    for(int i = size0-2; i > -1;  i = i-2)
    {

        if (TreeInfo.size() > 0)
        {
            double t =  r.nextDouble();
            if (t < probOfExtinction)
                removeSubtree(i);
        }
    }
}


//Drawing machinery
private static void drawTree()
{
    //Draw stem
    int x00 = scale * centerX;
    int y00 = scale * centerY + numberOfLeaves
            * verticalLength/2;
    int x11 = x00 + scale * horizontalLength;
    int y11 = y00;
    Line l = new Line(x00, verticalSize - y00,
            x11, verticalSize - y11);
    root.getChildren().add(l);

    //Horizontal and basic vertical strokes
    int H = scale * horizontalLength;
    int V;

    String path;
    char direction;

    //The tree is drawn:
    //A path from the origin of the tree to
    //each node is drawn
    for (int i = 0; i < TreeInfo.size(); i= i+2)
```

```
{
    //Addresses will be used as paths to  be drawn
    path = TreeInfo.get(i);


    //Reinitialization
    Point2D center = new Point2D(x11, y11);

    V = scale * verticalLength;
    //Path is interpreted
    for (int j = 0; j < path.length(); j++)
    {
        direction = path.charAt(j);
        if (direction == '0') //go up
        {
            int x0 = (int) center.getX();
            int y0 = (int) center.getY();
            int x1 = x0;
            int y1 = y0 + V;
            //Draw vertical stroke
            Line l2 = new Line(x0, verticalSize - y0, x1,
                    verticalSize - y1);
            root.getChildren().add(l2);
            //Draw horizontal stroke
            Line l3 = new Line(x1, verticalSize - y1,
                    x1 + H, verticalSize - y1);
            root.getChildren().add(l3);
            center = new Point2D(center.getX()
                    , center.getY() + V);
            //center.add(new Point2D(0,V));
            if (print2)
            {
                System.out.println("V = " + V);
                System.out.println("Center = "
                        + (int) center.getX() + "  "
                        + (int) center.getY());
            }

        } else // if direction == '1'  =  go down
        {
```

```
                int x0 = (int) center.getX();
                int y0 = (int) center.getY();
                int x1 = x0;

                int y1 = y0 - V;
                //Draw vertical stroke
                Line l4 = new Line(x0, verticalSize - y0,
                        x1, verticalSize - y1);
                root.getChildren().add(l4);
                //Draw horizontal stroke
                Line l5 = new Line(x1, verticalSize - y1,
                        x1 + H, verticalSize - y1);
                root.getChildren().add(l5);
                center = new Point2D(center.getX()
                        , center.getY() - V);

            } //End of if + else

            center = new Point2D(center.getX() + H
                        , center.getY());

            V = V / 2;
        }//End of for j
    }  //End of for: numberOfLeaves

    //Strings are drawn:
    for (int i = 0; i < treeInfoLength; i = i + 2)
    {
        //Addresses will be used as paths to the places
        //where strings must be drawn
        path = TreeInfo.get(i);
        String s = TreeInfo.get(i+1);
        //Reinitialization
        Point2D center = new Point2D(x00, y00);
        Point2D target = center;
        V = scale * verticalLength;
        //Path is interpreted
        for (int j = 0; j < path.length(); j++)
        {
            direction = path.charAt(j);
```

```
            if (direction == '0') //go up
            {
                target = new Point2D(target.getX()
                        , target.getY() + V);
            } else // if direction == '1'  =  go down
            {
                target = new Point2D(target.getX()
                        , target.getY() - V);
            }

            target = new Point2D(target.getX() + H
                    , target.getY());

            V = V / 2;
        }//End of for j
        if (print2)
            System.out.println("Target = "
                    + (int) target.getX() + "  "
                + (int) target.getY());
        //String is drawn
        Text text = new Text(s);
        text.setX(target.getX() + 1);

        text.setY(verticalSize - target.getY() - 1);
        text.setFill(Color.RED);

        root.getChildren().add(text);

    }
}

//Binary string s is decoded into an int
private static int stringToNumber(String s)
{
    int n = 0;
    int l = s.length();

    for(int j=0;j  < l;j++ )
    {
        char c  =   s.charAt(j);
```

```
        int  v = Character.getNumericValue(c);
        int q = j+1;
        int number1 = (int) (v * Math.pow(2,l-q));
        n  = n + number1;
    }
    return n;
}


private static int stringToInt(String address)
{
    int output = 0;
    //System.out.print("String = " + address );
    char c  = address.charAt(0);
    if  (c == '1') output = stringToNumber(address);
    if  (c == '0')
    {
        if (address.length() == 1) output = 0;
            else
        {
        address = address.replace('0', 'a');
        address = address.replace('1', '0');
        address = address.replace('a', '1');
        output = stringToNumber(address) + numberOfLeaves -2;
        }
    }
    //System.out.println("  int =  " + output);
    return output;
}


//A binary string is associated to the tree
//in a 1 to 1 fashion.
private static String treeToBinaryString()
{
    //This string is filled in zeros:
    //It represents the null tree
    String bitTree = "";
    for (int i = 0; i < 2*numberOfLeaves-1; i++)
        bitTree = bitTree + "0";
```

```
    if (print3)
        System.out.println(bitTree);
    String address;
    //Each listed address  corresponds
    //with an integer. The corresponding bit
    //in bitTree is set to 1.
    for (int i = 0; i < TreeInfo.size(); i= i+2)
    {

        address = TreeInfo.get(i);
        //"1" is appendex as prefix
        String s = "1" + address;
        int n = stringToInt(s);
        if (print3)
            System.out.println("String = " + address
                    + " int =  " + n);
        if (n < bitTree.length()-1)
         bitTree = bitTree.substring(0, n)
                + "1" + bitTree.substring(n+1);
        else bitTree = bitTree.substring(0, n)
                + "1" ;
        if (print3)
            System.out.println(bitTree);

    }

    return bitTree;
}


//The tree encoded by binary string b is
//decoded and drawn
private static void redrawTree(String b)
{
    TreeInfo.clear();
    int l = b.length();
    char c ;
    String s ;
    //First two bits are zero.
    for (Integer i = 2; i < l; i++)
```

```
    {
        c = b.charAt(i);
        if (c == '1')
        {
            if ( i < 2*numberOfLeaves)
            {
                s = Integer.toBinaryString(i);
                //prefix "1" is deleted
                s = s.substring(1);
                TreeInfo.add(s);
                s = s.replace('0', '+');
                s = s.replace('1', '-');
                TreeInfo.add(s);
            }
        }
    }
    treeInfoLength = TreeInfo.size();
    centerX = 40;
    drawTree();
}


@Override
public void start(Stage primaryStage)
{
    VBox box = new VBox();
    Scene scene = new Scene(box
            , horizontalSize, verticalSize);
    primaryStage.setTitle("Binary tree");
    primaryStage.setScene(scene);
    primaryStage.show();
    createTree();
    centerX = 10;
    System.out.println("After extinction");
    printTreeInfo();
    drawTree();

    String treeToBinaryString = treeToBinaryString();
    System.out.println("treeToBinaryString = \n"
                + treeToBinaryString);
```

```
    redrawTree(treeToBinaryString);

    pane.setContent(root);
    pane.setHbarPolicy(ScrollPane
            .ScrollBarPolicy.ALWAYS);
    pane.setVbarPolicy(ScrollPane
                .ScrollBarPolicy.ALWAYS);
    pane.setPrefSize(horizontalSize, verticalSize);
    box.getChildren().add(pane);
    //Automatic growth of container
    VBox.setVgrow(pane, Priority.ALWAYS);
}


public static void main(String[] args)
{
    launch(args);
}

}//End of Program  N84 BinaryBushJavaFX9
```

85 ***Exercise.*** *Run the previous program and play with the code.*

We can use now our trees to make biology. For our work we prefer our home-made trees.

4.4 Pleiotropy in action

The work through Vols X and XI allowed us to believe that pleitropy (one gene affecting many traits) generates a problem against which evolution is perfectly impotent. The reason is that gathering small change is anymore possible under pleitropy: you get great coordinated changes that amount to a solution of posed problem else you get nothing. That is why we are very willing to assess the role of pleitropy in regard with the stability of the shape of bones against the always working power of mutation.

86 ***Theorem.*** *Biological shape naturally gives rise to pleitropy.*

Proof: Shape in biology must be a natural byproduct of programmed reproduction, differentiation and death of cells. Now, the reproduction of cells is described

by a tree: the mother cell is the root and its descendant forms the arms of the tree. Let us assume that cell reproduction is equivalent to cell splitting so, our trees are binary. This representation of temporal events depicts a mother cell that disappears to give rise to two daughters. But if we pay attention to space events, a cell can give rise to many children that fill surrounding space. In space representation, reproduction would be best depicted by a tree with various children. The number of children may be supposed to be self regulated, at least partially, to fill in vacua.

On the other hand, the pleitropy that we investigated in Vols XI and XII resulted from the problem of fitting a tree to a distance matrix. Details are as follows: the distance in a tree from one leaf to another is defined as the minimum distance along all paths that connect the two leaves. So, we have a family of traits each one describing the distance through the tree from leaf to leaf. These distances were next plugged into a great fitness function that measures the difference of those distances with another ones represented by the target distance matrix. Let us notice that pleitropy results from the fact that changing any distance in within whatever pair of nodes implies that a lot of distances from leaf to leaf also change.

Now, we can apply the very same model to ontogeny: the tree is given by reproduction and the overall fitness function that depends on many interconnected traits is the shape itself that is possibly determined by the overall interaction of all cells that conform the shape with the rest of the body. Thus, pleiotropy is all pervading along biological shape if only shapes have some selective value. Nevertheless, we deny a selective meaning to the shape of the human hand of present populations but we give a very high one to the wings of birds.

We remark that we rely on our intuition meaning that statistics and Java simulations expect for people that want to pass from subjective feelings to objective results. This seems to be very difficult so, let us aim at showing something simple but important: binary bushes can generate detailed 3D shapes. The problem with 3D bushes is that 2015-JavaFX lacks 3D lines. The well known solution is to resort to thin cylinders, whose fundamental model is presented by the next code. Because of memory cost, cylinders might be left in favor of 3D boxes.

87 ***The following code shows a 3D cylinder.***

```
/*
   Program N87  TreeToShape
    This is a 3D template to use with this project.
    It just presents a cylinder, which will be
    the fundamental building block of our trees.
*/
```

```
package Programs;

//Files in the hosting project Programs
import Tools.Axes;
import Tools.Cameras;
import Tools.Xform;
//Used java resources
import java.lang.reflect.Field;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Cylinder;
import javafx.stage.Stage;



public class TreeToShape extends Application
{

    //Printing for debugging
    final static boolean printReadFile = false;
    private static final String title
            =  "3D bush";


    final Group root = new Group();
    private static final Cameras camera = new Cameras();
    final double cameraDistance = 5450;//450
    //Axes : ctrl + A
    private static final Axes axes = new Axes();
    final Xform world3DXform = new Xform();
    final Xform bonesXform = new Xform();
    //An array with all colors that are named in JavaFX
```

```
    private static final Color JavaFXColors[]
                                = new Color[148];
    //Generator of pseudo random numbers
    private static final Random rd = new Random();


@Override
public void start(Stage primaryStage)
{
    System.out.println("start");

    Scene scene = new Scene(root, 1200, 1000, true);


    primaryStage.setTitle(title
            + ". Drag the mouse to rotate. "
        + " Ctrl + A = Axis visibility"
            );
    scene.setFill(Color.AQUAMARINE);
    primaryStage.setScene(scene);
    primaryStage.show();
    axes.buildAxes();
    axes.handleKeyboard(scene);

    camera.handleMouse(scene);
    camera.buildCamera();
    scene.setCamera(camera.camera);


    final Group parent = new Group(create3dContent());
    world3DXform.getChildren().addAll(parent
            , axes.axesXform);

    root.getChildren().add(world3DXform);

    primaryStage.show();
    play();


}
```

```
public void play()
{

}

@Override public void stop()
{

}




private Node create3dContent()
{
    //Colors that have a name in JavaFX
    getColors();
    Project3D  h =  new Project3D();
    return new Group(h);
}

//Colors with a name in JavaFX are put into an array
static private Color[] getColors()
{
    //Colors with a name in JavaFX
    final Field[] fields = Color.class.getFields();


    int N = 0;
    //Colors are put into an array
    for (final Field field : fields)
    {
        if (field.getType() == Color.class)
        {
            try
            {
                final Color color = (Color) field.get(null);
                //Array to keep colors
```

```
                JavaFXColors[N] = color;
                /*
                //The name of the color
                final String colorName = field.getName();
                System.out.println("N = " + N + "  "
                        + color.toString() + " " + colorName);
                */
                N++;
            } catch (IllegalArgumentException
                    | IllegalAccessException ex)
            {
                Logger.getLogger(TreeToShape.class
                    .getName()).log(Level.SEVERE, null, ex);
            }
        }
    }

    return JavaFXColors;
}


//A random color is synthesized
 private static Color randomColor()
 {
    //A JavaFX named color is picked
    //but the first, Transparent, is ignored
    int n = rd.nextInt(JavaFXColors.length-1)+1;
    Color c = JavaFXColors[n];
    //Some light colors are not visualized so,
    //they might be darkened:
    //c = c.darker();
    return c;
 }


//=========================================================
//======= Beginning of Inner class Project3D ===========
//=========================================================
```

```
//The drawing is synthesized
private class Project3D extends Group
{

    //Constructor
    Project3D()
    {
        Cylinder cylinder = new Cylinder(20, 50);
        Color c = randomColor();
        PhongMaterial materialCylinder = new PhongMaterial(c);
        cylinder.setMaterial(materialCylinder);
        final Group g = new Group(cylinder);
        getChildren().addAll(g );

    }
}//End of inner class Project3D



public static void main(String[] args)
{
    launch(args);
}


}//End of program N87 TreeToShape
```

88 ***Exercise.*** *Run the previous program and play with the code.*

The previous program presented a template to be used in the sequel. Let us test it: our former 2D tree must be accepted, drawn and be capable of being rotated.

89 ***The following code shows a 2D tree that is inserted in 3D space.***

```
/*
   Program N89  TreeToShape2
   The program presents
```

```
   a 2D random binary bush
   that is inserted in 3D space.
*/

package Programs;



//Files in the hosting project
import Tools.Axes;
import Tools.Cameras;
import Tools.Xform;
//Java files
import java.lang.reflect.Field;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point2D;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Line;
import javafx.scene.text.Text;
import javafx.stage.Stage;



public class TreeToShape2 extends Application
{

    //This is data for a tree in our binary notation:
    //Even entries are the addresses in the tree of the nodes,
    //odd ones are the names or contents.
    private static   List<String> TreeInfo;
```

```
private static  int treeInfoLength;

private static final int numberOfGenerations = 5;
//The terminal  leaves define the complete tree
private static final int numberOfLeaves
        = (int) Math.pow(2, numberOfGenerations);

private static final String title
        =  "2D bush inserted into 3D space";


final Group root = new Group();
private static final Cameras camera = new Cameras();
final double cameraDistance = 5450;//450
//Axes : ctrl + A
private static final Axes axes = new Axes();
final Xform world3DXform = new Xform();
final Xform bonesXform = new Xform();
//An array with all colors that are named in JavaFX
private static final Color JavaFXColors[]
                            = new Color[148];
//Generator of pseudo random numbers
private static final Random r = new Random();


private static final boolean print = false;
private static final boolean print2 = false;
private static final boolean print3 = false;
private static final boolean print4 = false;
private static final boolean print5 = false;



//Vertical  size of the panel
private static final int verticalSize =
        (int) (5 * Math.pow(2, numberOfGenerations));

//Scale must be a power of 2
```

```
    private static final int scale = 16;
    //Symbolic initialization
    private static final int centerX = 1;
    private static final int centerY = 1;

    //Relative length of branches of the first U
    private static final int horizontalLength = 3;
    private static final int verticalLength = 8;

    final VBox vb = new VBox();



@Override
public void start(Stage primaryStage)
{

    primaryStage.setTitle(title);
    Scene scene = new Scene(root, 1200, 1000, true);

    scene.setFill(Color.AQUAMARINE);
    primaryStage.setScene(scene);
    primaryStage.show();
    axes.buildAxes();
    axes.handleKeyboard(scene);
    world3DXform.getChildren().addAll( axes.axesXform);

    camera.handleMouse(scene);
    camera.buildCamera();
    scene.setCamera(camera.camera);

    //TreeInfo is filled in:
    //A binary address to each node of the tree
    //is following by a content
    createTree();
    printTreeInfo();

    Group parent = new Group(create3dContent(TreeInfo));
        world3DXform.getChildren().addAll(parent);
        root.getChildren().add(world3DXform);
```

```
    primaryStage.show();
    play();


}

public void play()
{

}

@Override public void stop()
{

}




private Node create3dContent(List<String> TreeInfo)
{
    Project3D  p =  new Project3D(TreeInfo);
    return new Group(p);
}




private static void printTreeInfo()
{
    System.out.println("TreeInfo");

    int l = TreeInfo.size();
    for(int i = 0; i < l; i = i+2)
    {
        System.out.println(i/2 + "   " + TreeInfo.get(i)
        + "   " + TreeInfo.get(i+1));
    }
```

```
}


//A binary string with 2*numberOfLeaves-1 bits
private static String randomBinaryString()
{
    //A random binary string
    String bitTree = "";
    for (int i = 0; i < 2*numberOfLeaves-1; i++)
    {
        Integer j = r.nextInt(2);
        bitTree = bitTree + j.toString();

    }

    if (print3)
        System.out.println(bitTree);
    return bitTree;
}


//The tree encoded by binary string b is
//decoded and drawn.
//The tree corresponds with the lower
//part of a tree with one more generation.
//All addresses in there begin with 1.
private  void  createTree( )
{
    int n = numberOfLeaves;
    TreeInfo = new  ArrayList<>(2*n-1);
    //A random binary string:

    String b = randomBinaryString();

    int l = b.length();
    char c ;
    String s ;
    //First two bits do not matter
    for (Integer i = 2; i < l; i++)
    {
```

```
        //If bit i is 1, node i is added to the tree
        c = b.charAt(i);
        if (c == '1')
        {
            if ( i < 2*numberOfLeaves-1)
            {
                s = Integer.toBinaryString(i);
                //prefix "1" is deleted
                s = s.substring(1);
                TreeInfo.add(s);
                s = s.replace('0', '+');
                s = s.replace('1', '-');
                TreeInfo.add(s);
            }
        }
    }
    treeInfoLength = TreeInfo.size();
}


//=======================================================
//======= Beginning of Inner class Project3D ===========
//=======================================================



//The drawing is synthesized
private class Project3D extends Group
{

    //Constructor
    Project3D(List<String> TreeInfo)
    {
        System.out.println("List<String> TreeInfo");

        //Tree is drawn to expand a 3D bush
        Group g = drawTree(TreeInfo);
        getChildren().addAll(g );
    }
```

```
   private   Group drawTree(List<String> TreeInfo)
{
    Group myGroup  = new Group();
        //Draw stem
    int x00 = scale * centerX;
    int y00 = scale * centerY + numberOfLeaves
            * verticalLength/2;
    int x11 = x00 + scale * horizontalLength;
    int y11 = y00;
    Line l = new Line(x00, verticalSize - y00,
            x11, verticalSize - y11);
    myGroup.getChildren().add(l);

    //Horizontal and basic vertical strokes
    int H = scale * horizontalLength;
    int V;

    String path;
    char direction;

    //The tree is drawn:
    //A path from the origin of the tree to
    //each node is drawn
    for (int i = 0; i < TreeInfo.size(); i= i+2)
    {
        //Addresses will be used as paths to  be drawn
        path = TreeInfo.get(i);


        //Reinitialization
        Point2D center = new Point2D(x11, y11);

        V = scale * verticalLength;
        //Path is interpreted
        for (int j = 0; j < path.length(); j++)
        {
            direction = path.charAt(j);
            if (direction == '0') //go up
            {
                int x0 = (int) center.getX();
```

```
            int y0 = (int) center.getY();
            int x1 = x0;
            int y1 = y0 + V;
            //Draw vertical stroke
            Line l2 = new Line(x0, verticalSize - y0, x1,
                    verticalSize - y1);
            myGroup.getChildren().add(l2);
            //Draw horizontal stroke
            Line l3 = new Line(x1, verticalSize - y1,
                    x1 + H, verticalSize - y1);
            myGroup.getChildren().add(l3);
            center = new Point2D(center.getX()
                    , center.getY() + V);
            //center.add(new Point2D(0,V));
            if (print2)
            {
                System.out.println("V = " + V);
                System.out.println("Center = "
                        + (int) center.getX() + "  "
                        + (int) center.getY());
            }

        } else // if direction == '1'  =  go down
        {
            int x0 = (int) center.getX();
            int y0 = (int) center.getY();
            int x1 = x0;

            int y1 = y0 - V;
            //Draw vertical stroke
            Line l4 = new Line(x0, verticalSize - y0,
                    x1, verticalSize - y1);
            myGroup.getChildren().add(l4);
            //Draw horizontal stroke
            Line l5 = new Line(x1, verticalSize - y1,
                    x1 + H, verticalSize - y1);
            myGroup.getChildren().add(l5);
            center = new Point2D(center.getX()
                    , center.getY() - V);
```

```
            } //End of if + else

            center = new Point2D(center.getX() + H
                        , center.getY());

            V = V / 2;
        }//End of for j
    }  //End of for: numberOfLeaves

    //Strings are drawn:
    for (int i = 0; i < treeInfoLength; i = i + 2)
    {
        //Addresses will be used as paths to the places
        //where strings must be drawn
        path = TreeInfo.get(i);
        String s = TreeInfo.get(i+1);
        //Reinitialization
        Point2D center = new Point2D(x00, y00);
        Point2D target = center;
        V = scale * verticalLength;
        //Path is interpreted
        for (int j = 0; j < path.length(); j++)
        {
            direction = path.charAt(j);
            if (direction == '0') //go up
            {
                target = new Point2D(target.getX()
                        , target.getY() + V);
            } else // if direction == '1'  =  go down
            {
                target = new Point2D(target.getX()
                        , target.getY() - V);
            }

            target = new Point2D(target.getX() + H
                    , target.getY());

            V = V / 2;
        }//End of for j
        if (print2)
```

```
            System.out.println("Target = "
                    + (int) target.getX() + "  "
                + (int) target.getY());
        //String is drawn
        Text text = new Text(s);
        text.setX(target.getX() + 1);

        text.setY(verticalSize - target.getY() - 1);
        text.setFill(Color.RED);

        root.getChildren().add(text);

    }
    return myGroup;
}

}//End of inner class Project3D


public static void main(String[] args)
{
    launch(args);
}

}//End of programN89  TreeToShape2
```

90 ***Exercise.*** *Run the previous program and play with the code.*

91 ***Exercise.*** *Notice that in the previous program a node might appear but not some of its ancestors. Correct the code to insert all ancestors of each present node.* *Answer*

JavaFX naturally inserts a 2D drawing into 3D space. To go further, we need to implement 3D lines. Let us do it: among the many options, we consider that meshes is the best.

92 ***3D lines.*** *Let us use meshes to implement 3D lines as 3D boxes. Our convention for nodes follow:*

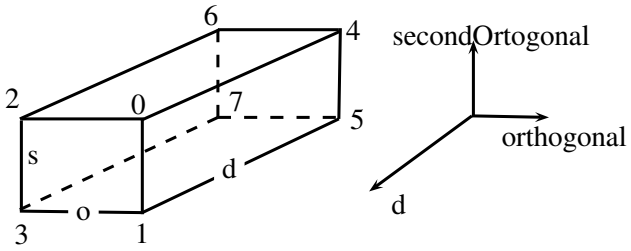


Figure 4.1. We see a box with measures d (the equivalent of x-coordinate), s, (orthogonal along the equivalent of y-coordinate) and s (secondOrthogonal along the equivalent of z-coordinate). The name of a vertex will be its place in the list of vertexes. The coordinates of vertexes will be the natural ones according to vector addition. Vertex number 7 coincides with p0, the first point of the line, d is the vector that represents the line properly, o and s are unitary.

The code for the class Line3D in package Tools follows:

```
/*
 * Program N92 Line3D
 * A 3D line modelled on a mesh of a box.
 * The line is determined by
 * a directional vector and a point.
 * The directional vector generates
 * an orthogonal triade,
 * which in turns generates a box.
 * The box is implemented as a mesh.

 * The program has no main so, it is not be run.
 * It is a tool to be used from other progams.
 */
package Tools;



import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.MeshView;
import javafx.scene.shape.TriangleMesh;
```

```
import javafx.stage.Stage;


public final class Line3D
{

public Group group;


public Line3D()
{}


//Line that connects P0 and P1
public  Line3D( Point3D P0, Point3D P1)
{
    group = Line3DBuilder( P0, P1);
}

//Line that connects P0 = (x0,y0,z0)
//and P1 = (x1,y1,z1)
public  Line3D( int x0, int y0, int z0,
                int x1, int y1, int z1)
{
    Point3D P0 = new Point3D(x0,y0,z0);
    Point3D P1 = new Point3D(x1,y1,z1);
    group = Line3DBuilder( P0, P1);
}


//Line is implemented as a box that is
//modelled on a mesh
public  Group Line3DBuilder( Point3D P0, Point3D P1)
{
    group = new Group();


    //Directional vector
    Point3D d = P1.subtract(P0);
    /*System.out.println("Directional  = "
```

```
                + d);*/
        double length = d.magnitude();
        Point3D orthogonal = new Point3D(1,1,1);
        Point3D secondOrthogonal = new Point3D(1,1,1);
        if (length < 0.0001)
        {
            System.out.println("Null line");
        }
        else
        {
            orthogonal = findOrthogonal(d).normalize();
            /*System.out.println("Orthogonal  = "
                    + orthogonal);*/
            secondOrthogonal
                    = (d.crossProduct(orthogonal)).normalize();
            /*System.out.println("SecondOrthogonal  = "
                    + secondOrthogonal);  */
        }

        return generateBox(P0, d, orthogonal, secondOrthogonal);

    }



    private Point3D findOrthogonal(Point3D v)
    {
        double x = v.getX();
        double y = v.getY();
        double z = v.getZ();
        //Let us find an orthonormal vector to v.
        //To that aim, we must solve
        //xm + yn + zo = 0.
        double m = 1;
        double n = 1;
        double o = 1;
        if (x == 0 )
        {
            if (y == 0 )
            {
```

```
            if ( z == 0)
                System.out.println("Null line");
            else //if z is not zero
            {
                m = 1;
                o = 1;
                n = -(x+y)/z;
            }
        }
        else //if y is not zero
        {
            m = 1;
            o = 1;
            n = -(x+z)/y;
        }
    }
    else //if x is not zero
    {
        n = 1;
        o = 1;
        m = -(y+z)/x;
    }
    return new Point3D(m,n,o);
}


//A box that has its main vertex at p0 and a triad
//composed of d, orthogonal and secondOrthogonal
private Group generateBox(Point3D p0, Point3D d,
        Point3D orthogonal, Point3D secondOrthogonal)
{

    Point3D P7 = p0;
    Point3D P3 = p0.add(d);
    Point3D P5 = p0.add(orthogonal);
    Point3D P1 = P5.add(d);
    Point3D P6 = p0.add(secondOrthogonal);
    Point3D P4 = P5.add(secondOrthogonal);
    Point3D P2 = P3.add(secondOrthogonal);
    Point3D P0 = P1.add(secondOrthogonal);
```

```
    //The mesh is declared and instantiated
    TriangleMesh mesh = new TriangleMesh();
    //Declaration of the points of the mesh.
    //Each point has 3 coordinates.
    //There is neither a tag for  points
    //or a special separator in within points!
    mesh.getPoints().addAll
    (

    (float) P0.getX(), (float) P0.getY(), (float) P0.getZ(),
    (float) P1.getX(), (float) P1.getY(), (float) P1.getZ(),
    (float) P2.getX(), (float) P2.getY(), (float) P2.getZ(),
    (float) P3.getX(), (float) P3.getY(), (float) P3.getZ(),
    (float) P4.getX(), (float) P4.getY(), (float) P4.getZ(),
    (float) P5.getX(), (float) P5.getY(), (float) P5.getZ(),
    (float) P6.getX(), (float) P6.getY(), (float) P6.getZ(),
    (float) P7.getX(), (float) P7.getY(), (float) P7.getZ()

    );

//This is for future use
    mesh.getTexCoords().addAll(0,0);

    /* List of triangles:

    //We have 12 triangles, two for each face of the box.
    //Each triangle is defined by a triade of points:
    //The upper face has two triangles:
    //The first is conformed by points 0, 4 and 2.
    //The second by points 2, 4, 6.

    0,   2,   1, //Frontal face
    2,   3,   1,
    4,   5,   6, //Back face
    6,   5,   7,
    0,   1,   4, //Right face
    4,   1,   5,
    2,   6,   3, //Left face
    3,   6,   7,
```

```
    0,   4,   2, //Upper face
    2,   4,   6,
    1,   3,   5, //Bottom face
    5,   3,   7

    */

    //The official form as  the list of triangles
    //must be declared for a mesh in JavaFX
    //demands additional information that we just fill
    //with a 0 intercalated after each vertex.
    mesh.getFaces().addAll
    (
        //Frontal face with vertexes 0,2, 1.
        0, 0, 2, 0, 1, 0, //Frontal face
        2, 0, 3, 0, 1, 0,
        4, 0, 5, 0, 6, 0, //Back face
        6, 0, 5, 0, 7, 0,
        0, 0, 1, 0, 4, 0, //Right face
        4, 0, 1, 0, 5, 0,
        2, 0, 6, 0, 3, 0, //Left face
        3, 0, 6, 0, 7, 0,
        0, 0, 4, 0, 2, 0, //Upper face
        2, 0, 4, 0, 6, 0,
        1, 0, 3, 0, 5, 0, //Bottom face
        5, 0, 3, 0, 7, 0
    );

    //A box is declared and instantiated as  a cube
    MeshView box = new MeshView(mesh);
    PhongMaterial earthMaterial
            = new PhongMaterial(Color.RED);
    box.setMaterial(earthMaterial);

    //Initialization
    final Group boxGroup = new Group(box);

  return boxGroup;
}
```

```
public  void start(Stage primaryStage)
{
    Line3D l = new Line3D(0,0,0,1,0,0);
}

}//End of main class N92 Line3D
```

93 ***Exercise.*** *Study the previous program and agree else disagree that it will produces 3D lines.*

Let us use our 3D lines.

94 ***Planar tree with 3D lines is presented by the next code.***

```
/*
   Program N94  TreeToShape4
   The program presents
   a planar random binary bush
   whose lines are 3D boxes modelled on  meshes.
*/

package Programs;



//Files in the hosting project
import Tools.Axes;
import Tools.Cameras;
import Tools.Line3D;
import Tools.Xform;

import java.lang.reflect.Field;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import static javafx.application.Application.launch;
```

```
import javafx.geometry.Point2D;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.ScrollPane;
import javafx.scene.paint.Color;
import javafx.scene.text.Text;
import javafx.stage.Stage;



public class TreeToShape4 extends Application
{

    //This is data for a tree in our binary notation:
    //Even entries are the addresses in the tree of the nodes,
    //odd ones are the names or contents.
    private static   List<String> TreeInfo;

    private static final int numberOfGenerations = 5;
    //The terminal  leaves define the complete tree
    private static final int numberOfLeaves
            = (int) Math.pow(2, numberOfGenerations);
    //Printing for debugging
    final static boolean printReadFile = false;
    private static final String title
            =  "3D bushh";


    final Group root = new Group();
    private static final Cameras camera = new Cameras();
    final double cameraDistance = 5450;//450
    //Axes : ctrl + A
    private static final Axes axes = new Axes();
    final Xform world3DXform = new Xform();
    final Xform bonesXform = new Xform();
    //An array with all colors that are named in JavaFX
    private static final Color JavaFXColors[]
                                = new Color[148];
    //Generator of pseudo random numbers
```

```
    private static final Random r = new Random();


    private static final boolean print = false;
    private static final boolean print2 = false;
    private static final boolean print3 = false;
    private static final boolean print4 = false;
    private static final boolean print5 = false;



    private static final ScrollPane pane = new ScrollPane();
    //Vertical  size of the panel
    private static final int verticalSize =
            (int) (5 * Math.pow(2, numberOfGenerations));
    //Scale must be a power of 2
    private static final int scale = 16;
    //Symbolic initialization
    private static final int centerX = 1;
    private static final int centerY = 1;

    //Relative length of branches of the first U
    private static final int horizontalLength = 3;
    private static final int verticalLength = 8;




@Override
public void start(Stage primaryStage)
{

    primaryStage.setTitle(title);
    Scene scene = new Scene(root, 1200, 1000, true);

    scene.setFill(Color.AQUAMARINE);
    primaryStage.setScene(scene);
    primaryStage.show();
    axes.buildAxes();
```

```
    axes.handleKeyboard(scene);
    world3DXform.getChildren().addAll( axes.axesXform);

    camera.handleMouse(scene);
    camera.buildCamera();
    scene.setCamera(camera.camera);

    //TreeInfo is filled in:
    //A binary address to each node of the tree
    //is following by a content
    createTree();
    printTreeInfo();

    Group parent = new Group(create3dContent(TreeInfo));
        world3DXform.getChildren().addAll(parent);
        root.getChildren().add(world3DXform);


    primaryStage.show();
    play();


}

public void play()
{

}

@Override public void stop()
{

}




private Node create3dContent(List<String> TreeInfo)
{
```

```
    //Colors that have a name in JavaFX
    getColors();
    Project3D  p =  new Project3D(TreeInfo);
    return new Group(p);
}

//Colors with a name in JavaFX are put into an array
static private Color[] getColors()
{
    //Colors with a name in JavaFX
    final Field[] fields = Color.class.getFields();


    int N = 0;
    //Colors are put into an array
    for (final Field field : fields)
    {
        if (field.getType() == Color.class)
        {
            try
            {
                final Color color = (Color) field.get(null);
                //Array to keep colors
                JavaFXColors[N] = color;
                /*
                //The name of the color
                final String colorName = field.getName();
                System.out.println("N = " + N + "  "
                        + color.toString() + " " + colorName);
                */
                N++;
            } catch (IllegalArgumentException
                    | IllegalAccessException ex)
            {
                Logger.getLogger(TreeToShape.class
                    .getName()).log(Level.SEVERE, null, ex);
            }
       }
    }
```

```
    return JavaFXColors;
}




private static void printTreeInfo()
{
    System.out.println("TreeInfo");

    int l = TreeInfo.size();
    for(int i = 0; i < l; i = i+2)
    {
        System.out.println(i/2 + "   " + TreeInfo.get(i)
        + "   " + TreeInfo.get(i+1));
    }

}



//A binary string with 2*numberOfLeaves-1 bits
private static String randomBinaryString()
{
    //A random binary string
    String bitTree = "";
    for (int i = 0; i < 2*numberOfLeaves-1; i++)
    {
        Integer j = r.nextInt(2);
        bitTree = bitTree + j.toString();

    }

    if (print3)
        System.out.println(bitTree);
    return bitTree;
}
```

```
//The tree encoded by binary string b is
//decoded and drawn.
//The tree corresponds with the lower
//part of a tree with one more generation.
private  void  createTree( )
{
    int n = numberOfLeaves;
    TreeInfo = new  ArrayList<>(2*n-1);
    //A random binary string:
    //If bit I is 1, the I node is added to the tree
    String b = randomBinaryString();

    int l = b.length();
    char c ;
    String s ;
    //First two bits do not matter
    for (Integer i = 2; i < l; i++)
    {
        c = b.charAt(i);
        if (c == '1')
        {
            if ( i < 2*numberOfLeaves-1)
            {
                s = Integer.toBinaryString(i);
                //prefix "1" is deleted
                s = s.substring(1);
                TreeInfo.add(s);
                s = s.replace('0', '+');
                s = s.replace('1', '-');
                TreeInfo.add(s);
            }
        }
    }
}


//=======================================================
//======= Beginning of Inner class Project3D ===========
//=======================================================
```

```
//The drawing is synthesized
private class Project3D extends Group
{

    //Constructor
    Project3D(List<String> TreeInfo)
    {
        System.out.println("List<String> TreeInfo");

        //Tree is drawn to expand a 3D bush
        Group g = drawTree(TreeInfo);




        getChildren().addAll(g );
    }

   private   Group drawTree(List<String> TreeInfo)
{
    Group myGroup  = new Group();
        //Draw stem
    int x00 = scale * centerX;
    int y00 = scale * centerY + numberOfLeaves
            * verticalLength/2;
    int x11 = x00 + scale * horizontalLength;
    int y11 = y00;
    Line3D l = new Line3D(x00, verticalSize - y00,0,
            x11, verticalSize - y11,0);
    myGroup.getChildren().add(l.group);

    //Horizontal and basic vertical strokes
    int H = scale * horizontalLength;
    int V;

    String path;
    char direction;
```

```
//The tree is drawn:
//A path from the origin of the tree to
//each node is drawn
for (int i = 0; i < TreeInfo.size(); i= i+2)
{
    //Addresses will be used as paths to  be drawn
    path = TreeInfo.get(i);


    //Reinitialization
    Point2D center = new Point2D(x11, y11);

    V = scale * verticalLength;
    //Path is interpreted
    for (int j = 0; j < path.length(); j++)
    {
        direction = path.charAt(j);
        if (direction == '0') //go up
        {
            int x0 = (int) center.getX();
            int y0 = (int) center.getY();
            int x1 = x0;
            int y1 = y0 + V;
            //Draw vertical stroke
            Line3D l2 = new Line3D(x0, verticalSize - y0, 0,
                    x1, verticalSize - y1,0);
            myGroup.getChildren().add(l2.group);
            //Draw horizontal stroke
            Line3D l3 = new Line3D(x1, verticalSize - y1,0,
                    x1 + H, verticalSize - y1,0);
            myGroup.getChildren().add(l3.group);
            center = new Point2D(center.getX()
                    , center.getY() + V);
            //center.add(new Point2D(0,V));
            if (print2)
            {
                System.out.println("V = " + V);
                System.out.println("Center = "
                        + (int) center.getX() + "  "
```

```
                        + (int) center.getY());
                }

            } else // if direction == '1'  =  go down
            {
                int x0 = (int) center.getX();
                int y0 = (int) center.getY();
                int x1 = x0;

                int y1 = y0 - V;
                //Draw vertical stroke
                Line3D l4 = new Line3D(x0, verticalSize - y0,0,
                        x1, verticalSize - y1,0);
                myGroup.getChildren().add(l4.group);
                //Draw horizontal stroke
                Line3D l5 = new Line3D(x1, verticalSize - y1,0,
                        x1 + H, verticalSize - y1,0);
                myGroup.getChildren().add(l5.group);
                center = new Point2D(center.getX()
                        , center.getY() - V);

            } //End of if + else

            center = new Point2D(center.getX() + H
                        , center.getY());

            V = V / 2;
        }//End of for j
    }  //End of for: numberOfLeaves

    //Strings are drawn:
    for (int i = 0; i < 0; i = i + 2)
    {
        //Addresses will be used as paths to the places
        //where strings must be drawn
        path = TreeInfo.get(i);
        String s = TreeInfo.get(i+1);
        //Reinitialization
        Point2D center = new Point2D(x00, y00);
        Point2D target = center;
```

```
        V = scale * verticalLength;
        //Path is interpreted
        for (int j = 0; j < path.length(); j++)
        {
            direction = path.charAt(j);
            if (direction == '0') //go up
            {
                target = new Point2D(target.getX()
                        , target.getY() + V);
            } else // if direction == '1'  =  go down
            {
                target = new Point2D(target.getX()
                        , target.getY() - V);
            }

            target = new Point2D(target.getX() + H
                    , target.getY());

            V = V / 2;
        }//End of for j
        if (print2)
            System.out.println("Target = "
                    + (int) target.getX() + "  "
                + (int) target.getY());
        //String is drawn
        Text text = new Text(s);
        text.setX(target.getX() + 1);

        text.setY(verticalSize - target.getY() - 1);
        text.setFill(Color.RED);

        root.getChildren().add(text);

    }
    return myGroup;
}

}//End of inner class Project3D
```

```
public static void main(String[] args)
{
    launch(args);
}


}//End of program N94  TreeToShape4
```

95 ***Exercise.*** *Run the previous program and play with the code.*

We are preparing the terrain to use a binary bush to create volume. This is possibly a complex task so, let us try to help ourselves by making some stylistic corrections to that previous program that draws a planar bush. Our purpose is to facilitate changes, to make the code evolvable. We opted for this option after too much vain suffering to pursue a useful modification of program N94 TreeToShape4 in order to generate volume.

96 ***The following code shows a planar bush built on 3D lines. Style has been improved to make the code evolvable, i.e., to facilitate upgrading to 3D.***

```
/*
   Program N96  TreeToShape5
   The program presents
   a planar random binary bush
   whose lines are 3D boxes modelled on meshes.
   Stylistic correction to previous program.
   This program is more evolvable.
*/

package Programs;


//Files in the hosting project
import Tools.Axes;
import Tools.Cameras;
import Tools.Line3D;
import Tools.Xform;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
```

```
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Sphere;
import javafx.stage.Stage;



public class TreeToShape5 extends Application
{

    //This is data for a tree in our binary notation:
    //Even entries are the addresses in the tree of the nodes,
    //odd ones are the names or contents.
    private static   List<String> TreeInfo;



    private static final int numberOfGenerations = 5;
    //The terminal  leaves define the complete tree
    private static final int numberOfLeaves
            = (int) Math.pow(2, numberOfGenerations);
    //Printing for debugging
    final static boolean printReadFile = false;
    private static final String title
            =  "3D bush";

    //Binary string that says whether or not  node i in bit i
    //is present
    private static String bitTree = "";

    final Group root = new Group();
    private static final Cameras camera = new Cameras();
    final double cameraDistance = 5450;//450
    //Axes : ctrl + A
    private static final Axes axes = new Axes();
```

```
    final Xform world3DXform = new Xform();
    final Xform bonesXform = new Xform();

    //Generator of pseudo random numbers
    private static final Random r = new Random();


    private static final boolean print = false;

    //Vertical  size of the panel
    private static final int verticalSize =
            (int) (5 * Math.pow(2, numberOfGenerations));
    //Scale must be a power of 2
    private static final int scale = 16;
    //Symbolic initialization
    private static final int centerX = 1;
    private static final int centerY = 1;

    //Relative length of branches of the first U
    private static final int horizontalLength = 3;
    private static final int verticalLength = 8;

    private static Point3D lastPoint;




@Override
public void start(Stage primaryStage)
{

    primaryStage.setTitle(title);
    Scene scene = new Scene(root, 1200, 1000, true);

    scene.setFill(Color.AQUAMARINE);
    primaryStage.setScene(scene);
    primaryStage.show();
    axes.buildAxes();
    axes.handleKeyboard(scene);
```

```
    world3DXform.getChildren().addAll( axes.axesXform);

    camera.handleMouse(scene);
    camera.buildCamera();
    scene.setCamera(camera.camera);

    //TreeInfo is filled in:
    //A binary address to each node of the tree
    //is following by a content
    createTree();
    printTreeInfo();
    //Tree is drawn
    Group parent = new Group(create3dContent(TreeInfo));
        world3DXform.getChildren().addAll(parent);
        root.getChildren().add(world3DXform);


    primaryStage.show();
    play();


}

public void play()
{

}

@Override public void stop()
{

}




private Node create3dContent(List<String> TreeInfo)
{
    Project3D  p =  new Project3D(TreeInfo);
```

```
    return new Group(p);
}









private static void printTreeInfo()
{
    System.out.println("TreeInfo");

    int l = TreeInfo.size();
    for(int i = 0; i < l; i = i+2)
    {
        System.out.println(i/2 + "   " + TreeInfo.get(i)
        + "   " + TreeInfo.get(i+1));
    }

}



//A binary string with 2*numberOfLeaves-1 bits
private static String randomBinaryString()
{
    //A random binary string
    bitTree = "";
    for (int i = 0; i < 2*numberOfLeaves-1; i++)
    {
        Integer j = r.nextInt(2);
        bitTree = bitTree + j.toString();

    }

    if (print)
        System.out.println(bitTree);
    return bitTree;
}
```

```
//Binary string s is decoded into an int
private static int stringToNumber(String s)
{
    int n = 0;
    int l = s.length();

    for(int j=0;j  < l;j++ )
    {
        char c  =   s.charAt(j);
        int  v = Character.getNumericValue(c);
        int q = j+1;
        int number1 = (int) (v * Math.pow(2,l-q));
        n  = n + number1;
    }
    return n;
}


private static int stringToInt(String address)
{
    int output = 0;
    //System.out.print("String = " + address );
    char c  = address.charAt(0);
    if  (c == '1') output = stringToNumber(address);
        else System.out.println("Bad encoding");
    //System.out.println("  int =  " + output);
    return output;
}


//All ancestors of node m are added
private  void  addAncestors(int m)
{
    String address = TreeInfo.get(m);
    int a = address.length();
    //If the address of  a node is 10001,
    //an ancestor has as address a left substring
    //of 0001, say 0, 00 or 000.
    //All of them must be added to TreeInfo
```

```
    for(int i = 0; i < a;  i++)
    {
        String leftSubstring
                = address.substring(0, i);
        leftSubstring = "1" + leftSubstring;
        int k = stringToInt(leftSubstring);
        //Updating of tree information
        if (k < bitTree.length()-1)
            bitTree = bitTree.substring(0, k)
                + "1" + bitTree.substring(k+1);
        else bitTree = bitTree.substring(0, k)
                + "1" ;
    }

}

//All ancestors of node m are deleted
private  void  deleteAncestors(int m)
{
    String address = TreeInfo.get(m);
    int a = address.length();
    //If the address of  a node is 10001,
    //an ancestor has as address a left substring
    //of 0001, say 0, 00 or 000.
    //All of them must be deleted from TreeInfo

    for(int i = 0; i < a;  i++)
    {
        String leftSubstring
                = address.substring(0, i);
        leftSubstring = "1" + leftSubstring;
        int k = stringToInt(leftSubstring);
        //Updating of tree information
        if (k < bitTree.length()-1)
            bitTree = bitTree.substring(0, k)
                + "0" + bitTree.substring(k+1);
        else bitTree = bitTree.substring(0, k)
                + "0" ;
    }
```

```
}

//The tree encoded by binary string b is
//decoded into an array with binary addresses and contents.
//The tree corresponds with the lower
//part of a tree with one more generation.
private  void  createTree(String b )
{


    int l = b.length();
    char c ;
    String s ;
    //First two bits do not matter
    for (Integer i = 2; i < l; i++)
    {
        c = b.charAt(i);
        if (c == '1')
        {
            if ( i < 2*numberOfLeaves-1)
            {
                s = Integer.toBinaryString(i);
                //prefix "1" is deleted
                s = s.substring(1);
                TreeInfo.add(s);
                s = s.replace('0', '+');
                s = s.replace('1', '-');
                TreeInfo.add(s);
            }
        }
    }
}


//The tree encoded by binary string b is
//decoded and drawn.
//The tree corresponds with the lower
//part of a tree with one more generation.
private  void  createTree( )
```

```
{
    int n = numberOfLeaves;
    TreeInfo = new  ArrayList<>(2*n-1);
    //A random binary string:
    //If bit I is 1, the I node is added to the tree
    bitTree = randomBinaryString();
    createTree(bitTree) ;

    /*
    //If a node exists, all its ancestors too
    for(int m = 0; m < TreeInfo.size()-1;   m = m+2)
        addAncestors(m);
    */

    //Simplify tree:
    //If a node exists, all its ancestors are erased
    for(int m = 0; m < TreeInfo.size()-1;   m = m+2)
        deleteAncestors(m);

    //Tree information is re-created from the bitTree
    TreeInfo.clear();
    createTree(bitTree);

}


//3D line that goes from lastPoint
//to thisPoint
private  Line3D  line3DTo(Point3D thisPoint)
{
    Line3D l = new Line3D(lastPoint, thisPoint);
    lastPoint = thisPoint;
    return l;
}

//=========================================================
//======= Beginning of Inner class Project3D ===========
//=========================================================
```

```
//The drawing is synthesized
private class Project3D extends Group
{

    //Constructor
    Project3D(List<String> TreeInfo)
    {
        System.out.println("List<String> TreeInfo");
        //Tree is drawn to expand a 3D bush
        Group g = drawTree(TreeInfo);
        getChildren().addAll(g );
    }

   private   Group drawTree(List<String> TreeInfo)
{
    Group myGroup  = new Group();
    //Draw stem
    int x00 = scale * centerX;
    int y00 = verticalSize - scale * centerY - numberOfLeaves
            * verticalLength/2;
    //Horizontal and basic vertical strokes
    int H = scale * horizontalLength;
    int V;
    int x11 = x00 + H;
    int y11 = y00;
    //The stalk is drawn
    Point3D P0=  new Point3D(x00, y00,0);
    Point3D   P1 = P0.add(H, 0,0);
    Line3D l = new Line3D(P0, P1);
    myGroup.getChildren().add(l.group);
    String path;
    char direction;

    //The tree is drawn:
    //A path from the origin of the tree to
    //each node is drawn
    for (int i = 0; i < TreeInfo.size(); i= i+2)
    {
        //Addresses will be used as paths to  be drawn
```

```
        path = TreeInfo.get(i);


        //Reinitialization
        lastPoint = new Point3D(x11, y11,0);

        V = scale * verticalLength;
        //Put a landmark for the 0th upper branch
        Sphere sphere = new Sphere();
        sphere.setRadius(10);
        sphere.setTranslateX((int) lastPoint.getX());
        sphere.setTranslateY(
            verticalSize - (int) lastPoint.getY() );
        root.getChildren().add(sphere);
        Point3D thisPoint;
        //Path is interpreted
        for (int j = 0; j < path.length(); j++)
        {
            direction = path.charAt(j);
            if (direction == '0') //go up
                thisPoint = lastPoint.add(0,V,0);
            else //if direction == '1'  =  go down
                thisPoint = lastPoint.add(0,-V,0);


            //Draw vertical stroke
            //from lastPoint to thisPoint
            Line3D l2 = line3DTo(thisPoint);
            thisPoint = lastPoint.add(H,0,0);
            //Draw horizontal stroke
            Line3D l3 = line3DTo(thisPoint);
            myGroup.getChildren().addAll(l2.group, l3.group);
            //Separation of minor branches decreases
            V = V / 2;
        }//End of for j
    }  //End of for: numberOfLeaves

    return myGroup;
}
```

```
}//End of inner class Project3D


public static void main(String[] args)
{
    launch(args);
}

}//End of programN96  TreeToShape5
```

97 ***Exercise.*** *Run the previous program and play with the code.*

98 ***To create volume we twist the branches of the planar tree that was shown by the previous program. The inspiring graphic and the code follows.***

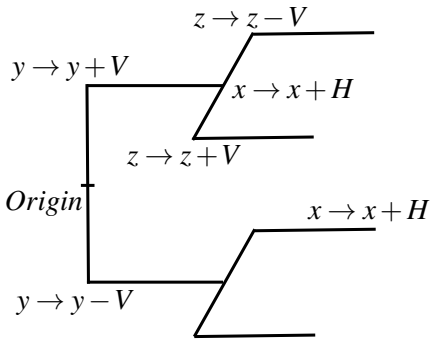


Figure 4.2. To find the coordinates of points in 3D one must add specific values to specific coordinates.

```
/*
   Program N98  TreeToShape6
   Branches of  a planar random binary bush
   are twisted recurrently to create volume.
*/

package Programs;


//Files in the hosting project
import Tools.Axes;
import Tools.Cameras;
```

```
import Tools.Line3D;
import Tools.Xform;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Sphere;
import javafx.stage.Stage;



public class TreeToShape6 extends Application
{

    //This is data for a tree in our binary notation:
    //Even entries are the addresses in the tree of the nodes,
    //odd ones are the names or contents.
    private static   List<String> TreeInfo;



    private static final int numberOfGenerations = 5;
    //The terminal  leaves define the complete tree
    private static final int numberOfLeaves
            = (int) Math.pow(2, numberOfGenerations);
    //Printing for debugging
    final static boolean printReadFile = false;
    private static final String title
            =  "3D bush";

    //Binary string that says whether or not  node i in bit i
    //is present
    private static String bitTree = "";
```

```
    final Group root = new Group();
    private static final Cameras camera = new Cameras();
    final double cameraDistance = 5450;//450
    //Axes : ctrl + A
    private static final Axes axes = new Axes();
    final Xform world3DXform = new Xform();
    final Xform bonesXform = new Xform();

    //Generator of pseudo random numbers
    private static final Random r = new Random();


    private static final boolean print = false;

    //Vertical  size of the panel
    private static final int verticalSize =
            (int) (5 * Math.pow(2, numberOfGenerations));
    //Scale must be a power of 2
    private static final int scale = 16;
    //Symbolic initialization
    private static final int centerX = 1;
    private static final int centerY = 1;

    //Relative length of branches of the first U
    private static final int horizontalLength = 3;
    private static final int verticalLength = 8;

    private static Point3D lastPoint;




@Override
public void start(Stage primaryStage)
{

    primaryStage.setTitle(title);
    Scene scene = new Scene(root, 1200, 1000, true);
```

```
    scene.setFill(Color.AQUAMARINE);
    primaryStage.setScene(scene);
    primaryStage.show();
    axes.buildAxes();
    axes.handleKeyboard(scene);
    world3DXform.getChildren().addAll( axes.axesXform);

    camera.handleMouse(scene);
    camera.buildCamera();
    scene.setCamera(camera.camera);

    //TreeInfo is filled in:
    //A binary address to each node of the tree
    //is following by a content
    createTree();
    printTreeInfo();
    //Tree is drawn
    Group parent = new Group(create3dContent(TreeInfo));
        world3DXform.getChildren().addAll(parent);
        root.getChildren().add(world3DXform);


    primaryStage.show();
    play();


}

public void play()
{

}

@Override public void stop()
{

}
```

```
private Node create3dContent(List<String> TreeInfo)
{
    Project3D  p =  new Project3D(TreeInfo);
    return new Group(p);
}
```

```
private static void printTreeInfo()
{
    System.out.println("TreeInfo");

    int l = TreeInfo.size();
    for(int i = 0; i < l; i = i+2)
    {
        System.out.println(i/2 + "   " + TreeInfo.get(i)
        + "   " + TreeInfo.get(i+1));
    }

}
```

```
//A binary string with 2*numberOfLeaves-1 bits
private static String randomBinaryString()
{
    //A random binary string
    bitTree = "";
    for (int i = 0; i < 2*numberOfLeaves-1; i++)
    {
        Integer j = r.nextInt(2);
        bitTree = bitTree + j.toString();
```

```
    }

    if (print)
        System.out.println(bitTree);
    return bitTree;
}

//Binary string s is decoded into an int
private static int stringToNumber(String s)
{
    int n = 0;
    int l = s.length();

    for(int j=0;j  < l;j++ )
    {
        char c  =   s.charAt(j);
        int  v = Character.getNumericValue(c);
        int q = j+1;
        int number1 = (int) (v * Math.pow(2,l-q));
        n  = n + number1;
    }
    return n;
}


private static int stringToInt(String address)
{
    int output = 0;
    //System.out.print("String = " + address );
    char c  = address.charAt(0);
    if  (c == '1') output = stringToNumber(address);
        else System.out.println("Bad encoding");
    //System.out.println("  int =  " + output);
    return output;
}


//All ancestors of node m are added
private  void  addAncestors(int m)
{
```

```
    String address = TreeInfo.get(m);
    int a = address.length();
    //If the address of  a node is 10001,
    //an ancestor has as address a left substring
    //of 0001, say 0, 00 or 000.
    //All of them must be added to TreeInfo

    for(int i = 0; i < a;  i++)
    {
        String leftSubstring
                = address.substring(0, i);
        leftSubstring = "1" + leftSubstring;
        int k = stringToInt(leftSubstring);
        //Updating of tree information
        if (k < bitTree.length()-1)
            bitTree = bitTree.substring(0, k)
                + "1" + bitTree.substring(k+1);
        else bitTree = bitTree.substring(0, k)
                + "1" ;
    }

}

//All ancestors of node m are deleted
private  void  deleteAncestors(int m)
{
    String address = TreeInfo.get(m);
    int a = address.length();
    //If the address of  a node is 10001,
    //an ancestor has as address a left substring
    //of 0001, say 0, 00 or 000.
    //All of them must be deleted from TreeInfo

    for(int i = 0; i < a;  i++)
    {
        String leftSubstring
                = address.substring(0, i);
        leftSubstring = "1" + leftSubstring;
        int k = stringToInt(leftSubstring);
        //Updating of tree information
```

```
        if (k < bitTree.length()-1)
            bitTree = bitTree.substring(0, k)
                + "0" + bitTree.substring(k+1);
        else bitTree = bitTree.substring(0, k)
                + "0" ;
    }

}

//The tree encoded by binary string b is
//decoded into an array with binary addresses and contents.
//The tree corresponds with the lower
//part of a tree with one more generation.
private  void  createTree(String b )
{


    int l = b.length();
    char c ;
    String s ;
    //First two bits do not matter
    for (Integer i = 2; i < l; i++)
    {
        c = b.charAt(i);
        if (c == '1')
        {
            if ( i < 2*numberOfLeaves-1)
            {
                s = Integer.toBinaryString(i);
                //prefix "1" is deleted
                s = s.substring(1);
                TreeInfo.add(s);
                s = s.replace('0', '+');
                s = s.replace('1', '-');
                TreeInfo.add(s);
            }
        }
    }
}
```

```
//The tree encoded by binary string b is
//decoded and drawn.
//The tree corresponds with the lower
//part of a tree with one more generation.
private  void  createTree( )
{
    int n = numberOfLeaves;
    TreeInfo = new  ArrayList<>(2*n-1);
    //A random binary string:
    //If bit I is 1, the I node is added to the tree
    bitTree = randomBinaryString();
    createTree(bitTree) ;

    /*
    //If a node exists, all its ancestors too
    for(int m = 0; m < TreeInfo.size()-1;   m = m+2)
        addAncestors(m);
    */

    //Simplify tree:
    //If a node exists, all its ancestors are erased
    for(int m = 0; m < TreeInfo.size()-1;   m = m+2)
        deleteAncestors(m);

    //Tree information is re-created from the bitTree
    TreeInfo.clear();
    createTree(bitTree);

}


//Returns a 3D line that goes from lastPoint
//to thisPoint
private  Line3D  line3DTo(Point3D thisPoint)
{
    Line3D l = new Line3D(lastPoint, thisPoint);
    lastPoint = thisPoint;
    return l;
}
```

```
//=======================================================
//======= Beginning of Inner class Project3D ===========
//=======================================================



//The drawing is synthesized
private class Project3D extends Group
{

    //Constructor
    Project3D(List<String> TreeInfo)
    {
        System.out.println("List<String> TreeInfo");
        //Tree is drawn to expand a 3D bush
        Group g = drawTree(TreeInfo);
        getChildren().addAll(g );
    }

   private   Group drawTree(List<String> TreeInfo)
{
    Group myGroup  = new Group();
    //Draw stem
    int x00 = scale * centerX;
    int y00 = verticalSize - scale * centerY - numberOfLeaves
            * verticalLength/2;
    //Horizontal and basic vertical strokes
    int H = scale * horizontalLength;
    int V;
    int x11 = x00 + H;
    int y11 = y00;
    //The stalk is drawn
    Point3D P0=  new Point3D(x00, y00,0);
    Point3D   P1 = P0.add(H, 0,0);
    Line3D l = new Line3D(P0, P1);
    myGroup.getChildren().add(l.group);
    String path;
    char direction;
```

```
//The tree is drawn:
//A path from the origin of the tree to
//each node is drawn
for (int i = 0; i < TreeInfo.size(); i= i+2)
{
    //Addresses will be used as paths to  be drawn
    path = TreeInfo.get(i);


    //Reinitialization
    lastPoint = new Point3D(x11, y11,0);
    V = scale * verticalLength;

    //Put a landmark for the 0th upper branch
    Sphere sphere = new Sphere();
    sphere.setRadius(10);
    sphere.setTranslateX((int) lastPoint.getX());
    sphere.setTranslateY(
        verticalSize - (int) lastPoint.getY() );
    root.getChildren().add(sphere);

    //To create volume,  strokes go
    //parallel to the XY-plane else
    //parallel to the YZ-plane
    boolean xyParallel = true;

    Point3D thisPoint;
    //Path is interpreted
    for (int j = 0; j < path.length(); j++)
    {
        direction = path.charAt(j);

        if (xyParallel)
        {
            if (direction == '0') //go up
                thisPoint = lastPoint.add(0,V,0);
            else //if direction == '1'  =  go down
                thisPoint = lastPoint.add(0,-V,0);
        }
        else
```

```
            {
                if (direction == '0') //go up
                    thisPoint = lastPoint.add(0,0,V);
                else //if direction == '1'  =  go down
                    thisPoint = lastPoint.add(0,0,-V);
            }

            //Draw vertical stroke
            //from lastPoint to thisPoint
            Line3D l2 = line3DTo(thisPoint);
            thisPoint = lastPoint.add(H,0,0);
            //Draw horizontal stroke
            Line3D l3 = line3DTo(thisPoint);
            myGroup.getChildren().addAll(l2.group, l3.group);
            //Separation of minor branches decreases
            V = V / 2;
            xyParallel = !xyParallel;
        }//End of for j
    }  //End of for: numberOfLeaves

    return myGroup;
}

}//End of inner class Project3D


public static void main(String[] args)
{
    launch(args);
}

}//End of program N98  TreeToShape6
```

99 ***Exercise.*** *Run the previous program and play with the code.*

100 ***Exercise.*** *Observe how natural and simple were the changes that were made to program N96 TreeToShape5 for 2D bushes to generate 3D bushes with volume. So, program N96 TreeToShape5 is an example of evolvable software. By contrast, program N94 TreeToShape4 is an example of a program that runs perfectly but that is not evolvable: the Author tried to adapt it to 3D but after too much suffering he*

opted for abandoning it to create a really evolvable version. Explain what makes program TreeToShape5 evolvable and what makes program N94 TreeToShape4 not evolvable. Answer

101 ***Exercise.*** *The previous code draws a bush in which a new stroke forms a right angle with its mother but the whole tree is highly symmetric. We will develop the code for a more chaotic tree, one more suitable for evolution, in three steps. Step one: Draw a random 3D self intersecting rectangular bundle of paths. Rectangular means that a new stroke can be added just straightforward else along any one of the 4 perpendicular possibilities dictated by the natural axes.* Answer

102 ***Exercise.*** *Reuse the previous program to develop the code for a self-intersecting rectangular random binary tree whose arms extends randomly in 3D. Step two of three.* Answer

To produce non self intersecting trees, we must learn to detect self-intersections.

103 ***The following code unveils self intersections of our trees.***

```
/*
    Program N103  TreeToShape9
    Self intersections are unveiled in 3D trees.

    When we think of a non-self-intersecting tree,
    we have in mind the  property
    that two cells
    cannot occupy the same space. So,
    we fill the tree with all ancestors of each node
    and we demand from any given position in space
    to be filled by just one node.

*/

package Programs;


//Files in the hosting project
import Tools.Axes;
import Tools.Cameras;
import Tools.Line3D;
import Tools.Xform;
```

```
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Sphere;
import javafx.stage.Stage;



public class TreeToShape9 extends Application
{

    private static final int numberOfGenerations = 5;
    //The terminal  leaves define the complete tree
    private static final int numberOfLeaves
            = (int) Math.pow(2, numberOfGenerations);

    //This is data for a tree in our binary notation:
    //Even entries are the addresses
    //of the nodes in the tree,
    //odd ones are the names or contents.
    //An address could be like this 100010,
    //it gives the structure of ramification.
    //A content could be any string with a message.
    private static   List<String> TreeInfo
            = new  ArrayList<>(2*numberOfLeaves-1);

    //List to keep the occupied places in 3D space
    private static   List<Point3D> Occupancy
            = new  ArrayList<>(2*numberOfLeaves-1);

    /*
    There are 6 directions for a rectangular tree:
```

```
0: +X, 1 = -X,  2 = +Y,  3 = -Y
4 = +Z, 5 = -Z
*/
private static int direction;

//Each node is localized in space thanks
//to a string of directions.
private static final   List<String> Directions
        = new  ArrayList<>(2*numberOfLeaves-1);

//Printing for debugging
final static boolean printReadFile = false;
private static final String title
        =  "3D bush";

//Binary string that says whether or not  node i in bit i
//is present
private static String bitTree = "";

final Group root = new Group();
private static final Cameras camera = new Cameras();
final double cameraDistance = 5450;//450
//Axes : ctrl + A
private static final Axes axes = new Axes();
final Xform world3DXform = new Xform();
final Xform bonesXform = new Xform();

//Generator of pseudo random numbers
private static final Random r = new Random();




//Vertical  size of the panel
private static final int verticalSize =
        (int) (5 * Math.pow(2, numberOfGenerations));
//Scale must be a power of 2
private static final int scale = 16;
//Symbolic initialization
private static final int centerX = 1;
```

```
    private static final int centerY = 1;

    //Relative length of branches of the first U
    private static final int horizontalLength = 3;
    private static final int verticalLength = 8;

    //Hidden argument of method lineTo(thisPoint)
    //to call a line creator method:
    //Line3D(LastPoint, thisPoint)
    private static Point3D lastPoint;

    //Length of strokes in 3D space
    private static final int delta = 50;

    private static final boolean print = false;
    private static final boolean print1 = true;
    private static final boolean print2 = false;
    private static final boolean print3 = false;




@Override
public void start(Stage primaryStage)
{

    primaryStage.setTitle(title);
    Scene scene = new Scene(root, 1200, 1000, true);

    scene.setFill(Color.AQUAMARINE);
    primaryStage.setScene(scene);
    primaryStage.show();
    axes.buildAxes();
    axes.handleKeyboard(scene);
    world3DXform.getChildren().addAll( axes.axesXform);

    camera.handleMouse(scene);
    camera.buildCamera();
```

```
    scene.setCamera(camera.camera);

    //TreeInfo is filled in:
    //A binary address to each node of the tree
    //is following by a content
    createTree();

    //Tree is drawn
    Group parent = new Group(create3dContent(TreeInfo));
        world3DXform.getChildren().addAll(parent);
        root.getChildren().add(world3DXform);

   if (print1) printInfo();
    primaryStage.show();
    play();


}

public void play()
{

}

@Override public void stop()
{

}




private Node create3dContent(List<String> TreeInfo)
{
    Project3D  p =  new Project3D(TreeInfo);
    return new Group(p);
}
```

```
private static void printInfo()
{
    System.out.println("TreeInfo");

    int l = TreeInfo.size();
    for(int i = 0; i < l; i = i+2)
    {
        System.out.println(i/2 + "   " + TreeInfo.get(i)
                + "   " + TreeInfo.get(i+1));
    }
    l = Occupancy.size();
    System.out.println("Occupancy");
    for(int i = 0; i < l; i++)
    {
        System.out.println(i  + "   "
                + Occupancy.get(i).toString());
        if (i > 0)
            for(int j = 0; j < i; j++)
            {
                if (Occupancy.get(i).equals(Occupancy.get(j)))
                    System.out.println(i
                            + " Repeated with " + j);
            }
    }

}


//A binary string with 2*numberOfLeaves-1 bits
private static String randomBinaryString()
{
    //A random binary string
    bitTree = "";
    for (int i = 0; i < 2*numberOfLeaves-1; i++)
    {
        Integer j = r.nextInt(2);
        bitTree = bitTree + j.toString();

    }
```

```
    if (print)
        System.out.println(bitTree);
    return bitTree;
}

//Binary string s is decoded into an int
private static int stringToNumber(String s)
{
    int n = 0;
    int l = s.length();

    for(int j=0;j  < l;j++ )
    {
        char c  =   s.charAt(j);
        int  v = Character.getNumericValue(c);
        int q = j+1;
        int number1 = (int) (v * Math.pow(2,l-q));
        n  = n + number1;
    }
    return n;
}


private static int stringToInt(String address)
{
    int output = 0;
    //System.out.print("String = " + address );
    char c  = address.charAt(0);
    if  (c == '1') output = stringToNumber(address);
        else System.out.println("Bad encoding");
    //System.out.println("  int =  " + output);
    return output;
}


//All ancestors of node m are added
private  void  addAncestors(int m)
{
    String address = TreeInfo.get(m);
```

```
    int a = address.length();
    //If the address of  a node is 10001,
    //an ancestor has as address a left substring
    //of 0001, say 0, 00 or 000.
    //All of them must be added to TreeInfo

    for(int i = 0; i < a;  i++)
    {
        String leftSubstring
                = address.substring(0, i);
        leftSubstring = "1" + leftSubstring;
        int k = stringToInt(leftSubstring);
        //Updating of tree information
        if (k < bitTree.length()-1)
            bitTree = bitTree.substring(0, k)
                + "1" + bitTree.substring(k+1);
        else bitTree = bitTree.substring(0, k)
                + "1" ;
    }

}

//All ancestors of node m are deleted
private  void  deleteAncestors(int m)
{
    String address = TreeInfo.get(m);
    int a = address.length();
    //If the address of  a node is 10001,
    //an ancestor has as address a left substring
    //of 0001, say 0, 00 or 000.
    //All of them must be deleted from TreeInfo

    for(int i = 0; i < a;  i++)
    {
        String leftSubstring
                = address.substring(0, i);
        leftSubstring = "1" + leftSubstring;
        int k = stringToInt(leftSubstring);
        //Updating of tree information
        if (k < bitTree.length()-1)
```

```
            bitTree = bitTree.substring(0, k)
                + "0" + bitTree.substring(k+1);
        else bitTree = bitTree.substring(0, k)
                + "0" ;
    }

}

//The tree encoded by binary string b is
//decoded into an array with binary addresses and contents.
//The tree corresponds with the lower
//part of a tree with one more generation.
private  void  createTree(String b )
{


    int l = b.length();
    char c ;
    String s ;
    //First two bits do not matter
    for (Integer i = 2; i < l; i++)
    {
        c = b.charAt(i);
        if (c == '1')
        {
            if ( i < 2*numberOfLeaves)
            {
                s = Integer.toBinaryString(i);
                //prefix "1" is deleted
                s = s.substring(1);
                TreeInfo.add(s);
                s = s.replace('0', '+');
                s = s.replace('1', '-');
                TreeInfo.add(s);
            }
        }
    }
}
```

```
//The tree encoded by binary string b is
//decoded and drawn.
//The tree corresponds with the lower
//part of a tree with one more generation.
private  void  createTree( )
{
    int n = numberOfLeaves;
    TreeInfo = new  ArrayList<>(2*n-1);
    //A random binary string:
    //If bit I is 1, the I node is added to the tree
    bitTree = randomBinaryString();
    createTree(bitTree) ;


    //If a node exists, all its ancestors too
    for(int m = 0; m < TreeInfo.size()-1;   m = m+2)
        addAncestors(m);

    /*
    //Simplify tree:
    //If a node exists, all its ancestors are erased
    for(int m = 0; m < TreeInfo.size()-1;   m = m+2)
        deleteAncestors(m);

    //Tree information is re-created from the bitTree
    TreeInfo.clear();
    /*
    //Test: full tree
    bitTree = "";
    for (int i = 0; i < 2*numberOfLeaves; i++)
    {

        bitTree = bitTree + "1";

    }
    */
    TreeInfo.clear();
    createTree(bitTree);

}
```

```
//Returns a 3D line that goes from lastPoint
//to thisPoint
private  Line3D  line3DTo(Point3D thisPoint)
{
    Line3D l = new Line3D(lastPoint, thisPoint);
    lastPoint = thisPoint;
    return l;
}


//=========================================================
//======= Beginning of Inner class Project3D ===========
//=========================================================



//The drawing is synthesized
private class Project3D extends Group
{

//Constructor
Project3D(List<String> TreeInfo)
{
    //Tree is drawn to expand a 3D bush
    Group g = drawTree(TreeInfo);
    getChildren().addAll(g );
}


/*
There are 6 directions:
0: +X, 1 = -X,  2 = +Y,  3 = -Y
4 = +Z, 5 = -Z.
One distinct than d is chosen
*/
private int distinctDirection(int d)
{
    int i = r.nextInt(6);
```

```
    //System.out.println(i + "  " + d);
    if (i == d) distinctDirection(d);
    return i;
}



private  Point3D  move3D(int d)
{

    Point3D p = new Point3D(0,0,0);
    switch (d)
    {
        case 0: p = lastPoint.add( delta,0,0); break;
        case 1: p = lastPoint.add(-delta,0,0); break;
        case 2: p = lastPoint.add(0, delta,0); break;
        case 3: p = lastPoint.add(0,-delta,0); break;
        case 4: p = lastPoint.add(0,0, delta); break;
        case 5: p = lastPoint.add(0,0,-delta); break;
    }
    return p;
}

//Common initial substring to strings s and t
private  String commonSubstring(String s, String t)
{
    int a = s.length();
    int b = t.length();
    int min = a;
    if (b < a ) min = b;
    String c = "";
    for(int i  = 0; i < min; i++)
    {
        if (s.charAt(i) == t.charAt(i))
            c = c + s.charAt(i);
        else break;
    }
    return c;
}
```

```
//TreeInfo in the tree
//gives the ramification structure but not
//the 3D presentation which arises from
//3D directions chosen at random   but
//if two addresses have an initial common segment,
//so do their initial 3D subpaths.
//Example: 00110 and 00101 have in common 001
//so, their 3D paths share
//an initial subpath with 3 strokes
private  void harmonizeTree()
{

    for (int i = Directions.size()-2;
            i > -1; i= i-2)
    {
        String s = TreeInfo.get(i);
        for (int j = i-2;   j > -1; j = j-2)
        {
            String t = TreeInfo.get(j);

            String common = commonSubstring(s,t);

            int l = common.length();
            if (l > 0)
            {

                int m = t.length();
                String ds = Directions.get(i);
                String dt = Directions.get(j);
                if (print2)
                    System.out.print(s + "  " + t + "  "
                            + common + "  "
                            + ds + "   " + dt);

                String prefix = ds.substring(0, l);
                String insertion1 = "";
                String insertion2 = "";
                String suffix = "";

                if (m > l)
```

```
                {
                    insertion2 = dt.substring(l, l+1);
                    if (m > l+1) suffix = dt.substring(l+1);

                    if ( ds.charAt(l) == dt.charAt(l))
                    {
                        insertion2 = ""
                            + distinctDirection(t.charAt(l));
                    }
                }

                dt = prefix + insertion1
                        + insertion2 + suffix;

                if (print2)
                    System.out.println( "   " + dt);

                Directions.set(j, dt);
            }
        }

    }
}



//Information is created to realize tree into 3D space.
private  void treeTo3D()
{
    //Directions are concatenated at random
    for (int i = 0; i < TreeInfo.size(); i= i+2)
    {
        String s = "";
        for (int j = 0; j < TreeInfo.get(i).length(); j++)
        {
            //A new direction is chosen
            direction = distinctDirection(direction);
            s = s + direction;
        }//End of for j
        Directions.add(s);
```

```
        Directions.add(s);
    }
    //The tree structure is recovered
    harmonizeTree();

}


   private   Group drawTree(List<String> TreeInfo)
{
    Group myGroup  = new Group();
    //Draw stem
    int x00 = scale * centerX;
    int y00 = verticalSize - scale * centerY - numberOfLeaves
            * verticalLength/2;
    //Horizontal and basic vertical strokes
    int H = scale * horizontalLength;


    //The stalk is drawn
    Point3D P0=  new Point3D(x00, y00,0);
    Point3D   P1 = P0.add(H, 0,0); //+X direction
    /*
    There are 6 directions:
    0: +X, 1 = -X,  2 = +Y,  3 = -Y
    4 = +Z, 5 = -Z
    */
    direction = r.nextInt(6);




    //Put a landmark at the root
    Sphere sphere = new Sphere();
    sphere.setRadius(10);
    sphere.setTranslateX((int) P1.getX());
    sphere.setTranslateY((int) P1.getY() );
    sphere.setTranslateZ((int) P1.getZ() );
    root.getChildren().add(sphere);
```

```
    //Information is created to realize tree into 3D space
    treeTo3D();
    //The tree is drawn:
    for (int i = 0; i < TreeInfo.size(); i= i+2)
    {

        //Reinitialization
        lastPoint = P1;
        Point3D thisPoint = P1;
        String d = Directions.get(i);
        if (print3)
            System.out.println( "Path = " + d);
        int length = d.length();

        for (int j = 0; j < length; j++)
        {
            //Direction at j is read
            direction = Integer.parseInt("" + d.charAt(j));
            if (print3)
                System.out.println(direction + "   ");
            //thisPoint moves  along the chosen direction
            thisPoint = move3D(direction);
            //Draw a stroke
            //from lastPoint to thisPoint
            Line3D l2 = line3DTo(thisPoint);
            myGroup.getChildren().add(l2.group);
        }//End of for j
        Occupancy.add(thisPoint);

    }  //End of for: numberOfLeaves

    return myGroup;
}

}//End of inner class Project3D


public static void main(String[] args)
{
    launch(args);
```

```
}

}//End of program N103  TreeToShape9
```

104 ***Exercise.*** *Run the previous program and play with the code.*

105 ***Exercise.*** *Develop the code for a non-self-intersecting rectangular random binary tree whose arms extends randomly in 3D. Step three of three.* Answer

106 ***Our plan.*** *We know now how to devise a random binary bush whose arms extends in 3D space at random. This enables us to simulate evolution as follows: we will program evolution to devise a binary 3D bush that fits into a cube. If this task is easy for evolution, we will infer that biological shape is an evolutionary fluid. But if this problem is difficult for evolution, we will claim that biological shape is by its very nature evolution-proof. In the first case, everyone is allowed to think that evolution is a candidate to be the explanation of his or her corporeal form. To make a complete proof of that belief, it remains to explain the stability of biological shape when it is not shielded by selection. In the second case, everyone is invited to work further to show that there is no reason to consider evolution as our creator. This exclusion is mandatory because a problem is difficult when it leaves too many tracks along its path towards the final and perfect solution. Therefore, were evolution our creator and the problem of shape formation difficult, the fossil record should have been filled in every kind of half cooked products. But these are not found so, if the shape problem is difficult for evolution, the Evolutionary Theory would be mortally falsified. We need some few steps to implement our plan. The first is to test how much randomness can give of itself.*

107 ***Can randomness explain the unitary cube?*** *The following program allows us to tackle this question: Can a random binary tree whose arms extends randomly in 3D explain the unitary cube?*

```
/*
    Program N107  TreeToShapes11
    A unitary cube is fitted by a random
    3D binary tree.

*/

package Programs;
```

```
//Files in the hosting project
import Tools.Axes;
import Tools.Cameras;
import Tools.Line3D;
import Tools.Xform;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Sphere;
import javafx.stage.Stage;



public class TreeToShapes11 extends Application
{

    private static int numberOfGenerations;
    //The terminal  leaves define the complete tree
    private static int numberOfLeaves;

    //This is data for a tree in our binary notation:
    //Even entries are the addresses
    //of the nodes in the tree,
    //odd ones are the names or contents.
    //An address could be like this 100010,
    //it gives the structure of ramification.
    //A content could be any string with a message.
    private static List<String> TreeInfo;

    //List to keep the occupied places in 3D space
    private static List<Point3D> Occupancy;
```

```
/*
There are 6 directions for a rectangular tree:
0: +X, 1 = -X,  2 = +Y,  3 = -Y
4 = +Z, 5 = -Z
*/
private static int direction;

//Array with  strings of directions that define a path
//to localize each node in 3D space.
private static List<String> Paths3D;

//Printing for debugging
final static boolean printReadFile = false;
private static final String title
        =  "3D binary bush";

//Binary string that says whether or not  node i
//is present (bit i = 1)
private static String bitTree = "";

//Root of the   tree of the JavaFX application
final Group root = new Group();
private static final Cameras camera = new Cameras();
final double cameraDistance = 5450;//450
//Axes : ctrl + A
private static final Axes axes = new Axes();
final Xform world3DXform = new Xform();
final Xform bonesXform = new Xform();

//Generator of pseudo random numbers
private static final Random r = new Random();


//Hidden argument of method lineTo(thisPoint)
//to call a line creator method:
//Line3D(LastPoint, thisPoint)
private static Point3D lastPoint;
private static final Point3D root3D
        = new Point3D(0,0,0);
```

```
    //Length of strokes in 3D space
    private static final int delta = 50;

    //Boolean printing flags
    private static final boolean print = false;
    private static final boolean printInfo = false;
    private static final boolean printIsOccupied = false;
    private static final boolean printGetAncestralPath
            = false;
    private static final boolean printTryToGetPath = false;
    private static final boolean printDrawTree = false;
    private static final boolean printTreeTo3D = false;
    private static final boolean printIsPathRepeated
            = false;
    //Maximal number of trials to find
    //a path that is not repeated and that ends
    // in a point  that is not occupied
    private static final int limit = 50;
    //Counter of trials to find a path
    private static  int counterOfTrials = 0;
    //Counter of trials to fit the unitary cube
    private static  int numberOfTrialsCube = 0;




//Fundamental parameters are given a value
private void initialization()
{
    numberOfGenerations = r.nextInt(5);
    //The terminal  leaves define the complete tree
    numberOfLeaves
            = (int) Math.pow(2, numberOfGenerations);

    //This is data for a tree in our binary notation:
    //Even entries are the addresses
    //of the nodes in the tree,
    //odd ones are the names or contents.
    //An address could be like this 100010,
```

```
    //it gives the structure of ramification.
    //A content could be any string with a message.
    TreeInfo  = new  ArrayList<>(2*numberOfLeaves-1);


    //List to keep the occupied places in 3D space
    Occupancy = new  ArrayList<>(2*numberOfLeaves-1);
    Paths3D  = new  ArrayList<>(2*numberOfLeaves-1);
}


@Override
public void start(Stage primaryStage)
{


    primaryStage.setTitle(title);
    Scene scene = new Scene(root, 1200, 1000, true);

    scene.setFill(Color.AQUAMARINE);
    primaryStage.setScene(scene);
    primaryStage.show();
    axes.buildAxes();
    axes.handleKeyboard(scene);
    world3DXform.getChildren().addAll( axes.axesXform);

    camera.handleMouse(scene);
    camera.buildCamera();
    scene.setCamera(camera.camera);



    //Tree is drawn
    Group parent = new Group(create3dContent());
        world3DXform.getChildren().addAll(parent);
        root.getChildren().add(world3DXform);

    if (printInfo) printInfo();
    primaryStage.show();
    play();
```

```
}

public void play()
{

}

@Override public void stop()
{

}



//Master of 3D graphics
private Node create3dContent( )
{
    Project3D  p =  new Project3D( );
    return new Group(p);
}



private static void printAddresses()
{
    System.out.println("Addresses");

    int l = TreeInfo.size();
    for(int i = 0; i < l; i = i+2)
    {
        System.out.println(i/2 + "   " + TreeInfo.get(i)
                + "   " + TreeInfo.get(i+1));
    }
}


private static void printOccupancy()
```

```
{
    System.out.println("Record of occupancy");

    int l = Occupancy.size();
    System.out.println("Occupancy");
    for(int i = 0; i < l; i++)
    {
        System.out.println(i
                + "Address = " + TreeInfo.get(2*i)
                + "   Path = " + Paths3D.get(i)
                + "   point =    "
                + Occupancy.get(i).toString());
        if (i > 0)
            for(int j = 0; j < i; j++)
            {
                if (Occupancy.get(i)
                        .equals(Occupancy.get(j)))
                    System.out.println(i
                            + " Repeated with " + j);
            }
    }
}


private static void printInfo()
{
    printAddresses();
    printOccupancy();

}


//A binary string with 2*numberOfLeaves-1 bits
private static String randomBinaryString()
{
    //A random binary string
    bitTree = "";
    for (int i = 0; i < 2*numberOfLeaves-1; i++)
    {
        Integer j = r.nextInt(2);
```

```
        bitTree = bitTree + j.toString();

    }

    if (print)
        System.out.println(bitTree);
    return bitTree;
}

//Binary string s is decoded into an int
private static int stringToNumber(String s)
{
    int n = 0;
    int l = s.length();

    for(int j=0;j  < l;j++ )
    {
        char c  =   s.charAt(j);
        int  v = Character.getNumericValue(c);
        int q = j+1;
        int number1 = (int) (v * Math.pow(2,l-q));
        n  = n + number1;
    }
    return n;
}


//String address is interpreted as a binary number
//that corresponds with an int
private static int stringToInt(String address)
{
    int output = 0;
    //System.out.print("String = " + address );
    char c  = address.charAt(0);
    if  (c == '1') output = stringToNumber(address);
        else System.out.println("Bad encoding");
    //System.out.println("  int =  " + output);
    return output;
}
```

```
//All ancestors of node m are added
private  void  addAncestors(int m)
{
    String address = TreeInfo.get(m);
    int a = address.length();
    //If the address of  a node is 10001,
    //an ancestor has as address a left substring
    //of 0001, say 0, 00 or 000.
    //All of them must be added to TreeInfo

    for(int i = 0; i < a;  i++)
    {
        String leftSubstring
                = address.substring(0, i);
        leftSubstring = "1" + leftSubstring;
        int k = stringToInt(leftSubstring);
        //Updating of tree information
        if (k < bitTree.length()-1)
            bitTree = bitTree.substring(0, k)
                + "1" + bitTree.substring(k+1);
        else bitTree = bitTree.substring(0, k)
                + "1" ;
    }

}



//The tree encoded by binary string b is
//decoded into an array with binary addresses and contents.
//The tree corresponds with the lower
//part of a tree with one more generation.
private  void  createTree(String b )
{


    int l = b.length();
    char c ;
    String s ;
```

```
    //First two bits do not matter
    for (Integer i = 2; i < l; i++)
    {
        c = b.charAt(i);
        if (c == '1')
        {
            if ( i < 2*numberOfLeaves)
            {
                s = Integer.toBinaryString(i);
                //prefix "1" is deleted
                s = s.substring(1);
                TreeInfo.add(s);
                s = s.replace('0', '+');
                s = s.replace('1', '-');
                TreeInfo.add(s);
            }
        }
    }
}


//The tree encoded by binary string b is
//decoded and drawn.
//The tree corresponds with the lower
//part of a tree with one more generation.
private  void  createTree( )
{


    //A random binary string:
    //If bit I is 1, the I node is added to the tree
    bitTree = randomBinaryString();
    createTree(bitTree) ;


    //If a node exists, all its ancestors too.
    //bitTree will be modified in consequence.
    for(int m = 0; m < TreeInfo.size()-1;   m = m+2)
        addAncestors(m);
```

```
    TreeInfo.clear();
    createTree(bitTree);
}




//=========================================================
//======= Beginning of Inner class Project3D ===========
//=========================================================



//The drawing is synthesized
private class Project3D extends Group
{

//Constructor
Project3D()
{
    //Tree is drawn to expand a 3D bush
    Group g = makeTree( );
    getChildren().addAll(g );
}

//Returns a 3D line that goes from lastPoint
//to thisPoint
private  Line3D  line3DTo(Point3D thisPoint)
{
    Line3D l = new Line3D(lastPoint, thisPoint);
    lastPoint = thisPoint;
    return l;
}

/*
There are 6 directions given by
//the natural basis.
One distinct than d is chosen
*/
```

```
private int distinctDirection(int d)
{
    int i = r.nextInt(6);
    //System.out.println(i + "  " + d);
    if (i == d) distinctDirection(d);
    return i;
}


//A direction distinct than  backwards is chosen
private int notBackwards(int d)
{
    int backwards  = 0;
    switch (d)
    {
        case 0: backwards = 1; break;
        case 1: backwards = 0 ; break;
        case 2: backwards = 3; break;
        case 3: backwards = 2; break;
        case 4: backwards = 5; break;
        case 5: backwards = 4; break;
    }
    int i = distinctDirection(backwards);
    return i;
}

//Direction d is realized as a move
//There are 6 directions:
//0: +X, 1 = -X,  2 = +Y,  3 = -Y
//4 = +Z, 5 = -Z.
private  Point3D  move3D(int d)
{

    Point3D p = new Point3D(0,0,0);
    switch (d)
    {
        case 0: p = new Point3D( delta,0,0); break;
        case 1: p = new Point3D(-delta,0,0); break;
        case 2: p = new Point3D(0, delta,0); break;
        case 3: p = new Point3D(0,-delta,0); break;
```

```
        case 4: p = new Point3D(0,0, delta); break;
        case 5: p = new Point3D(0,0,-delta); break;
    }
    return p;
}

//Common initial substring to strings s and t
private  String commonSubstring(String s, String t)
{
    int a = s.length();
    int b = t.length();
    int min = a;
    if (b < a ) min = b;
    String c = "";
    for(int i  = 0; i < min; i++)
    {
        if (s.charAt(i) == t.charAt(i))
            c = c + s.charAt(i);
        else break;
    }
    return c;
}


//Geometric barriers are defined here
private  boolean isForbidden(Point3D thisPoint)
{
    boolean b = false;
    double x = thisPoint.getX();
    double y = thisPoint.getY();
    double z = thisPoint.getZ();
    //Codification of the unitary cube of edge 50
    if ((x < 0) || (x > 90)) b = true;
    if ((y < 0) || (y > 90)) b = true;
    if ((z < 0) || (z > 90)) b = true;
    return b;
}
```

```
//Is thisPoint already occupied?
private  boolean isOccupied(Point3D thisPoint)
{
    //printOccupancy();
    int l = Occupancy.size();
    boolean occupied = false;
    if (printIsOccupied)
        System.out.println("isOccupied?");
    if (isForbidden(thisPoint) )
    {
        if (printIsOccupied)
            System.out.println("Forbidden" + thisPoint);
        return true;
    }
    if (root3D.equals(thisPoint))
        {
            return true;
        }
    else
    {
        for(int i = 0; i < l; i++)
        {
            if (Occupancy.get(i).equals(thisPoint))
            {
                occupied = true;
                break;
            }
        }
        return occupied;
    }
}


//Is thisPoint already occupied?
private  boolean isPathRepeated(String path)
{
    int l = Paths3D.size();
    boolean repeated = false;
    if (printIsPathRepeated)
```

```
        System.out.println("isPathRepeated?");
    for(int i = 0; i < l; i++)
    {
        if (Paths3D.get(i).equals(path))
        {
            repeated = true;
            break;
        }
    }
    return repeated;
}


//A node has a path that must coincide with
//the paths of its ancestors.
//So, we look for the longest one
private String getAncestralPath(int i)
{
    String addressi = TreeInfo.get(i);

    String ancestralPath = "";
    String candidatePath;
    int f = Paths3D.size();
    List<String> Candidates
            = new  ArrayList<>(2*numberOfLeaves-1);
    for (int j = f-1; j >= 0; j--)
    {
        String addressj = TreeInfo.get(2*j);
        if (printGetAncestralPath)
            System.out.print(" addressi =  " + addressi
                + " addressj = " + addressj);
        String commonAddress
                = commonSubstring(addressi, addressj);


        int k = commonAddress.length();
        if (k == 0)
        {
            if (printGetAncestralPath)
                System.out.println(" No common subaddress ");
```

```
        }
        else
        {
            String path = Paths3D.get(j);
            candidatePath = path.substring(0, k);
            Candidates.add(candidatePath);
            if (printGetAncestralPath)
                System.out.println(" Common address = "
                    + commonAddress
                    + " Pathj = " + path
                    + "  candidate path = "
                    + candidatePath);

        }

    }
    int l = Candidates.size();
    int max = 0;
    for(int k = 0; k < l; k++)
    {
        String s = Candidates.get(k);
        int p = s.length();
        if (p > max)
        {

            max = p;
            ancestralPath = s;
        }
    }
    if (printGetAncestralPath)
                System.out.println(" ancestral path = "
                    + ancestralPath);
    return ancestralPath;
}


//Node i in the list TreeInfo
//needs an exclusive location in 3D.
//If possible, one is found.
private  String tryToGetPath(int i)
```

```
{
    //This is the root of the tree
    Point3D   P1 = new Point3D(0, 0,0);
    //Reinitialization
    lastPoint = P1;
    Point3D thisPoint = P1;
    //A child must share subpaths with their ancestors

    String address = TreeInfo.get(i);
    int l = address.length();

    String ancestralPath = getAncestralPath(i);
    String path = ancestralPath;
    int lengthPath = path.length();
    for (int j = 0; j < lengthPath; j++)
    {
        //Direction at j is read
        direction = Integer.parseInt("" + path.charAt(j));

        //thisPoint moves  along the chosen direction
        thisPoint = thisPoint.add(move3D(direction));

    }

    //New directions are added to complete path
    if (counterOfTrials < limit) //Overflow is interdicted.
    {
        //Directions are concatenated
        for (int j = lengthPath; j < l; j++)
        {
            //A non backwards new direction is chosen
            direction = notBackwards(direction);
            path = path + direction;
            //Chosen direction is executed in memory
            thisPoint = thisPoint.add( move3D(direction));
        }

        if (printTryToGetPath)
            System.out.println("\nAddress to try = "
                + address
```

```
                 + "   Ancetralpath = " + ancestralPath
                + " counterOfTrials = "
                + counterOfTrials
                +   " \nThisPoint = " + thisPoint);

        if ( (isPathRepeated(path))
                || (isOccupied(thisPoint)) )
        {
            if (printTryToGetPath)
                System.out.println("    Path is repeated.");
            counterOfTrials++;
            tryToGetPath(i);
        }
        else
        {
            if (printTryToGetPath)
                System.out.println(
                        "    Path is NOT repeated.");
            Occupancy.add(thisPoint);
            Paths3D.add(path);
        }
    }
    if (counterOfTrials == limit)
    {
        if (printTryToGetPath)
            System.out.println(
                    "    Path not found at trial "
                + counterOfTrials);
        path =  "-1";
    }

    return path;
}


//Information is created to realize
//and draw tree into 3D space.
private  void treeTo3D()
{
    String path ;
```

```
    //Directions are proposed at random,
    //and concatenated if no double occupancy appears
    for (int i = 0; i < TreeInfo.size(); i= i+2)
    {
        if (printTreeTo3D)
            System.out.print("Node i = " + i
                + "  Address = " + TreeInfo.get(i));
        counterOfTrials = 0;
        path = tryToGetPath(i);
        if (printTreeTo3D)
            System.out.println("  Path at treeTo3D = "
            + path);
        if (path.contentEquals("-1"))
        {
            if (printTreeTo3D)
                System.out.println("Truncation");
        }
    }
    if (printTreeTo3D)
        System.out.println("Paths3D size = "
                + Paths3D.size());
    if (printTreeTo3D) printOccupancy();
}



private   Group drawTree( )
{
    Group myGroup  = new Group();
    //This is the root of the tree
    Point3D   P1 = new Point3D(0, 0,0); //+X direction


    //Put a landmark at the root
    Sphere sphere = new Sphere();
    sphere.setRadius(10);
    sphere.setTranslateX((int) P1.getX());
    sphere.setTranslateY((int) P1.getY() );
```

```
    sphere.setTranslateZ((int) P1.getZ() );
    root.getChildren().add(sphere);

    //Information is created to realize
    //and draw tree into 3D space
    treeTo3D();
    Occupancy.clear();
    //The tree is drawn:
    for (String Paths3D1 : Paths3D)
    {
        //Reinitialization
        lastPoint = P1;
        Point3D thisPoint = P1;
        String path = Paths3D1;
        if (printDrawTree)
            System.out.println( "Path = " + path);
        int length = path.length();
        for (int j = 0; j < length; j++)
        {
            //Direction at j is read
            direction = Integer.parseInt("" + path.charAt(j));
            if (printDrawTree)
                System.out.print(direction + "   ");
            //thisPoint moves  along the chosen direction
            thisPoint = thisPoint.add(move3D(direction));
            //Draw a stroke
            //from lastPoint to thisPoint
            Line3D l = line3DTo(thisPoint);
            myGroup.getChildren().add(l.group);
        }
        if (printDrawTree)
            System.out.println();
        Occupancy.add(thisPoint);
    } //End of for: numberOfLeaves
    return myGroup;
}

private   Group makeTree( )
{
```

```
    Group myGroup;
    numberOfTrialsCube++;
    initialization();
    //TreeInfo is filled in:
    //A binary string is composed:
    //if bit i is 1, node i is added to the tree.
    //A binary address to each node of the tree
    //is following by a content in TrreInfo
    createTree();
    //Information is created to realize
    //and draw tree into 3D space
    treeTo3D();

    int n = Paths3D.size();
    //A fitting tree has 7 edges
    if (n < 7)
    {
        TreeInfo.clear();
        Occupancy.clear();
        Paths3D.clear();
        makeTree();
    }
    else
    {
        System.out.println("Number of trials to fit "
            + "the unitary cube = " + numberOfTrialsCube );
    }
    myGroup  = drawTree( );
    return myGroup;
}

}//End of inner class Project3D



public static void main(String[] args)
{
    launch(args);
}
```

```
}//End of program N107  TreeToShapes11
```

108 ***Exercise.*** *Run the previous program and play with the code. Play enough to agree else disagree with the conclusion of the Author: the fitting of a unitary cube is for randomness an extremely easy task.*

109 ***Exercise.*** *Develop the code to study the power of randomness to create a tree whose nodes populate the corners and lateral integral points of a cube of size 2. Decide whether or not a cubic shape of edge 2 is a realizable task for randomness* *Answer*

We have witnessed in the previous exercise that randomness is not suitable to explain biological shape. Nevertheless, randomness is good to guess more than 70% of the shape. So, in view of the extreme perfection of the bone structure of our own bodies, we are authorized to investigate the power of evolution to explain it. We expect from evolution to produce an improvement because our simulations show that random trees could easily be enlarged to get more fit to the cube. It is now the correct time to remember that evolution needs evolvable codes.

110 ***Previous program is rewritten to make it more clear, evolvable.***

```
/*
    Program N110  TreeToShapes13
    A  cube of size 2 is fitted by a random
    3D binary tree.

    Restructured code, more clear, more evolvable.

    The cube has a corner at the origin
    and the others extend over the positive octant.
    The root of the tree is at the origin and it can
    extend its arms in within the cube only.
    The task assigned to randomness
    consists in visiting  with its nodes at least
    a predefined number of points of the cube
    with integer coordinates.

    Our   goal is to know whether or not
    randomness  can explain a cube of edge 2,
    a simple model of   biological shape.
```

```
*/


package Programs;


//Files in the hosting project
import Tools.Axes;
import Tools.Cameras;
import Tools.Line3D;
import Tools.Xform;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Sphere;
import javafx.stage.Stage;



public class TreeToShapes13 extends Application
{
    //A task is denoted by the required
    //number of visited points of the cube, less than 26.
    private static final  int
            RequiredNumberOfVisitedPoints = 22;
    //Maximal number of trials to get the task done
    private static  final int maxNumberOfTrials = 100;

    //Maximal number of trials to find
    //a path that is not repeated and that ends
    // in a point  that is not occupied
    private static final int limit = 50;
```

```
//Counter of trials to find a path
private static  int counterOfTrialsToFindPath= 0;
//Counter of trials to fit the unitary cube
private static  int counterOfTrialsCube = 0;
//Maximal length or number of links along arms of trees
private static int maxHeight;
//The terminal  leaves define the complete tree
private static int numberOfLeaves;

//This is data for a tree in our binary notation:
//Even entries are the addresses
//of the nodes in the tree,
//odd ones are the names or contents.
//An address could be like this 100010,
//it gives the structure of ramification.
//A content could be any string with a message.
private static List<String> TreeInfo;

//List to keep the occupied places in 3D space
private static List<Point3D> Occupancy;

/*
There are 6 directions for a rectangular tree:
0: +X, 1 = -X,  2 = +Y,  3 = -Y
4 = +Z, 5 = -Z
*/
private static int direction;

//Array with  strings of directions that define a path
//to localize each node in 3D space.
private static List<String> Paths3D;

//Printing for debugging
final static boolean printReadFile = false;
private static final String title
        =  "3D binary bush";

//Binary string that says whether or not  node i
```

```
//is present (bit i = 1)
private static String bitTree = "";

//Root of the   tree of the JavaFX application
final Group root = new Group();
private static final Cameras camera = new Cameras();
final double cameraDistance = 5450;//450
//Axes : ctrl + A
private static final Axes axes = new Axes();
final Xform world3DXform = new Xform();
final Xform bonesXform = new Xform();

//Generator of pseudo random numbers
private static final Random r = new Random();


//Hidden argument of method lineTo(thisPoint)
//to call a line creator method:
//Line3D(LastPoint, thisPoint)
private static Point3D lastPoint;
private static final Point3D root3D
        = new Point3D(0,0,0);

//Length of strokes in 3D space
private static final int delta = 50;

//Boolean printing flags
private static final boolean print = false;
private static final boolean printInfo = false;
private static final boolean printIsOccupied = false;
private static final boolean printGetAncestralPath
        = false;
private static final boolean printTryToGetPath = false;
private static final boolean printDrawTree = false;
private static final boolean printTreeTo3D = false;
private static final boolean printIsPathRepeated
        = false;
```

```
//Fundamental parameters are given a value
private void initialization()
{
    maxHeight = r.nextInt(5);
    counterOfTrialsCube = 0;

    //The terminal  leaves define the complete tree
    numberOfLeaves
            = (int) Math.pow(2, maxHeight);

    //This is data for a tree in our binary notation:
    //Even entries are the addresses
    //of the nodes in the tree,
    //odd ones are the names or contents.
    //An address could be like this 100010,
    //it gives the structure of ramification.
    //A content could be any string with a message.
    TreeInfo  = new  ArrayList<>(2*numberOfLeaves-1);


    //List to keep the occupied places in 3D space
    Occupancy = new  ArrayList<>(2*numberOfLeaves-1);
    Paths3D  = new  ArrayList<>(2*numberOfLeaves-1);
}


@Override
public void start(Stage primaryStage)
{


    primaryStage.setTitle(title);
    Scene scene = new Scene(root, 1200, 1000, true);

    scene.setFill(Color.AQUAMARINE);
    primaryStage.setScene(scene);
    primaryStage.show();
    axes.buildAxes();
```

```
    axes.handleKeyboard(scene);
    world3DXform.getChildren().addAll( axes.axesXform);

    camera.handleMouse(scene);
    camera.buildCamera();
    scene.setCamera(camera.camera);

    //Various random trees are tried out
    variousTrees( );

    //The last tree is drawn
    Group parent = new Group(create3dContent());
        world3DXform.getChildren().addAll(parent);
        root.getChildren().add(world3DXform);

    if (printInfo) printInfo();
    primaryStage.show();
    play();


}

public void play()
{

}

@Override public void stop()
{

}
```

```
private static void printAddresses()
```

```
{
    System.out.println("Addresses");

    int l = TreeInfo.size();
    for(int i = 0; i < l; i = i+2)
    {
        System.out.println(i/2 + "   " + TreeInfo.get(i)
                + "   " + TreeInfo.get(i+1));
    }
}


private static void printOccupancy()
{
    System.out.println("Record of occupancy");

    int l = Occupancy.size();
    System.out.println("Occupancy");
    for(int i = 0; i < l; i++)
    {
        System.out.println(i
                + "Address = " + TreeInfo.get(2*i)
                + "   Path = " + Paths3D.get(i)
                + "   point =    "
                + Occupancy.get(i).toString());
        if (i > 0)
            for(int j = 0; j < i; j++)
            {
                if (Occupancy.get(i)
                        .equals(Occupancy.get(j)))
                    System.out.println(i
                            + " Repeated with " + j);
            }
    }
}


private static void printInfo()
{
    printAddresses();
```

```
    printOccupancy();

}

//=========================================================
//========= Tree as a ramificating structure is built=====
//=========================================================

//A binary string with 2*numberOfLeaves-1 bits
private static String randomBinaryString()
{
    //A random binary string
    bitTree = "";
    for (int i = 0; i < 2*numberOfLeaves-1; i++)
    {
        Integer j = r.nextInt(2);
        bitTree = bitTree + j.toString();

    }

    if (print)
        System.out.println(bitTree);
    return bitTree;
}

//Binary string s is decoded into an int
private static int stringToNumber(String s)
{
    int n = 0;
    int l = s.length();

    for(int j=0;j  < l;j++ )
    {
        char c  =   s.charAt(j);
        int  v = Character.getNumericValue(c);
        int q = j+1;
        int number1 = (int) (v * Math.pow(2,l-q));
        n  = n + number1;
    }
    return n;
```

```
}


//String address is interpreted as a binary number
//that corresponds with an int
private static int stringToInt(String address)
{
    int output = 0;
    //System.out.print("String = " + address );
    char c  = address.charAt(0);
    if  (c == '1') output = stringToNumber(address);
        else System.out.println("Bad encoding");
    //System.out.println("  int =  " + output);
    return output;
}


//All ancestors of node m are added
private  void  addAncestors(int m)
{
    String address = TreeInfo.get(m);
    int a = address.length();
    //If the address of  a node is 10001,
    //an ancestor has as address a left substring
    //of 0001, say 0, 00 or 000.
    //All of them must be added to TreeInfo

    for(int i = 0; i < a;  i++)
    {
        String leftSubstring
                = address.substring(0, i);
        leftSubstring = "1" + leftSubstring;
        int k = stringToInt(leftSubstring);
        //Updating of tree information
        if (k < bitTree.length()-1)
            bitTree = bitTree.substring(0, k)
                + "1" + bitTree.substring(k+1);
        else bitTree = bitTree.substring(0, k)
                + "1" ;
    }
```

```
}



//The tree encoded by binary string b is
//decoded into an array with binary addresses and contents.
//The tree corresponds with the lower
//part of a tree with one more generation.
private  void  createTree(String b )
{


    int l = b.length();
    char c ;
    String s ;
    //First two bits do not matter
    for (Integer i = 2; i < l; i++)
    {
        c = b.charAt(i);
        if (c == '1')
        {
            if ( i < 2*numberOfLeaves)
            {
                s = Integer.toBinaryString(i);
                //prefix "1" is deleted
                s = s.substring(1);
                TreeInfo.add(s);
                s = s.replace('0', '+');
                s = s.replace('1', '-');
                TreeInfo.add(s);
            }
        }
    }
}


//The tree encoded by binary string b is
//decoded and drawn.
//The tree corresponds with the lower
```

```
//part of a tree with one more generation.
private  void  createTree( )
{


    //A random binary string:
    //If bit I is 1, the I node is added to the tree
    bitTree = randomBinaryString();
    createTree(bitTree) ;


    //If a node exists, all its ancestors too.
    //bitTree will be modified in consequence.
    for(int m = 0; m < TreeInfo.size()-1;   m = m+2)
        addAncestors(m);

    TreeInfo.clear();
    createTree(bitTree);
}


//=========================================================
//========= Tree is realized as a virtual 3D shape =======
//=========================================================

/*
There are 6 directions given by
//the natural basis.
One distinct than d is chosen
*/
private int distinctDirection(int d)
{
    int i = r.nextInt(6);
    //System.out.println(i + "  " + d);
    if (i == d) distinctDirection(d);
    return i;
}


//A direction distinct than  backwards is chosen
```

```
private int notBackwards(int d)
{
    int backwards  = 0;
    switch (d)
    {
        case 0: backwards = 1; break;
        case 1: backwards = 0 ; break;
        case 2: backwards = 3; break;
        case 3: backwards = 2; break;
        case 4: backwards = 5; break;
        case 5: backwards = 4; break;
    }
    int i = distinctDirection(backwards);
    return i;
}

//Direction d is realized as a move
//There are 6 directions:
//0: +X, 1 = -X,  2 = +Y,  3 = -Y
//4 = +Z, 5 = -Z.
private  Point3D  move3D(int d)
{

    Point3D p = new Point3D(0,0,0);
    switch (d)
    {
        case 0: p = new Point3D( delta,0,0); break;
        case 1: p = new Point3D(-delta,0,0); break;
        case 2: p = new Point3D(0, delta,0); break;
        case 3: p = new Point3D(0,-delta,0); break;
        case 4: p = new Point3D(0,0, delta); break;
        case 5: p = new Point3D(0,0,-delta); break;
    }
    return p;
}

//Common initial substring to strings s and t
private  String commonSubstring(String s, String t)
{
    int a = s.length();
```

```
    int b = t.length();
    int min = a;
    if (b < a ) min = b;
    String c = "";
    for(int i  = 0; i < min; i++)
    {
        if (s.charAt(i) == t.charAt(i))
            c = c + s.charAt(i);
        else break;
    }
    return c;
}


//Geometric barriers are defined here
private  boolean isForbidden(Point3D thisPoint)
{
    boolean b = false;
    double x = thisPoint.getX();
    double y = thisPoint.getY();
    double z = thisPoint.getZ();
    //Codification of  cube of size 2
    //Unitary edge = 50
    if ((x < 0) || (x > 110)) b = true;
    if ((y < 0) || (y > 110)) b = true;
    if ((z < 0) || (z > 110)) b = true;
    return b;
}




//Is thisPoint already occupied?
private  boolean isOccupied(Point3D thisPoint)
{
    //printOccupancy();
    int l = Occupancy.size();
    boolean occupied = false;
    if (printIsOccupied)
        System.out.println("isOccupied?");
```

```
    //Points at the boundary of the shape are occupied
    if (isForbidden(thisPoint) )
    {
        if (printIsOccupied)
            System.out.println("Forbidden" + thisPoint);
        return true;
    }
    //(0,0,0) is occupied
    if (root3D.equals(thisPoint))
        {
            return true;
        }
    else
    {
        for(int i = 0; i < l; i++)
        {
            if (Occupancy.get(i).equals(thisPoint))
            {
                occupied = true;
                break;
            }
        }
        return occupied;
    }
}


//Is thisPoint already occupied?
private  boolean isPathRepeated(String path)
{
    int l = Paths3D.size();
    boolean repeated = false;
    if (printIsPathRepeated)
        System.out.println("isPathRepeated?");
    for(int i = 0; i < l; i++)
    {
        if (Paths3D.get(i).equals(path))
        {
            repeated = true;
            break;
```

```
        }
    }
    return repeated;
}


//A node has a path that must coincide with
//the paths of its ancestors.
//So, we look for the longest one
private String getAncestralPath(int i)
{
    String addressi = TreeInfo.get(i);

    String ancestralPath = "";
    String candidatePath;
    int f = Paths3D.size();
    List<String> Candidates
            = new  ArrayList<>(2*numberOfLeaves-1);
    for (int j = f-1; j >= 0; j--)
    {
        String addressj = TreeInfo.get(2*j);
        if (printGetAncestralPath)
            System.out.print(" addressi =  " + addressi
                + " addressj = " + addressj);
        String commonAddress
                = commonSubstring(addressi, addressj);


        int k = commonAddress.length();
        if (k == 0)
        {
            if (printGetAncestralPath)
                System.out.println(" No common subaddress ");
        }
        else
        {
            String path = Paths3D.get(j);
            candidatePath = path.substring(0, k);
            Candidates.add(candidatePath);
            if (printGetAncestralPath)
```

```
                System.out.println(" Common address = "
                    + commonAddress
                    + " Pathj = " + path
                    + "  candidate path = "
                    + candidatePath);

        }

    }
    int l = Candidates.size();
    int max = 0;
    for(int k = 0; k < l; k++)
    {
        String s = Candidates.get(k);
        int p = s.length();
        if (p > max)
        {

            max = p;
            ancestralPath = s;
        }
    }
    if (printGetAncestralPath)
                System.out.println(" ancestral path = "
                    + ancestralPath);
    return ancestralPath;
}


//Node i in the list TreeInfo
//needs an exclusive location in 3D.
//If possible, one is found.
private  String tryToGetPath(int i)
{
    //This is the root of the tree
    Point3D   P1 = new Point3D(0, 0,0);
    //Reinitialization
    lastPoint = P1;
    Point3D thisPoint = P1;
    //A child must share subpaths with their ancestors
```

```
String address = TreeInfo.get(i);
int l = address.length();

String ancestralPath = getAncestralPath(i);
String path = ancestralPath;
int lengthPath = path.length();
for (int j = 0; j < lengthPath; j++)
{
    //Direction at j is read
    direction = Integer.parseInt("" + path.charAt(j));

    //thisPoint moves  along the chosen direction
    thisPoint = thisPoint.add(move3D(direction));

}

//New directions are added to complete path
if (counterOfTrialsToFindPath< limit) //Overflow is interdicted.
{
    //Directions are concatenated
    for (int j = lengthPath; j < l; j++)
    {
        //A non backwards new direction is chosen
        direction = notBackwards(direction);
        path = path + direction;
        //Chosen direction is executed in memory
        thisPoint = thisPoint.add( move3D(direction));
    }

    if (printTryToGetPath)
        System.out.println("\nAddress to try = "
            + address
             + "   Ancetralpath = " + ancestralPath
            + " counterOfTrialsToFindPath= "
            + counterOfTrialsToFindPath
            +   " \nThisPoint = " + thisPoint);

    if ( (isPathRepeated(path))
            || (isOccupied(thisPoint)) )
```

```
        {
            if (printTryToGetPath)
                System.out.println("    Path is repeated.");
            counterOfTrialsToFindPath++;
            tryToGetPath(i);
        }
        else
        {
            if (printTryToGetPath)
                System.out.println(
                        "    Path is NOT repeated.");
            Occupancy.add(thisPoint);
            Paths3D.add(path);
        }
    }
    if (counterOfTrialsToFindPath== limit)
    {
        if (printTryToGetPath)
            System.out.println(
                    "    Path not found at trial "
                + counterOfTrialsToFindPath);
        path =  "-1";
    }

    return path;
}


//Information is created to realize
//and draw tree into 3D space.
private  void treeTo3D()
{
    String path ;

    //Directions are proposed at random,
    //and concatenated if no double occupancy appears
    for (int i = 0; i < TreeInfo.size(); i= i+2)
    {
        if (printTreeTo3D)
            System.out.print("Node i = " + i
```

```
                + "  Address = " + TreeInfo.get(i));
        counterOfTrialsToFindPath= 0;
        path = tryToGetPath(i);
        if (printTreeTo3D)
            System.out.println("  Path at treeTo3D = "
            + path);
        if (path.contentEquals("-1"))
        {
            if (printTreeTo3D)
                System.out.println("Truncation");
        }
    }
    if (printTreeTo3D)
        System.out.println("Paths3D size = "
                + Paths3D.size());
    if (printTreeTo3D) printOccupancy();
}


//============================
//========= Master Method=====
//============================


private   void makeTree( )
{
    initialization();
    //TreeInfo is filled in.
    //A binary string is composed:
    //if bit i is 1, node i is added to the tree.
    //A binary address to each node of the tree
    //is following by a content in TreeInfo.
    createTree();
    //Information is created to realize
    //and draw tree into 3D space
    treeTo3D();
}


//Is the task done?
```

```
//Task: populate lateral and inner points
//with integer coordinates
//of a cube of edge 2
private   boolean weAreDone()
{
    boolean b = false;
    int n = Occupancy.size();
    System.out.println("Number of occupied points = " + n);
    if (n == RequiredNumberOfVisitedPoints) b = true;
    return b;
}


private   void  variousTrees( )
{
    for(int i = 0; i < maxNumberOfTrials; i++)
    {
        makeTree();
        counterOfTrialsCube = i;
        if (weAreDone())   break;
        if (counterOfTrialsCube < maxNumberOfTrials-1)
        {
            TreeInfo.clear();
            Occupancy.clear();
            Paths3D.clear();
        }
    }
    System.out.println("Demanded number of visitied points = " +
            RequiredNumberOfVisitedPoints
            + "\nNumber of done trials  = "
            + counterOfTrialsCube );
}




//================================
//======= 3D graphics ===========
//================================
```

```
//Instantiation of the  3D graphics class
private Node create3dContent( )
{
    Project3D  p =  new Project3D( );
    return new Group(p);
}


//The drawing is synthesized
private class Project3D extends Group
{

//Constructor
Project3D()
{
    //Tree is drawn to expand a 3D bush
    Group g = drawTree( );
    getChildren().addAll(g );
}

//Returns a 3D line that goes from lastPoint
//to thisPoint
private  Line3D  line3DTo(Point3D thisPoint)
{
    Line3D l = new Line3D(lastPoint, thisPoint);
    lastPoint = thisPoint;
    return l;
}


private   Group drawTree( )
{
    Group myGroup  = new Group();
    //This is the root of the tree
    Point3D   P1 = new Point3D(0, 0,0); //+X direction


    //Put a landmark at the root
```

```
    Sphere sphere = new Sphere();
    sphere.setRadius(10);
    sphere.setTranslateX((int) P1.getX());
    sphere.setTranslateY((int) P1.getY() );
    sphere.setTranslateZ((int) P1.getZ() );
    root.getChildren().add(sphere);

    //Information is created to realize
    //and draw tree into 3D space
    treeTo3D();
    Occupancy.clear();
    //The tree is drawn:
    for (String Paths3D1 : Paths3D)
    {
        //Reinitialization
        lastPoint = P1;
        Point3D thisPoint = P1;
        String path = Paths3D1;
        if (printDrawTree)
            System.out.println( "Path = " + path);
        int length = path.length();
        for (int j = 0; j < length; j++)
        {
            //Direction at j is read
            direction = Integer.parseInt("" + path.charAt(j));
            if (printDrawTree)
                System.out.print(direction + "   ");
            //thisPoint moves  along the chosen direction
            thisPoint = thisPoint.add(move3D(direction));
            //Draw a stroke
            //from lastPoint to thisPoint
            Line3D l = line3DTo(thisPoint);
            myGroup.getChildren().add(l.group);
        }
        if (printDrawTree)
            System.out.println();
        Occupancy.add(thisPoint);
    }
    return myGroup;
}
```

```
}//End of inner class Project3D



public static void main(String[] args)
{
    launch(args);
}

}//End of program N110  TreeToShapes13
```

111 ***Exercise.*** *Run the previous program and play with the code. We know how to generate a tree that fits inside a cube. Encapsulate it into a class to make the code evolvable: the class will allow multiple instantiations to generate a population of trees to be evolved.* *Answer*

112 ***Testing the power of Evolution to explain biological shape.***

We plan to implement evolution in the space of trees to fit a cubic shape with edge of arbitrary length. The task is to visit a given proportion of points of a cube, both inner and lateral, with integral coordinates. We assume that reproduction of cells naturally fill inner spaces. That is why we demand to fill the inner points also. The criterion to decide that the task is done is the following: the total number of points of a cube of edge l that must be visited is $(l+1)^3-1$ (the origin is given for granted so, it must not be visited). Thus, one can pose the task of visiting a number of points less that this value. Our trees have two parts. First: the ramification structure, given by a binary string. Second: the 3D realization given by an array of strings of 3D directions, one string for one path or arm. We evolve only the first.

113 ***Battling with bugs.***

Trying to implement aforementioned program became a terrible battle with bugs, some of which were traced to programs that perfectly ran for the cube of size 2. The most severe ones are those when the program compiles and runs but its output is not what one expects so, the possible cause is completely unknown. In a case like this one must make the program write to the console relevant information from within possible causing methods: **software will be known by its output**. In second place, one must try to test precise instructions or methods one at a time: one makes a simple change in one line while keeping everything else constant to see what happens. These two rules allow one to advance. Another option might be to learn to run programs under the debugger facility that comes with both Netbeans

and Eclipse, which allow to track the values of predetermined variables. In this case, consult the web for help.

You can visit the battle field by running programs `N113 TreeToShapes15` and `N113b TreeToShapes16`. They run but produces graphs that are not binary trees because they may have a trifurcation or a cycle. A cause of bugs was found in the encoding of our variables. Let us see: a tree is determined by 2 items. The first is a ramification structure that is at last encoded in a list of strings, each one pointing to an address of a node in the tree and the second as a realization of each path into 3D space, which is also a string. The problem is that addresses and 3D paths are kept in separate lists that must be synchronized to allow the retrieving of corresponding entries. Now, when one passes from a cube of size 2 to one of size 3 or higher the probability of finding no realization for a path increases and so one has an address that has no pairing with 3D paths. This causes an avalanche of problems that one can solve in many ways, say, with new classes or with maps as in hashmaps the modern version of hashtables. Which could be the simplest one?

The simplest solution is that in which no problem appears and so no solution is needed.

It is delicious to battle to win even now when so much endurance is needed.

114 ***Implementation.***

Let us try to achieve the disappearance of problems by changing our encoding and our way of attacking: an address and its 3D realization will conform one single string. Thus, the correspondence will be retrieved at once. This, and other changes, are implemented in the following code:

```
/*
    Program N114  TreeToShapes17
    A  cube of edge with arbitrary length
    is tried to be fitted by evolution of 3D binary trees.

    The cube has a corner at the origin
    and the others extend over the positive octant.
    The root of the tree is at the origin and it can
    extend its arms in within the cube only.
    The task assigned to randomness
    consists in visiting  with its nodes at least
    a predefined number of points of the cube
    with integer coordinates.
```

```
Our   goal is to know whether or not
evolution  can explain a cube of arbitray edge,
our model of   biological shape.

Our trees have two parts:
1. A genotype given by the ramification structure,
    which is given by a binary string.
2. The phenotype which is the 3D realization
    of the genotype.
    It is given by an array of strings of 3D directions.

The two items are encoded into one single string.
In that way, the correspondence will be apparent
and no problem will spring after adding or deleting items
from the lists where strings are kept.

Only the first item is evolved here.

This program illustrates a powerful concept:
the best strategy to battle bugs is to
interdict its appearance.
Example 1:
Lists were used to keep strings that were
intended to be paired. But it happens
that some pairs get singled.
This causes an avalanche of problems
that the Author was unable to solve.
Solution: encode pairs in a single string
and say good by to ensuing problems and bugs.
Example 2:
A redundant encoding of a tree was used
but to so something many troubles arouse.
Solution: turn to a nonredundant encoding
and prepare before  apperance of troubles
and not after.
In the language of TRIZ this reads as follows:
the best solution is no solution at all. See
http://www.triz-journal.com/triz-what-is-triz/
```

```
    This abstract task corresponds with the formulation
    and simulation of the following theory about
    the genetic encoding of biological shape:
    cell reproduction is genetically instructed
    to follow a binary pattern that stops  reproduction along
    each branch after a perfectly predefined
    number of  divisions. Cellular division poses daughter
    cells at  some place in 3D space and this is not encoded.
    So,  position is 3D space of each cell is a matter of
    randomness. It is supposed that some process gives
    from above the directives for the binary pattern, say,
    the shape of a bone is determined by its performance
    in overall movements. This constraint plus
    a tree structure results in pliotropy and chaos:
    a tiny change in any given place
    may give rise to a completely different shape.
*/


package Programs;


//Files in the hosting project
import Tools.Axes;
import Tools.Cameras;
import Tools.Line3D;
import Tools.Xform;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Sphere;
import javafx.stage.Stage;
```

```
public class TreeToShapes17 extends Application
{
    //=========The cube===========

    //The measure of the edge of the cube, 1,2,...n...
    private static final int lengthOfEdge = 2;
    //Maximal  number of links along arms of a tree.
    //Trees have a random depth or height but this is the max.
    private static final int maxHeight =  3* lengthOfEdge + 2;
    //A task is denoted by the required
    //porportion of occupied points of the cube, less than 1.
    private static final double demandedFractOfPoints = 0.7;



    private static int totalNumberOfPoints;

    private static int  RequiredNumberOfOccupiedPoints;

    //======The tree===============

    //Probability with which each node
    //is inserted into the tree
    private static final double probOfNodeInsertion = 0.1;
    //Number of inserted nodes
    private static int numberOfInsertedNodes;
    //Maximal number of trials to find
    //a path that is not repeated and that ends
    // in a point  that is not occupied
    final int maxNumberOfTrials = 15;

    //======= Evolution============

    //Number of generations
    private static int nGenerations = 10000;
    //Number of individuals in the population
    private static  final int nIndividuals = 5;
```

```
    private static final double pointMutationRate  = 0.1;

    //Population of chromosomes (treated as individuals)
    private static ABinaryTree[]  Pop
                    = new ABinaryTree[nIndividuals] ;
    private static ABinaryTree[]  Pop2
                    = new ABinaryTree[nIndividuals] ;

    //Actual generation
    private static int gen = -1;

    //Printing flags
    private static final boolean printRec = false ;
    private static final boolean printMut = false ;
    private static final boolean printReproduction = false;
    private static final boolean printInitialPopulation
                                            = false;
    private static boolean printFindTheBest = false;
    private static boolean printAssessFitness = false;
    private static boolean printEvolution = false;
    private static boolean printExecutePath = false;


    //==============JavaFX===========


    //Root of the   tree of the JavaFX application
    private static final Group root = new Group();
    private static final Cameras camera = new Cameras();

    //Axes : ctrl + A
    private static final Axes axes = new Axes();
    private static final Xform world3DXform = new Xform();


    //Generator of pseudo random numbers
    final private static Random r = new Random();
    private static final String title
            =  "3D binary bush";
```

```
    //Hidden argument of method lineTo(thisPoint)
    //to call a line creator method:
    //Line3D(LastPoint, thisPoint)
    private static Point3D lastPoint;
    private static final boolean printDrawTree = false;

    /*
    There are 6 directions for a rectangular tree:
    0: +X, 1 = -X,  2 = +Y,  3 = -Y
    4 = +Z, 5 = -Z
    */
    Integer direction;




@Override
public void start(Stage primaryStage)
{


    primaryStage.setTitle(title);
    Scene scene = new Scene(root, 1200, 1000, true);

    scene.setFill(Color.AQUAMARINE);
    primaryStage.setScene(scene);
    primaryStage.show();
    axes.buildAxes();
    axes.handleKeyboard(scene);
    world3DXform.getChildren().addAll( axes.axesXform);

    camera.handleMouse(scene);
    camera.buildCamera();
    scene.setCamera(camera.camera);


    //An optimization task is given to evolution.
    //The index of the best solution is returned
    int theBest = evolution();
```

```
    //The best tree is drawn
    Group parent = new Group(create3dContent(Pop[theBest]));
        world3DXform.getChildren().addAll(parent);
        root.getChildren().add(world3DXform);


    primaryStage.show();
    play();
}


public void play()
{

}

@Override public void stop()
{

}


private   void printParameters()
{
System.out.println("FITTING A CUBE WITH A 3D BINARY TREE."
    + "\nLength of edge = " + lengthOfEdge
    + "\nMaximal depth of tree (number of links along arms of tree) = "
    + maxHeight
    + "\nDemanded fraction of occupied points = "
    + demandedFractOfPoints
    + "\nProbabilty of Node Insertion in the tree = "
    + probOfNodeInsertion
    + "\nMaximal Number Of Trials to get "
    + "a link in the 3D path = " +  maxNumberOfTrials
    + "\nNumber of Generations = " + nGenerations
    + "\nNumber of individuals in the population = "
    + nIndividuals
    + "\nProbability of mutation per bit (in "
    + "the binary string that  represents the nodes"
```

```
    + " of the tree) per generation = "
    + pointMutationRate);
}

//================================
//======= Tree as a class =========
//================================


//Encapsulation of the tree
private class ABinaryTree
{
    //Maximal  number of links along arms of a tree
    int myHeight =  r.nextInt( maxHeight) + 1;



    //Counter of trials to find a path
    int counterOfTrialsToFindPath= 0;
    //Counter of trials to fit the unitary cube
    int counterOfTrialsCube = 0;

    //The terminal  leaves define the complete tree
    int numberOfLeaves;
    //Total number of nodes in the full binary tree
    int totalNumberOfNodes;
    //This is data for a tree in our binary notation.
    //We have the ramification structure:
    //Even entries are the addresses
    //of the nodes in the tree,
    //odd ones are the names or contents.
    //An address could be like this 100010,
    //it gives the structure of ramification.
    //A content could be any string with a message.
    //A ramification structure is next realized as a 3D bush:
    //a 3D path, given by a string of directions in 3D space,
    //is associated to each address
    //if only that is possible.
    //The 3D  path is concatenated to the address
    //and they conform one single string.
```

```
List<String> TreeInfo;

//List to keep the occupied places in 3D space
List<Point3D> Occupancy;

//List for local use

List<Point3D> Occupancy2 ;

//Binary string that says whether or not  node i
//is present (bit i = 1)
String bitTree = "";

double fitness;

final Point3D root3D
        = new Point3D(0,0,0);

//Length of strokes in 3D space
final int delta = 50;

//Boolean printing flags

final boolean printRandomBinaryString = false;
final boolean printInfo = false;
final boolean printIsOccupied = false;
private boolean printUpdateNodes  = false;
private boolean printTryToGetPath = false;
private boolean printexecutePathVirtual = false;
final boolean printTreeTo3D = false;
final boolean printIsPathRepeated  = false;
boolean printFillTreeInfo = false;
final boolean printTestAddressDecoding = false;
final boolean printTestAddAncetors = false;
final boolean printGetAddress = false;
final boolean printGetPath = false;
//Tet the method TryToGetPath
final boolean testTryToGetPath = true;
final boolean  printDeleteSubpaths = false;
```

```
//Constructor = mandatory initialization
ABinaryTree()
{
    initialization();

    if (printTestAddressDecoding)
        testAddressDecoding();
    if (printTestAddAncetors) testAddAncetors();



    //A tree with all ancestors for each node
    //with address and content for each node
    createTree();
}


private void testAddressDecoding()
{
    System.out.println("testAddressDecoding()");
    String s = "10101";
    System.out.println(" s = " + s
            +"  Address = " + getAddress(s)
    + " Path = " + getPath(s));

    s = "10101aa34567";
    System.out.println(" s = " + s
            +"  Address = " + getAddress(s)
    + " Path = " + getPath(s));

    s = "00";
    System.out.println(" s = " + s
            +"  Address = " + getAddress(s)
    + " Path = " + getPath(s));

    s = "00aa11";
```

```
    System.out.println(" s = " + s
            +"  Address = " + getAddress(s)
    + " Path = " + getPath(s));

    s = "00aa1";
    System.out.println(" s = " + s
            +"  Address = " + getAddress(s)
    + " Path = " + getPath(s));

    s = "00aa1111";
    System.out.println(" s = " + s
            +"  Address = " + getAddress(s)
    + " Path = " + getPath(s));

    s = "1101000aa0030222";
    System.out.println(" s = " + s
            +"  Address = " + getAddress(s)
    + " Path = " + getPath(s));
    executePathVirtual(getPath(s));

    s = "1101aa2544";
    System.out.println(" s = " + s
            +"  Address = " + getAddress(s)
    + " Path = " + getPath(s));
    executePathVirtual(getPath(s));



}

private void testAddAncetors()
{
    System.out.println("testAddAncetors()");
    //Fake initialization
    numberOfLeaves = 40;
    //String BitTree = "1111110";
    String BitTree = "1111111";

    //bitTree is decoded into an array
    //with addresses of nodes and content
```

```
    fillTreeInfo(BitTree);
    //All ancestors are added to each node
    addAncestors();
    //True initialization
    numberOfLeaves
            = (int) Math.pow(2, myHeight);

}


//Fundamental parameters are given a value
private void initialization()
{

    myHeight = r.nextInt( maxHeight) + 1;
    //Total number of points of the shape,
    //both  inner and superficial.
    //For a cube: (lengthOfEdge + 1)^3
    totalNumberOfPoints
            = (int) Math.pow(lengthOfEdge + 1, 3) ;
    numberOfInsertedNodes = 0;

    RequiredNumberOfOccupiedPoints
        = (int)(demandedFractOfPoints * totalNumberOfPoints);
    //The terminal  leaves define the complete tree
    numberOfLeaves
            = (int) Math.pow(2, myHeight);
    totalNumberOfNodes = 2*numberOfLeaves-1;
    //This is data for a tree in our binary notation:
    //Even entries are the addresses
    //of the nodes in the tree,
    //odd ones are the names or contents.
    //An address could be like this 100010,
    //it gives the structure of ramification.
    //A content could be any string with a message.
    TreeInfo  = new  ArrayList<>(totalNumberOfNodes);


    //List to keep the occupied places in 3D space
    Occupancy = new  ArrayList<>(totalNumberOfNodes);
```

```
    //List to keep the occupied places in 3D space
    Occupancy2 = new  ArrayList<>(totalNumberOfNodes);
    counterOfTrialsCube = 0;

    //Prevention against garbage annoyance
    TreeInfo.clear();
    Occupancy.clear();
    Occupancy2.clear();

}




private void printAll(String subtitle)
{
    System.out.println("printAll() " + subtitle);

    System.out.println("TreeInfo.size() = " + TreeInfo.size()
            + "     Occupancy.size() = " + Occupancy.size());
    for(int i = 0; i < TreeInfo.size(); i = i + 2)
       // for(int i = 0; i < 20; i = i + 2)
    {
        String point;
        Point3D P = new Point3D(0,0,0);
        if (executePathVirtual(getPath(i)).equals(P))
            point = "";
        else point
            = executePathVirtual(getPath(i)).toString();
        /*
        System.out.println(i
                + "   TreeInfo = " + TreeInfo.get(i)
                + "   Address = " + getAddress(i)
                + "   Path = " + getPath(i)
                + "   Point3D = " + point);
        */
        System.out.println(TreeInfo.get(i)
                + "\n" + TreeInfo.get(i+1));
    }
```

```
    System.out.println("Occupancy.size() = "
            + Occupancy.size());
    for(int i = 1; i < Occupancy.size(); i++)
    {

        for(int j = 0; j < i; j++)
        {

            if (Occupancy.get(i).equals(Occupancy.get(j)))
                System.out.println(i
                        + " RRRRRRRRepeated with " + j);
        }
    }
}


private void printOccupancy(List<Point3D> O, String subtitle)
{
    System.out.println("printOccupancy() " + subtitle);


    for(int i = 0; i < O.size(); i++)
    {
        System.out.println(i
                        + "    " + O.get(i));
        for(int j = 0; j < i; j++)
        {

            if (O.get(i).equals(O.get(j)))
                System.out.println(i
                        + " RRRRRRRRepeated with " + j);
        }
    }
}


//=========================================================
//========= Tree as a ramificating structure is built=====
//=========================================================
```

```
//A binary string with 2*numberOfLeaves-1 bits
private String randomBinaryString()
{
    //A random binary string
    bitTree = "";


    for (int i = 0; i < totalNumberOfNodes; i++)
    {
        double p = r.nextDouble();

        Integer j;
        if (p < probOfNodeInsertion )
        {
            j = 1;
            numberOfInsertedNodes++;
        }
        else j = 0;
        bitTree = bitTree + j.toString();

    }
    System.out.println(" Number Of Inserted Nodes = "
        + numberOfInsertedNodes
        + " Maximal Number Of Possible Nodes = "
            + totalNumberOfNodes);
    if (printRandomBinaryString)
        System.out.println(bitTree);
    return bitTree;
}



//Binary string s is decoded into an int
private int stringToNumber(String s)
{
    int n = 0;
    int l = s.length();

    for(int j=0;j  < l;j++ )
```

```
    {
        char c  =   s.charAt(j);
        int  v = Character.getNumericValue(c);
        int q = j+1;
        int number1 = (int) (v * Math.pow(2,l-q));
        n  = n + number1;
    }
    return n;
}


//String address is interpreted as a binary number
//that corresponds with an int
private int stringToInt(String address)
{
    int output = 0;
    //System.out.print("String = " + address );
    char c  = address.charAt(0);
    if  (c == '1') output = stringToNumber(address);
        else System.out.println("Bad encoding");
    //System.out.println("  int =  " + output);
    return output;
}



private String getAddress(String s)
{
    String address = s;
    if (address.contains("aa"))
    {
        int l = s.indexOf("aa");
        address = s.substring(0, l);
    }
    if (printGetAddress)
        System.out.println("input = " + s
            + "   address = " + address);
    return address;
}
```

```
private String getAddress( int i)
{
    String s = TreeInfo.get(i);
    String address = getAddress(s);
    return address;
}

private String getPath(String s)
{

    String path = "";
    if (s.contains("aa"))
    {
        int l = s.indexOf("aa");
        path = s.substring( l+2);
    }
    if (printGetPath)
        System.out.println("input = " + s
            + "   path    = " + path);
    return path;
}

private String getPath( int i)
{
    String s = TreeInfo.get(i);
    String path = getPath(s);
    return path;
}


//All ancestors of node m are added to bitTree
private  void  addAncestors(int m)
{
    String address = getAddress(m);
    //System.out.println("address = " + address);
    //If the address of  a node is 10001,
    //an ancestor has as address a left substring
    //of 0001, say 0, 00 or 000.
    //All of them must be added to TreeInfo
```

```
    String bitTreeInput = bitTree;
    String leftSubAddress = "";
    int k = 0;
    for(int i = 1; i < address.length();  i++)
    {
        leftSubAddress
                = address.substring(0, i);

        leftSubAddress = "1" + leftSubAddress;
        k = stringToInt(leftSubAddress);



        //Updating of tree information.
        //Last bit does not need updating
        if (k < bitTree.length()-1)
        {
            bitTree = bitTree.substring(0, k)
                + "1" + bitTree.substring(k+1);

        }
    }

    String bitTreeOutput = bitTree;

    if (bitTreeInput.length() >= bitTreeOutput.length())
        System.out.println("\nERROR IN ADDANCESTORS"
                + " m =  " + m
                + "\nAddress  = " + address
                + "\n1+leftSubAddress = " + leftSubAddress
                + "\nk = " + k
                + "\nbitTree input = " + bitTreeInput
                +  "\nbitTree outpt = " + bitTreeOutput);
}

//If a node exists, all its ancestors too.
//bitTree is modified in consequence.
private  void addAncestors()
{
    for(int m = 0; m < TreeInfo.size()-1;   m = m+2)
```

```
    {
        addAncestors(m);

    }
}



//The tree encoded by binary string b is
//decoded into an array with binary addresses and contents.
//The tree corresponds with the lower
//part of a tree with one more generation.
private  void  fillTreeInfo(String b )
{
    int l = b.length();
    //printFillTreeInfo = true;
    if(printFillTreeInfo)
        System.out.println("fillTreeInfo: \nbitTree "
                + " = " + b
                +  " bLength = " + l);
    char c ;
    String s ;
    int bound = 2*numberOfLeaves;
    //First two bits do not matter
    for (Integer i = 2; i < l; i++)
    {
        c = b.charAt(i);
        if (c == '1')
        {
            //In spite of different depth,
            //trees are encoded by bit strings
            //of the same length
            if ( i < bound)
            {
                if(printFillTreeInfo)
                    System.out.println(" i = " + i);
                s = Integer.toBinaryString(i);
                if(printFillTreeInfo)
                    System.out.println("s0 = " + s);
```

```
                //prefix "1" is deleted from place 0
                s = s.substring(1);
                //Address is added
                TreeInfo.add(s);
                if(printFillTreeInfo)
                    System.out.println("s1 = " + s);
                s = s.replace('0', '+');

                //Content
                s = s.replace('1', '-');
                TreeInfo.add(s);
                if(printFillTreeInfo)
                    System.out.println("s2 = " + s);
            }

        }
    }
}



//All ancestors of node m are deleted
private  void  deleteAncestors(int m)
{
    String address = getAddress(m);
    //If the address of  a node is 10001,
    //an ancestor has as address a left substring
    //of 0001, say 0, 00 or 000.
    //All of them must be deleted from TreeInfo

    for(int i = 0; i < address.length();  i++)
    {
        String leftSubstring
                = address.substring(0, i);
        leftSubstring = "1" + leftSubstring;
        int k = stringToInt(leftSubstring);
        //Deletion is registered in bitTree
        if (k < bitTree.length()-1)
            bitTree = bitTree.substring(0, k)
                + "0" + bitTree.substring(k+1);
```

```
        else
            if (k == bitTree.length()-1)
                bitTree = bitTree.substring(0, k)
                + "0" ;
        else
            System.out.println("Error in deleteAncestors"
            + " k = " + k
            + "  bitTreeLength = " + bitTree.length());
    }









}


//The tree is simplified. To that aim,
//all ancestors of each node  are deleted
private  void  simplifyTree()
{
  //If a node exists, all its ancestors are erased
    for(int m = 0; m < TreeInfo.size()-1;   m = m+2)
        deleteAncestors(m);
    //Tree information is re-created from the bitTree
    TreeInfo.clear();
    Occupancy.clear();
    fillTreeInfo(bitTree);
}

//The tree encoded by binary string b is
//decoded and drawn in memmory.
//The tree corresponds with the lower
//part of a tree with one more generation.
//All ancestors of eacn node are added.
private  void  createTree()
{
```

```
    //A random binary string:
    //If bit I is 1, the I node is added to the tree
    bitTree = randomBinaryString();
    //bitTree is decoded into an array
    //with addresses of nodes and content
    fillTreeInfo(bitTree);
    //Simplify tree:
    simplifyTree();

}


//=========================================================
//========= Tree is realized as a virtual 3D shape =======
//=========================================================

/*
There are 6 directions given by
the natural basis.
One distinct than d is chosen
*/
private Integer distinctDirection(Integer d)
{
    Integer i = r.nextInt(6);
    //System.out.println(i + "  " + d);
    if (i.equals(d)) distinctDirection(d);
    return i;
}


//A direction distinct than  backwards is chosen
private Integer notBackwards(Integer d)
{
    int backwards  = 0;
    switch (d)
    {
        case 0: backwards = 1; break;
        case 1: backwards = 0 ; break;
        case 2: backwards = 3; break;
        case 3: backwards = 2; break;
```

```
        case 4: backwards = 5; break;
        case 5: backwards = 4; break;
    }
    Integer i = distinctDirection(backwards);
    return i;
}

//Direction d is realized as a move
//There are 6 directions:
//0: +X, 1 = -X,  2 = +Y,  3 = -Y
//4 = +Z, 5 = -Z.
private  Point3D  move3D(int d)
{

    Point3D p = new Point3D(0,0,0);
    switch (d)
    {
        case 0: p = new Point3D( delta,0,0); break;
        case 1: p = new Point3D(-delta,0,0); break;
        case 2: p = new Point3D(0, delta,0); break;
        case 3: p = new Point3D(0,-delta,0); break;
        case 4: p = new Point3D(0,0, delta); break;
        case 5: p = new Point3D(0,0,-delta); break;
    }
    return p;
}

//Common initial substring to strings s and t
private  String commonSubstring(String s, String t)
{
    int a = s.length();
    int b = t.length();
    int min = a;
    if (b < a ) min = b;
    String c = "";
    for(int i  = 0; i < min; i++)
    {
        if (s.charAt(i) == t.charAt(i))
            c = c + s.charAt(i);
        else break;
```

```
    }
    return c;
}


//Geometric barriers given by the required shape
//are defined here
private  boolean isBarrier(Point3D thisPoint)
{
    boolean b = false;
    double x = thisPoint.getX();
    double y = thisPoint.getY();
    double z = thisPoint.getZ();
    //Codification of  cube of size lengthOfEdge
    //Unitary length = 50
    int barrier = 50 * lengthOfEdge + 10;
    if ((x < 0) || (x > barrier)) b = true;
    if ((y < 0) || (y > barrier)) b = true;
    if ((z < 0) || (z > barrier)) b = true;
    return b;
}


//Is thisPoint already occupied?
private  boolean isOccupied1(Point3D thisPoint)
{
    //printOccupancy();
    int l = Occupancy.size();
    boolean occupied = false;
    if (printIsOccupied)
        System.out.println("isOccupied?");
    //Points at the boundary of the shape are occupied
    if (isBarrier(thisPoint) )
    {
        if (printIsOccupied)
            System.out.println("Forbidden" + thisPoint);
        return true;
    }
    //(0,0,0) is occupied
    if (root3D.equals(thisPoint)) return true;
```

```
    else
    {
        for(int i = 0; i < l; i++)
        {
            if (Occupancy.get(i).equals(thisPoint))
            {
                occupied = true;
                break;
            }
        }
        return occupied;
    }
}


//Is thisPoint already occupied?
private  boolean isOccupied2(Point3D thisPoint)
{
    //printOccupancy();
    int l = Occupancy2.size();
    boolean occupied = false;
    if (printIsOccupied)
        System.out.println("isOccupied?");
    for(int i = 0; i < l; i++)
    {
        if (Occupancy2.get(i).equals(thisPoint))
        {
            occupied = true;
            break;
        }
    }
    return occupied;

}

//Is thisPoint already occupied?
private  boolean isOccupied(Point3D thisPoint)
{
    boolean occupied = false;
    if (isOccupied1(thisPoint)) return true;
```

```
    if (isOccupied2(thisPoint)) return true;
    return occupied;
}

//Is thisPoint already occupied?
private  boolean isPathRepeated(String path)
{
    boolean isRepeated = false;
    if (printIsPathRepeated)
        System.out.println("isPathRepeated?");
    for(int i = 0; i < TreeInfo.size(); i = i+2)
    {

        if (getPath(i).equals(path))
        {
            isRepeated = true;
            break;
        }
    }
    return isRepeated;
}

//If a path  to node i has not been found,
// delete branching paths
private  void  deleteBlindAlleys(int i)
{
    String addressI = TreeInfo.get(i);
    //If   node with address  10001 is a blind alley,
    //all its branchings  must be deleted.
    //Example: node 10001 is extended by 1000101.
    //If the first is a blind alley, the second also
    //and must be deleted.
    String addressJ;

    for(int j = TreeInfo.size() -2;  j > i; j =j-2)
    {
        addressJ = TreeInfo.get(j);
        if (addressJ.length() > addressI.length())
            if (TreeInfo.get(j)
                .substring(0, addressI.length())
```

```
                    .contains(addressI))
            {
                TreeInfo.remove(j);//Address
                TreeInfo.remove(j);//Content
            }
    }

}


//3D bush must reflect the binary structure of the tree
//so, path to node i might share a part with
//all others with a common subaddress.
private void updateNodes(int i)
{
    String addressI = getAddress(i);
    String pathI = getPath(i);


    String copiedPath;
    printUpdateNodes = false;

    if (i < TreeInfo.size()-2)
    {
        for(int j = i+2; j < TreeInfo.size(); j = j+2)
        {
            String addressJ = getAddress(j);
            String pathJ = getPath(j);
            if (printUpdateNodes)
                System.out.println("i = " + i
                    + " TreeInfo i = " + TreeInfo.get(i)
                    + " addressI =  " + addressI
                    + " pathI =  " + pathI
                    + " j = " + j
                    + " TreeInfo j = " + TreeInfo.get(j)

                    + " addressj = " + addressJ
                    + " pathJ =  " + pathJ);

            String commonAddress
```

```
        = commonSubstring(addressI, addressJ);
int k = commonAddress.length();
if (k == 0)
{
    if (printUpdateNodes)
        System.out.println(
                " No common subaddress ");
}
else
{
    int pIl = pathI.length();
    int pJl = pathJ.length();


    if (printUpdateNodes)
        System.out.print(
            " Common subaddress = "
                    + commonAddress
        +  " PathJ = " + pathJ );

    if ((pIl > 0) && (pJl < k ))
    {
        if (pJl < pIl)
        {
            int border = k;
            if (k > pIl) border = pIl;
            copiedPath
                = pathI.substring(0, border);
            TreeInfo.set(j
                , addressJ + "aa" + copiedPath);
            if (printUpdateNodes)
                System.out.println(
                    "  copied path = "
                            + copiedPath
                + " TreeInfo j = "
                            + TreeInfo.get(j));
        }

    }
}
```

```
        }//End of for (int j..)
    }//End of if (i < TreeInfo.size()-2)

    if (addressI.length() > pathI.length())
        deleteBlindAlleys(i);


}


//Path is executed and final point is reported
private Point3D executePathVirtual(String path)
{
    Point3D   thisPoint = new Point3D(0, 0,0);

    if (path.contentEquals("")) direction = 0;
    else
    {
        if (printexecutePathVirtual)
                System.out.println("Path = " + path );
        //Reinitialization
        lastPoint = thisPoint;
        for (int j = 0; j < path.length(); j++)
        {
            //char j is read as integer
            direction = Integer.parseInt(
                    ""  + path.charAt(j));

            //thisPoint moves  along the chosen direction
            thisPoint = thisPoint.add(move3D(direction));
            if (printexecutePathVirtual)
                System.out.println("direction = " + direction
                        + " thisPoint =  " + thisPoint);
        }

    }

    return thisPoint;
}
```

```
//Node i in the list TreeInfo
//needs an exclusive path to get it a location in 3D.
//If it is possible, one is found.
private  String tryToGetPath(int i)
{


    if (printTryToGetPath)
        System.out.println("Method tryToGetPath(int i)");


    String addressI = getAddress(i);
    //Initialization of direction and path
    //for the 3D point that node i will occupy.
    String oldPath = getPath(i);
    String path0 = oldPath;
    Point3D  thisPoint0 = executePathVirtual(path0);
    if (printTryToGetPath)
        System.out.println(
            " \n\n**** Input at tryToGetPath(int i)"
            + "TreeInfo(" + i + ") = "
            + TreeInfo.get(i)
            + "  address " + addressI
            + "  path = " + path0
            + " this point " + thisPoint0
        + "  direction = " + direction);
    int direction0 = direction;

    int l0 = path0.length();
    int l1 = addressI.length();
    Point3D thisPoint;
    String path;

    //Directions are concatenated to path.
    //One direction for one char of address
    //-if only that is possible
    for (int j = l0; j < l1; j++)
```

```
{

    //Some trials to find the next link of path are made
    for (int w = 0; w < maxNumberOfTrials; w++)
    {
        direction = direction0;
        path = path0;
        thisPoint = thisPoint0;
        //A random non backwards new direction is chosen
        direction = notBackwards(direction);
        path = path + direction.toString();


        //if path is not suitable
        if ( isPathRepeated(path))
        {

            if (printTryToGetPath)
                System.out.println("    Path = "
                        + path + " YES repeated "
                + "TRY again");

        }
        else//path is suitable: is point occupied?
        {
            if (printTryToGetPath)
                System.out.println("    Path = "
                        + path + "  is NOT repeated ");

            //new direction is executed
            thisPoint = thisPoint.add(move3D(direction));
            //if point is not suitable
            if ( isOccupied(thisPoint) )
            {
                if (printTryToGetPath)
                    System.out.println(
                        "   Point = " + thisPoint
                                + "  is YES occupied."
                    + " Try AGAIN");
```

```
                }//Path and Point are suitable:
                else//Path and Point get registered
                {

                    Occupancy.add(thisPoint);
                    if (printTryToGetPath)
                        System.out.println(
                            "   Point = " + thisPoint
                            + "  is NOT occupied."
                        +"    partial Path YES has been found.");
                    direction0 = direction;
                    path0 = path;
                    thisPoint0 = thisPoint;
                    //System.out.println("·····");
                    break; //get out of trials' loop
                }
            }



        }//End of trials' loop

        //System.out.println("////////");

    }//End of concatenation's loop
    //New path gets registered
    if (oldPath.contentEquals(path0));
        else
        {
            TreeInfo.set(i, addressI + "aa" + path0);
            updateNodes(i);
            if (printTryToGetPath)
            printAll( "at TryToGetPath after updateNodes");
        }



    if (path0.contentEquals("")) path0 = "null";
```

```
    return path0;
}

//If a paths exists, all its subpaths are erased
private  void deleteSubpaths(int m)
{
   String path = getPath(m);
    //If the path to   node m is 12345,
    //a subpath is  a left substring:
    //1, 12, 123, 1234.
    //All of them must be deleted from TreeInfo

    for(int i = 1; i < path.length();  i++)
    {
        String pattern
                = path.substring(0, i);
        for(int j = TreeInfo.size()-2; j >=0;   j = j-2)
        {
            if (printDeleteSubpaths)

                System.out.println (i + "  " + j
                    + " Pattern = " + pattern
                    + "  pathj = " + getPath(j));
            if (pattern.contentEquals(getPath(j)))
            {
                TreeInfo.remove(j);
                TreeInfo.remove(j);
                if (printDeleteSubpaths)
                    System.out.println ("Deleted");
            }

        }
    }

}


//If a paths exists, all its subpaths are erased
private  void simplifyPaths()
```

```
{

   int i;
    for(i = TreeInfo.size()-2; i > 0;   i = i-2)
    {


        if (getPath(i).contentEquals(""))
        {
            TreeInfo.remove(i);
            TreeInfo.remove(i);

        }


    }

    int k = TreeInfo.size()-2;

    for(int m = k; m > 0;   m = m-2)
    {
        if (m > TreeInfo.size()-2) m = TreeInfo.size()-2;
        deleteSubpaths(m);

    }




}

//Information is created to realize
//and draw tree into 3D space.
private  void treeTo3D()
{
    /* Tiny test
    TreeInfo.clear();
    TreeInfo.add(0, "10");
    TreeInfo.add(1, "++");
    TreeInfo.add(2, "110");
```

```
TreeInfo.add(3, "++");
TreeInfo.add(4, "++");
TreeInfo.add(5, "++");
TreeInfo.add(6, "1110");
TreeInfo.add(7, "++");*/


if (printTreeTo3D)
    System.out.println("treeTo3D()");
int size = TreeInfo.size();
//Directions are proposed at random,
//and concatenated if no double occupancy appears
for (int i = 0; i < size; i= i+2)
{
    if (printTreeTo3D)
        System.out.println(
                "Input TreeInfo  Node " + i
            + " =   " + TreeInfo.get(i));
    counterOfTrialsToFindPath= 0;
    String addressIn = getAddress(i);
    //A realization as a 3D path of
    //addressIn  is looked for
    String path = tryToGetPath(i);
    if (printTreeTo3D)
        System.out.println("Output,   TreeInfo  Node   "
            + i +  " =   " + TreeInfo.get(i)
            + "  Address = " + getAddress(i)
            + "  Path = "  + getPath(i)
            + " Returned path  "  +  path);
    String addressOut = getAddress(i);
    if (printTreeTo3D)
        if (addressIn.length() == addressOut.length());
        else
            System.out.println("MUTILATION else NOISE");
    size = TreeInfo.size();
}
if (printTreeTo3D)
{
    System.out.println(
        "     TreeInfo.size() = " + TreeInfo.size());
```

```
        printOccupancy(Occupancy, "from treeTo3D()");
    }
    simplifyPaths();

}


}//End of class ABinaryTree




//=====================================
//============Evolutionary part========
//=====================================



//Population is initialized.

private    int initialPopulation( )
{
    //if(printInitialPopulation)
        System.out.println("Initial population");
    for(int i = 0; i <  nIndividuals; i++)
    {

        Pop[i]  = new ABinaryTree();
        if(printInitialPopulation)
            System.out.println("i = " + i
                    + "  BitTree = " + Pop[i].bitTree);
    }
    //The fitness of each individual is assessed
    assessFitness();
    //The best is returned
    int theBest = findTheBest();
    if(printInitialPopulation)
        Pop[theBest].printAll("The best of "
                + "Initial population");
```

```
    Pop2 = Pop;
    return theBest;
}



private   int evolution()
{
    printParameters();


    //Initial population.
    int theBest = initialPopulation( );
    if (isTaskDone(theBest)) nGenerations = 0;
    //Loop for nGenerations
    for(int i = 0; i < nGenerations; ++i)
    {
        gen = i;
        oneGeneration();

        //The best is returned
        theBest = findTheBest();
        if (isTaskDone(theBest)) break ;
    }
    if (printEvolution)
    {
        System.out.println();
        Pop[theBest].printAll("The final "
                + "Best after evolution");
    }
    return theBest;
}



//One generation = expression + assessement
//+ reproduction,
//recombination and mutation.
//The index of the best exponent is returned
```

```
private   void  oneGeneration()
{

    System.out.println("\nGeneration: " + gen);


    reproduction();

    recombination();

    mutation();



    //The fitness of each individual is assessed
    assessFitness();




    /*
    //report
    Pop[theBest].printOccupancy("from oneGeneration"
            + " after assessFitness() " );
    */

}



private int  findTheBest()
{
    double max = 0;
    int theBest = 0;
    for(int i = 0; i <  nIndividuals; i++)
    {

        if (Pop[i].fitness > max)
        {
            max = Pop[i].fitness;
```

```
            theBest = i;
        }
        if(printFindTheBest)
            System.out.println(i + " Fitness = "
                + Pop[i].fitness  + " max = " + max);
    }
    if(printFindTheBest)
        System.out.println("TheBest = " + theBest
            + " Fitness = " + Pop[theBest].fitness
                + " max = " + max);

    return theBest;
}

//Task: for a cube, populate lateral and inner points
//that have integer coordinates
private   boolean isTaskDone(int theBest)
{
    boolean b = false;
    //The root is given for granted
    int n = Pop[theBest].Occupancy.size() + 1;
    if (n >= RequiredNumberOfOccupiedPoints)
    {
        b = true;
        System.out.println("Task is accomplished");
    }

    double AchievedFractOfPoints = n;
    double d = totalNumberOfPoints;
    AchievedFractOfPoints = n/d;
    System.out.println(
        "\nThe best of generation " + gen
        + "\nTotalNumberOfPoints in the cube = "
        + totalNumberOfPoints
        + "\nNumber of occupied points = " + n
        + "\nRequired number of occupied points =  "
        + RequiredNumberOfOccupiedPoints
        + "\nAchieved Fraction of points = "
        + AchievedFractOfPoints
        + "\ndemanded Fraction of points = "
```

```
        + demandedFractOfPoints

        );
    return b;
}


//The fitness of a tree is the number of sites
//in the cube that it can occupy.
//Sites are listed in Occupancy.
private void  assessFitness()
{
    for(int i = 0; i <  nIndividuals; i++)
    {
        //TreeInfo is filled in.
        //If bit i of bitTree is 1,
        //node i is added to the tree.
        //A binary address is added to each node of the tree,
        //which is followed by a content in TreeInfo.
        Pop[i].fillTreeInfo(Pop[i].bitTree);
        Pop[i].simplifyTree();
        //Information is created to realize
        //and draw tree into 3D space
        Pop[i].treeTo3D();
        /*
        Pop[i].printOccupancy("from oneGeneration"
            + " after treeTo3D() ++++++ "  + i);
        */
        double totalNumberOfPointsD = totalNumberOfPoints;
        Pop[i].fitness
            = Pop[i].Occupancy.size()/totalNumberOfPointsD;
        if (printAssessFitness)
            System.out.println("\nINDIVIDUAL  = " + i
                    + " Fitness = "  + Pop[i].fitness);
    }
}


//Recombination is enabled.
//Indivual i-th is recombined with
```

```
//another one chose at random
private  String rec(int i)
{

    String s = Pop[i].bitTree;

    int sl= s.length();
    //First recombinant substring
    String  s1= "";
    //Place of recombination
     int m1 = 0;
    if (sl > 0)
    {
        m1 = r.nextInt(sl) ;
        s1= s.substring(0,m1);
    }
    //Partner for recombination
    int c = r.nextInt(nIndividuals);
    String t = Pop[c].bitTree;
    int tl= t.length();
    //Second recombinant substring
    String   t2;
    if (m1 < tl)
    {
        t2 = t.substring(m1);//Recombination
    }
    else t2 = s.substring(m1);//No recombination
    String rec = s1 + t2;
    if (printRec)
        System.out.println(i
                + "\nInput 1    = " + s
                + "\nInput 2    = " + t
                + "\nSubstring1 = " + s1
                + "\nSubstring2 = " + t2
                + "\nOutput     = " + rec);
    //Recombinant
    return rec;
}
```

```
//The definitive population  is a recombinant one.
private  void recombination()
{
    if (printRec)
        System.out.println("RECOMBINATION");
    for(int i = 0; i <  nIndividuals; i++)
    {
        //
        Pop2[i].bitTree = rec(i);

    }


    //Transfer to Pop[]
    Pop = Pop2;
    /*
    for(int i = 0; i <  nIndividuals; i++)
    {
        Pop[i].bitTree = Pop2[i].bitTree;
    }
    */
}


//Mutation of individual Pop[i]
private  String mut(int j)
{
    String s = Pop[j].bitTree;
    String mutant = "";
    for(int i = 0; i <  s.length(); i++)
    {
        char c = s.charAt(i);
        double p = r.nextDouble();
        if (p < pointMutationRate)
        {

            if (c == '1') c = '0';
            else c = '1';
```

```
        }

  mutant = mutant + c;
    }
    if (printMut)
            {
                System.out.println("Input  = " + s

                + "\nmutant = " + mutant);

            }
    return mutant;
}




private  void mutation()
{
    //Mutants are saved to Pop2[]
    for(int i = 0; i <  nIndividuals; i++)
    {
        //
        Pop2[i].bitTree = mut(i);

    }

   //Transfer to Pop[]
    Pop = Pop2;
    /*
    for(int i = 0; i <  nIndividuals; i++)
    {
        Pop[i].bitTree = Pop2[i].bitTree;
    }*/
}


/*  Reproduction by selection:
    The next generation consists of
```

```
    individuals that are sampled at random
    from the old population
    and cloned
    with a probability that is
    proportional to its fitness.
    */
private void reproduction()
{
    if (printReproduction)
    {
        System.out.println("\nReproduction by selection");
    }
    //Numbers of clones of each individual
    int count[] = new int[nIndividuals];
    //The probability of reproduction
    //is proportional to fitness.
    //Cumulative distribution
    double cumulate[] = new double[nIndividuals];
    cumulate[0] =   Pop[0].fitness;
    for (int i = 1; i < nIndividuals; i++)
        cumulate[i]= cumulate[i-1] + Pop[i].fitness;


    if (printReproduction)
    {
        System.out.println( "i \tfitness cumulate ");
        for (int i = 0; i < nIndividuals; i++)
        {
            System.out.println( i + "\t"
                    + Pop[i].fitness + " \t "
                    + cumulate[i]);
        }
    }

    //Maximal value
    double max = cumulate[nIndividuals-1];
    //Counter of chosen individuals
    int W = 0;

    //System.out.println("Generated numbers: ");
```

```
for (int i = 0; i < nIndividuals;i++)
{
    //A random number less than max is generated
    //with a uniform distribution
    double number = max * r.nextFloat();
    if (printReproduction)
        System.out.println("Random number =  " + number);
    //Probability of cloning is made
    //in proportion to fitness.
    //Is indiv 0 to be chosen?
    if (number < cumulate[0])
    {
        if (printReproduction)
            System.out.println(
                    "chosen individual  = 0 ");
        count[0]++;
        //New population
        Pop2[W] = Pop[0];
        W++;
    }
    //Is worm l to be chosen?
    for (int j=0; j < nIndividuals-1;j++)
    {

        int l = j+1;
        if ( ( cumulate[j] <= number)
                & (number < cumulate[l]) )
        {
            if (printReproduction)
                System.out.println(
                    "Chosen individual = " + l);
            count[l]++;
            //New population
            Pop2[W] = Pop[l];
            W++;
        }
    }

}
```

```
   //Output
    if (printReproduction)
    {
    System.out.println("\nEvents and their observed "
            + "and expected frequencies: ");
    int sum1 = 0;
    Double sum2 = 0.0;
    for (int i=0; i < nIndividuals;i++)
    {
        System.out.println(i + "\t"
            + count[i] + "\t"
            +  Pop[i].fitness
                *nIndividuals/cumulate[nIndividuals-1]);
        sum1 = sum1 + count[i];
        sum2 = sum2 + Pop[i].fitness
                *nIndividuals/cumulate[nIndividuals-1];

    }
    System.out.println("Totals  " + sum1 + "\t"
                +  sum2);
    }
    //New population becomes current one
    Pop = Pop2;
}




//================================
//======= 3D graphics ===========
//================================



//Instantiation of the  3D graphics class
private Node create3dContent(ABinaryTree t )
{
    Project3D  p =  new Project3D( t );
```

```
    return new Group(p);
}


//The drawing is synthesized
private class Project3D extends Group
{

    Group myGroup  = new Group();

//Constructor
Project3D(ABinaryTree t)
{
    //Tree is drawn to expand a 3D bush
    Group g = drawTree(t );
    getChildren().addAll(g );
}

//Returns a 3D line that goes from lastPoint
//to thisPoint
private  Line3D  line3DTo(Point3D thisPoint)
{
    Line3D l = new Line3D(lastPoint, thisPoint);
    lastPoint = thisPoint;
    return l;
}


private Point3D executePath(ABinaryTree t, String path)
{
    Point3D   thisPoint = new Point3D(0, 0,0);
    if (path.contentEquals(""));
    else
    {
        if (printExecutePath)
                System.out.println("\npath = " + path);
        //Reinitialization
        lastPoint = thisPoint;
        for (int j = 0; j < path.length(); j++)
        {
```

```
            //char j is read as integer
            direction = Integer.parseInt(
                    "" + path.charAt(j));

            //thisPoint moves  along the chosen direction
            thisPoint = thisPoint.add(t.move3D(direction));
            if (printExecutePath)
                System.out.println(" direction = " + direction
                        + " thisPoint HHHHH = " + thisPoint);
            t.Occupancy.add(thisPoint);
            //Draw a stroke
            //from lastPoint to thisPoint
            Line3D l = line3DTo(thisPoint);
            myGroup.getChildren().add(l.group);

        }

    }

    return thisPoint;
}



private   Group drawTree(ABinaryTree t )
{
    if (printDrawTree)
        t.printOccupancy(t.Occupancy,
                "drawTree at start line");
    //t.printAll("DrawTree");


    //This is the root of the tree
    Point3D   rootOfTree = new Point3D(0, 0,0);


    //Put a landmark at the root
    Sphere sphere = new Sphere();
    sphere.setRadius(10);
```

```
    sphere.setTranslateX((int) rootOfTree.getX());
    sphere.setTranslateY((int) rootOfTree.getY() );
    sphere.setTranslateZ((int) rootOfTree.getZ() );
    root.getChildren().add(sphere);

    for (int i = 0; i < t.TreeInfo.size(); i= i+2)
    {
        //Reinitialization
        lastPoint = rootOfTree;
        Point3D thisPoint;
        String path = t.getPath(i);
        if (printDrawTree)
            System.out.println("Index number " + i  + "  =  "
                + t.TreeInfo.get(i) + " path  " + path);
        if (path.contentEquals(""));
        else
        {
            thisPoint = executePath(t, path);
            t.Occupancy.add(thisPoint);
            if (printDrawTree)
                System.out.println();
        }
    }



    //t.printOccupancy(t.Occupancy, "DRawTree at the end");
    return myGroup;
}
}//End of inner class Project3D



public static void main(String[] args)
{
    launch(args);
}

}//End of program N114  TreeToShapes17
```

115 ***Exercise.*** *Run the previous program and play with the code.*

116 ***Exercise.*** *Use the previous program to test the following claims of the Author:*

1. *We have proposed and simulated the following theory of the arising of biological form: cell reproduction is genetically instructed to follow a binary pattern that stops reproduction along each branch after a perfectly predefined number of divisions. Cellular division poses daughter cells at some place in 3D space but this is not encoded. So, position is 3D space of each cell is a matter of randomness. It is supposed that some process gives from above the directives for the binary pattern, say, the shape of a bone is determined by its performance in overall movements. This constraint plus a tree structure results in pliotropy and chaos: a tiny change in any given place may give rise to a completely different shape. In regard with this theory, we have:*

2. *Randomness alone can explain very simple shapes that are modeled here by cubes whose edge measures one, two or three units.*

3. *Randomness and evolution can easily explain poorly cooked solutions to the problem of fitting shapes by binary trees.*

4. *A disordered tree has short strokes and many turns in contrast with ordered trees that has long strokes and few turns. Randomness and evolution produce in general disordered trees but for tiny cubes ordered trees sometimes appear. So, evolution predicts extremely disordered process of shape forging but by contrast ontological processes depicted by embryology seem to be extremely ordered.*

5. *Needed time of our algorithm to get complete execution rapidly overwhelms patience. This immediately implies that evolution predicts a huge amount of fossils with crooked shapes.*

117 ***Graduation.*** *Our algorithm is composed of two phases. In the first, the binary structure of a tree is defined and, in the second, a 3D realization of the resultant binary structure is looked for. The 3D realization is made arm by arm, each arm fully researched, serially. This process causes that the length of 3D arms decreases in function of its order of presentation. Now, this does not sound like embryology (although the brain and the eye have long connectors and cancer in metastasis throw very long branches). Instead, we imagine that in general cell reproduction happens at the same time in many fronts so, reproduction runs more in parallel*

than serially. So, your task is to switch from the simulation of serial reproduction to parallel one. Use your new code to test anew the aforementioned claims of the Author: all must remain true.

4.5 Conclusion

The excellent implementation of deformable meshes in JavaFX enabled a straightforward proof that mutation is extremely powerful to erode the form of shapes. This would imply that shapes must be maintained actively in biology: natural selection would immediately solve the problem were not for the fact that it is absent in a very important case: the human hand. The reason is that human mothers seem to care for not severely defective babies even more than for perfect ones. Our first option to solve the puzzle of shape constancy and living fossils was to adhere to the belief that recombination is a palliative remedy if only an already perfect shape is in action. To go beyond this we propose to invoke the conservative power of pleitropy that arises naturally from constrained cell reproduction. Our preliminary results in this regard are impressive: evolution predicts extremely disordered process of shape forging but by contrast ontological processes depicted by embryology seem to be extremely ordered. Furthermore, our algorithms run very slowly so running them rapidly overwhelms patience. This immediately implies that evolution predicts a huge amount of fossils with crooked shapes.

Chapter 5

On the evolution of biological shape

To write this chapter as a tentative conclusion of the work along this volume is offered as a challenge for the reader.

Chapter 6

Answers to exercises

Problems of Chapter 1

4, page 10. . To enlarge a cube just in the x-direction, we need to change the implementation of the changePoints() method by the following one

```
//Program N4 EvoBrick2
....
public class EvoBrick2 extends Application
....
private void changePoints()
{
    int n = mesh.getPoints().size();
    float [] pointsFloat = new float[n];
    float factor = 1.001f;
    for(int i = 0; i < n; i = i+3)
    {
        float x =  mesh.getPoints().get(i);
        float y =  mesh.getPoints().get(i+1);
        float z =  mesh.getPoints().get(i+2);
        //Only the x-coordinate is enlarged
        pointsFloat[i]    = factor*x;
        pointsFloat[i+1]  = y;
        pointsFloat[i+2]  = z;
    }
```

```
    mesh.getPoints().setAll(pointsFloat, 0, n);
}
```

5, page 10. The code presents an animation of a cube in which the texture softly evolves over time from covering a quarter of the image to its full size. The method that makes the change in the texture coordinates follows:

```
//Program N5 EvoBrick3
....
public class EvoBrick3 extends Application
....
//The texture of the mesh gets transformed.
//To that aim, the coordinates of texture points are read,
//modified and then reinserted into the mesh
 private void changeTexture()
{
    boolean go = true;
    int n = mesh.getTexCoords().size();
    float [] coordFloat = new float[n];
    float factor = 1.001f;
    //Texture coordinates must not surpass 1
    for(int i = 0; i < n; i++)
    {
        coordFloat[i]    = mesh.getTexCoords().get(i);
        //System.out.println(coordFloat[i]);
        if (coordFloat[i] == 1) go = false;
    }
    //Reading and modifying.
    if (go)
    {
        for(int i = 0; i < n; i++)
        {
            coordFloat[i]    = mesh.getTexCoords().get(i);
            float f    = factor * coordFloat[i];
            if (f <= 1) coordFloat[i]    = f;
            else coordFloat[i]    = 1;
```

```
        }
    }
    //Reinsertion
    mesh.getTexCoords().setAll(coordFloat, 0, n);
}
```

8, page 17. The following program shows a planet with rotation and translation.

```
/*
    Program N8 Planet2.
    Planet Earth is modelled as
    a spherical mesh with texture.
    It can rotate about itself and
    translate along an orbit.
 */


package Programs;
//Files in the hosting project
import Tools.Axes;
import Tools.Cameras;
import Tools.Xform;
//Files in JavaFX
import javafx.animation.Animation;
import static javafx.animation.Animation.INDEFINITE;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.SequentialTransition;
import javafx.animation.Timeline;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.paint.Color;
```

```
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.CullFace;
import javafx.scene.shape.DrawMode;
import javafx.scene.shape.MeshView;
import javafx.scene.shape.TriangleMesh;
import javafx.stage.Stage;
import javafx.util.Duration;


public class Planet2 extends Application
{

    private Timeline animation;
    final Group root = new Group();
    Cameras camera = new Cameras();
    Axes axes = new Axes();




    private static final boolean print = false;
    //The radius of the mother circular orbit
    private static final Integer radiusOrbit = 100;
    //Angular velocity
    private static final double omega = 0.001;
    //Paramters of the ellipse
    private static final double a = 2;
    private static final double b = 1;
    //Time is discrete
    private static Integer t = 0;


@Override
public void start(Stage primaryStage)
{
    primaryStage.setTitle("Planet Earth 3D");
    //The whole drawing  is created
    //as a JavaFX space-like object
```

```
    final Group parent = new Group(create3dContent());

    //Window of the scene
    final Scene scene = new Scene(root, 900, 900, true);
    //Background
    scene.setFill(Color.BLACK);

    camera.buildCamera();
    camera.handleMouse(scene);
    primaryStage.setScene(scene);
    primaryStage.show();
    axes.buildAxes();
    axes.handleKeyboard(scene);
    scene.setCamera(camera.camera);
    root.getChildren().addAll(parent, axes.axesXform);
}


//Manager of graphic
private Node create3dContent()
{
    //Shape
    Planet p = new Planet();


    return new Group(p);
}

public void play()
{
    animation.play();
}


@Override public void stop()
{
    animation.pause();
}
```

```
//A planet with rotation + translation
private class Planet extends Group
{


//Constructor
//A mesh is converted into an anmated meshView.
private  Planet()
  {

    //Planet Earth as a meshView with texture
    Xform planet = MeshSphere(100f,10, Color.GREEN);


    animation = new Timeline();
    //Rotation
    animation.getKeyFrames().addAll
    (
        new KeyFrame//Initial states at initial time
        (
            Duration.ZERO,
            //new KeyValue(b.ry.angleProperty(), 0d),
            //new KeyValue(b.rx.angleProperty(), 0d),
            new KeyValue(planet.rz.angleProperty(), 0d)
        ),
        new KeyFrame//Final states at final time
        (
            Duration.seconds(10),
            //new KeyValue(b.ry.angleProperty(), 360d),
            //new KeyValue(b.rx.angleProperty(), 360d),
            new KeyValue(planet.rz.angleProperty(), 360d)
        )
    );
    animation.setCycleCount(Animation.INDEFINITE);


    animation.play();
```

```
//Translation
//Duration of each iteration
Duration duration = Duration.millis(1);

//Transitions are nested:
//a translation tT is nested into
//a sequential transition s
//that defines the kinematics
//for one cycle = one year.
SequentialTransition s = new SequentialTransition();
s.setCycleCount(INDEFINITE);
//Iterations for one cycle.
//6275 found by trial and error to complete a cycle
for(int i = 0; i < 6275; i++)
{
    //Shifting values for an ellipse
    double byX =  - a*omega*radiusOrbit
                * Math.sin(omega* t);
    double byY =  b*omega*radiusOrbit
                * Math.cos(omega* t);
    //Translation of the planet
    TranslateTransition tT =
         new TranslateTransition(duration, planet);
    tT.setByX(byX);
    tT.setByY(byY);
    tT.setCycleCount(1);
    //Nesting of tT into s
    s.getChildren().add(tT);
    //Time updating
    t++;
}
s.play();

//Booking
getChildren().addAll(planet, axes.axesXform);
```

```
  }
}//End of class Graphic



//The mesh of a sphere.
//Pi has  n subdivisions
 Xform MeshSphere(float rho,
        int nSubdivisions,
        Color color)
{
    TriangleMesh mesh = new TriangleMesh();

    float v = 0;

    float deltaAngle = (float) Math.PI /nSubdivisions;

    System.out.println(deltaAngle);
    //Faces are built.
    //Two triangles for each rectangle.
    int n = nSubdivisions;
    int m = 2*n;
    //Points of mesh and of texture are built
    for (int j = 0; j <= n ; j++)
    {
        float u = 0;
        for (int i = 0; i <= m ; i++)
        {

            //if (j == 0) sign = 1; else sign = -1;
            //A minus is inserted to get the correct orientation
            //of resultant triangles
            float x
                = (float) (rho * Math.sin(v) * Math.cos(-u));
            float y
                = (float) (rho * Math.sin(v) * Math.sin( -u));
            float z
```

```
            = (float) (rho * Math.cos(v));


        mesh.getPoints().addAll(x,y,z);
        //Texture coordinates
        float tx = i* 1.0f/m;
        float ty = j* 1.0f/n;
        //Booking
        mesh.getTexCoords().addAll(tx, ty);

        if(print)
            System.out.println(
                x + "  "  + y + "  "  + z + "  " +
                tx + "  "  + ty + "  " );

        u  = u + deltaAngle ;

    }
    v =  v + deltaAngle ;
}


//There are nm tiny rectangles.
//Each rectangle has 2 triangles

for (int j = 0; j < n; j++)
{
    for (int i = 0; i < m; i++)
    {
        //Points of rectangle of the mesh
        int p00 = j * (m+1) + i;
        int p01 = p00 + 1;
        int p10 = p00 + (m+1);
        int p11 = p10 + 1;
        //Points of rectangle of texture
        int tc00 = j * (m+1) + i;
        int tc01 = tc00 + 1;
        int tc10 = tc00 + (m+1);
```

```
            int tc11 = tc10 + 1;

            if ( j < n-1)
            {
                //First triangle
                mesh.getFaces().addAll(
                        p00, tc00, p10, tc10, p11, tc11);
                //second triangle
                mesh.getFaces().addAll(
                        p11, tc11, p01, tc01, p00, tc00);
            }
            else
            //second triangle alone
            mesh.getFaces().addAll(
                    p11, tc11, p01, tc01, p00, tc00);
        }
    }


     //Wallpaper is gotten
    //Mapa Mundi clip art, shared by Mirian 06-13-2013
    //http://www.clker.com/clipart-mapa-mundi.html
    Image wallpaper = new Image(getClass()
            .getResourceAsStream("PlanetEarth.png"));
    //Declaration of material for painting the pyramid
    PhongMaterial material = new PhongMaterial();
    //Wallpaper is declared as covering material
    material.setDiffuseMap(wallpaper);
    //material.setDiffuseColor(Color.ALICEBLUE);
    MeshView  meshView = new MeshView(mesh);
    meshView.setMaterial(material);
    //meshView.setDrawMode(DrawMode.LINE);
    meshView.setDrawMode(DrawMode.FILL);
    meshView.setCullFace(CullFace.BACK);


    final Xform meshViewXform = new Xform();
```

```
    meshViewXform.getChildren().add(meshView);
    //Initial position
    meshViewXform.setTranslate(radiusOrbit, 0, 0);
    //Initial inclination in degrees
    meshViewXform.rx.setAngle(0);
    meshViewXform.ry.setAngle(50);
    meshViewXform.rz.setAngle(0);
    return meshViewXform;
}


public static void main(String[] args)
{

    launch(args);
}

}//End of main class  N8 Planet2
```

9, page 17. A round planet evolves into an enlarging ellipsoid:

```
/*
    Program N9 Planet3.
    Planet Earth is modelled as
    a spherical mesh with texture.
    It can rotate about itself and
    translate around a sun in an elliptic orbit.
    The earth evolves into an ellipsoid
    but, with this code,
    the coordinates of the points of the mesh
    cannot be written.
    The reason is that printing is in a thread
    with low priority so,
    its fields might be updated but not  necessarily
    or not at the right time.
 */
```

```
package Programs;
//Files in the hosting project
import Tools.Axes;
import Tools.Cameras;
import Tools.Xform;

//Files in JavaFX
import javafx.animation.Animation;
import static javafx.animation.Animation.INDEFINITE;
import javafx.animation.AnimationTimer;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.SequentialTransition;
import javafx.animation.Timeline;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.CullFace;
import javafx.scene.shape.DrawMode;
import javafx.scene.shape.MeshView;
import javafx.scene.shape.Sphere;
import javafx.scene.shape.TriangleMesh;
import javafx.stage.Stage;
import javafx.util.Duration;


public class Planet3 extends Application
{

    private static Timeline animation;
    //The texture of the mesh changes over time
    //private static AnimationTimer timer;
```

```
    private final Group root = new Group();
    Cameras camera = new Cameras();
    Axes axes = new Axes();



    private static final boolean print = false;
    //The radius of the mother circular orbit
    private static final Integer radiusOrbit = 100;
    //Angular velocity
    private static final double omega = 0.001;
    //Paramters of the ellipse
    private static final double a = 2;
    private static final double b = 1;
    //Time is discrete
    private static Integer t = 0;

    //The radius of the sun
    private static final Integer rSun = 10;



@Override
public void start(Stage primaryStage)
{
    primaryStage.setTitle("Planet Earth 3D");
    //The whole drawing  is created
    //as a JavaFX space-like object
    final Group parent = new Group(create3dContent());

    //Window of the scene
    final Scene scene = new Scene(root, 900, 900, true);
    //Background
    scene.setFill(Color.BLACK);

    camera.buildCamera();
```

```
    camera.handleMouse(scene);
    primaryStage.setScene(scene);
    primaryStage.show();
    axes.buildAxes();
    axes.handleKeyboard(scene);
    scene.setCamera(camera.camera);
    root.getChildren().addAll(parent, axes.axesXform);
}


//Manager of graphic
private Node create3dContent()
{
    //Shape
    Planet p = new Planet();

    Sphere sun = new Sphere(rSun);
    PhongMaterial material = new PhongMaterial(Color.GOLD);
    sun.setMaterial(material);
    //Position
    sun.setTranslateX(0);
    sun.setTranslateY(0);
    sun.setTranslateZ(0);
    sun.setCullFace(CullFace.BACK);



    return new Group(p, sun);
}

public void play()
{
    animation.play();
}


@Override public void stop()
{
```

```
    animation.pause();
}

//A planet with rotation + translation
private class Planet extends Group
{

private  TriangleMesh mesh;

//Constructor
//A mesh is converted into an anmated meshView.
private  Planet()
  {

    //Planet Earth as a meshView with texture
    Xform planet = MeshSphere(30f,10);


    animation = new Timeline();
    //Rotation
    animation.getKeyFrames().addAll
    (
        new KeyFrame//Initial states at initial time
        (
            Duration.ZERO,
            //new KeyValue(b.ry.angleProperty(), 0d),
            //new KeyValue(b.rx.angleProperty(), 0d),
            new KeyValue(planet.rz.angleProperty(), 0d)
        ),
        new KeyFrame//Final states at final time
        (
            Duration.seconds(10),
            //new KeyValue(b.ry.angleProperty(), 360d),
            //new KeyValue(b.rx.angleProperty(), 360d),
            new KeyValue(planet.rz.angleProperty(), 360d)
        )
    );
    animation.setCycleCount(Animation.INDEFINITE);
```

```
animation.play();


//Translation
//Duration of each iteration
Duration duration = Duration.millis(1);

//Transitions are nested:
//a translation tT is nested into
//a sequential transition s
//that defines the kinematics
//for one cycle = one year.
SequentialTransition s = new SequentialTransition();
s.setCycleCount(INDEFINITE);
//Iterations for one cycle.
//6275 found by trial and error to complete a cycle
for(int i = 0; i < 6275; i++)
{
    //Shifting values for an ellipse
    double byX =  - a*omega*radiusOrbit
                * Math.sin(omega* t);
    double byY =  b*omega*radiusOrbit
                * Math.cos(omega* t);
    //Translation of the planet
    TranslateTransition tT =
         new TranslateTransition(duration, planet);
    tT.setByX(byX);
    tT.setByY(byY);
    tT.setCycleCount(1);
    //Nesting of tT into s
    s.getChildren().add(tT);
    //Time updating
    t++;
}
s.play();
```

```
    //Booking
    getChildren().addAll(planet, axes.axesXform);

  }




//The mesh of a sphere of radius rho
//Pi has  n subdivisions
 Xform MeshSphere(float rho,
        int nSubdivisions)
{
    mesh = new TriangleMesh();

    float v = 0;

    float deltaAngle = (float) Math.PI /nSubdivisions;

    System.out.println(deltaAngle);
    //Faces are built.
    //Two triangles for each rectangle.
    int n = nSubdivisions;
    int m = 2*n;
    //Points of mesh and of texture are built
    for (int j = 0; j <= n ; j++)
    {
        float u = 0;
        for (int i = 0; i <= m ; i++)
        {

            //if (j == 0) sign = 1; else sign = -1;
            //A minus is inserted to get the correct orientation
            //of resultant triangles
            float x
                = (float) (rho * Math.sin(v) * Math.cos(-u));
            float y
```

```
                = (float) (rho * Math.sin(v) * Math.sin( -u));
            float z
                = (float) (rho * Math.cos(v));


            mesh.getPoints().addAll(x,y,z);
            //Texture coordinates
            float tx = i* 1.0f/m;
            float ty = j* 1.0f/n;
            //Booking
            mesh.getTexCoords().addAll(tx, ty);

            if(print)
                System.out.println(
                    x + "  "  + y + "  "  + z + "  " +
                    tx + "  "  + ty + "  " );

            u  = u + deltaAngle ;

        }
        v =  v + deltaAngle ;
    }


    //There are nm tiny rectangles.
    //Each rectangle has 2 triangles

    for (int j = 0; j < n; j++)
    {
        for (int i = 0; i < m; i++)
        {
            //Points of rectangle of the mesh
            int p00 = j * (m+1) + i;
            int p01 = p00 + 1;
            int p10 = p00 + (m+1);
            int p11 = p10 + 1;
            //Points of rectangle of texture
            int tc00 = j * (m+1) + i;
```

```
            int tc01 = tc00 + 1;
            int tc10 = tc00 + (m+1);
            int tc11 = tc10 + 1;

            if ( j < n-1)
            {
                //First triangle
                mesh.getFaces().addAll(
                        p00, tc00, p10, tc10, p11, tc11);
                //second triangle
                mesh.getFaces().addAll(
                        p11, tc11, p01, tc01, p00, tc00);
            }
            else
            //second triangle alone
            mesh.getFaces().addAll(
                    p11, tc11, p01, tc01, p00, tc00);
        }
    }


    //Wallpaper is gotten
    //Mapa Mundi clip art, shared by Mirian 06-13-2013
    //http://www.clker.com/clipart-mapa-mundi.html
    Image wallpaper = new Image(getClass()
            .getResourceAsStream("PlanetEarth.png"));
    //Declaration of material for painting the pyramid
    PhongMaterial material = new PhongMaterial();
    //Wallpaper is declared as covering material
    material.setDiffuseMap(wallpaper);
    //material.setDiffuseColor(Color.ALICEBLUE);
    MeshView  meshView = new MeshView(mesh);
    meshView.setMaterial(material);
    //meshView.setDrawMode(DrawMode.LINE);
    meshView.setDrawMode(DrawMode.FILL);
    meshView.setCullFace(CullFace.BACK);
```

```
    final Xform meshViewXform = new Xform();

    meshViewXform.getChildren().add(meshView);
    //Initial position
    meshViewXform.setTranslate(radiusOrbit, 0, 0);
    //Initial inclination in degrees
    meshViewXform.rx.setAngle(0);
    meshViewXform.ry.setAngle(3);
    meshViewXform.rz.setAngle(0);



    //The mesh changes over time
    AnimationTimer timer = new AnimationTimer()
    {
        @Override
        public void handle(long l)
        {
            changePoints();
        }
    };

    timer.start();
    return meshViewXform;
}



//The earth becomes an ellipsoid so,
//its points change. Nevertheless,
//they are not allowed to be known
//by the printing thread.
 private void changePoints()
{
    int n = mesh.getPoints().size();
    float [] pointsFloat = new float[n];
    float factor = 1.001f;
    for(int i = 0; i < n; i = i+3)
```

```
    {
        float x =  mesh.getPoints().get(i);
        float y =  mesh.getPoints().get(i+1);
        float z =  mesh.getPoints().get(i+2);
        if (i == 36)
            System.out.println("i = " + i + "   x === " + x
            + "  " +  System.nanoTime());
        //Only the x-coordinate is enlarged
        pointsFloat[i]    = factor*x;
        pointsFloat[i+1]  = y;
        pointsFloat[i+2]  = z;
    }

    mesh.getPoints().setAll(pointsFloat, 0, n);
}

}//End of inner class Planet



public static void main(String[] args)
{

    launch(args);
}

}//End of main class N9 Planet3
```

12, page 19. A program that draws an image pixel by pixel but with a random color of the red tonality.

```
/*
 * Program N12 PixelLab2
 * A JavaFX application to built an image
 * pixel by pixel with a red palette.
 */
package pixellab2;
```

```
import java.util.Random;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.image.PixelWriter;
import javafx.scene.image.WritableImage;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;


public class PixelLab2 extends Application
{

@Override
public void start(Stage primaryStage)
{
    Image image = randomPixels(300,400);
    ImageView imageView = new ImageView();
    imageView.setImage(image);
    StackPane root = new StackPane();
    root.getChildren().add(imageView);
    Scene scene = new Scene(root, 300, 250);
    primaryStage.setTitle("Image is built pixel by pixel");
    primaryStage.setScene(scene);
    primaryStage.show();
}


//An image is built pixel by pixel
private Image randomPixels(int width, int height)
{
    WritableImage img = new WritableImage(width, height);
    PixelWriter   pw  = img.getPixelWriter();
    Random        rnd = new Random();
```

```
    for (int y = 0 ; y < height ; y++)
    {
        for (int x = 0 ; x < width ; x++)
        {
            //Pixel manipulation: red, green, blue.
            //Level of each color moves from 0 to 225.
            pw.setColor(x, y, Color.rgb(225,
                                        rnd.nextInt(255),
                                        rnd.nextInt(255)));
        }
    }
return img;
}


public static void main(String[] args)
{
    launch(args);
}

}//End of program N12 PixelLab2
```

16, page 36. To prove that the code is wrong it is enough to turn off the translation transition by silencing the corresponding playing command. The ensuing flux must show a periodic sequence as the Earth turns around but this does not happen.

22, page 66. From

$$a_n = \frac{\frac{x_{n+1}-x_n}{\Delta t_n} - \frac{x_n - x_{n-1}}{\Delta t_{n-1}}}{\frac{\Delta t_n + \Delta t_{n-1}}{2}}$$

we get

$$a_n \frac{\Delta t_n + \Delta t_{n-1}}{2} = \frac{x_{n+1}-x_n}{\Delta t_n} - \frac{x_n - x_{n-1}}{\Delta t_{n-1}}$$

$$\frac{x_{n+1}-x_n}{\Delta t_n} = a_n \frac{\Delta t_n + \Delta t_{n-1}}{2} + \frac{x_n - x_{n-1}}{\Delta t_{n-1}}$$

$$x_{n+1} = x_n + (x_n - x_{n-1})\frac{\Delta t_n}{\Delta t_{n-1}} + a_n \frac{\Delta t_n + \Delta t_{n-1}}{2}\Delta t_n.$$

23, page 66. To define initial conditions, we must give suitable values for 2 positions and 2 deltas of time. So, we take to begin with

$$\Delta t_0 = \Delta t_1 = 1ms$$
$$x_0 = (100,0,0)$$
$$x_1 = x_0 + \Delta x = (100,0,0) + (0,10,0)$$
$$x_{n+1} = x_n + (x_n - x_{n-1})\frac{\Delta t_n}{\Delta t_{n-1}} + a_n \frac{\Delta t_n + \Delta t_{n-1}}{2}\Delta t_n.$$
$$x_{n+1} - x_n = (x_n - x_{n-1})\frac{\Delta t_n}{\Delta t_{n-1}} + a_n \frac{\Delta t_n + \Delta t_{n-1}}{2}\Delta t_n.$$

24, page 66. We took program N18 Planet 5 and changed the part corresponding to Earth translation. This can be found in program N24 Planet 6. The pertinent changes read:

```
//Program N24 Planet6.
...

public class Planet6 extends Application
{

...


//JavaFX thread for  rotation + translation of the Earth
private void updateMesh()
{

    mesh.getPoints().setAll(rotPoints);

    //Duration of each iteration
    Duration duration = Duration.millis(deltaTn/3);

    //Transitions are nested:
    //a translation tT is nested into
    //a sequential transition s
    //that defines the kinematics
    //for one cycle = one year.
    SequentialTransition s = new SequentialTransition();
    s.setCycleCount(1);

    if (printUpdatePoints)
```

```
    System.out.println("time = " + time
            + "  deltaT = " + deltaTn);
Point3D centralUnitaryVector = centerOfEarthN
        .multiply(1/centerOfEarthN.magnitude());
//force = centralUnitaryVector
//      * (-k*massOfSun*massOfEarth/
//          (centerOfEarthN.magnitude()^2)  )
force = centralUnitaryVector
        .multiply(-k*massOfSun*massOfEarth/
            (centerOfEarthN.magnitude()
                * centerOfEarthN.magnitude() ));

acceleration = force.multiply(1.0/massOfEarth);

//term1 =  (centerOfEarthN- centerOfEarthNMinus1)
//*deltaTn/deltaTnMinus1
Point3D term1 = ( centerOfEarthN
        .subtract(centerOfEarthNMinus1) )
        .multiply(deltaTn/deltaTnMinus1);
//term2 = acceleration
//      * deltaTn * (deltaTn + deltaTnMinus1) /2)
Point3D term2 = acceleration
        .multiply(deltaTn * (deltaTn + deltaTnMinus1) /2);
// displacement = term1 + term2;
displacement = term1.add(term2);
System.out.println("Force.X = " + force.getX()

        + " displacement.X =  " + displacement.getX());


//centerOfEarthNPlus1 =
//Shifting values for an ellipse
double byX =  displacement.getX();
double byY =  displacement.getY();
double byZ =  displacement.getZ();
//Translation of the planet
TranslateTransition tT =
 new TranslateTransition(duration, planetXform);
tT.setByX(byX );
tT.setByY(byY);
```

```
    tT.setByZ(byZ);
    tT.setCycleCount(1);
    //Nesting of tT into s
    s.getChildren().add(tT);

    if (translation)
    {
        s.play();
        centerOfEarthNMinus1 = centerOfEarthN;
        centerOfEarthN = centerOfEarthN.add(displacement);
    }

    wallpaper = drawThermometers();

}
```

Problems of Chapter 2

27, page 69. A sphere (planet Earth) is disfigured by random mutations of the positions of points of its mesh. Disfiguring and only disfiguring is the output:

```
/*
    Program N27 DisfiguredSphere
 *  The points of a sphere ( planet Earth) are subjected to
    random mutations in tandem.
 */

package Programs;



//Files in the hosting project
import Tools.Axes;
import Tools.Cameras;
import Tools.Xform;
import java.util.Random;
```

```
//Files in JavaFX
import javafx.animation.Animation;
import static javafx.animation.Animation.INDEFINITE;
import javafx.animation.AnimationTimer;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.SequentialTransition;
import javafx.animation.Timeline;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.CullFace;
import javafx.scene.shape.DrawMode;
import javafx.scene.shape.MeshView;
import javafx.scene.shape.Sphere;
import javafx.scene.shape.TriangleMesh;
import javafx.stage.Stage;
import javafx.util.Duration;


public class DisfiguredSphere extends Application
{

    private static Timeline animation;
    //The texture of the mesh changes over time
    //private static AnimationTimer timer;
    private final Group root = new Group();
    Cameras camera = new Cameras();
    Axes axes = new Axes();
```

```
    private static final boolean print = false;
    //The radius of the mother circular orbit
    private static final Integer radiusOrbit = 100;
    //Angular velocity
    private static final double omega = 0.001;
    //Paramters of the ellipse
    private static final double a = 2;
    private static final double b = 1;
    //Time is discrete
    private static Integer t = 0;

    //The radius of the sun
    private static final Integer rSun = 10;

    private static  Random r = new Random();



@Override
public void start(Stage primaryStage)
{
    primaryStage.setTitle("Planet Earth 3D");
    //The whole drawing  is created
    //as a JavaFX space-like object
    final Group parent = new Group(create3dContent());

    //Window of the scene
    final Scene scene = new Scene(root, 900, 900, true);
    //Background
    scene.setFill(Color.BLACK);

    camera.buildCamera();
    camera.handleMouse(scene);
    primaryStage.setScene(scene);
    primaryStage.show();
    axes.buildAxes();
    axes.handleKeyboard(scene);
```

```
    scene.setCamera(camera.camera);
    root.getChildren().addAll(parent, axes.axesXform);
}


//Manager of graphic
private Node create3dContent()
{
    //Shape
    Planet p = new Planet();

    Sphere sun = new Sphere(rSun);
    PhongMaterial material = new PhongMaterial(Color.GOLD);
    sun.setMaterial(material);
    //Position
    sun.setTranslateX(0);
    sun.setTranslateY(0);
    sun.setTranslateZ(0);
    sun.setCullFace(CullFace.BACK);



    return new Group(p, sun);
}

public void play()
{
    animation.play();
}


@Override public void stop()
{
    animation.pause();
}

//A planet with rotation + translation
private class Planet extends Group
```

```
{

private  TriangleMesh mesh;

//Constructor
//A mesh is converted into an anmated meshView.
private  Planet()
  {

    //Planet Earth as a meshView with texture
    Xform planet = MeshSphere(30f,10);


    animation = new Timeline();
    //Rotation
    animation.getKeyFrames().addAll
    (
        new KeyFrame//Initial states at initial time
        (
            Duration.ZERO,
            //new KeyValue(b.ry.angleProperty(), 0d),
            //new KeyValue(b.rx.angleProperty(), 0d),
            new KeyValue(planet.rz.angleProperty(), 0d)
        ),
        new KeyFrame//Final states at final time
        (
            Duration.seconds(10),
            //new KeyValue(b.ry.angleProperty(), 360d),
            //new KeyValue(b.rx.angleProperty(), 360d),
            new KeyValue(planet.rz.angleProperty(), 360d)
        )
    );
    animation.setCycleCount(Animation.INDEFINITE);


    animation.play();
```

```
  //Translation
  //Duration of each iteration
  Duration duration = Duration.millis(10);

  //Transitions are nested:
  //a translation tT is nested into
  //a sequential transition s
  //that defines the kinematics
  //for one cycle = one year.
  SequentialTransition s = new SequentialTransition();
  s.setCycleCount(INDEFINITE);
  //Iterations for one cycle.
  //6275 found by trial and error to complete a cycle
  for(int i = 0; i < 6275; i++)
  {
      //Shifting values for an ellipse
      double byX =  - a*omega*radiusOrbit
                  * Math.sin(omega* t);
      double byY =  b*omega*radiusOrbit
                  * Math.cos(omega* t);
      //Translation of the planet
      TranslateTransition tT =
           new TranslateTransition(duration, planet);
      tT.setByX(byX);
      tT.setByY(byY);
      tT.setCycleCount(1);
      //Nesting of tT into s
      s.getChildren().add(tT);
      //Time updating
      t++;
  }
  s.play();

  //Booking
  getChildren().addAll(planet, axes.axesXform);

}
```

```
//The mesh of a sphere of radius rho
//Pi has  n subdivisions
 Xform MeshSphere(float rho,
        int nSubdivisions)
{
    mesh = new TriangleMesh();

    float v = 0;

    float deltaAngle = (float) Math.PI /nSubdivisions;

    System.out.println(deltaAngle);
    //Faces are built.
    //Two triangles for each rectangle.
    int n = nSubdivisions;
    int m = 2*n;
    //Points of mesh and of texture are built
    for (int j = 0; j <= n ; j++)
    {
        float u = 0;
        for (int i = 0; i <= m ; i++)
        {

            //if (j == 0) sign = 1; else sign = -1;
            //A minus is inserted to get the correct orientation
            //of resultant triangles
            float x
                = (float) (rho * Math.sin(v) * Math.cos(-u));
            float y
                = (float) (rho * Math.sin(v) * Math.sin( -u));
            float z
                = (float) (rho * Math.cos(v));
```

```
        mesh.getPoints().addAll(x,y,z);
        //Texture coordinates
        float tx = i* 1.0f/m;
        float ty = j* 1.0f/n;
        //Booking
        mesh.getTexCoords().addAll(tx, ty);

        if(print)
            System.out.println(
                x + "  "  + y + "  "  + z + "  " +
                tx + "  "  + ty + "  " );

        u  = u + deltaAngle ;

    }
    v =  v + deltaAngle ;
}


//There are nm tiny rectangles.
//Each rectangle has 2 triangles

for (int j = 0; j < n; j++)
{
    for (int i = 0; i < m; i++)
    {
        //Points of rectangle of the mesh
        int p00 = j * (m+1) + i;
        int p01 = p00 + 1;
        int p10 = p00 + (m+1);
        int p11 = p10 + 1;
        //Points of rectangle of texture
        int tc00 = j * (m+1) + i;
        int tc01 = tc00 + 1;
        int tc10 = tc00 + (m+1);
        int tc11 = tc10 + 1;

        if ( j < n-1)
```

```
            {
                //First triangle
                mesh.getFaces().addAll(
                        p00, tc00, p10, tc10, p11, tc11);
                //second triangle
                mesh.getFaces().addAll(
                        p11, tc11, p01, tc01, p00, tc00);
            }
            else
            //second triangle alone
            mesh.getFaces().addAll(
                    p11, tc11, p01, tc01, p00, tc00);
        }
    }


    //Wallpaper is gotten
    //Mapa Mundi clip art, shared by Mirian 06-13-2013
    //http://www.clker.com/clipart-mapa-mundi.html
    Image wallpaper = new Image(getClass()
            .getResourceAsStream("PlanetEarth.png"));
    //Declaration of material for painting the pyramid
    PhongMaterial material = new PhongMaterial();
    //Wallpaper is declared as covering material
    material.setDiffuseMap(wallpaper);
    //material.setDiffuseColor(Color.ALICEBLUE);
    MeshView  meshView = new MeshView(mesh);
    meshView.setMaterial(material);
    //meshView.setDrawMode(DrawMode.LINE);
    meshView.setDrawMode(DrawMode.FILL);
    meshView.setCullFace(CullFace.BACK);


    final Xform meshViewXform = new Xform();

    meshViewXform.getChildren().add(meshView);
    //Initial position
    meshViewXform.setTranslate(radiusOrbit, 0, 0);
```

```
    //Initial inclination in degrees
    meshViewXform.rx.setAngle(0);
    meshViewXform.ry.setAngle(3);
    meshViewXform.rz.setAngle(0);



    //The mesh changes over time
    AnimationTimer timer = new AnimationTimer()
    {
        @Override
        public void handle(long l)
        {
            changePoints();
        }
    };

    timer.start();
    return meshViewXform;
}



//The earth is difigured by random mutations in tandem
//of the positions of the points of its mesh.
 private void changePoints()
{
    int n = mesh.getPoints().size();
    float [] pointsFloat = new float[n];
    for(int i = 0; i < n; i = i+3)
    {
        //Reading
        float x =  mesh.getPoints().get(i);
        float y =  mesh.getPoints().get(i+1);
        float z =  mesh.getPoints().get(i+2);
        if (i == 36)
            System.out.println("i = " + i + "   x === " + x
            + "  " +  System.nanoTime());
```

```
        //Transforming:
        //Random shifts are included
        pointsFloat[i]    = x + r.nextInt(3) - 1;
        pointsFloat[i+1]  = y + r.nextInt(3) - 1;
        pointsFloat[i+2]  = z + r.nextInt(3) - 1;
    }
    //Resetting
    mesh.getPoints().setAll(pointsFloat, 0, n);
}

}//End of inner class Planet



public static void main(String[] args)
{

    launch(args);
}

}//End of Program N27 DisfiguredSphere
```

28, page 69. The following program N28 DisfiguredSphere2 implements a sequence of radial mutations of a sphere. The output is somehow artistic. Relevant changes are here:

```
/*
    Program N28 DisfiguredSphere2
 *  The points of a sphere ( planet Earth) are subjected to
    radial random mutations in tandem.
*/


//The earth is difigured by random mutations in tandem
//of the positions of the points of its mesh.
 private void changePoints()
{
```

```
    int n = mesh.getPoints().size();
    float [] pointsFloat = new float[n];
    for(int i = 0; i < n; i = i+3)
    {
        //Reading
        float x =  mesh.getPoints().get(i);
        float y =  mesh.getPoints().get(i+1);
        float z =  mesh.getPoints().get(i+2);
        Point3D p = new Point3D(x,y,z);
        double magnitude = p.magnitude();
        //Unitary vector in the radial direction
        Point3D u = p.multiply(1.0/magnitude);
        //Transforming:
        //Magnitude suffers mutation
        magnitude = magnitude + r.nextInt(5) - 2;
        //Disappearing is not allowed
        if (magnitude > 10)
            p = u.multiply(magnitude);
        //Random shifts are included
        pointsFloat[i]    =  (int) p.getX();
        pointsFloat[i+1]  =  (int) p.getY();
        pointsFloat[i+2]  =  (int) p.getZ();
    }
    //Resetting
    mesh.getPoints().setAll(pointsFloat, 0, n);
}
```

Problems of Chapter 3

31, page 72. The following code shows an animated skeleton of the hand.

```
/*
    Program N31 AnimatedHandSkeleton
    This program imports a set  of meshes that  encode
    for the bones of the hand.
    ".obj" format.
```

```
    Content is shown in a textArea and
    encoded meshes are drawn and animated

    One can select multiple files in the file chooser.

    Use of licensed files:
    particularly  with those of the BodyParts3D project.
    http://lifesciencedb.jp/bp3d/


    The license for this database is specified in the
    Creative Commons Attribution-Share Alike 2.1 Japan.
    If you use data from this database, please be sure
    attribute this database as follows: "BodyParts3D, (c)
    The Database Center for Life Science licensed under
    CC Attribution-Share Alike 2.1 Japan".
    http://dbguidehive.biosciencedbc.jp/en/bodyparts3d/lic.html

 */

package Programs;

//Files in the hosting project Programs
import Tools.Axes;
import Tools.Cameras;
import Tools.Xform;
import Tools.XformWithPivot;


import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
```

```
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.binding.NumberBinding;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.Slider;
import javafx.scene.control.TextArea;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.MeshView;;
import javafx.scene.shape.TriangleMesh;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import javafx.util.Duration;



public class AnimatedHandSkeleton extends Application
{
```

```
//Visibility of the mesh
//might depend on the correct choice
//of this factor. Move it in within 1 and 500.
private static double scaleFactor = 4;
//Printing for debugging
final static boolean printReadFile = false;
private static final String title
        =  "Importer of meshes in format ";

//Tool to read files
BufferedReader br ;
private static  String textOfFile = "";

private static int counterOfFaces = 0;
private static int counterOfVertexes = 0;
private static int counterOfNormals = 0;

private static int[] destFaces ;

private static TriangleMesh[] bonesMeshesArray  = new Triang
//Array of Xforms
private static Xform[] bonesXformsArray  = new Xform[100];
private static MeshView[] bonesViewArray  = new MeshView[100
private static final String[] names1  = new String[100];
private static final String[] names2  = new String[100];
//Group of Xforms
private static Xform bonesXforms  = new Xform();
//Counter
private static int counterOfBones = 0;

//*****************

final static Group root = new Group();

final static Group axisGroup = new Group();


final static Xform cameraXform = new Xform();
```

```
    final static Xform cameraXform2 = new Xform();
    final static Xform cameraXform3 = new Xform();

    final static double cameraDistance = 0; //1450;//450
    final static Xform world3DXform = new Xform();
    final static Xform bonesXform = new Xform();

    //Axes : ctrl + A
    private static final Axes axes = new Axes();
    private static final Cameras camera = new Cameras();

    boolean timelinePlaying = false;
    double ONE_FRAME = 1.0/24.0;
    double DELTA_MULTIPLIER = 200.0;
    double CONTROL_MULTIPLIER = 0.1;
    double SHIFT_MULTIPLIER = 0.1;
    double ALT_MULTIPLIER = 0.5;

    double mousePosX;
    double mousePosY;
    double mouseOldX;
    double mouseOldY;
    double mouseDeltaX;
    double mouseDeltaY;




   //Menubar declaration and initialization
    MenuBar menuBar = new MenuBar();


//This is the outer pane to place tabs
private static BorderPane outerPane = new BorderPane();
private static Pane paneHand = new Pane();

//Slider for zooming
    //private  static Slider zSlider;
```

```
//Position of graphic along the z-axis,
//normal to the screen.
//It is a double property because
//it will get involved into a binding
//private  static final DoubleProperty zPos
//        = new SimpleDoubleProperty(200);

//Slider for angle of rotation of fingers
private  static Slider closeHandSlider;
//To close the hand,
//each phalange suffers a rotation.
//It is a double property because
//it will get involved into a binding
private  static final DoubleProperty closeHandProperty
        = new SimpleDoubleProperty(0);

//Slider for opening fingers thanks to
//their transversal rotation
private  static Slider openFingersSlider;
//To open the fingers,
//each phalange suffers a transversal rotation.
//It is a double property because
//it will get involved into a binding
private  static final DoubleProperty openFingersProperty
        = new SimpleDoubleProperty(0);

//Slider for opening/closing the thumb
private  static Slider closeThumbSlider;
//To close the thumb we need a rotation.
//It is a double property because
//it will get involved into a binding
private  static final DoubleProperty closeThumbProperty
        = new SimpleDoubleProperty(0);

//Slider for rotating the thumb to form the grip
private  static Slider rotateThumbSlider;
//To form the grip we need a rotation of the thumb.
//It is a double property because
```

```
//it will get involved into a binding
private  static final DoubleProperty rotateThumbProperty
        = new SimpleDoubleProperty(0);


//Nested definition of   nodes is implemented
//to have, say,  the palm dragging the fingers,
//and a phalanx dragging the rest of its the finger.


//Each finger generates various natural groups:
//Group with the extreme phalanx of finger0
//Symbolic initialization
private static final XformWithPivot finger0Ph2Group
        = new XformWithPivot();
//Group with the two extreme phalanges
private static final XformWithPivot finger0Ph21Group
        = new XformWithPivot();
//Group with all three phalanges. It is the finger
private static final XformWithPivot finger0Ph210Group
        = new XformWithPivot();
//Finger1
private static final XformWithPivot finger1Ph2Group
        = new XformWithPivot();
private static final XformWithPivot finger1Ph21Group
        = new XformWithPivot();
private static final XformWithPivot finger1Ph210Group
        = new XformWithPivot();
//Finger2
private static final XformWithPivot finger2Ph2Group
        = new XformWithPivot();
private static final XformWithPivot finger2Ph21Group
        = new XformWithPivot();
private static final XformWithPivot finger2Ph210Group
        = new XformWithPivot();
//Finger3
private static final XformWithPivot finger3Ph2Group
        = new XformWithPivot();
```

```
    private static final XformWithPivot finger3Ph21Group
            = new XformWithPivot();
    private static final XformWithPivot finger3Ph210Group
            = new XformWithPivot();
    //Finger4
    private static final XformWithPivot finger4Ph2Group
            = new XformWithPivot();
    private static final XformWithPivot finger4Ph21Group
            = new XformWithPivot();
    private static final XformWithPivot finger4Ph210Group
            = new XformWithPivot();
    //Thumb
    private static final XformWithPivot thumbPh2Group
            = new XformWithPivot();
    private static final XformWithPivot thumbPh21Group
            = new XformWithPivot();
    private static final XformWithPivot thumbPh210Group
            = new XformWithPivot();

     private static final XformWithPivot wristGroup
            = new XformWithPivot();
    //Group with wrist and fingers
    private static final XformWithPivot palmGroup
            = new XformWithPivot();

    private static final Timeline animation  = new Timeline();

    private static double[][] coordBoxes = new double[100][6];

@Override
public void start(Stage primaryStage)
{
    System.out.println("start");

    Scene scene = new Scene(root, 1200, 1000, true);

    paneHand.setPrefSize(500, 500);
```

```
outerPane.setOpacity(0);
outerPane.setLayoutX(700);
outerPane.setLayoutY(700);

/*
   Multiple file loader
*/

//Content of files will be shown in textArea
final TextArea textArea = new TextArea();
textArea.prefWidth(400);
textArea.setWrapText(true);
textArea.setText("");
//Container  for textArea.
ScrollPane scrollPane = new ScrollPane();
scrollPane.getStyleClass().add("noborder-scroll-pane");
scrollPane.setContent(textArea);
scrollPane.setFitToWidth(true);
scrollPane.setPrefWidth(400);
scrollPane.setPrefHeight(180);
VBox vBox = new VBox();
vBox.visibleProperty().setValue(false);
vBox.getChildren().addAll(scrollPane);
root.getChildren().addAll(vBox);


// --- Menu files
Menu filesMenu = new Menu("Files");
//Menu to load multiple files
MenuItem loadFiles
        = new MenuItem("Load + process multiple files ");
//MenuItem added to menu
filesMenu.getItems().addAll(loadFiles);
menuBar.getMenus().add(filesMenu);
root.getChildren().add(menuBar);

//Listener
loadFiles.setOnAction((ActionEvent t) ->
```

```
{
    FileChooser fileChooser = new FileChooser();

    //Name of file with extension "
    FileChooser.ExtensionFilter extFilter
            = new FileChooser
                    .ExtensionFilter("TXT files (*.obj)"
                            , "*.obj");
    fileChooser.getExtensionFilters().add(extFilter);

    //Files will be kept in a list
    List<File> files = new FileChooser()
            .showOpenMultipleDialog(primaryStage);



        //Delete possible repetitions
        //files = deleteRepetitions(files);
    //counterOfBones = 0;



    for(int j = 0; j < files.size(); j++)
    {
        File file = files.get(j);
        String name = file.getName();
        names1[j] = name;
    }
    counterOfBones = 0;
    //Each file is processed
    for(int j = 0; j < files.size(); j++)
    {

        String name = names1[j];
        boolean repeated = false;
        for (int i = 0; i <  j; i++)
        {
            if (name.equals(names1[i]))
```

```
            {
                repeated = true;

                //System.out.println("Repeated" + name);
            }
        }

        if (repeated);
        else
        try
        {
            File file = files.get(j);
            //Text of file is read and mesh is extracted
            textOfFile = readFile(file);
            //Text copied to textArea
            textArea.appendText(textOfFile);

        } catch (IOException ex)
            {
                Logger.getLogger(AnimatedHandSkeleton
                        .class.getName())
                        .log(Level.SEVERE, null, ex);
            }




    }

    defineGroups();



//To remedy a not understood bug
//( = menuBar is added to the graphic )
root.getChildren().remove(menuBar);

vBox.visibleProperty().set(true);
```

```
});

//====== End of File loader

primaryStage.setTitle(title
        + ". Drag the mouse to rotate. "
    + " Ctrl + A = Axis visibility"
        );
scene.setFill(Color.BLUE);
primaryStage.setScene(scene);
primaryStage.show();
axes.buildAxes();
axes.handleKeyboard(scene);
//camera.handleMouse(scene);
camera.handleMouse(paneHand);

//Hide textArea and show graphic
Button hideTextArea = new Button();
hideTextArea.setText("View graphic");
//Listener
hideTextArea.setOnAction((ActionEvent t) ->
{
    vBox.visibleProperty().set(false);
    camera.buildCamera();
    bonesXform.setScale(scaleFactor);



    /*
    //Centering: found by trial and error.
    double xMax = bonesXform.getBoundsInLocal().getMaxX();
    double yMax = bonesXform.getBoundsInLocal().getMaxY();
    double zMax = bonesXform.getBoundsInLocal().getMaxZ();
    double xMin = bonesXform.getBoundsInLocal().getMinX();
    double yMin = bonesXform.getBoundsInLocal().getMinY();
    double zMin = bonesXform.getBoundsInLocal().getMinZ();
    double x = scaleFactor *  0.82 * (xMax + xMin) ;
    double y = scaleFactor * 0.78 * (yMax + yMin);
```

```
    double z = scaleFactor * 0.9 * (zMax + zMin) ;

    world3DXform.setTranslate(-x, -y, -z);

    */
    world3DXform.getChildren().add(bonesXform);
    world3DXform.setScale(scaleFactor);


    //world3DXform.setRx(180);
    //world3DXform.setRy(90);
    world3DXform.setScale(2);
    world3DXform.setTranslate(-1300, -800, 0);
    paneHand.getChildren().add(world3DXform);
    root.getChildren().add(paneHand);
    scene.setCamera(camera.camera);
});

vBox.getChildren().addAll(hideTextArea);



//This slider   controls the closing of the hand.
closeHandSlider = new Slider();
closeHandSlider.setMax(90);
closeHandSlider.setMin(0);
closeHandSlider.setValue(10);

//To close the hand, an angle of rotation is enabled.
//Bind angleProperty to the reading of the slider
closeHandSlider.valueProperty()
        .bindBidirectional(closeHandProperty);


//This slider controls the openning of fingers.
openFingersSlider = new Slider();
openFingersSlider.setMax(15);
openFingersSlider.setMin(0);
```

```
openFingersSlider.setValue(10);

//To open fingers, an angle of rotation is enabled.
//Bind openFingersProperty to the reading of the slider
openFingersSlider.valueProperty()
        .bindBidirectional(openFingersProperty);


//Slider to open  the thumb
closeThumbSlider = new Slider();
closeThumbSlider.setMax(55);
closeThumbSlider.setMin(0);
closeThumbSlider.setValue(-0);
closeThumbSlider.adjustValue(closeThumbSlider.getValue());

//To open the thumb, an angle of rotation is enabled.
//Bind closeThumbProperty to the reading of the slider
closeThumbSlider.valueProperty().bindBidirectional(closeThum


//Slider to rotate  the thumb
rotateThumbSlider = new Slider();
rotateThumbSlider.setMax(50);
rotateThumbSlider.setMin(0);
rotateThumbSlider.setValue(0);
rotateThumbSlider.adjustValue(rotateThumbSlider.getValue());

//To rotate the thumb, an angle of rotation is enabled.
//Bind closeThumbProperty to the reading of the slider
rotateThumbSlider.valueProperty()
        .bindBidirectional(rotateThumbProperty);



//Other JavaFX controls, labels and containers
GridPane gridPane = new GridPane();
gridPane.setVgap(4);
gridPane.setHgap(10);
```

```
        gridPane.setPadding(new Insets(5, 5, 5, 5));
        //Column 0, Row   0
        gridPane.add(new Label("Zoom: "), 0, 0);
        //gridPane.add(zSlider, 1, 0);
        gridPane.add(new Label("Close hand: "), 2, 0);
        gridPane.add(closeHandSlider, 3, 0);
        gridPane.add(new Label("Open fingers: "), 0, 1);
        gridPane.add(openFingersSlider, 1, 1);
        gridPane.add(new Label("Close thumb: "), 0, 2);
        gridPane.add(closeThumbSlider, 1, 2);
        gridPane.add(new Label("Rotate thumb: "), 2, 2);
        gridPane.add(rotateThumbSlider, 3,2);



        SubScene subScene
                = new SubScene(gridPane, 880, 200);
        subScene.setLayoutX(10);
        subScene.setLayoutY(800);
        paneHand.getChildren().add(subScene);









        primaryStage.show();
}
```

```
//File is read and its content is returned as a string.
//The associated mesh is recovered
//and correct orientation of faces is provided.
//The mesh is a global variable.
private  String readFile(File file)
        throws FileNotFoundException, IOException
{
    //Name of file is printed
    String name = file.getName();
    if (printReadFile)
        System.out.println(name);
    names2[counterOfBones] = name;
    //Buffer to keep the content of file
    //StringBuilder stringBuffer = new StringBuilder();
    StringBuilder stringBuffer  = new StringBuilder();
    //Tool to read file
    BufferedReader bufferedReader = null;

    //A mesh is declared and instantiated
    TriangleMesh mesh  = new TriangleMesh();
    //The mesh has no predefined texture
    mesh.getTexCoords().addAll(0,0);
    //In the " format, counting begins with 1
    //but in Java, we begin with 0.
    //So, an additional inocuos point is added.
    mesh.getPoints().addAll(0,0,0);

    if (printReadFile)
        System.out.println("File content: ");
//File is read
try
{
//
bufferedReader = new BufferedReader(new FileReader(file));
```

```
String text;



while( (text = bufferedReader.readLine()) != null)
{
    //Text for the textArea
    stringBuffer.append(text);
    //Print each line
    //System.out.println(text);

    //Vertexes are read
    if (((text.length() > 0 ) && text.charAt(0) == 'v')
            && text.charAt(1) == ' ')
    {
        if (printReadFile)
            System.out.println("Vertex  = "
                    + counterOfVertexes + " = "+text);
        counterOfVertexes++;
        text = text.substring(2);
        int j = 0;
        //Three vertexes are read
        for(int i = 0; i < 3; i++)
        {
            double d ;
            String s = "";
            Boolean go = true;

            while (go)
            {
                char thisChar = text.charAt(j);
                s = s + thisChar;
                char nextChar = ' ';
                //++j means add one to j
                //and then evaluate expression with <
                if (++j < text.length())
```

```
                nextChar = text.charAt(j);
                if ((nextChar == ' ') ||
                        (j >= text.length()))
                        go = false;

            }
            //string to double
            d = Double.parseDouble(s);
            float df = (float) d;
            if (printReadFile)
                System.out.println("Float number = " + df);
            mesh.getPoints().addAll(df);

        }
    }


    //Normal vectors are counted and displayed
    if ((text.length() > 0 ) && (text.charAt(0) == 'v')
            && (text.charAt(1) == 'n'))
    {

        if (printReadFile)
            System.out.println("Normal " + counterOfNormals
                + " = " + text);
        counterOfNormals++;
    }


    //Face is read, one face per line.
    //A face has 3 or more vertexes. These are extracted.
    //Additional information is ignored
    if ( (text.length() > 0 ) && (text.charAt(0) == 'f'))
    {

        if (printReadFile)
            System.out.println("Face  = " + counterOfFaces
                + " = "+text);
```

```
counterOfFaces++;
text = text.substring(2);

int vertex[] = new int[4];
int j = 0;

Boolean isNotBlank ;
//Three vertexes are extracted
for(int i = 0; i < 3; i++)
{

    double d ;
    String s = "";
    Boolean go1 = true;


    while (go1)
    {
        if (j >= text.length()-2)
            break;

        char thisChar = text.charAt(j);
        s = s + thisChar;
        /*
        System.out.println(" j = "
                + j + "    s = " + s);
                */
        char nextChar = ' ';
        j++;
        if (j < text.length())
         nextChar = text.charAt(j);
        char nextNextChar = ' ';
        isNotBlank = true;
        if (nextChar == '/' )
        {
            go1 = false;
            //go until the following blank
            while ((isNotBlank)
```

```
                        && (j < text.length()-1))
                {
                    j++;
                    nextNextChar = text.charAt(j);
                    if (nextNextChar == ' ' )
                        isNotBlank = false;

                }
            }

        }

        d = Double.parseDouble(s);
        int v = (int) d;
        vertex[i] = v;
        if (printReadFile)
            System.out.println("Int number = " + v);
    }//End of cycle for 3 vertexes

    //A face with 3 vertexes is added to mesh
    mesh.getFaces().addAll(vertex[0], 0, vertex[1], 0,
        vertex[2], 0);

    if (printReadFile)
    {
        for(int i = 0; i < 3; i++)
         System.out.print(vertex[i] + " . ");
        System.out.println();
    }

    //Is there a four vertex in the face?
    if   ( (text.length() - j )> 3)
    {
        String s = "";
        Boolean go1 = true;
        while (go1)
        {
```

```
                char thisChar = text.charAt(j);
                s = s + thisChar;
                j++;
                char nextChar = ' ';
                if (j < text.length())
                 nextChar = text.charAt(j);
                if (nextChar == '/' ) go1 = false;
            }
            if (printReadFile)
                System.out.println("4th = " + s);
            double d = Double.parseDouble(s);
            int v = (int) d;
            vertex[3] = v;

            //A polyface with 4 vertex, say, a,b,c,d,
            //can be covered by 2 triangles:
            //Triangle 1: a,b,c (Already added to mesh)
            //and triangle 2: a,c,d.
            //Triangle 2 with 3 vertexes is added to mesh
            mesh.getFaces().addAll(vertex[0], 0, vertex[2],
                    0, vertex[3], 0);
        }
    }//End of face cycle

    //Bounds of containing box are read and displayed
    if ( (text.length() > 0 )
            && (text.contains("# Bounds(mm):")))
    {
        //System.out.println(text);

        String a = "# Bounds(mm): (";
        int m = a.length();
        text = text.substring(m);
        text = text.replace(")-(", ",");
        text = text.replace(")", ",");
        //System.out.println(text);
```

```
int j = 0;


//Six vertexes are extracted
for(int i = 0; i < 6; i++)
{

    double d ;
    String s = "";
    Boolean go1 = true;


    while (go1)
    {
        if (j >= text.length()-2)
            break;

        char thisChar = text.charAt(j);
        s = s + thisChar;
        /*
        System.out.println(" j = "
                + j + "    s = " + s);
        */
        char nextChar = ' ';
        j++;
        if (j < text.length())
         nextChar = text.charAt(j);
        if (nextChar == ',' )
        {
            go1 = false;
           if ( j < text.length() ) j++;
        }

    }

    d = Double.parseDouble(s);

    coordBoxes[counterOfBones][i] = d;
```

```
            if (printReadFile)
                 System.out.println("Coordinate = " + i + "= " + d);
        }//End of cycle for 3 vertexes
    }

}

} //End of  try.
    //Exception management
    catch (FileNotFoundException ex)
    {
        Logger.getLogger(AnimatedHandSkeleton
        .class.getName()).log(Level.SEVERE, null, ex);
    }
    catch (IOException ex)
    {
        Logger.getLogger(AnimatedHandSkeleton
        .class.getName()).log(Level.SEVERE, null, ex);
    }
    finally
    {
        try
        {
            bufferedReader.close();
        } catch (IOException ex)
        {
            Logger.getLogger(AnimatedHandSkeleton
            .class.getName()).log(Level.SEVERE, null, ex);
        }
    }



    if (printReadFile)
    {
        System.out.println("End of file content. ");
        //Other indicators are reported
        report(mesh);
```

```
    }


    //======MeshWork=======

    MeshView boneMesh =
            new MeshView(mesh);

    PhongMaterial goldenMaterial
            = new PhongMaterial(Color.GOLDENROD);
    boneMesh.setMaterial(goldenMaterial);
    Group boneGroup = new Group(boneMesh);
    bonesViewArray[counterOfBones] = boneMesh;

    Xform boneXform = new Xform();
    boneXform.getChildren().add(boneGroup);
    bonesXformsArray[counterOfBones] = boneXform;
    bonesXform.getChildren().add(boneXform);
    counterOfBones++;

    return stringBuffer.toString();
}


//Indicators of retrieved data are printed
private void report(TriangleMesh mesh)
{
    float[] destPoints = new float[mesh.getPoints().size()];
    mesh.getPoints().toArray(destPoints);
    System.out.println("POINTS");
    for(int j = 0; j <  destPoints.length/3; j++)
    {
        System.out.print(j + "   " );
        for(int k = 0; k < 3; k++)
        {
            System.out.print( destPoints[j*3 + k] + "   " );
        }
        System.out.println();
```

```
    }

    destFaces = new int[mesh.getFaces().size()];
    //Faces go to a matrix
    mesh.getFaces().toArray(destFaces);
    System.out.println("FACES WITH TEXTURE COORDINATES");
    for(int j = 0; j <  destFaces.length/6; j++)
    {
        System.out.print(j + "   " );
        for(int k = 0; k < 6; k++)
        {
            System.out.print( destFaces[j*6 + k] + "   " );
        }
        System.out.println();
    }

    System.out.println("counterOfPoints = "
            + counterOfVertexes );
    System.out.println("counterOfFaces = "
            + counterOfFaces );
    System.out.println("counterOfNormals = "
            + counterOfNormals);
    System.out.println("mesh.getPoints().size() = "
            + mesh.getPoints().size() );
    System.out.println("mesh.getFaces().size() = "
            + mesh.getFaces().size() );

    System.out.println("getPointElementSize = "
            + mesh.getPointElementSize() );
    System.out.println("getfaceElementSize = "
            + mesh.getFaceElementSize() );

}


private static Point3D pivot(int i)
{
    double x = (coordBoxes[i][0] + coordBoxes[i][3])/2;
```

```
    double y = (coordBoxes[i][1] + coordBoxes[i][4])/2;
    double z = coordBoxes[i][5];
    Point3D p = new Point3D(x,y,z);
    return p;
}

private static Point3D pivot2(int i)
{
    double x = (coordBoxes[i][0] + coordBoxes[i][3])/2;
    double y = (coordBoxes[i][1] + coordBoxes[i][4])/2;
    double z = coordBoxes[i][2];
    Point3D p = new Point3D(x,y,z);
    return p;
}

private static void    defineGroups()
{
    //Pivots of groups
    Point3D finger0Ph2GroupPivot = new Point3D(0,0,0);
    Point3D finger0Ph21GroupPivot  = new Point3D(0,0,0);
    Point3D finger0Ph210GroupPivot = new Point3D(0,0,0);
    Point3D finger1Ph2GroupPivot = new Point3D(0,0,0);
    Point3D finger1Ph21GroupPivot = new Point3D(0,0,0);
    Point3D finger1Ph210GroupPivot = new Point3D(0,0,0);
    Point3D finger2Ph2GroupPivot = new Point3D(0,0,0);
    Point3D finger2Ph21GroupPivot = new Point3D(0,0,0);
    Point3D finger2Ph210GroupPivot = new Point3D(0,0,0);
    Point3D finger3Ph2GroupPivot = new Point3D(0,0,0);
    Point3D finger3Ph21GroupPivot = new Point3D(0,0,0);
    Point3D finger3Ph210GroupPivot = new Point3D(0,0,0);
    Point3D thumbPh2GroupPivot = new Point3D(0,0,0);
    Point3D thumbPh21GroupPivot = new Point3D(0,0,0);
    Point3D thumbPh210GroupPivot = new Point3D(0,0,0);
    Point3D wristGroupPivot = new Point3D(0,0,0);
```

```
//Natural groups are instantiated

for(int i = 0; i < counterOfBones; i++)
{
    String name = names2[i];
    System.out.println("i = " + i + "   " +name);


    //Finger 0 = little and its subgroups
    if (name.contains("little"))
    {
        if (name.contains("Distal"))
        {
            finger0Ph2Group
                .getChildren().add(bonesViewArray[i]);
            finger0Ph2GroupPivot = pivot(i);

        }
        if (name.contains("Middle"))
        {
            finger0Ph21Group
                .getChildren()
                .addAll(finger0Ph2Group, bonesViewArray[i]);
            finger0Ph21GroupPivot = pivot(i);
        }
        if (name.contains("Proximal"))
        {
            finger0Ph210Group
                .getChildren()
                .addAll(finger0Ph21Group, bonesViewArray[i]);
            finger0Ph210GroupPivot = pivot(i);
        }
    }

    //Finger 1 = ring and its subgroups
    else if (name.contains("ring"))
    {
        if (name.contains("Distal"))
```

```
            {
                finger1Ph2Group
                    .getChildren().add(bonesViewArray[i]);
                finger1Ph2GroupPivot = pivot(i);
            }
            if (name.contains("Middle"))
            {
                finger1Ph21Group
                    .getChildren()
                    .addAll(finger1Ph2Group, bonesViewArray[i]);
                finger1Ph21GroupPivot = pivot(i);
            }
            if (name.contains("Proximal"))
            {
                finger1Ph210Group
                    .getChildren()
                    .addAll(finger1Ph21Group, bonesViewArray[i])
                finger1Ph210GroupPivot = pivot(i);
            }
        }

        //Finger 2 = middle and its subgroups
        else if (name.contains("middle"))
        {
            if (name.contains("Distal"))
            {
                finger2Ph2Group
                .getChildren().add(bonesViewArray[i]);
                finger2Ph2GroupPivot = pivot(i);
            }
            if (name.contains("Middle"))
            {
                finger2Ph21Group
                    .getChildren()
                    .addAll(finger2Ph2Group, bonesViewArray[i]);
                finger2Ph21GroupPivot = pivot(i);
            }
            if (name.contains("Proximal"))
```

```
    {
        finger2Ph210Group
            .getChildren()
            .addAll(finger2Ph21Group, bonesViewArray[i]);
        finger2Ph210GroupPivot = pivot(i);
    }
}

//Finger 3 = index and its subgroups
else if (name.contains("index"))
{
    if (name.contains("Distal"))
    {
        finger3Ph2Group
            .getChildren().add(bonesViewArray[i]);
        finger3Ph2GroupPivot = pivot(i);
    }
    if (name.contains("Middle"))
    {
        finger3Ph21Group
            .getChildren()
            .addAll(finger3Ph2Group, bonesViewArray[i]);
        finger3Ph21GroupPivot = pivot(i);
    }
    if (name.contains("Proximal"))
    {
        finger3Ph210Group
            .getChildren()
            .addAll(finger3Ph21Group, bonesViewArray[i]);
        finger3Ph210GroupPivot = pivot(i);
    }
}

//Thumb and its subgroups
else if (name.contains("thumb"))
{
    if (name.contains("Distal"))
    {
```

```
                thumbPh2Group
                    .getChildren().add(bonesViewArray[i]);
                    thumbPh2GroupPivot = pivot(i);
            }
            if (name.contains("Proximal"))
            {
                thumbPh21Group
                    .getChildren()
                    .addAll(thumbPh2Group, bonesViewArray[i]);
                thumbPh21GroupPivot = pivot(i);
            }
        }

        //Thumb and its subgroups
        else if (name.contains("first metacarpal"))
        {
            thumbPh210Group
                .getChildren()
                .addAll(thumbPh21Group, bonesViewArray[i]);
            //thumbPh210GroupPivot = pivot(i);
        }

        //Thumb and its subgroups
        else if (name.contains("trapezium"))
        {
           thumbPh210GroupPivot = pivot(i);
           wristGroup.getChildren().add(bonesViewArray[i]);
        }
        else

        {
            wristGroup.getChildren().add(bonesViewArray[i]);
            if (name.contains("lunate"))
                wristGroupPivot = pivot(i);
        }
}
```

```
//Fingers are attached to the palm

    palmGroup.getChildren().addAll(wristGroup
        , finger0Ph210Group
        , finger1Ph210Group
        , finger2Ph210Group
        , finger3Ph210Group
        , thumbPh210Group);

    world3DXform.getChildren().addAll(palmGroup);


//========OPENING/CL0SING OF THE HAND============

NumberBinding closeHandPropertyHalf
        = closeHandProperty.multiply(0.5);

//The slider that closes the hand extends its
//control to the palm:
//it must rotate around
//the end of the arm that is parallel to the X_AXIS
//This is the pivot of rotation
palmGroup.rwpx.pivotZProperty()
        .setValue(wristGroupPivot.getZ());
palmGroup.rwpx.pivotYProperty()
        .setValue(wristGroupPivot.getY());
//The angle of rotation is controlled by a slider
palmGroup.rwpx.angleProperty()
        .bind(closeHandPropertyHalf);



//Specifications of movement for subgroups of finger 0
finger0Ph2Group.rwpx.pivotZProperty()
        .setValue(finger0Ph2GroupPivot.getZ());
finger0Ph2Group.rwpx.pivotYProperty()
        .setValue(finger0Ph2GroupPivot.getY());
finger0Ph2Group.rwpx.angleProperty()
```

```
            .bindBidirectional(closeHandProperty);

        finger0Ph21Group.rwpx.pivotZProperty()
                .setValue(finger0Ph21GroupPivot.getZ());
        finger0Ph21Group.rwpx.pivotYProperty()
                .setValue(finger0Ph21GroupPivot.getY());
        finger0Ph21Group.rwpx.angleProperty()
                .bindBidirectional(closeHandProperty);

        finger0Ph210Group.rwpx.pivotZProperty()
                .setValue(finger0Ph210GroupPivot.getZ());
        finger0Ph210Group.rwpx.pivotYProperty()
                .setValue(finger0Ph210GroupPivot.getY());
        finger0Ph210Group.rwpx.angleProperty()
                .bindBidirectional(closeHandProperty);

        //Specifications of movement for subgroups of finger 1
        finger1Ph2Group.rwpx.pivotZProperty()
                .setValue(finger1Ph2GroupPivot.getZ());
        finger1Ph2Group.rwpx.pivotYProperty()
                .setValue(finger1Ph2GroupPivot.getY());
        finger1Ph2Group.rwpx.angleProperty()
                .bindBidirectional(closeHandProperty);

        finger1Ph21Group.rwpx.pivotZProperty()
                .setValue(finger1Ph21GroupPivot.getZ());
        finger1Ph21Group.rwpx.pivotYProperty()
                .setValue(finger1Ph21GroupPivot.getY());
        finger1Ph21Group.rwpx.angleProperty()
                .bindBidirectional(closeHandProperty);

        finger1Ph210Group.rwpx.pivotZProperty()
                .setValue(finger1Ph210GroupPivot.getZ());
        finger1Ph210Group.rwpx.pivotYProperty()
                .setValue(finger1Ph210GroupPivot.getY());
        finger1Ph210Group.rwpx.angleProperty()
                .bindBidirectional(closeHandProperty);
```

```
//Specifications of movement for subgroups of finger 2
finger2Ph2Group.rwpx.pivotZProperty()
        .setValue(finger2Ph2GroupPivot.getZ());
finger2Ph2Group.rwpx.pivotYProperty()
        .setValue(finger2Ph2GroupPivot.getY());
finger2Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger2Ph21Group.rwpx.pivotZProperty()
        .setValue(finger2Ph21GroupPivot.getZ());
finger2Ph21Group.rwpx.pivotYProperty()
        .setValue(finger2Ph21GroupPivot.getY());
finger2Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger2Ph210Group.rwpx.pivotZProperty()
        .setValue(finger2Ph210GroupPivot.getZ());
finger2Ph210Group.rwpx.pivotYProperty()
        .setValue(finger2Ph210GroupPivot.getY());
finger2Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

//Specifications of movement for subgroups of finger 3
finger3Ph2Group.rwpx.pivotZProperty()
        .setValue(finger3Ph2GroupPivot.getZ());
finger3Ph2Group.rwpx.pivotYProperty()
        .setValue(finger3Ph2GroupPivot.getY());
finger3Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger3Ph21Group.rwpx.pivotZProperty()
        .setValue(finger3Ph21GroupPivot.getZ());
finger3Ph21Group.rwpx.pivotYProperty()
        .setValue(finger3Ph21GroupPivot.getY());
finger3Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger3Ph210Group.rwpx.pivotZProperty()
```

```
        .setValue(finger3Ph210GroupPivot.getZ());
finger3Ph210Group.rwpx.pivotYProperty()
        .setValue(finger3Ph210GroupPivot.getY());
finger3Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);



//======OPENING/CLOSING OF FINGERS========

//Fingers that are at the border must
//open twice faster as those
//near the center.
NumberBinding openFingersPropertyTwo
        = openFingersProperty.multiply(2);
//Specifications of movement for subgroups of finger 0
finger0Ph210Group.rwpy.pivotXProperty()
        .setValue(finger0Ph210GroupPivot.getX());
finger0Ph210Group.rwpy.pivotYProperty()
        .setValue(finger0Ph210GroupPivot.getY());
finger0Ph210Group.rwpy.pivotZProperty()
        .setValue(finger0Ph210GroupPivot.getZ());
finger0Ph210Group.rwpy.angleProperty()
        .bind(openFingersPropertyTwo);


//Specifications of movement for subgroups of finger 1
finger1Ph210Group.rwpy.pivotXProperty()
        .setValue(finger1Ph210GroupPivot.getX());
finger1Ph210Group.rwpy.pivotYProperty()
        .setValue(finger1Ph210GroupPivot.getY());
finger1Ph210Group.rwpy.pivotZProperty()
        .setValue(finger1Ph210GroupPivot.getZ());

finger1Ph210Group.rwpy.angleProperty()
        .bind(openFingersProperty);

//Finger two remains invariant under opening of fingers
```

```
//Opening in the contrary direction of fingers 3 and 4
NumberBinding openFingersPropertyMinus
        = openFingersProperty.multiply(-1);

//Specifications of movement for finger 3
finger3Ph210Group.rwpy.pivotXProperty()
        .setValue(finger3Ph210GroupPivot.getX());
finger3Ph210Group.rwpy.pivotYProperty()
        .setValue(finger3Ph210GroupPivot.getY());
finger3Ph210Group.rwpy.pivotZProperty()
        .setValue(finger3Ph210GroupPivot.getZ());
finger3Ph210Group.rwpy.angleProperty()
        .bind(openFingersPropertyMinus);

NumberBinding openFingersPropertyMinusTwo
        = openFingersProperty.multiply(-2);



//====== The thumb ========
//Specifications of movement for the thumb


NumberBinding rotateThumbPropertyDown
        = rotateThumbProperty.multiply(-0.4);

NumberBinding rotateThumbPropertyUp
        = rotateThumbProperty.multiply(-2);


NumberBinding closeThumbPropertyMinus
        = closeThumbProperty.multiply(-1);


//Traversal movement
thumbPh210Group.rwpy.pivotXProperty()
        .setValue(thumbPh210GroupPivot.getX());
```

```
thumbPh210Group.rwpy.pivotZProperty()
        .setValue(thumbPh210GroupPivot.getZ());
thumbPh210Group.rwpy.angleProperty()
        .bind(closeThumbProperty);

/*
//Rotations of phalanges 2 and 1 to make the grip.
//Rotations around X_AXIS and Y_AXIS at the same time
//Phalanx 2
thumbPh2Group.rwpx.pivotXProperty()
        .setValue(thumbPh2GroupPivot.getX());
thumbPh2Group.rwpx.pivotYProperty()
        .setValue(thumbPh2GroupPivot.getY());
thumbPh2Group.rwpx.pivotZProperty()
        .setValue(thumbPh2GroupPivot.getZ());
thumbPh2Group.rwpx.angleProperty()
        .bind(rotateThumbPropertyDown);

//Phalanges 2 and 1
thumbPh21Group.rwpx.pivotXProperty()
        .setValue(thumbPh21GroupPivot.getX());
thumbPh21Group.rwpx.pivotYProperty()
        .setValue(thumbPh21GroupPivot.getY());
thumbPh21Group.rwpx.pivotZProperty()
        .setValue(thumbPh21GroupPivot.getZ());
thumbPh21Group.rwpx.angleProperty()
        .bind(rotateThumbPropertyDown);

//Phalanx 2
thumbPh2Group.rwpy.pivotXProperty()
        .setValue(thumbPh2GroupPivot.getX());
thumbPh2Group.rwpy.pivotYProperty()
        .setValue(thumbPh2GroupPivot.getY());
thumbPh2Group.rwpy.pivotZProperty()
        .setValue(thumbPh2GroupPivot.getZ());
thumbPh2Group.rwpy.angleProperty()
        .bind(rotateThumbPropertyDown);
//Phalanges 2 and 1
```

```
thumbPh21Group.rwpy.pivotXProperty()
        .setValue(thumbPh21GroupPivot.getX());
thumbPh21Group.rwpy.pivotYProperty()
        .setValue(thumbPh21GroupPivot.getY());
thumbPh21Group.rwpy.pivotZProperty()
        .setValue(thumbPh21GroupPivot.getZ());
thumbPh21Group.rwpy.angleProperty()
        .bind(rotateThumbPropertyDown);
*/

//Rotation of the whole thumb
//to get opposite to other fingers
thumbPh210Group.rwpz.pivotXProperty()
        .setValue(thumbPh210GroupPivot.getX());
thumbPh210Group.rwpz.pivotYProperty()
        .setValue(thumbPh210GroupPivot.getY());
thumbPh210Group.rwpz.pivotZProperty()
        .setValue(thumbPh210GroupPivot.getZ());
thumbPh210Group.rwpz.angleProperty()
        .bind(rotateThumbProperty.multiply(-2));



//===========Animation=========

double endTime1 = 10000.0; ////1000 = 1 second
double endTime2 = 2 * endTime1;
double endTime3 = 3 * endTime1;
double endTime4 = 4 * endTime1;
double endTime5 = 5 * endTime1;
double endTime6 = 6 * endTime1;
double endTime7 = 7 * endTime1;
double endTime8 = 8 * endTime1;
double endTime9 = 9 * endTime1;
```

```
    animation.getKeyFrames().addAll
    (

//The hand gets closed
        //closeHandProperty: initial position at initial time
        new KeyFrame(Duration.ZERO,
            //Permanent condition
            new KeyValue(closeThumbProperty, 0)),
        //closeHandProperty: final position at final time
        new KeyFrame(new Duration(endTime1),
            new KeyValue(closeThumbProperty, -50)),

        //The hand gets closed
        //closeHandProperty: initial position at initial time
        new KeyFrame(new Duration(endTime1),
            //Permanent condition
            new KeyValue(closeHandProperty, 0)),
        //closeHandProperty: final position at final time
        new KeyFrame(new Duration(endTime2),
            new KeyValue(closeHandProperty, -90))
            ,


        //The thumb closes the grip
        //rotateThumbProperty: initial position at initial time
        new KeyFrame(new Duration(endTime2),
            //Permanent condition
            new KeyValue(rotateThumbProperty, 0)),
        //rotateThumbProperty: final position at final time
        new KeyFrame(new Duration(endTime3),
            new KeyValue(rotateThumbProperty, 37)),

        //The thumb gets reopened
        //rotateThumbProperty: initial position at initial time
        new KeyFrame(new Duration(endTime3),
            //Permanent condition
            new KeyValue(rotateThumbProperty, 37)),
        //rotateThumbProperty: final position at final time
```

```
new KeyFrame(new Duration(endTime4),
    new KeyValue(rotateThumbProperty, 0)),

//The hand get reopened
//closeHandProperty: initial position at initial time
new KeyFrame(new Duration(endTime4),
    //Permanent condition
    new KeyValue(closeHandProperty, -90)),
//closeHandProperty: final position at final time
new KeyFrame(new Duration(endTime5),
    new KeyValue(closeHandProperty, 0)),

//Fingers get reopened
//openFingersProperty: initial position at initial time
new KeyFrame(new Duration(endTime5),
//Permanent condition
new KeyValue(openFingersProperty, 0)),
//openFingersProperty: final position at final time
new KeyFrame(new Duration(endTime6),
    new KeyValue(openFingersProperty, 15)),

//Fingers get closed
//openFingersProperty: initial position at initial time
new KeyFrame(new Duration(endTime6),
//Permanent condition
new KeyValue(openFingersProperty, 15)),
//openFingersProperty: final position at final time
new KeyFrame(new Duration(endTime7),
    new KeyValue(openFingersProperty, 0)),

//The thumb gets closed
//closeThumbProperty: initial position at initial time
new KeyFrame(new Duration(endTime7),
    //Permanent condition
    new KeyValue(closeThumbProperty, -55)),
//closeThumbProperty: final position at final time
new KeyFrame(new Duration(endTime8),
    new KeyValue(closeThumbProperty, 0)),
```

```
        //The thumb gets reopen
        //closeThumbProperty: initial position at initial time
        new KeyFrame(new Duration(endTime8),
            //Permanent condition
            new KeyValue(closeThumbProperty, 0)),
        //closeThumbProperty: final position at final time
        new KeyFrame(new Duration(endTime9),
            new KeyValue(closeThumbProperty, -25))

    );




animation.play();

}

public void play()
{

}

@Override public void stop()
{

}

public static void main(String[] args)
{

    launch(args);
}


}//End of main class N31 AnimatedHandSkeleton
```

42, page 159. The next code implements PCA mutations of a bone:

```
/*
    Program N42 PCAmutations
    This program produces PCA mutations of a bone.
    These are scale mutations along the principal axes
    of the bone.

    Principal axes can be understood as the directions
    of the bone that mark its length, height and width.

    Tested only with some  files,
    particularly  with those of the BodyParts3D project.
    http://lifesciencedb.jp/bp3d/
    Those files are licensed:

    The license for this database is specified in the
    Creative Commons Attribution-Share Alike 2.1 Japan.
    If you use data from this database, please be sure
    attribute this database as follows: "BodyParts3D, (c)
    The Database Center for Life Science licensed under
    CC Attribution-Share Alike 2.1 Japan".
    http://dbguidehive.biosciencedbc.jp/en/bodyparts3d/lic.html
 */
package Programs;
import Jama.*;



//Files in the hosting project Programs
import Tools.Axes;
import Tools.Cameras;
import Tools.Xform;
//import Tools.XformWithPivot;
```

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.event.ActionEvent;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.TextArea;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.MeshView;;
import javafx.scene.shape.TriangleMesh;
import javafx.stage.FileChooser;
import javafx.stage.Stage;



public class PCAmutations extends Application
{
    //This is a global scale factor.
    //Visibility of the mesh
    //might depend on the correct choice
    //of this factor. Move it in within 1 and 500.

    private static final float scaleFactor = 10;
```

```
//Printing for debugging
final static boolean printReadFile = false;
private static final String title
        =  "Importer of meshes in format .obj";

//Tool to read files
BufferedReader br ;
private static  String textOfFile = "";

private static int counterOfFaces = 0;
private static int counterOfVertexes = 0;
private static int counterOfNormals = 0;

private static int[] destFaces ;


//*****************

final Group root = new Group();


private static final Cameras camera = new Cameras();


final Xform cameraXform = new Xform();
final Xform cameraXform2 = new Xform();
final Xform cameraXform3 = new Xform();

//Axes : ctrl + A
private static final Axes axes = new Axes();


final double cameraDistance = 1450;//450
final Xform world3DXform = new Xform();
final Xform bonesXform = new Xform();
```

```
    boolean timelinePlaying = false;
    double ONE_FRAME = 1.0/24.0;
    double DELTA_MULTIPLIER = 200.0;
    double CONTROL_MULTIPLIER = 0.1;
    double SHIFT_MULTIPLIER = 0.1;
    double ALT_MULTIPLIER = 0.5;

    double mousePosX;
    double mousePosY;
    double mouseOldX;
    double mouseOldY;
    double mouseDeltaX;
    double mouseDeltaY;



   //Menubar declaration and initialization
    MenuBar menuBar = new MenuBar();

    //Scal factors along width, height and length
    final static private float scaleFactorW = 5;
    final static private float scaleFactorH = 0.3f;
    final static private float scaleFactorL = 2;




@Override
public void start(Stage primaryStage)
{
    System.out.println("start");

    Scene scene = new Scene(root, 1200, 1000, true);
```

```
/*
   Multiple file loader
*/

//Content of files will be shown in textArea
final TextArea textArea = new TextArea();
textArea.prefWidth(400);
textArea.setWrapText(true);
textArea.setText("");
//Container  for textArea.
ScrollPane scrollPane = new ScrollPane();
scrollPane.getStyleClass().add("noborder-scroll-pane");
scrollPane.setContent(textArea);
scrollPane.setFitToWidth(true);
scrollPane.setPrefWidth(400);
scrollPane.setPrefHeight(180);
VBox vBox = new VBox();
vBox.visibleProperty().setValue(false);
vBox.getChildren().addAll(scrollPane);
root.getChildren().addAll(vBox);


// --- Menu files
Menu filesMenu = new Menu("Files");
//Menu to load multiple files
MenuItem loadFiles
        = new MenuItem("Load + process multiple files ");
//MenuItem added to menu
filesMenu.getItems().addAll(loadFiles);
menuBar.getMenus().add(filesMenu);
root.getChildren().add(menuBar);

//Listener
loadFiles.setOnAction((ActionEvent t) ->
{
    FileChooser fileChooser = new FileChooser();
```

```
        //Name of file with extension .obj
        FileChooser.ExtensionFilter extFilter
                = new FileChooser
                        .ExtensionFilter("TXT files (*.obj)"
                                , "*.obj");
        fileChooser.getExtensionFilters().add(extFilter);
        //Files will be kept in a list
        List<File> files = new FileChooser()
                .showOpenMultipleDialog(primaryStage);
        //Each file is processed
        files.stream().forEach((File file) -> { try
        {
            //Text of file is read and mesh is extracted
            textOfFile = readFile(file);
            //Text copied to textArea
            textArea.appendText(textOfFile);

            } catch (IOException ex)
            {
                Logger.getLogger(PCAofHandBones
                        .class.getName())
                        .log(Level.SEVERE, null, ex);
            }
        });

    //To remedy a not understood bug
    //( = menuBar is removed from the graphic )
    root.getChildren().remove(menuBar);

    vBox.visibleProperty().set(true);
    });

    //====== End of File loader

    primaryStage.setTitle(title
            + ". Drag the mouse to rotate. "
        + " Ctrl + A = Axis visibility"
            );
```

```
        scene.setFill(Color.AQUAMARINE);
        primaryStage.setScene(scene);
        primaryStage.show();
        axes.buildAxes();
        axes.handleKeyboard(scene);
        root.getChildren().addAll(axes.axesXform);
        camera.handleMouse(scene);
        camera.handleMouse(scene);


        //Hide textArea
        Button hideTextArea = new Button();
        hideTextArea.setText("View graphic");
        //Listener
        hideTextArea.setOnAction((ActionEvent t) ->
        {
            vBox.visibleProperty().set(false);
            camera.buildCamera();

            world3DXform.getChildren().add(bonesXform);
            root.getChildren().add(world3DXform);
            scene.setCamera(camera.camera);
        });




        vBox.getChildren().addAll(hideTextArea);




        primaryStage.show();
    }
```

```
//File is read and its content is returned as a string.
//The associated mesh is recovered
//and correct orientation fo faces is provided.
//The mesh is a global variable.


private String readFile(File file)
        throws FileNotFoundException, IOException
{
    //Name of file is printed
    String name = file.getName();
    System.out.println(name);
    //Buffer to keep the content of file
    //StringBuilder stringBuffer = new StringBuilder();
    StringBuilder stringBuffer  = new StringBuilder();
    //Tool to read file
    BufferedReader bufferedReader = null;

    //A mesh is declared and instantiated
    TriangleMesh mesh  = new TriangleMesh();
    //The mesh has no predefined texture
    mesh.getTexCoords().addAll(0,0);
    //In the .obj format, counting begins with 1
    //but in Java, we begin with 0.
    //So, an additional inocuos point is added.
    mesh.getPoints().addAll(0,0,0);

    if (printReadFile)
        System.out.println("File content: ");
//File is read
try
{
//
bufferedReader = new BufferedReader(new FileReader(file));

String text;
```

```
while( (text = bufferedReader.readLine()) != null)
{
    //Text for the textArea
    stringBuffer.append(text);

    //Vertexes are read
    if (((text.length() > 0 ) && text.charAt(0) == 'v')
            && text.charAt(1) == ' ')
    {
        if (printReadFile)
            System.out.println("Vertex  = "
                    + counterOfVertexes + " = "+text);
        counterOfVertexes++;
        text = text.substring(2);
        int j = 0;
        //Three vertexes are read
        for(int i = 0; i < 3; i++)
        {
            double d ;
            String s = "";
            Boolean go = true;

            while (go)
            {
                char thisChar = text.charAt(j);
                s = s + thisChar;
                char nextChar = ' ';
                //++j means add one to j
                //and then evaluate expression with <
                if (++j < text.length())
                 nextChar = text.charAt(j);
                if ((nextChar == ' ') ||
                        (j >= text.length()))
                        go = false;
```

```
            }
            //string to double
            d = Double.parseDouble(s);
            float df = (float) d;
            if (printReadFile)
                System.out.println("Float number = " + df);
            mesh.getPoints().addAll(df);

        }
    }


    //Normal vectors are counted and displayed
    if ((text.length() > 0 ) && (text.charAt(0) == 'v')
            && (text.charAt(1) == 'n'))
    {

        if (printReadFile)
            System.out.println("Normal " + counterOfNormals
                + " = " + text);
        counterOfNormals++;
    }


    //Face is read, one face per line.
    //A face has 3 or more vertexes. These are extracted.
    //Additional information is ignored
    if ( (text.length() > 0 ) && (text.charAt(0) == 'f'))
    {

        if (printReadFile)
            System.out.println("Face  = " + counterOfFaces
                + " = "+text);
        counterOfFaces++;
        text = text.substring(2);

        int vertex[] = new int[4];
        int j = 0;
```

```
Boolean isNotBlank ;
//Three vertexes are extracted
for(int i = 0; i < 3; i++)
{

    double d ;
    String s = "";
    Boolean go1 = true;


    while (go1)
    {
        if (j >= text.length()-2)
            break;

        char thisChar = text.charAt(j);
        s = s + thisChar;
        /*
        System.out.println(" j = "
                + j + "    s = " + s);
                */
        char nextChar = ' ';
        j++;
        if (j < text.length())
         nextChar = text.charAt(j);
        char nextNextChar;
        isNotBlank = true;
        if (nextChar == '/' )
        {
            go1 = false;
            //go until the following blank
            while ((isNotBlank)
                    && (j < text.length()-1))
            {
                j++;
                nextNextChar = text.charAt(j);
                if (nextNextChar == ' ' )
```

```
                        isNotBlank = false;

                }
            }

        }

        d = Double.parseDouble(s);
        int v = (int) d;
        vertex[i] = v;
        if (printReadFile)
            System.out.println("Int number = " + v);
    }//End of cycle for 3 vertexes

    //A face with 3 vertexes is added to mesh
    mesh.getFaces().addAll(vertex[0], 0, vertex[1], 0,
        vertex[2], 0);

    if (printReadFile)
    {
        for(int i = 0; i < 3; i++)
         System.out.print(vertex[i] + " . ");
        System.out.println();
    }

    //Is there a four vertex in the face?
    if   ( (text.length() - j )> 3)
    {
        String s = "";
        Boolean go1 = true;
        while (go1)
        {


            char thisChar = text.charAt(j);
            s = s + thisChar;
            j++;
            char nextChar = ' ';
```

```
            if (j < text.length())
             nextChar = text.charAt(j);
            if (nextChar == '/' ) go1 = false;
        }
        if (printReadFile)
            System.out.println("4th = " + s);
        double d = Double.parseDouble(s);
        int v = (int) d;
        vertex[3] = v;

        //A polyface with 4 vertex, say, a,b,c,d,
        //can be covered by 2 triangles:
        //Triangle 1: a,b,c (Already added to mesh)
        //and triangle 2: a,c,d.
        //Triangle 2 with 3 vertexes is added to mesh
        mesh.getFaces().addAll(vertex[0], 0, vertex[2],
                0, vertex[3], 0);
    }
}//End of face cycle


    //Bounds of containing box are read and displayed
if ( (text.length() > 0 )
        && (text.contains("# Bounds(mm):")))
{
    System.out.println(text);

    String a = "# Bounds(mm): (";
    int m = a.length();
    text = text.substring(m);
    text = text.replace(")-(", ",");
    text = text.replace(")", ",");
    //System.out.println(text);

    int j = 0;


    //Six vertexes are extracted
```

```
        for(int i = 0; i < 6; i++)
        {

            double d ;
            String s = "";
            Boolean go1 = true;


            while (go1)
            {
                if (j >= text.length()-2)
                    break;

                char thisChar = text.charAt(j);
                s = s + thisChar;
                /*
                System.out.println(" j = "
                        + j + "    s = " + s);
                */
                char nextChar = ' ';
                j++;
                if (j < text.length())
                 nextChar = text.charAt(j);
                if (nextChar == ',' )
                {
                    go1 = false;
                   if ( j < text.length() ) j++;
                }

            }

            d = Double.parseDouble(s);


           if (printReadFile)
                System.out.println("Coordinate = " + i + "= " +
        }//End of cycle for 3 vertexes
    }
```

```
}

} //End of  try.
    //Exception management
    catch (FileNotFoundException ex)
    {
        Logger.getLogger(PCAofHandBones
        .class.getName()).log(Level.SEVERE, null, ex);
    }
    catch (IOException ex)
    {
        Logger.getLogger(PCAofHandBones
        .class.getName()).log(Level.SEVERE, null, ex);
    }
    finally
    {
        try
        {
            bufferedReader.close();
        } catch (IOException ex)
        {
            Logger.getLogger(PCAofHandBones
            .class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    if (printReadFile)
    {
        System.out.println("End of file content. ");
        //Other indicators are reported
        report(mesh);
    }

   //mesh = centerMesh(mesh);
    //======MeshWork=======
```

```
    PCA(mesh);

    MeshView boneMesh =
            new MeshView(mesh);

    PhongMaterial goldenMaterial
            = new PhongMaterial(Color.GOLDENROD);
    boneMesh.setMaterial(goldenMaterial);
    Group boneGroup = new Group( boneMesh);
    //Mesh is added to bonesXform
    bonesXform.getChildren().add(boneGroup);



    return stringBuffer.toString();
}
```

```
//Indicators of retrieved data are printed
private void report(TriangleMesh mesh)
{
    float[] destPoints = new float[mesh.getPoints().size()];
    mesh.getPoints().toArray(destPoints);
    System.out.println("POINTS");
    for(int j = 0; j <  destPoints.length/3; j++)
    {
        System.out.print(j + "   " );
        for(int k = 0; k < 3; k++)
        {
            System.out.print( destPoints[j*3 + k] + "   " );
        }
        System.out.println();
```

```
    }

    destFaces = new int[mesh.getFaces().size()];
    //Faces go to a matrix
    mesh.getFaces().toArray(destFaces);
    System.out.println("FACES WITH TEXTURE COORDINATES");
    for(int j = 0; j <  destFaces.length/6; j++)
    {
        System.out.print(j + "   " );
        for(int k = 0; k < 6; k++)
        {
            System.out.print( destFaces[j*6 + k] + "   " );
        }
        System.out.println();
    }

    System.out.println("counterOfPoints = "
            + counterOfVertexes );
    System.out.println("counterOfFaces = "
            + counterOfFaces );
    System.out.println("counterOfNormals = "
            + counterOfNormals);
    System.out.println("mesh.getPoints().size() = "
            + mesh.getPoints().size() );
    System.out.println("mesh.getFaces().size() = "
            + mesh.getFaces().size() );

    System.out.println("getPointElementSize = "
            + mesh.getPointElementSize() );
    System.out.println("getfaceElementSize = "
            + mesh.getFaceElementSize() );

}



public static String fixedWidthDoubletoString
```

```
        (double x, int w, int d) {
   java.text.DecimalFormat fmt
           = new java.text.DecimalFormat();
   fmt.setMaximumFractionDigits(d);
   fmt.setMinimumFractionDigits(d);
   fmt.setGroupingUsed(false);
   String s = fmt.format(x);
   while (s.length() < w) {
      s = " " + s;
   }
   return s;
}

//Principal component analysis of a mesh
private void PCA(TriangleMesh mesh)
{
    //Point are declared as a matrix of coordinates
    int n = mesh.getPoints().size();
        float nPoints = n;
    nPoints = nPoints/3;
    float[][] CoordinatesFloat = new float[(int) nPoints][3];


    /*
    //Test: adapted from
    //Stat Trek: Variance-Covariance Matrix
    //http://stattrek.com/matrix-algebra/covariance-matrix.aspx

    float[] data = {90,  60,  90,
                    90,  90,  30,
                    60,  60,  60,
                    60,  60,  90,
                    30,  30,  30 };


        n = 15;
        nPoints = 5;
    int counter = 0;
```

```
for(int i = 0; i < n; i = i+3)
{
    //Reading
    float x =  data[i];
    float y =  data[i+1] ;
    float z =  data[i+2] ;
    CoordinatesFloat[counter][0] = x;
    CoordinatesFloat[counter][1] = y;
    CoordinatesFloat[counter][2] = z;
    counter++;
}

//Mean is found
float[] mean = new float[3];

for(int i = 0; i < 3; i++)
{
    mean[i] = 0;
    for (int j = 0; j <  nPoints; j++)
        mean[i] = mean[i] + CoordinatesFloat[j][i];
    mean[i] = mean[i]/(nPoints);
}

System.out.println("meanX = " + mean[0]);
System.out.println("meanY = " + mean[1]);
System.out.println("meanZ = " + mean[2]);




//Calculate covariance matrix
double[][] Cov = new double[3][3];
for(int i = 0; i < 3; i++)
{
    for (int j = 0; j <  3; j++)
    {
        Cov[i][j] = 0;
        for (int k = 0; k <  nPoints; k++)
```

```
                Cov[i][j] = Cov[i][j] +
                        (CoordinatesFloat[k][i] - mean[i])
                        * (CoordinatesFloat[k][j] - mean[j]);

            Cov[i][j] = Cov[i][j]/nPoints;

        }

    }

    printMatrix(Cov, "Cov");

    //Answer: cov matrix must be:
    //504  360  180
    //360  360  0
    //180  0  720
    */

    //=============

    int counter = 0;
    for(int i = 0; i < n; i = i+3)
    {
        //Reading
        float x =  mesh.getPoints().get(i);
        float y =  mesh.getPoints().get(i+1);
        float z =  mesh.getPoints().get(i+2);
        CoordinatesFloat[counter][0] = x;
        CoordinatesFloat[counter][1] = y;
        CoordinatesFloat[counter][2] = z;
        counter++;
        //Warning: the zeroth point is artificial,
        //filled with zeros.
        if (i == 0)
            System.out.println("Zeroth point: x = " + x + "   y =
                    + y + "   z =  "  + z);
    }
```

```
//Mean is found. It is vector with 3 coordinates
float[] mean = new float[3];

for(int i = 0; i < 3; i++)
{
    mean[i] = 0;
    //Zeroth point does not count
    for (int j = 1; j <  nPoints; j++)
        mean[i] = mean[i] + CoordinatesFloat[j][i];
    mean[i] = mean[i]/(nPoints -1);
}

System.out.println("meanX = " + mean[0]);
System.out.println("meanY = " + mean[1]);
System.out.println("meanZ = " + mean[2]);



//Calculate covariance matrix
double[][] Cov = new double[3][3];
for(int i = 0; i < 3; i++)
{
    for (int j = 0; j <  3; j++)
    {
        Cov[i][j] = 0;
        for (int k = 1; k <  nPoints; k++)
            Cov[i][j] = Cov[i][j] +
                    (CoordinatesFloat[k][i] - mean[i])
                    * (CoordinatesFloat[k][j] - mean[j]);

        Cov[i][j] = Cov[i][j]/(nPoints-1);

    }

}

printMatrix(Cov, "Cov");
```

```
//=========

//Eigenvalues of Cov matrix are found
Matrix M = new Matrix(Cov);
EigenvalueDecomposition E =
     new EigenvalueDecomposition(M);
double[] d = E.getRealEigenvalues();

for(int i = 0; i < 3; i++)
    System.out.println("Eigenvalue " + i + " : " + d[i]);
double[][] EV = E.getV().getArray();
for(int i = 0; i < 3; i++)
{
    System.out.println("Eigenvector " + i + " : " );
    for (int j = 0; j <  3; j++)
    {
        System.out.println(EV[i][j]);
    }
    System.out.println();
}

//Test: eigenvectors of a real symmetric matrix
//are orthogonal one to another: dot product must be zero
System.out.println("Orthogonality test : matrix I is expecte
double[][] Ort = new double[3][3];
for(int i = 0; i < 3; i++)
{

    for (int j = 0; j <  3; j++)
    {

        //sum = sum + EV[i][j]*EV[i+1][j];
        Ort[i][j] = dotProduct(EV[i], EV[j]);

    }
    //System.out.println(sum);
```

```
}

printMatrix(Ort, "Ort test");

//Determinant must be 1.
System.out.println("Orientation test: "
        + "+1 is expected. Det E = " + E.getV().det() + " \n");


 //Mesh is centered
float [] pointsFloat = new float[n];
for(int i = 0; i < n; i = i+3)
{
    //Reading
    float x =  mesh.getPoints().get(i);
    float y =  mesh.getPoints().get(i+1);
    float z =  mesh.getPoints().get(i+2);

    //Mean is subtracted
    pointsFloat[i]    =  scaleFactor*(x - mean[0]);
    pointsFloat[i+1]  =  15*(y - mean[1]);
    pointsFloat[i+2]  =  15*(z - mean[2]);
}
//Resetting of points of mesh
mesh.getPoints().setAll(pointsFloat, 0, n);

axes.axesGroup.setVisible(true);




//1D array for recording
float[]rotPoints = new float[n];
//Mesh is aligned with the new basis
//given by the PCA eigenvetors
for(int i = 0; i < n; i = i+3)
{
     //Reading
```

```
        float x =  mesh.getPoints().get(i);
        float y =  mesh.getPoints().get(i+1);
        float z =  mesh.getPoints().get(i+2);
        //Three coordinates define a point
        double[] w = {x,y,z};
        //Coordinates of point w are transformed
        //to the new basis
        w = matrixTimesVector(E.getV()
                .transpose().getArray(),w);

        //Point is added to the  1-dimensional array
        //PCA scaling is applied
        rotPoints[i ] = (float) (scaleFactorW * w[0] );
        rotPoints[i + 1] = (float) (scaleFactorH * w[1] );
        rotPoints[i + 2] = (float) (scaleFactorL * w[2] );
    }
    //Points are reset
    mesh.getPoints().setAll(rotPoints, 0, n);

}

//============== Matrix algebra ==============


private static void printMatrix(double m[][], String title)
{
    System.out.println("This is the matrix " + title);
    for(int i=0;i<3;i++)
        System.out.println(m[i][0] + " " + m[i][1]+ " "
                                            + m[i][2]);
    System.out.println();
}




//The multiplication of a matrix times a vector
private static  double[] matrixTimesVector(double m[][],
```

```
                                                double v[])
{
    double[]  helpV = new double[3];
    for(int i=0; i<3; i++)
        helpV[i] =0;
    for(int i=0; i<3; i++)
        for(int j=0; j<3; j++)
            helpV[i]  =helpV[i]  + m[i][j]*v[j];
    return helpV;
}




//Normalized crux product of vectors v and w
private static  double dotProduct(double[] v,
                                        double[] w)
{
    return v[0]*w[0] + v[1]*w[1] + v[2]*w[2];

}




public static void main(String[] args)
{

    launch(args);
}

}//End of main class N42 PCAmutations
```

43, page 159. To implement cylindrical mutations one just need to give the same value to scaleFactorW and scalefactorH of PCA mutations. Cylindrical mutations also produce monstrosities.

44, page 159. The following program implements PCA mutations over the bones of the hand separately, shows the mutant hand and animates it to see the effect of mutations. Our program allows a great span of PCA mutations, magnifying up to 9 times and zooming out by the same factor. It is clear that PCA mutations must have a very small effect to be admitted by nature.

```
/*
    Program N44 PCAmutations2
    This program imports a set  of meshes that  encode
    for the bones of the hand.
    ".obj" format.

    Content is shown in a textArea and
    encoded meshes are drawn, PCA mutated and animated.
    PCA mutations change scale along the natural
    direction of the bone: width, height and length.

    One can select multiple files in the file chooser.

    Tested only with some  files,
    particularly  with those of the BodyParts3D project.
    http://lifesciencedb.jp/bp3d/
    Those files are licensed:

    The license for this database is specified in the
    Creative Commons Attribution-Share Alike 2.1 Japan.
    If you use data from this database, please be sure
    attribute this database as follows: "BodyParts3D, (c)
    The Database Center for Life Science licensed under
    CC Attribution-Share Alike 2.1 Japan".
    http://dbguidehive.biosciencedbc.jp/en/bodyparts3d/lic.html

 */

package Programs;

//Files in the hosting project Programs
```

```
import Jama.EigenvalueDecomposition;
import Jama.Matrix;
import Tools.Axes;
import Tools.Cameras;
import Tools.Xform;
import Tools.XformWithPivot;


import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.List;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.binding.NumberBinding;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.Slider;
```

```
import javafx.scene.control.TextArea;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.MeshView;;
import javafx.scene.shape.TriangleMesh;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import javafx.util.Duration;



public class PCAmutations2 extends Application
{
    //Visibility of the mesh
    //might depend on the correct choice
    //of this factor. Move it in within 1 and 500.
    private static final double scaleFactor = 4;

    //Scale factors along width, height and length
    static private float scaleFactorW = 1;
    static private float scaleFactorH = 1;
    static private float scaleFactorL = 1;

    //Printing for debugging
    final static boolean printReadFile = false;
    private static final String title
            =  "Importer of meshes in format ";

    //Tool to read files
    BufferedReader br ;
    private static  String textOfFile = "";

    private static int counterOfFaces = 0;
    private static int counterOfVertexes = 0;
```

```
private static int counterOfNormals = 0;

private static int[] destFaces ;

private static TriangleMesh[] bonesMeshesArray  = new TriangleMesh[
//Array of Xforms
private static Xform[] bonesXformsArray  = new Xform[100];
private static MeshView[] bonesViewArray  = new MeshView[100];
private static final String[] names1  = new String[100];
private static final String[] names2  = new String[100];
//Group of Xforms
private static Xform bonesXforms  = new Xform();
//Counter
private static int counterOfBones = 0;

//*****************

final static Group root = new Group();

final static Group axisGroup = new Group();


final static Xform cameraXform = new Xform();
final static Xform cameraXform2 = new Xform();
final static Xform cameraXform3 = new Xform();

final static double cameraDistance = 0; //1450;//450
final static Xform world3DXform = new Xform();
final static Xform bonesXform = new Xform();

//Axes : ctrl + A
private static final Axes axes = new Axes();
private static final Cameras camera = new Cameras();

boolean timelinePlaying = false;
double ONE_FRAME = 1.0/24.0;
double DELTA_MULTIPLIER = 200.0;
double CONTROL_MULTIPLIER = 0.1;
```

```
    double SHIFT_MULTIPLIER = 0.1;
    double ALT_MULTIPLIER = 0.5;

    double mousePosX;
    double mousePosY;
    double mouseOldX;
    double mouseOldY;
    double mouseDeltaX;
    double mouseDeltaY;




   //Menubar declaration and initialization
    MenuBar menuBar = new MenuBar();


//This is the outer pane to place tabs
private static BorderPane outerPane = new BorderPane();
private static Pane paneHand = new Pane();

//Slider for zooming
    //private  static Slider zSlider;
    //Position of graphic along the z-axis,
    //normal to the screen.
    //It is a double property because
    //it will get involved into a binding
    //private  static final DoubleProperty zPos
    //        = new SimpleDoubleProperty(200);

    //Slider for angle of rotation of fingers
    private  static Slider closeHandSlider;
    //To close the hand,
    //each phalange suffers a rotation.
    //It is a double property because
    //it will get involved into a binding
    private  static final DoubleProperty closeHandProperty
            = new SimpleDoubleProperty(0);
```

```
//Slider for opening fingers thanks to
//their transversal rotation
private  static Slider openFingersSlider;
//To open the fingers,
//each phalange suffers a transversal rotation.
//It is a double property because
//it will get involved into a binding
private  static final DoubleProperty openFingersProperty
        = new SimpleDoubleProperty(0);

//Slider for opening/closing the thumb
private  static Slider closeThumbSlider;
//To close the thumb we need a rotation.
//It is a double property because
//it will get involved into a binding
private  static final DoubleProperty closeThumbProperty
        = new SimpleDoubleProperty(0);

//Slider for rotating the thumb to form the grip
private  static Slider rotateThumbSlider;
//To form the grip we need a rotation of the thumb.
//It is a double property because
//it will get involved into a binding
private  static final DoubleProperty rotateThumbProperty
        = new SimpleDoubleProperty(0);


//Nested definition of   nodes is implemented
//to have, say,  the palm dragging the fingers,
//and a phalanx dragging the rest of its the finger.


//Each finger generates various natural groups:
//Group with the extreme phalanx of finger0
//Symbolic initialization
private static final XformWithPivot finger0Ph2Group
        = new XformWithPivot();
```

```
    //Group with the two extreme phalanges
    private static final XformWithPivot finger0Ph21Group
            = new XformWithPivot();
    //Group with all three phalanges. It is the finger
    private static final XformWithPivot finger0Ph210Group
            = new XformWithPivot();
    //Finger1
    private static final XformWithPivot finger1Ph2Group
            = new XformWithPivot();
    private static final XformWithPivot finger1Ph21Group
            = new XformWithPivot();
    private static final XformWithPivot finger1Ph210Group
            = new XformWithPivot();
    //Finger2
    private static final XformWithPivot finger2Ph2Group
            = new XformWithPivot();
    private static final XformWithPivot finger2Ph21Group
            = new XformWithPivot();
    private static final XformWithPivot finger2Ph210Group
            = new XformWithPivot();
    //Finger3
    private static final XformWithPivot finger3Ph2Group
            = new XformWithPivot();
    private static final XformWithPivot finger3Ph21Group
            = new XformWithPivot();
    private static final XformWithPivot finger3Ph210Group
            = new XformWithPivot();
    //Finger4
    private static final XformWithPivot finger4Ph2Group
            = new XformWithPivot();
    private static final XformWithPivot finger4Ph21Group
            = new XformWithPivot();
    private static final XformWithPivot finger4Ph210Group
            = new XformWithPivot();
    //Thumb
    private static final XformWithPivot thumbPh2Group
            = new XformWithPivot();
    private static final XformWithPivot thumbPh21Group
```

```
            = new XformWithPivot();
    private static final XformWithPivot thumbPh210Group
            = new XformWithPivot();

     private static final XformWithPivot wristGroup
            = new XformWithPivot();
    //Group with wrist and fingers
    private static final XformWithPivot palmGroup
            = new XformWithPivot();

    private static final Timeline animation  = new Timeline();

    private static double[][] coordBoxes = new double[100][6];



    Random r = new Random();

@Override
public void start(Stage primaryStage)
{
    System.out.println("start");

    Scene scene = new Scene(root, 1200, 1000, true);

    paneHand.setPrefSize(500, 500);

    outerPane.setOpacity(0);
    outerPane.setLayoutX(700);
    outerPane.setLayoutY(700);

    /*
       Multiple file loader
    */

    //Content of files will be shown in textArea
    final TextArea textArea = new TextArea();
    textArea.prefWidth(400);
```

```
textArea.setWrapText(true);
textArea.setText("");
//Container  for textArea.
ScrollPane scrollPane = new ScrollPane();
scrollPane.getStyleClass().add("noborder-scroll-pane");
scrollPane.setContent(textArea);
scrollPane.setFitToWidth(true);
scrollPane.setPrefWidth(400);
scrollPane.setPrefHeight(180);
VBox vBox = new VBox();
vBox.visibleProperty().setValue(false);
vBox.getChildren().addAll(scrollPane);
root.getChildren().addAll(vBox);


// --- Menu files
Menu filesMenu = new Menu("Files");
//Menu to load multiple files
MenuItem loadFiles
        = new MenuItem("Load + process multiple files ");
//MenuItem added to menu
filesMenu.getItems().addAll(loadFiles);
menuBar.getMenus().add(filesMenu);
root.getChildren().add(menuBar);

//Listener
loadFiles.setOnAction((ActionEvent t) ->
{
    FileChooser fileChooser = new FileChooser();

    //Name of file with extension "
    FileChooser.ExtensionFilter extFilter
            = new FileChooser
                    .ExtensionFilter("TXT files (*.obj)"
                            , "*.obj");
    fileChooser.getExtensionFilters().add(extFilter);

    //Files will be kept in a list
```

```
List<File> files = new FileChooser()
        .showOpenMultipleDialog(primaryStage);



    //Delete possible repetitions
    //files = deleteRepetitions(files);
//counterOfBones = 0;



for(int j = 0; j < files.size(); j++)
{
    File file = files.get(j);
    String name = file.getName();
    names1[j] = name;
}
counterOfBones = 0;
//Each file is processed
for(int j = 0; j < files.size(); j++)
{

    String name = names1[j];
    boolean repeated = false;
    for (int i = 0; i <  j; i++)
    {
        if (name.equals(names1[i]))
        {
            repeated = true;

            //System.out.println("Repeated" + name);
        }
    }

    if (repeated);
    else
    try
    {
```

```
                File file = files.get(j);
                //Text of file is read and mesh is extracted
                textOfFile = readFile(file);
                //Text copied to textArea
                textArea.appendText(textOfFile);

            } catch (IOException ex)
                {
                    Logger.getLogger(AnimatedHandSkeleton
                            .class.getName())
                            .log(Level.SEVERE, null, ex);
                }



        }

        defineGroups();



    //To remedy a not understood bug
    //( = menuBar is added to the graphic )
    root.getChildren().remove(menuBar);

    vBox.visibleProperty().set(true);
    });

    //====== End of File loader

    primaryStage.setTitle(title
            + ". Drag the mouse to rotate. "
        + " Ctrl + A = Axis visibility"
            );
    scene.setFill(Color.BLUE);
    primaryStage.setScene(scene);
    primaryStage.show();
```

```
axes.buildAxes();
axes.handleKeyboard(scene);
//camera.handleMouse(scene);
camera.handleMouse(paneHand);

//Hide textArea and show graphic
Button hideTextArea = new Button();
hideTextArea.setText("View graphic");
//Listener
hideTextArea.setOnAction((ActionEvent t) ->
{
    vBox.visibleProperty().set(false);
    camera.buildCamera();
    bonesXform.setScale(scaleFactor);



    /*
    //Centering: found by trial and error.
    double xMax = bonesXform.getBoundsInLocal().getMaxX();
    double yMax = bonesXform.getBoundsInLocal().getMaxY();
    double zMax = bonesXform.getBoundsInLocal().getMaxZ();
    double xMin = bonesXform.getBoundsInLocal().getMinX();
    double yMin = bonesXform.getBoundsInLocal().getMinY();
    double zMin = bonesXform.getBoundsInLocal().getMinZ();
    double x = scaleFactor *  0.82 * (xMax + xMin) ;
    double y = scaleFactor * 0.78 * (yMax + yMin);
    double z = scaleFactor * 0.9 * (zMax + zMin) ;

    world3DXform.setTranslate(-x, -y, -z);

    */
    world3DXform.getChildren().add(bonesXform);
    world3DXform.setScale(scaleFactor);


    //world3DXform.setRx(180);
    //world3DXform.setRy(90);
```

```
        world3DXform.setScale(2);
        world3DXform.setTranslate(-1300, -800, 0);
        paneHand.getChildren().add(world3DXform);
        root.getChildren().add(paneHand);
        scene.setCamera(camera.camera);
    });

    vBox.getChildren().addAll(hideTextArea);



    //This slider   controls the closing of the hand.
    closeHandSlider = new Slider();
    closeHandSlider.setMax(90);
    closeHandSlider.setMin(0);
    closeHandSlider.setValue(10);

    //To close the hand, an angle of rotation is enabled.
    //Bind angleProperty to the reading of the slider
    closeHandSlider.valueProperty()
            .bindBidirectional(closeHandProperty);


    //This slider controls the openning of fingers.
    openFingersSlider = new Slider();
    openFingersSlider.setMax(15);
    openFingersSlider.setMin(0);
    openFingersSlider.setValue(10);

    //To open fingers, an angle of rotation is enabled.
    //Bind openFingersProperty to the reading of the slider
    openFingersSlider.valueProperty()
            .bindBidirectional(openFingersProperty);


    //Slider to open  the thumb
    closeThumbSlider = new Slider();
    closeThumbSlider.setMax(55);
```

```
closeThumbSlider.setMin(0);
closeThumbSlider.setValue(-0);
closeThumbSlider.adjustValue(closeThumbSlider.getValue());

//To open the thumb, an angle of rotation is enabled.
//Bind closeThumbProperty to the reading of the slider
closeThumbSlider.valueProperty().bindBidirectional(closeThumbProper


//Slider to rotate  the thumb
rotateThumbSlider = new Slider();
rotateThumbSlider.setMax(50);
rotateThumbSlider.setMin(0);
rotateThumbSlider.setValue(0);
rotateThumbSlider.adjustValue(rotateThumbSlider.getValue());

//To rotate the thumb, an angle of rotation is enabled.
//Bind closeThumbProperty to the reading of the slider
rotateThumbSlider.valueProperty()
        .bindBidirectional(rotateThumbProperty);



//Other JavaFX controls, labels and containers
GridPane gridPane = new GridPane();
gridPane.setVgap(4);
gridPane.setHgap(10);
gridPane.setPadding(new Insets(5, 5, 5, 5));
//Column 0, Row   0
gridPane.add(new Label("Zoom: "), 0, 0);
//gridPane.add(zSlider, 1, 0);
gridPane.add(new Label("Close hand: "), 2, 0);
gridPane.add(closeHandSlider, 3, 0);
gridPane.add(new Label("Open fingers: "), 0, 1);
gridPane.add(openFingersSlider, 1, 1);
gridPane.add(new Label("Close thumb: "), 0, 2);
gridPane.add(closeThumbSlider, 1, 2);
gridPane.add(new Label("Rotate thumb: "), 2, 2);
```

```
    gridPane.add(rotateThumbSlider, 3,2);



    SubScene subScene
            = new SubScene(gridPane, 880, 200);
    subScene.setLayoutX(10);
    subScene.setLayoutY(800);
    paneHand.getChildren().add(subScene);









    primaryStage.show();
}




//File is read and its content is returned as a string.
//The associated mesh is recovered
//and correct orientation of faces is provided.
//The mesh is a global variable.
private  String readFile(File file)
        throws FileNotFoundException, IOException
{
```

```
    //Name of file is printed
    String name = file.getName();
    if (printReadFile)
        System.out.println(name);
    names2[counterOfBones] = name;
    //Buffer to keep the content of file
    //StringBuilder stringBuffer = new StringBuilder();
    StringBuilder stringBuffer  = new StringBuilder();
    //Tool to read file
    BufferedReader bufferedReader = null;

    //A mesh is declared and instantiated
    TriangleMesh mesh  = new TriangleMesh();
    //The mesh has no predefined texture
    mesh.getTexCoords().addAll(0,0);
    //In the " format, counting begins with 1
    //but in Java, we begin with 0.
    //So, an additional inocuos point is added.
    mesh.getPoints().addAll(0,0,0);

    if (printReadFile)
        System.out.println("File content: ");
//File is read
try
{
//
bufferedReader = new BufferedReader(new FileReader(file));

String text;



while( (text = bufferedReader.readLine()) != null)
{
    //Text for the textArea
    stringBuffer.append(text);
    //Print each line
```

```
//System.out.println(text);

//Vertexes are read
if (((text.length() > 0 ) && text.charAt(0) == 'v')
        && text.charAt(1) == ' ')
{
    if (printReadFile)
        System.out.println("Vertex  = "
                + counterOfVertexes + " = "+text);
    counterOfVertexes++;
    text = text.substring(2);
    int j = 0;
    //Three vertexes are read
    for(int i = 0; i < 3; i++)
    {
        double d ;
        String s = "";
        Boolean go = true;

        while (go)
        {
            char thisChar = text.charAt(j);
            s = s + thisChar;
            char nextChar = ' ';
            //++j means add one to j
            //and then evaluate expression with <
            if (++j < text.length())
             nextChar = text.charAt(j);
            if ((nextChar == ' ') ||
                    (j >= text.length()))
                    go = false;

        }
        //string to double
        d = Double.parseDouble(s);
        float df = (float) d;
        if (printReadFile)
            System.out.println("Float number = " + df);
```

```
        mesh.getPoints().addAll(df);

    }
}


//Normal vectors are counted and displayed
if ((text.length() > 0 ) && (text.charAt(0) == 'v')
        && (text.charAt(1) == 'n'))
{

    if (printReadFile)
        System.out.println("Normal " + counterOfNormals
            + " = " + text);
    counterOfNormals++;
}


//Face is read, one face per line.
//A face has 3 or more vertexes. These are extracted.
//Additional information is ignored
if ( (text.length() > 0 ) && (text.charAt(0) == 'f'))
{

    if (printReadFile)
        System.out.println("Face  = " + counterOfFaces
            + " = "+text);
    counterOfFaces++;
    text = text.substring(2);

    int vertex[] = new int[4];
    int j = 0;

    Boolean isNotBlank ;
    //Three vertexes are extracted
    for(int i = 0; i < 3; i++)
    {
```

```
double d ;
String s = "";
Boolean go1 = true;


while (go1)
{
    if (j >= text.length()-2)
        break;

    char thisChar = text.charAt(j);
    s = s + thisChar;
    /*
    System.out.println(" j = "
            + j + "    s = " + s);
            */
    char nextChar = ' ';
    j++;
    if (j < text.length())
     nextChar = text.charAt(j);
    char nextNextChar = ' ';
    isNotBlank = true;
    if (nextChar == '/' )
    {
        go1 = false;
        //go until the following blank
        while ((isNotBlank)
                && (j < text.length()-1))
        {
            j++;
            nextNextChar = text.charAt(j);
            if (nextNextChar == ' ' )
                isNotBlank = false;

        }
    }

}
```

```
    d = Double.parseDouble(s);
    int v = (int) d;
    vertex[i] = v;
    if (printReadFile)
        System.out.println("Int number = " + v);
}//End of cycle for 3 vertexes

//A face with 3 vertexes is added to mesh
mesh.getFaces().addAll(vertex[0], 0, vertex[1], 0,
    vertex[2], 0);

if (printReadFile)
{
    for(int i = 0; i < 3; i++)
     System.out.print(vertex[i] + " . ");
    System.out.println();
}

//Is there a four vertex in the face?
if   ( (text.length() - j )> 3)
{
    String s = "";
    Boolean go1 = true;
    while (go1)
    {


        char thisChar = text.charAt(j);
        s = s + thisChar;
        j++;
        char nextChar = ' ';
        if (j < text.length())
         nextChar = text.charAt(j);
        if (nextChar == '/' ) go1 = false;
    }
    if (printReadFile)
        System.out.println("4th = " + s);
```

```
            double d = Double.parseDouble(s);
            int v = (int) d;
            vertex[3] = v;

            //A polyface with 4 vertex, say, a,b,c,d,
            //can be covered by 2 triangles:
            //Triangle 1: a,b,c (Already added to mesh)
            //and triangle 2: a,c,d.
            //Triangle 2 with 3 vertexes is added to mesh
            mesh.getFaces().addAll(vertex[0], 0, vertex[2],
                    0, vertex[3], 0);
        }
    }//End of face cycle

    //Bounds of containing box are read and displayed
    if ( (text.length() > 0 )
            && (text.contains("# Bounds(mm):")))
    {
        //System.out.println(text);

        String a = "# Bounds(mm): (";
        int m = a.length();
        text = text.substring(m);
        text = text.replace(")-(", ",");
        text = text.replace(")", ",");
        //System.out.println(text);

        int j = 0;


        //Six vertexes are extracted
        for(int i = 0; i < 6; i++)
        {

            double d ;
            String s = "";
            Boolean go1 = true;
```

```
            while (go1)
            {
                if (j >= text.length()-2)
                    break;

                char thisChar = text.charAt(j);
                s = s + thisChar;
                /*
                System.out.println(" j = "
                        + j + "    s = " + s);
                */
                char nextChar = ' ';
                j++;
                if (j < text.length())
                 nextChar = text.charAt(j);
                if (nextChar == ',' )
                {
                    go1 = false;
                   if ( j < text.length() ) j++;
                }

            }

            d = Double.parseDouble(s);

            coordBoxes[counterOfBones][i] = d;
           if (printReadFile)
                System.out.println("Coordinate = " + i + "= " + d);
        }//End of cycle for 3 vertexes
    }

}

} //End of  try.
    //Exception management
    catch (FileNotFoundException ex)
    {
```

```
        Logger.getLogger(AnimatedHandSkeleton
        .class.getName()).log(Level.SEVERE, null, ex);
    }
    catch (IOException ex)
    {
        Logger.getLogger(AnimatedHandSkeleton
        .class.getName()).log(Level.SEVERE, null, ex);
    }
    finally
    {
        try
        {
            bufferedReader.close();
        } catch (IOException ex)
        {
            Logger.getLogger(AnimatedHandSkeleton
            .class.getName()).log(Level.SEVERE, null, ex);
        }
    }



    if (printReadFile)
    {
        System.out.println("End of file content. ");
        //Other indicators are reported
        report(mesh);
    }


    //======MeshWork=======

    PCA(mesh);

    MeshView boneMesh =
            new MeshView(mesh);

    PhongMaterial goldenMaterial
```

```
            = new PhongMaterial(Color.GOLDENROD);
    boneMesh.setMaterial(goldenMaterial);
    Group boneGroup = new Group(boneMesh);
    bonesViewArray[counterOfBones] = boneMesh;

    Xform boneXform = new Xform();
    boneXform.getChildren().add(boneGroup);
    bonesXformsArray[counterOfBones] = boneXform;
    bonesXform.getChildren().add(boneXform);
    counterOfBones++;

    return stringBuffer.toString();
}


//Indicators of retrieved data are printed
private void report(TriangleMesh mesh)
{
    float[] destPoints = new float[mesh.getPoints().size()];
    mesh.getPoints().toArray(destPoints);
    System.out.println("POINTS");
    for(int j = 0; j <  destPoints.length/3; j++)
    {
        System.out.print(j + "   " );
        for(int k = 0; k < 3; k++)
        {
            System.out.print( destPoints[j*3 + k] + "   " );
        }
        System.out.println();
    }

    destFaces = new int[mesh.getFaces().size()];
    //Faces go to a matrix
    mesh.getFaces().toArray(destFaces);
    System.out.println("FACES WITH TEXTURE COORDINATES");
    for(int j = 0; j <  destFaces.length/6; j++)
    {
        System.out.print(j + "   " );
```

```
        for(int k = 0; k < 6; k++)
        {
            System.out.print( destFaces[j*6 + k] + "   " );
        }
        System.out.println();
    }

    System.out.println("counterOfPoints = "
            + counterOfVertexes );
    System.out.println("counterOfFaces = "
            + counterOfFaces );
    System.out.println("counterOfNormals = "
            + counterOfNormals);
    System.out.println("mesh.getPoints().size() = "
            + mesh.getPoints().size() );
    System.out.println("mesh.getFaces().size() = "
            + mesh.getFaces().size() );

    System.out.println("getPointElementSize = "
            + mesh.getPointElementSize() );
    System.out.println("getfaceElementSize = "
            + mesh.getFaceElementSize() );

}



public static String fixedWidthDoubletoString
        (double x, int w, int d) {
   java.text.DecimalFormat fmt
           = new java.text.DecimalFormat();
   fmt.setMaximumFractionDigits(d);
   fmt.setMinimumFractionDigits(d);
   fmt.setGroupingUsed(false);
   String s = fmt.format(x);
   while (s.length() < w) {
      s = " " + s;
   }
```

```
   return s;
}

//Principal component analysis of a mesh
private void PCA(TriangleMesh mesh)
{
    //Point are declared as a matrix of coordinates
    int n = mesh.getPoints().size();
        float nPoints = n;
    nPoints = nPoints/3;
    float[][] CoordinatesFloat = new float[(int) nPoints][3];


    /*
    //Test: adapted from
    //Stat Trek: Variance-Covariance Matrix
    //http://stattrek.com/matrix-algebra/covariance-matrix.aspx

    float[] data = {90,  60,  90,
                    90,  90,  30,
                    60,  60,  60,
                    60,  60,  90,
                    30,  30,  30 };


        n = 15;
        nPoints = 5;
    int counter = 0;
    for(int i = 0; i < n; i = i+3)
    {
        //Reading
        float x =  data[i];
        float y =  data[i+1] ;
        float z =  data[i+2] ;
        CoordinatesFloat[counter][0] = x;
        CoordinatesFloat[counter][1] = y;
        CoordinatesFloat[counter][2] = z;
        counter++;
```

```
}

//Mean is found
float[] mean = new float[3];

for(int i = 0; i < 3; i++)
{
    mean[i] = 0;
    for (int j = 0; j <  nPoints; j++)
        mean[i] = mean[i] + CoordinatesFloat[j][i];
    mean[i] = mean[i]/(nPoints);
}

System.out.println("meanX = " + mean[0]);
System.out.println("meanY = " + mean[1]);
System.out.println("meanZ = " + mean[2]);



//Calculate covariance matrix
double[][] Cov = new double[3][3];
for(int i = 0; i < 3; i++)
{
    for (int j = 0; j <  3; j++)
    {
        Cov[i][j] = 0;
        for (int k = 0; k <  nPoints; k++)
            Cov[i][j] = Cov[i][j] +
                    (CoordinatesFloat[k][i] - mean[i])
                    * (CoordinatesFloat[k][j] - mean[j]);

        Cov[i][j] = Cov[i][j]/nPoints;

    }

}
```

```
printMatrix(Cov, "Cov");

//Answer: cov matrix must be:
//504  360  180
//360  360  0
//180  0  720
*/

//=============

int counter = 0;
for(int i = 0; i < n; i = i+3)
{
    //Reading
    float x =  mesh.getPoints().get(i);
    float y =  mesh.getPoints().get(i+1);
    float z =  mesh.getPoints().get(i+2);
    CoordinatesFloat[counter][0] = x;
    CoordinatesFloat[counter][1] = y;
    CoordinatesFloat[counter][2] = z;
    counter++;
    //Warning: the zeroth point is artificial,
    //filled with zeros.
    if (i == 0)
        System.out.println("Zeroth point: x = " + x + "   y =   "
                + y + "   z =  "  + z);
}

//Mean is found. It is vector with 3 coordinates
float[] mean = new float[3];

for(int i = 0; i < 3; i++)
{
    mean[i] = 0;
    //Zeroth point does not count
    for (int j = 1; j <  nPoints; j++)
        mean[i] = mean[i] + CoordinatesFloat[j][i];
    mean[i] = mean[i]/(nPoints -1);
```

```
}

System.out.println("meanX = " + mean[0]);
System.out.println("meanY = " + mean[1]);
System.out.println("meanZ = " + mean[2]);



//Calculate covariance matrix
double[][] Cov = new double[3][3];
for(int i = 0; i < 3; i++)
{
    for (int j = 0; j <  3; j++)
    {
        Cov[i][j] = 0;
        for (int k = 1; k <  nPoints; k++)
            Cov[i][j] = Cov[i][j] +
                    (CoordinatesFloat[k][i] - mean[i])
                    * (CoordinatesFloat[k][j] - mean[j]);

        Cov[i][j] = Cov[i][j]/(nPoints-1);

    }

}

printMatrix(Cov, "Cov");

//=========

//Eigenvalues of Cov matrix are found
Matrix M = new Matrix(Cov);
EigenvalueDecomposition E =
     new EigenvalueDecomposition(M);
double[] d = E.getRealEigenvalues();

for(int i = 0; i < 3; i++)
```

```
    System.out.println("Eigenvalue " + i + " : " + d[i]);
double[][] EV = E.getV().getArray();
for(int i = 0; i < 3; i++)
{
    System.out.println("Eigenvector " + i + " : " );
    for (int j = 0; j <  3; j++)
    {
        System.out.println(EV[i][j]);
    }
    System.out.println();
}

//Test: eigenvectors of a real symmetric matrix
//are orthogonal one to another: dot product must be zero
System.out.println("Orthogonality test : matrix I is expected");
double[][] Ort = new double[3][3];
for(int i = 0; i < 3; i++)
{

    for (int j = 0; j <  3; j++)
    {

        //sum = sum + EV[i][j]*EV[i+1][j];
        Ort[i][j] = dotProduct(EV[i], EV[j]);

    }
    //System.out.println(sum);

}

printMatrix(Ort, "Ort test");

//Determinant must be 1.
System.out.println("Orientation test: "
        + "+1 is expected. Det E = " + E.getV().det() + " \n");


 //Mesh is centered
```

```
float [] pointsFloat = new float[n];
for(int i = 0; i < n; i = i+3)
{
    //Reading
    float x =  mesh.getPoints().get(i);
    float y =  mesh.getPoints().get(i+1);
    float z =  mesh.getPoints().get(i+2);

    //Mesh is centered
    pointsFloat[i]    =  (x - mean[0]);
    pointsFloat[i+1]  =  (y - mean[1]);
    pointsFloat[i+2]  =  (z - mean[2]);
}
//Resetting of points of mesh
mesh.getPoints().setAll(pointsFloat, 0, n);

axes.axesGroup.setVisible(true);

//Scale factors are composed
scaleFactors();


/*
//Test: slight magnification
scaleFactorW = 1.1f;
scaleFactorH = 1.1f;
scaleFactorL = 1.1f;
*/

//Scale factors as a diagonal matrix
double[][] scaleMatrix = {{scaleFactorW, 0,0},
                          {0, scaleFactorH, 0},
                          {0,0, scaleFactorL}};



//1D array for recording
float[]newPoints = new float[n];
```

```
    //Mesh is aligned with the new basis
    //given by the PCA eigenvetors
    for(int i = 0; i < n; i = i+3)
    {
         //Reading
        float x =  mesh.getPoints().get(i);
        float y =  mesh.getPoints().get(i+1);
        float z =  mesh.getPoints().get(i+2);
        //Three coordinates define a point
        double[] w = {x,y,z};
        //Coordinates of point w are transformed
        //to the new basis
        w = matrixTimesVector(E.getV()
                .transpose().getArray(),w);
        //PCA scaling is applied
        w = matrixTimesVector(scaleMatrix, w);
        //Coordinates of point w are transformed
        //to the old, natural basis
        w = matrixTimesVector(E.getV().getArray(),w);

        //Initial center is restored.
        //Point is added to the  1-dimensional array
        newPoints[i ] = (float) ( w[0] + mean[0]);
        newPoints[i + 1] = (float) ( w[1]  + mean[1] );
        newPoints[i + 2] = (float) ( w[2] + mean[2]);
    }
    //Points are reset
    mesh.getPoints().setAll(newPoints, 0, n);

}


//Returns a random number that can be
//a digit or 1/digit
private float randomFactor()
{
    float s = r.nextInt(10);
    int d = r.nextInt(2);
```

```
    if (d == 0) s = 1.0f/s;
    return s;
}


//Return random values for scale factors.
//Factors can be greater than or less than 1
private void scaleFactors()
{
    scaleFactorW = randomFactor();
    scaleFactorH = randomFactor();
    scaleFactorL = randomFactor();
}


private static Point3D pivot(int i)
{
    double x = (coordBoxes[i][0] + coordBoxes[i][3])/2;
    double y = (coordBoxes[i][1] + coordBoxes[i][4])/2;
    double z = coordBoxes[i][5];
    Point3D p = new Point3D(x,y,z);
    return p;
}

private static Point3D pivot2(int i)
{
    double x = (coordBoxes[i][0] + coordBoxes[i][3])/2;
    double y = (coordBoxes[i][1] + coordBoxes[i][4])/2;
    double z = coordBoxes[i][2];
    Point3D p = new Point3D(x,y,z);
    return p;
}

private static void    defineGroups()
{
    //Pivots of groups
    Point3D finger0Ph2GroupPivot = new Point3D(0,0,0);
    Point3D finger0Ph21GroupPivot  = new Point3D(0,0,0);
```

```
Point3D finger0Ph210GroupPivot = new Point3D(0,0,0);
Point3D finger1Ph2GroupPivot = new Point3D(0,0,0);
Point3D finger1Ph21GroupPivot = new Point3D(0,0,0);
Point3D finger1Ph210GroupPivot = new Point3D(0,0,0);
Point3D finger2Ph2GroupPivot = new Point3D(0,0,0);
Point3D finger2Ph21GroupPivot = new Point3D(0,0,0);
Point3D finger2Ph210GroupPivot = new Point3D(0,0,0);
Point3D finger3Ph2GroupPivot = new Point3D(0,0,0);
Point3D finger3Ph21GroupPivot = new Point3D(0,0,0);
Point3D finger3Ph210GroupPivot = new Point3D(0,0,0);
Point3D thumbPh2GroupPivot = new Point3D(0,0,0);
Point3D thumbPh21GroupPivot = new Point3D(0,0,0);
Point3D thumbPh210GroupPivot = new Point3D(0,0,0);
Point3D wristGroupPivot = new Point3D(0,0,0);



//Natural groups are instantiated

for(int i = 0; i < counterOfBones; i++)
{
    String name = names2[i];
    System.out.println("i = " + i + "  " +name);


    //Finger 0 = little and its subgroups
    if (name.contains("little"))
    {
        if (name.contains("Distal"))
        {
            finger0Ph2Group
                .getChildren().add(bonesViewArray[i]);
            finger0Ph2GroupPivot = pivot(i);

        }
        if (name.contains("Middle"))
        {
```

```
                finger0Ph21Group
                    .getChildren()
                    .addAll(finger0Ph2Group, bonesViewArray[i]);
                finger0Ph21GroupPivot = pivot(i);
            }
            if (name.contains("Proximal"))
            {
                finger0Ph210Group
                    .getChildren()
                    .addAll(finger0Ph21Group, bonesViewArray[i])
                finger0Ph210GroupPivot = pivot(i);
            }
        }

        //Finger 1 = ring and its subgroups
        else if (name.contains("ring"))
        {
            if (name.contains("Distal"))
            {
                finger1Ph2Group
                    .getChildren().add(bonesViewArray[i]);
                finger1Ph2GroupPivot = pivot(i);
            }
            if (name.contains("Middle"))
            {
                finger1Ph21Group
                    .getChildren()
                    .addAll(finger1Ph2Group, bonesViewArray[i]);
                finger1Ph21GroupPivot = pivot(i);
            }
            if (name.contains("Proximal"))
            {
                finger1Ph210Group
                    .getChildren()
                    .addAll(finger1Ph21Group, bonesViewArray[i])
                finger1Ph210GroupPivot = pivot(i);
            }
        }
```

```
//Finger 2 = middle and its subgroups
else if (name.contains("middle"))
{
    if (name.contains("Distal"))
    {
        finger2Ph2Group
        .getChildren().add(bonesViewArray[i]);
        finger2Ph2GroupPivot = pivot(i);
    }
    if (name.contains("Middle"))
    {
        finger2Ph21Group
            .getChildren()
            .addAll(finger2Ph2Group, bonesViewArray[i]);
        finger2Ph21GroupPivot = pivot(i);
    }
    if (name.contains("Proximal"))
    {
        finger2Ph210Group
            .getChildren()
            .addAll(finger2Ph21Group, bonesViewArray[i]);
        finger2Ph210GroupPivot = pivot(i);
    }
}

//Finger 3 = index and its subgroups
else if (name.contains("index"))
{
    if (name.contains("Distal"))
    {
        finger3Ph2Group
            .getChildren().add(bonesViewArray[i]);
        finger3Ph2GroupPivot = pivot(i);
    }
    if (name.contains("Middle"))
    {
        finger3Ph21Group
```

```
                .getChildren()
                .addAll(finger3Ph2Group, bonesViewArray[i]);
            finger3Ph21GroupPivot = pivot(i);
        }
        if (name.contains("Proximal"))
        {
            finger3Ph210Group
                .getChildren()
                .addAll(finger3Ph21Group, bonesViewArray[i])
            finger3Ph210GroupPivot = pivot(i);
        }
    }

    //Thumb and its subgroups
    else if (name.contains("thumb"))
    {
        if (name.contains("Distal"))
        {
            thumbPh2Group
                .getChildren().add(bonesViewArray[i]);
                thumbPh2GroupPivot = pivot(i);
        }
        if (name.contains("Proximal"))
        {
            thumbPh21Group
                .getChildren()
                .addAll(thumbPh2Group, bonesViewArray[i]);
            thumbPh21GroupPivot = pivot(i);
        }
    }

    //Thumb and its subgroups
    else if (name.contains("first metacarpal"))
    {
        thumbPh210Group
            .getChildren()
            .addAll(thumbPh21Group, bonesViewArray[i]);
        //thumbPh210GroupPivot = pivot(i);
```

```
        }

        //Thumb and its subgroups
        else if (name.contains("trapezium"))
        {
           thumbPh210GroupPivot = pivot(i);
           wristGroup.getChildren().add(bonesViewArray[i]);
        }
        else

        {
            wristGroup.getChildren().add(bonesViewArray[i]);
            if (name.contains("lunate"))
                wristGroupPivot = pivot(i);
        }
}


    //Fingers are attached to the palm

        palmGroup.getChildren().addAll(wristGroup
            , finger0Ph210Group
            , finger1Ph210Group
            , finger2Ph210Group
            , finger3Ph210Group
            , thumbPh210Group);

        world3DXform.getChildren().addAll(palmGroup);


    //========OPENING/CL0SING OF THE HAND===========

    NumberBinding closeHandPropertyHalf
            = closeHandProperty.multiply(0.5);

    //The slider that closes the hand extends its
    //control to the palm:
    //it must rotate around
```

```
//the end of the arm that is parallel to the X_AXIS
//This is the pivot of rotation
palmGroup.rwpx.pivotZProperty()
        .setValue(wristGroupPivot.getZ());
palmGroup.rwpx.pivotYProperty()
        .setValue(wristGroupPivot.getY());
//The angle of rotation is controlled by a slider
palmGroup.rwpx.angleProperty()
        .bind(closeHandPropertyHalf);



//Specifications of movement for subgroups of finger 0
finger0Ph2Group.rwpx.pivotZProperty()
        .setValue(finger0Ph2GroupPivot.getZ());
finger0Ph2Group.rwpx.pivotYProperty()
        .setValue(finger0Ph2GroupPivot.getY());
finger0Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger0Ph21Group.rwpx.pivotZProperty()
        .setValue(finger0Ph21GroupPivot.getZ());
finger0Ph21Group.rwpx.pivotYProperty()
        .setValue(finger0Ph21GroupPivot.getY());
finger0Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger0Ph210Group.rwpx.pivotZProperty()
        .setValue(finger0Ph210GroupPivot.getZ());
finger0Ph210Group.rwpx.pivotYProperty()
        .setValue(finger0Ph210GroupPivot.getY());
finger0Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

//Specifications of movement for subgroups of finger 1
finger1Ph2Group.rwpx.pivotZProperty()
        .setValue(finger1Ph2GroupPivot.getZ());
finger1Ph2Group.rwpx.pivotYProperty()
```

```
        .setValue(finger1Ph2GroupPivot.getY());
finger1Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger1Ph21Group.rwpx.pivotZProperty()
        .setValue(finger1Ph21GroupPivot.getZ());
finger1Ph21Group.rwpx.pivotYProperty()
        .setValue(finger1Ph21GroupPivot.getY());
finger1Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger1Ph210Group.rwpx.pivotZProperty()
        .setValue(finger1Ph210GroupPivot.getZ());
finger1Ph210Group.rwpx.pivotYProperty()
        .setValue(finger1Ph210GroupPivot.getY());
finger1Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

//Specifications of movement for subgroups of finger 2
finger2Ph2Group.rwpx.pivotZProperty()
        .setValue(finger2Ph2GroupPivot.getZ());
finger2Ph2Group.rwpx.pivotYProperty()
        .setValue(finger2Ph2GroupPivot.getY());
finger2Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger2Ph21Group.rwpx.pivotZProperty()
        .setValue(finger2Ph21GroupPivot.getZ());
finger2Ph21Group.rwpx.pivotYProperty()
        .setValue(finger2Ph21GroupPivot.getY());
finger2Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger2Ph210Group.rwpx.pivotZProperty()
        .setValue(finger2Ph210GroupPivot.getZ());
finger2Ph210Group.rwpx.pivotYProperty()
        .setValue(finger2Ph210GroupPivot.getY());
finger2Ph210Group.rwpx.angleProperty()
```

```
        .bindBidirectional(closeHandProperty);

//Specifications of movement for subgroups of finger 3
finger3Ph2Group.rwpx.pivotZProperty()
        .setValue(finger3Ph2GroupPivot.getZ());
finger3Ph2Group.rwpx.pivotYProperty()
        .setValue(finger3Ph2GroupPivot.getY());
finger3Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger3Ph21Group.rwpx.pivotZProperty()
        .setValue(finger3Ph21GroupPivot.getZ());
finger3Ph21Group.rwpx.pivotYProperty()
        .setValue(finger3Ph21GroupPivot.getY());
finger3Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger3Ph210Group.rwpx.pivotZProperty()
        .setValue(finger3Ph210GroupPivot.getZ());
finger3Ph210Group.rwpx.pivotYProperty()
        .setValue(finger3Ph210GroupPivot.getY());
finger3Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);



//======OPENING/CLOSING OF FINGERS========

//Fingers that are at the border must
//open twice faster as those
//near the center.
NumberBinding openFingersPropertyTwo
        = openFingersProperty.multiply(2);
//Specifications of movement for subgroups of finger 0
finger0Ph210Group.rwpy.pivotXProperty()
        .setValue(finger0Ph210GroupPivot.getX());
finger0Ph210Group.rwpy.pivotYProperty()
        .setValue(finger0Ph210GroupPivot.getY());
```

```
finger0Ph210Group.rwpy.pivotZProperty()
        .setValue(finger0Ph210GroupPivot.getZ());
finger0Ph210Group.rwpy.angleProperty()
        .bind(openFingersPropertyTwo);


//Specifications of movement for subgroups of finger 1
finger1Ph210Group.rwpy.pivotXProperty()
        .setValue(finger1Ph210GroupPivot.getX());
finger1Ph210Group.rwpy.pivotYProperty()
        .setValue(finger1Ph210GroupPivot.getY());
finger1Ph210Group.rwpy.pivotZProperty()
        .setValue(finger1Ph210GroupPivot.getZ());

finger1Ph210Group.rwpy.angleProperty()
        .bind(openFingersProperty);

//Finger two remains invariant under opening of fingers

//Opening in the contrary direction of fingers 3 and 4
NumberBinding openFingersPropertyMinus
        = openFingersProperty.multiply(-1);

//Specifications of movement for finger 3
finger3Ph210Group.rwpy.pivotXProperty()
        .setValue(finger3Ph210GroupPivot.getX());
finger3Ph210Group.rwpy.pivotYProperty()
        .setValue(finger3Ph210GroupPivot.getY());
finger3Ph210Group.rwpy.pivotZProperty()
        .setValue(finger3Ph210GroupPivot.getZ());
finger3Ph210Group.rwpy.angleProperty()
        .bind(openFingersPropertyMinus);

NumberBinding openFingersPropertyMinusTwo
        = openFingersProperty.multiply(-2);
```

```
//====== The thumb ========
//Specifications of movement for the thumb


NumberBinding rotateThumbPropertyDown
        = rotateThumbProperty.multiply(-0.4);

NumberBinding rotateThumbPropertyUp
        = rotateThumbProperty.multiply(-2);


NumberBinding closeThumbPropertyMinus
        = closeThumbProperty.multiply(-1);


//Traversal movement
thumbPh210Group.rwpy.pivotXProperty()
        .setValue(thumbPh210GroupPivot.getX());
thumbPh210Group.rwpy.pivotZProperty()
        .setValue(thumbPh210GroupPivot.getZ());
thumbPh210Group.rwpy.angleProperty()
        .bind(closeThumbProperty);

/*
//Rotations of phalanges 2 and 1 to make the grip.
//Rotations around X_AXIS and Y_AXIS at the same time
//Phalanx 2
thumbPh2Group.rwpx.pivotXProperty()
        .setValue(thumbPh2GroupPivot.getX());
thumbPh2Group.rwpx.pivotYProperty()
        .setValue(thumbPh2GroupPivot.getY());
thumbPh2Group.rwpx.pivotZProperty()
        .setValue(thumbPh2GroupPivot.getZ());
thumbPh2Group.rwpx.angleProperty()
        .bind(rotateThumbPropertyDown);

//Phalanges 2 and 1
thumbPh21Group.rwpx.pivotXProperty()
```

```
        .setValue(thumbPh21GroupPivot.getX());
thumbPh21Group.rwpx.pivotYProperty()
        .setValue(thumbPh21GroupPivot.getY());
thumbPh21Group.rwpx.pivotZProperty()
        .setValue(thumbPh21GroupPivot.getZ());
thumbPh21Group.rwpx.angleProperty()
        .bind(rotateThumbPropertyDown);

//Phalanx 2
thumbPh2Group.rwpy.pivotXProperty()
        .setValue(thumbPh2GroupPivot.getX());
thumbPh2Group.rwpy.pivotYProperty()
        .setValue(thumbPh2GroupPivot.getY());
thumbPh2Group.rwpy.pivotZProperty()
        .setValue(thumbPh2GroupPivot.getZ());
thumbPh2Group.rwpy.angleProperty()
        .bind(rotateThumbPropertyDown);
//Phalanges 2 and 1
thumbPh21Group.rwpy.pivotXProperty()
        .setValue(thumbPh21GroupPivot.getX());
thumbPh21Group.rwpy.pivotYProperty()
        .setValue(thumbPh21GroupPivot.getY());
thumbPh21Group.rwpy.pivotZProperty()
        .setValue(thumbPh21GroupPivot.getZ());
thumbPh21Group.rwpy.angleProperty()
        .bind(rotateThumbPropertyDown);
*/

//Rotation of the whole thumb
//to get opposite to other fingers
thumbPh210Group.rwpz.pivotXProperty()
        .setValue(thumbPh210GroupPivot.getX());
thumbPh210Group.rwpz.pivotYProperty()
        .setValue(thumbPh210GroupPivot.getY());
thumbPh210Group.rwpz.pivotZProperty()
        .setValue(thumbPh210GroupPivot.getZ());
thumbPh210Group.rwpz.angleProperty()
        .bind(rotateThumbProperty.multiply(-2));
```

```
    //===========Animation=========

    double endTime1 = 10000.0; ////1000 = 1 second
    double endTime2 = 2 * endTime1;
    double endTime3 = 3 * endTime1;
    double endTime4 = 4 * endTime1;
    double endTime5 = 5 * endTime1;
    double endTime6 = 6 * endTime1;
    double endTime7 = 7 * endTime1;
    double endTime8 = 8 * endTime1;
    double endTime9 = 9 * endTime1;




    animation.getKeyFrames().addAll
    (

//The hand gets closed
        //closeHandProperty: initial position at initial time
        new KeyFrame(Duration.ZERO,
            //Permanent condition
            new KeyValue(closeThumbProperty, 0)),
        //closeHandProperty: final position at final time
        new KeyFrame(new Duration(endTime1),
            new KeyValue(closeThumbProperty, -50)),

        //The hand gets closed
        //closeHandProperty: initial position at initial time
        new KeyFrame(new Duration(endTime1),
            //Permanent condition
            new KeyValue(closeHandProperty, 0)),
        //closeHandProperty: final position at final time
        new KeyFrame(new Duration(endTime2),
            new KeyValue(closeHandProperty, -90))
```

```
    ,


//The thumb closes the grip
//rotateThumbProperty: initial position at initial time
new KeyFrame(new Duration(endTime2),
    //Permanent condition
    new KeyValue(rotateThumbProperty, 0)),
//rotateThumbProperty: final position at final time
new KeyFrame(new Duration(endTime3),
    new KeyValue(rotateThumbProperty, 37)),

//The thumb gets reopened
//rotateThumbProperty: initial position at initial time
new KeyFrame(new Duration(endTime3),
    //Permanent condition
    new KeyValue(rotateThumbProperty, 37)),
//rotateThumbProperty: final position at final time
new KeyFrame(new Duration(endTime4),
    new KeyValue(rotateThumbProperty, 0)),

//The hand get reopened
//closeHandProperty: initial position at initial time
new KeyFrame(new Duration(endTime4),
    //Permanent condition
    new KeyValue(closeHandProperty, -90)),
//closeHandProperty: final position at final time
new KeyFrame(new Duration(endTime5),
    new KeyValue(closeHandProperty, 0)),

//Fingers get reopened
//openFingersProperty: initial position at initial time
new KeyFrame(new Duration(endTime5),
//Permanent condition
new KeyValue(openFingersProperty, 0)),
//openFingersProperty: final position at final time
new KeyFrame(new Duration(endTime6),
    new KeyValue(openFingersProperty, 15)),
```

```
        //Fingers get closed
        //openFingersProperty: initial position at initial time
        new KeyFrame(new Duration(endTime6),
        //Permanent condition
        new KeyValue(openFingersProperty, 15)),
        //openFingersProperty: final position at final time
        new KeyFrame(new Duration(endTime7),
            new KeyValue(openFingersProperty, 0)),

        //The thumb gets closed
        //closeThumbProperty: initial position at initial time
        new KeyFrame(new Duration(endTime7),
            //Permanent condition
            new KeyValue(closeThumbProperty, -55)),
        //closeThumbProperty: final position at final time
        new KeyFrame(new Duration(endTime8),
            new KeyValue(closeThumbProperty, 0)),

        //The thumb gets reopen
        //closeThumbProperty: initial position at initial time
        new KeyFrame(new Duration(endTime8),
            //Permanent condition
            new KeyValue(closeThumbProperty, 0)),
        //closeThumbProperty: final position at final time
        new KeyFrame(new Duration(endTime9),
            new KeyValue(closeThumbProperty, -25))

    );



animation.play();

}
```

```
private static void printMatrix(double m[][], String title)
{
    System.out.println("This is the matrix " + title);
    for(int i=0;i<3;i++)
        System.out.println(m[i][0] + " " + m[i][1]+ " "
                                              + m[i][2]);
    System.out.println();
}


//The multiplication of a matrix times a vector
  private static  double[] matrixTimesVector(double m[][],
                                             double v[])
  {
    double[]  helpV = new double[3];
    for(int i=0; i<3; i++)
        helpV[i] =0;
    for(int i=0; i<3; i++)
        for(int j=0; j<3; j++)
            helpV[i]  =helpV[i]  + m[i][j]*v[j];
    return helpV;
  }


//Normalized crux product of vectors v and w
  private static  double dotProduct(double[] v,
                                    double[] w)
  {
    return v[0]*w[0] + v[1]*w[1] + v[2]*w[2];

  }


public void play()
{

}
```

```
@Override public void stop()
{

}

public static void main(String[] args)
{

    launch(args);
}


}//End of main class N44 PCAmutations2
```

45, page 159. The following code shows the effect of small cylindrical mutations over the bones of the hand. There are people whose hands have shorter phalanges for the ring finger, this mutation is cylindrical. But apart from this, the opinion of the Author is that independent cylindrical mutations must be preferentially tiny to be compatible with what he sees in nature. The code follows:

```
/*
    Program N45 Cylindricalmutations
    This program produces
    small cylindrical mutations over the bones of the hand.
    A cylindrical mutation over a cylinder
    may enlarge its radius or its height.
    PCA allows to fit the best cylinder over a bone
    and then it is mutated cylindrically.

    Tested only with some  files,
    particularly  with those of the BodyParts3D project.
    http://lifesciencedb.jp/bp3d/
    Those files are licensed:

    The license for this database is specified in the
    Creative Commons Attribution-Share Alike 2.1 Japan.
```

```
    If you use data from this database, please be sure
    attribute this database as follows: "BodyParts3D, (c)
    The Database Center for Life Science licensed under
    CC Attribution-Share Alike 2.1 Japan".
    http://dbguidehive.biosciencedbc.jp/en/bodyparts3d/lic.html

 */

package Programs;

//Files in the hosting project Programs
import Jama.EigenvalueDecomposition;
import Jama.Matrix;
import Tools.Axes;
import Tools.Cameras;
import Tools.Xform;
import Tools.XformWithPivot;


import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.List;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.binding.NumberBinding;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
```

```
import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.Slider;
import javafx.scene.control.TextArea;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.MeshView;;
import javafx.scene.shape.TriangleMesh;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import javafx.util.Duration;



public class CylindricalMutations extends Application
{
    //Visibility of the mesh
    //might depend on the correct choice
    //of this factor. Move it in within 1 and 500.
    private static final float scaleFactor = 5;

    //Scale factors along width, height and length
    static private float scaleFactorW = 1;
    static private float scaleFactorH = 1;
    static private float scaleFactorL = 1;
```

```
//Printing for debugging
final static boolean printReadFile = false;
private static final String title
        =  "Importer of meshes in format ";

//Tool to read files
BufferedReader br ;
private static  String textOfFile = "";

private static int counterOfFaces = 0;
private static int counterOfVertexes = 0;
private static int counterOfNormals = 0;

private static int[] destFaces ;

private static TriangleMesh[] bonesMeshesArray  = new TriangleMesh[
//Array of Xforms
private static Xform[] bonesXformsArray  = new Xform[100];
private static MeshView[] bonesViewArray  = new MeshView[100];
private static final String[] names1  = new String[100];
private static final String[] names2  = new String[100];
//Group of Xforms
private static Xform bonesXforms  = new Xform();
//Counter
private static int counterOfBones = 0;

//*****************

final static Group root = new Group();

final static Group axisGroup = new Group();


final static Xform cameraXform = new Xform();
final static Xform cameraXform2 = new Xform();
final static Xform cameraXform3 = new Xform();

final static double cameraDistance = 0; //1450;//450
```

```
    final static Xform world3DXform = new Xform();
    final static Xform bonesXform = new Xform();

    //Axes : ctrl + A
    private static final Axes axes = new Axes();
    private static final Cameras camera = new Cameras();

    boolean timelinePlaying = false;
    double ONE_FRAME = 1.0/24.0;
    double DELTA_MULTIPLIER = 200.0;
    double CONTROL_MULTIPLIER = 0.1;
    double SHIFT_MULTIPLIER = 0.1;
    double ALT_MULTIPLIER = 0.5;

    double mousePosX;
    double mousePosY;
    double mouseOldX;
    double mouseOldY;
    double mouseDeltaX;
    double mouseDeltaY;




   //Menubar declaration and initialization
    MenuBar menuBar = new MenuBar();


//This is the outer pane to place tabs
private static BorderPane outerPane = new BorderPane();
private static Pane paneHand = new Pane();

//Slider for zooming
    //private  static Slider zSlider;
    //Position of graphic along the z-axis,
    //normal to the screen.
    //It is a double property because
    //it will get involved into a binding
```

```
//private  static final DoubleProperty zPos
//        = new SimpleDoubleProperty(200);

//Slider for angle of rotation of fingers
private  static Slider closeHandSlider;
//To close the hand,
//each phalange suffers a rotation.
//It is a double property because
//it will get involved into a binding
private  static final DoubleProperty closeHandProperty
        = new SimpleDoubleProperty(0);

//Slider for opening fingers thanks to
//their transversal rotation
private  static Slider openFingersSlider;
//To open the fingers,
//each phalange suffers a transversal rotation.
//It is a double property because
//it will get involved into a binding
private  static final DoubleProperty openFingersProperty
        = new SimpleDoubleProperty(0);

//Slider for opening/closing the thumb
private  static Slider closeThumbSlider;
//To close the thumb we need a rotation.
//It is a double property because
//it will get involved into a binding
private  static final DoubleProperty closeThumbProperty
        = new SimpleDoubleProperty(0);

//Slider for rotating the thumb to form the grip
private  static Slider rotateThumbSlider;
//To form the grip we need a rotation of the thumb.
//It is a double property because
//it will get involved into a binding
private  static final DoubleProperty rotateThumbProperty
        = new SimpleDoubleProperty(0);
```

```
//Nested definition of   nodes is implemented
//to have, say,  the palm dragging the fingers,
//and a phalanx dragging the rest of its the finger.


//Each finger generates various natural groups:
//Group with the extreme phalanx of finger0
//Symbolic initialization
private static final XformWithPivot finger0Ph2Group
        = new XformWithPivot();
//Group with the two extreme phalanges
private static final XformWithPivot finger0Ph21Group
        = new XformWithPivot();
//Group with all three phalanges. It is the finger
private static final XformWithPivot finger0Ph210Group
        = new XformWithPivot();
//Finger1
private static final XformWithPivot finger1Ph2Group
        = new XformWithPivot();
private static final XformWithPivot finger1Ph21Group
        = new XformWithPivot();
private static final XformWithPivot finger1Ph210Group
        = new XformWithPivot();
//Finger2
private static final XformWithPivot finger2Ph2Group
        = new XformWithPivot();
private static final XformWithPivot finger2Ph21Group
        = new XformWithPivot();
private static final XformWithPivot finger2Ph210Group
        = new XformWithPivot();
//Finger3
private static final XformWithPivot finger3Ph2Group
        = new XformWithPivot();
private static final XformWithPivot finger3Ph21Group
        = new XformWithPivot();
private static final XformWithPivot finger3Ph210Group
        = new XformWithPivot();
```

```
    //Finger4
    private static final XformWithPivot finger4Ph2Group
            = new XformWithPivot();
    private static final XformWithPivot finger4Ph21Group
            = new XformWithPivot();
    private static final XformWithPivot finger4Ph210Group
            = new XformWithPivot();
    //Thumb
    private static final XformWithPivot thumbPh2Group
            = new XformWithPivot();
    private static final XformWithPivot thumbPh21Group
            = new XformWithPivot();
    private static final XformWithPivot thumbPh210Group
            = new XformWithPivot();

     private static final XformWithPivot wristGroup
            = new XformWithPivot();
    //Group with wrist and fingers
    private static final XformWithPivot palmGroup
            = new XformWithPivot();

    private static final Timeline animation  = new Timeline();

    private static double[][] coordBoxes = new double[100][6];



    Random r = new Random();

@Override
public void start(Stage primaryStage)
{
    System.out.println("start");

    Scene scene = new Scene(root, 1200, 1000, true);

    paneHand.setPrefSize(500, 500);
```

```
outerPane.setOpacity(0);
//outerPane.setLayoutX(700);
//outerPane.setLayoutY(700);

/*
   Multiple file loader
*/

//Content of files will be shown in textArea
final TextArea textArea = new TextArea();
textArea.prefWidth(400);
textArea.setWrapText(true);
textArea.setText("");
//Container  for textArea.
ScrollPane scrollPane = new ScrollPane();
scrollPane.getStyleClass().add("noborder-scroll-pane");
scrollPane.setContent(textArea);
scrollPane.setFitToWidth(true);
scrollPane.setPrefWidth(400);
scrollPane.setPrefHeight(180);
VBox vBox = new VBox();
vBox.visibleProperty().setValue(false);
vBox.getChildren().addAll(scrollPane);
root.getChildren().addAll(vBox);


// --- Menu files
Menu filesMenu = new Menu("Files");
//Menu to load multiple files
MenuItem loadFiles
        = new MenuItem("Load + process multiple files ");
//MenuItem added to menu
filesMenu.getItems().addAll(loadFiles);
menuBar.getMenus().add(filesMenu);
root.getChildren().add(menuBar);

//Listener
loadFiles.setOnAction((ActionEvent t) ->
```

```
{
    FileChooser fileChooser = new FileChooser();

    //Name of file with extension "
    FileChooser.ExtensionFilter extFilter
            = new FileChooser
                    .ExtensionFilter("TXT files (*.obj)"
                            , "*.obj");
    fileChooser.getExtensionFilters().add(extFilter);

    //Files will be kept in a list
    List<File> files = new FileChooser()
            .showOpenMultipleDialog(primaryStage);



        //Delete possible repetitions
        //files = deleteRepetitions(files);
    //counterOfBones = 0;



    for(int j = 0; j < files.size(); j++)
    {
        File file = files.get(j);
        String name = file.getName();
        names1[j] = name;
    }
    counterOfBones = 0;
    //Each file is processed
    for(int j = 0; j < files.size(); j++)
    {

        String name = names1[j];
        boolean repeated = false;
        for (int i = 0; i <  j; i++)
        {
            if (name.equals(names1[i]))
```

```
                {
                    repeated = true;

                    //System.out.println("Repeated" + name);
                }
            }

            if (repeated);
            else
            try
            {
                File file = files.get(j);
                //Text of file is read and mesh is extracted
                textOfFile = readFile(file);
                //Text copied to textArea
                textArea.appendText(textOfFile);

            } catch (IOException ex)
                {
                    Logger.getLogger(AnimatedHandSkeleton
                            .class.getName())
                            .log(Level.SEVERE, null, ex);
                }



        }

        defineGroups();



    //To remedy a not understood bug
    //( = menuBar is added to the graphic )
    root.getChildren().remove(menuBar);

    vBox.visibleProperty().set(true);
```

```
});

//====== End of File loader

primaryStage.setTitle(title
        + ". Drag the mouse to rotate. "
    + " Ctrl + A = Axis visibility"
        );
scene.setFill(Color.BLUE);
primaryStage.setScene(scene);
primaryStage.show();
axes.buildAxes();
axes.handleKeyboard(scene);
//camera.handleMouse(scene);
camera.handleMouse(paneHand);

//Hide textArea and show graphic
Button hideTextArea = new Button();
hideTextArea.setText("View graphic");
//Listener
hideTextArea.setOnAction((ActionEvent t) ->
{
    vBox.visibleProperty().set(false);
    camera.buildCamera();


    world3DXform.getChildren().add(bonesXform);


    //world3DXform.setRx(180);
    //world3DXform.setRy(90);
    world3DXform.setScale(scaleFactor);
    world3DXform.setTranslate(-3500, -1400, 0);

    paneHand.getChildren().add(world3DXform);
    root.getChildren().add(paneHand);
    scene.setCamera(camera.camera);
});
```

```
vBox.getChildren().addAll(hideTextArea);



//This slider   controls the closing of the hand.
closeHandSlider = new Slider();
closeHandSlider.setMax(90);
closeHandSlider.setMin(0);
closeHandSlider.setValue(10);

//To close the hand, an angle of rotation is enabled.
//Bind angleProperty to the reading of the slider
closeHandSlider.valueProperty()
        .bindBidirectional(closeHandProperty);


//This slider controls the openning of fingers.
openFingersSlider = new Slider();
openFingersSlider.setMax(15);
openFingersSlider.setMin(0);
openFingersSlider.setValue(10);

//To open fingers, an angle of rotation is enabled.
//Bind openFingersProperty to the reading of the slider
openFingersSlider.valueProperty()
        .bindBidirectional(openFingersProperty);


//Slider to open  the thumb
closeThumbSlider = new Slider();
closeThumbSlider.setMax(55);
closeThumbSlider.setMin(0);
closeThumbSlider.setValue(-0);
closeThumbSlider.adjustValue(closeThumbSlider.getValue());

//To open the thumb, an angle of rotation is enabled.
//Bind closeThumbProperty to the reading of the slider
```

```
closeThumbSlider.valueProperty().bindBidirectional(closeThumbProper


//Slider to rotate  the thumb
rotateThumbSlider = new Slider();
rotateThumbSlider.setMax(50);
rotateThumbSlider.setMin(0);
rotateThumbSlider.setValue(0);
rotateThumbSlider.adjustValue(rotateThumbSlider.getValue());

//To rotate the thumb, an angle of rotation is enabled.
//Bind closeThumbProperty to the reading of the slider
rotateThumbSlider.valueProperty()
        .bindBidirectional(rotateThumbProperty);



//Other JavaFX controls, labels and containers
GridPane gridPane = new GridPane();
gridPane.setVgap(4);
gridPane.setHgap(10);
gridPane.setPadding(new Insets(5, 5, 5, 5));
//Column 0, Row   0
gridPane.add(new Label("Zoom: "), 0, 0);
//gridPane.add(zSlider, 1, 0);
gridPane.add(new Label("Close hand: "), 2, 0);
gridPane.add(closeHandSlider, 3, 0);
gridPane.add(new Label("Open fingers: "), 0, 1);
gridPane.add(openFingersSlider, 1, 1);
gridPane.add(new Label("Close thumb: "), 0, 2);
gridPane.add(closeThumbSlider, 1, 2);
gridPane.add(new Label("Rotate thumb: "), 2, 2);
gridPane.add(rotateThumbSlider, 3,2);



SubScene subScene
        = new SubScene(gridPane, 880, 200);
```

```
    subScene.setLayoutX(10);
    subScene.setLayoutY(800);
    paneHand.getChildren().add(subScene);









    primaryStage.show();
}




//File is read and its content is returned as a string.
//The associated mesh is recovered
//and correct orientation of faces is provided.
//The mesh is a global variable.
private  String readFile(File file)
        throws FileNotFoundException, IOException
{
    //Name of file is printed
    String name = file.getName();
    //if (printReadFile)
        System.out.println(name);
    names2[counterOfBones] = name;
    //Buffer to keep the content of file
```

```
    //StringBuilder stringBuffer = new StringBuilder();
    StringBuilder stringBuffer  = new StringBuilder();
    //Tool to read file
    BufferedReader bufferedReader = null;

    //A mesh is declared and instantiated
    TriangleMesh mesh  = new TriangleMesh();
    //The mesh has no predefined texture
    mesh.getTexCoords().addAll(0,0);
    //In the " format, counting begins with 1
    //but in Java, we begin with 0.
    //So, an additional inocuos point is added.
    mesh.getPoints().addAll(0,0,0);

    if (printReadFile)
        System.out.println("File content: ");
//File is read
try
{
//
bufferedReader = new BufferedReader(new FileReader(file));

String text;




while( (text = bufferedReader.readLine()) != null)
{
    //Text for the textArea
    stringBuffer.append(text);
    //Print each line
    //System.out.println(text);

    //Vertexes are read
    if (((text.length() > 0 ) && text.charAt(0) == 'v')
            && text.charAt(1) == ' ')
    {
```

```
        if (printReadFile)
            System.out.println("Vertex  = "
                    + counterOfVertexes + " = "+text);
        counterOfVertexes++;
        text = text.substring(2);
        int j = 0;
        //Three vertexes are read
        for(int i = 0; i < 3; i++)
        {
            double d ;
            String s = "";
            Boolean go = true;

            while (go)
            {
                char thisChar = text.charAt(j);
                s = s + thisChar;
                char nextChar = ' ';
                //++j means add one to j
                //and then evaluate expression with <
                if (++j < text.length())
                 nextChar = text.charAt(j);
                if ((nextChar == ' ') ||
                        (j >= text.length()))
                        go = false;

            }
            //string to double
            d = Double.parseDouble(s);
            float df = (float) d;
            if (printReadFile)
                System.out.println("Float number = " + df);
            mesh.getPoints().addAll(df);

        }
    }
```

```
//Normal vectors are counted and displayed
if ((text.length() > 0 ) && (text.charAt(0) == 'v')
        && (text.charAt(1) == 'n'))
{

    if (printReadFile)
        System.out.println("Normal " + counterOfNormals
            + " = " + text);
    counterOfNormals++;
}


//Face is read, one face per line.
//A face has 3 or more vertexes. These are extracted.
//Additional information is ignored
if ( (text.length() > 0 ) && (text.charAt(0) == 'f'))
{

    if (printReadFile)
        System.out.println("Face  = " + counterOfFaces
            + " = "+text);
    counterOfFaces++;
    text = text.substring(2);

    int vertex[] = new int[4];
    int j = 0;

    Boolean isNotBlank ;
    //Three vertexes are extracted
    for(int i = 0; i < 3; i++)
    {

        double d ;
        String s = "";
        Boolean go1 = true;


        while (go1)
```

```
{
    if (j >= text.length()-2)
        break;

    char thisChar = text.charAt(j);
    s = s + thisChar;
    /*
    System.out.println(" j = "
            + j + "    s = " + s);
            */
    char nextChar = ' ';
    j++;
    if (j < text.length())
     nextChar = text.charAt(j);
    char nextNextChar = ' ';
    isNotBlank = true;
    if (nextChar == '/' )
    {
        go1 = false;
        //go until the following blank
        while ((isNotBlank)
                && (j < text.length()-1))
        {
            j++;
            nextNextChar = text.charAt(j);
            if (nextNextChar == ' ' )
                isNotBlank = false;

        }
    }

}

d = Double.parseDouble(s);
int v = (int) d;
vertex[i] = v;
if (printReadFile)
    System.out.println("Int number = " + v);
```

```
}//End of cycle for 3 vertexes

//A face with 3 vertexes is added to mesh
mesh.getFaces().addAll(vertex[0], 0, vertex[1], 0,
    vertex[2], 0);

if (printReadFile)
{
    for(int i = 0; i < 3; i++)
     System.out.print(vertex[i] + " . ");
    System.out.println();
}

//Is there a four vertex in the face?
if   ( (text.length() - j )> 3)
{
    String s = "";
    Boolean go1 = true;
    while (go1)
    {


        char thisChar = text.charAt(j);
        s = s + thisChar;
        j++;
        char nextChar = ' ';
        if (j < text.length())
         nextChar = text.charAt(j);
        if (nextChar == '/' ) go1 = false;
    }
    if (printReadFile)
        System.out.println("4th = " + s);
    double d = Double.parseDouble(s);
    int v = (int) d;
    vertex[3] = v;

    //A polyface with 4 vertex, say, a,b,c,d,
    //can be covered by 2 triangles:
```

```
            //Triangle 1: a,b,c (Already added to mesh)
            //and triangle 2: a,c,d.
            //Triangle 2 with 3 vertexes is added to mesh
            mesh.getFaces().addAll(vertex[0], 0, vertex[2],
                    0, vertex[3], 0);
        }
    }//End of face cycle

    //Bounds of containing box are read and displayed
    if ( (text.length() > 0 )
            && (text.contains("# Bounds(mm):")))
    {
        //System.out.println(text);

        String a = "# Bounds(mm): (";
        int m = a.length();
        text = text.substring(m);
        text = text.replace(")-(", ",");
        text = text.replace(")", ",");
        System.out.println(text);

        int j = 0;


        //Six vertexes are extracted
        for(int i = 0; i < 6; i++)
        {

            double d ;
            String s = "";
            Boolean go1 = true;


            while (go1)
            {
                if (j >= text.length()-2)
                    break;
```

```
                char thisChar = text.charAt(j);
                s = s + thisChar;
                /*
                System.out.println(" j = "
                        + j + "    s = " + s);
                */
                char nextChar = ' ';
                j++;
                if (j < text.length())
                 nextChar = text.charAt(j);
                if (nextChar == ',' )
                {
                    go1 = false;
                   if ( j < text.length() ) j++;
                }

            }

            d = Double.parseDouble(s);

            coordBoxes[counterOfBones][i] = d;
           if (printReadFile)
                System.out.println("Coordinate = " + i + "= " + d);
        }//End of cycle for 3 vertexes
    }

}

} //End of  try.
    //Exception management
    catch (FileNotFoundException ex)
    {
        Logger.getLogger(AnimatedHandSkeleton
        .class.getName()).log(Level.SEVERE, null, ex);
    }
    catch (IOException ex)
    {
        Logger.getLogger(AnimatedHandSkeleton
```

```
        .class.getName()).log(Level.SEVERE, null, ex);
    }
    finally
    {
        try
        {
            bufferedReader.close();
        } catch (IOException ex)
        {
            Logger.getLogger(AnimatedHandSkeleton
            .class.getName()).log(Level.SEVERE, null, ex);
        }
    }



    if (printReadFile)
    {
        System.out.println("End of file content. ");
        //Other indicators are reported
        report(mesh);
    }


    //======MeshWork=======

    PCA(mesh);

    MeshView boneMesh =
            new MeshView(mesh);

    PhongMaterial goldenMaterial
            = new PhongMaterial(Color.GOLDENROD);
    boneMesh.setMaterial(goldenMaterial);
    Group boneGroup = new Group(boneMesh);
    bonesViewArray[counterOfBones] = boneMesh;

    Xform boneXform = new Xform();
```

```
    boneXform.getChildren().add(boneGroup);
    bonesXformsArray[counterOfBones] = boneXform;
    bonesXform.getChildren().add(boneXform);
    counterOfBones++;

    return stringBuffer.toString();
}


//Indicators of retrieved data are printed
private void report(TriangleMesh mesh)
{
    float[] destPoints = new float[mesh.getPoints().size()];
    mesh.getPoints().toArray(destPoints);
    System.out.println("POINTS");
    for(int j = 0; j <  destPoints.length/3; j++)
    {
        System.out.print(j + "   " );
        for(int k = 0; k < 3; k++)
        {
            System.out.print( destPoints[j*3 + k] + "   " );
        }
        System.out.println();
    }

    destFaces = new int[mesh.getFaces().size()];
    //Faces go to a matrix
    mesh.getFaces().toArray(destFaces);
    System.out.println("FACES WITH TEXTURE COORDINATES");
    for(int j = 0; j <  destFaces.length/6; j++)
    {
        System.out.print(j + "   " );
        for(int k = 0; k < 6; k++)
        {
            System.out.print( destFaces[j*6 + k] + "   " );
        }
        System.out.println();
    }
```

```
    System.out.println("counterOfPoints = "
            + counterOfVertexes );
    System.out.println("counterOfFaces = "
            + counterOfFaces );
    System.out.println("counterOfNormals = "
            + counterOfNormals);
    System.out.println("mesh.getPoints().size() = "
            + mesh.getPoints().size() );
    System.out.println("mesh.getFaces().size() = "
            + mesh.getFaces().size() );

    System.out.println("getPointElementSize = "
            + mesh.getPointElementSize() );
    System.out.println("getfaceElementSize = "
            + mesh.getFaceElementSize() );

}



public static String fixedWidthDoubletoString
        (double x, int w, int d) {
   java.text.DecimalFormat fmt
           = new java.text.DecimalFormat();
   fmt.setMaximumFractionDigits(d);
   fmt.setMinimumFractionDigits(d);
   fmt.setGroupingUsed(false);
   String s = fmt.format(x);
   while (s.length() < w) {
      s = " " + s;
   }
   return s;
}

//Principal component analysis of a mesh
private void PCA(TriangleMesh mesh)
{
```

```
//Point are declared as a matrix of coordinates
int n = mesh.getPoints().size();
    float nPoints = n;
nPoints = nPoints/3;
float[][] CoordinatesFloat = new float[(int) nPoints][3];


/*
//Test: adapted from
//Stat Trek: Variance-Covariance Matrix
//http://stattrek.com/matrix-algebra/covariance-matrix.aspx

float[] data = {90,  60,  90,
                90,  90,  30,
                60,  60,  60,
                60,  60,  90,
                30,  30,  30 };


    n = 15;
    nPoints = 5;
int counter = 0;
for(int i = 0; i < n; i = i+3)
{
    //Reading
    float x =  data[i];
    float y =  data[i+1] ;
    float z =  data[i+2] ;
    CoordinatesFloat[counter][0] = x;
    CoordinatesFloat[counter][1] = y;
    CoordinatesFloat[counter][2] = z;
    counter++;
}

//Mean is found
float[] mean = new float[3];

for(int i = 0; i < 3; i++)
```

```
{
    mean[i] = 0;
    for (int j = 0; j <  nPoints; j++)
        mean[i] = mean[i] + CoordinatesFloat[j][i];
    mean[i] = mean[i]/(nPoints);
}

System.out.println("meanX = " + mean[0]);
System.out.println("meanY = " + mean[1]);
System.out.println("meanZ = " + mean[2]);



//Calculate covariance matrix
double[][] Cov = new double[3][3];
for(int i = 0; i < 3; i++)
{
    for (int j = 0; j <  3; j++)
    {
        Cov[i][j] = 0;
        for (int k = 0; k <  nPoints; k++)
            Cov[i][j] = Cov[i][j] +
                    (CoordinatesFloat[k][i] - mean[i])
                    * (CoordinatesFloat[k][j] - mean[j]);

        Cov[i][j] = Cov[i][j]/nPoints;

    }

}

printMatrix(Cov, "Cov");

//Answer: cov matrix must be:
//504  360  180
//360  360  0
//180  0  720
```

```
*/

//=============

int counter = 0;
for(int i = 0; i < n; i = i+3)
{
    //Reading
    float x =  mesh.getPoints().get(i);
    float y =  mesh.getPoints().get(i+1);
    float z =  mesh.getPoints().get(i+2);
    CoordinatesFloat[counter][0] = x;
    CoordinatesFloat[counter][1] = y;
    CoordinatesFloat[counter][2] = z;
    counter++;
    //Warning: the zeroth point is artificial,
    //filled with zeros.
    if (i == 0)
        System.out.println("Zeroth point: x = " + x + "   y =   "
                + y + "   z =  "  + z);
}

//Mean is found. It is vector with 3 coordinates
float[] mean = new float[3];

for(int i = 0; i < 3; i++)
{
    mean[i] = 0;
    //Zeroth point does not count
    for (int j = 1; j <  nPoints; j++)
        mean[i] = mean[i] + CoordinatesFloat[j][i];
    mean[i] = mean[i]/(nPoints -1);
}

System.out.println("meanX = " + mean[0]);
System.out.println("meanY = " + mean[1]);
System.out.println("meanZ = " + mean[2]);
```

```
//Calculate covariance matrix
double[][] Cov = new double[3][3];
for(int i = 0; i < 3; i++)
{
    for (int j = 0; j <  3; j++)
    {
        Cov[i][j] = 0;
        for (int k = 1; k <  nPoints; k++)
            Cov[i][j] = Cov[i][j] +
                    (CoordinatesFloat[k][i] - mean[i])
                    * (CoordinatesFloat[k][j] - mean[j]);

        Cov[i][j] = Cov[i][j]/(nPoints-1);

    }

}

printMatrix(Cov, "Cov");

//=========

//Eigenvalues of Cov matrix are found
Matrix M = new Matrix(Cov);
EigenvalueDecomposition E =
     new EigenvalueDecomposition(M);
double[] d = E.getRealEigenvalues();

for(int i = 0; i < 3; i++)
    System.out.println("Eigenvalue " + i + " : " + d[i]);
double[][] EV = E.getV().getArray();
for(int i = 0; i < 3; i++)
{
    System.out.println("Eigenvector " + i + " : " );
    for (int j = 0; j <  3; j++)
```

```
    {
        System.out.println(EV[i][j]);
    }
    System.out.println();
}

//Test: eigenvectors of a real symmetric matrix
//are orthogonal one to another: dot product must be zero
System.out.println("Orthogonality test : matrix I is expected");
double[][] Ort = new double[3][3];
for(int i = 0; i < 3; i++)
{

    for (int j = 0; j <  3; j++)
    {

        //sum = sum + EV[i][j]*EV[i+1][j];
        Ort[i][j] = dotProduct(EV[i], EV[j]);

    }
    //System.out.println(sum);

}

printMatrix(Ort, "Ort test");

//Determinant must be 1.
System.out.println("Orientation test: "
        + "+1 is expected. Det E = " + E.getV().det() + " \n");


 //Mesh is centered
float [] pointsFloat = new float[n];
for(int i = 0; i < n; i = i+3)
{
    //Reading
    float x =  mesh.getPoints().get(i);
    float y =  mesh.getPoints().get(i+1);
```

```
        float z =  mesh.getPoints().get(i+2);

        //Mesh is centered
        pointsFloat[i]    =  (x - mean[0]);
        pointsFloat[i+1]  =  (y - mean[1]);
        pointsFloat[i+2]  =  (z - mean[2]);
    }
    //Resetting of points of mesh
    mesh.getPoints().setAll(pointsFloat, 0, n);

    axes.axesGroup.setVisible(true);

    //Scale factors are composed
    cylindricalScaleFactors();


    /*
    //Test: slight magnification
    scaleFactorW = 1.1f;
    scaleFactorH = 1.1f;
    scaleFactorL = 1.1f;
    */

    //Scale factors as a diagonal matrix
    double[][] scaleMatrix = {{scaleFactorW, 0,0},
                              {0, scaleFactorH, 0},
                              {0,0, scaleFactorL}};



    //1D array for recording
    float[]newPoints = new float[n];
    //Mesh is aligned with the new basis
    //given by the PCA eigenvetors
    for(int i = 0; i < n; i = i+3)
    {
         //Reading
        float x =  mesh.getPoints().get(i);
```

```
            float y =  mesh.getPoints().get(i+1);
            float z =  mesh.getPoints().get(i+2);
            //Three coordinates define a point
            double[] w = {x,y,z};
            //Coordinates of point w are transformed
            //to the new basis
            w = matrixTimesVector(E.getV()
                    .transpose().getArray(),w);
            //PCA scaling is applied
            w = matrixTimesVector(scaleMatrix, w);
            //Coordinates of point w are transformed
            //to the old, natural basis
            w = matrixTimesVector(E.getV().getArray(),w);

            //Initial center is restored.
            //Point is added to the  1-dimensional array
            newPoints[i ] = (float) ( w[0] + mean[0]);
            newPoints[i + 1] = (float) ( w[1]  + mean[1] );
            newPoints[i + 2] = (float) ( w[2] + mean[2]);
        }
        //Points are reset
        mesh.getPoints().setAll(newPoints, 0, n);

}


//Returns a random float number
//uniformly distributed in the interval
//(1-epsilon, 1 + epsilon)
private float randomFactor(float epsilon)
{
    float s = 2 * epsilon * r.nextFloat();
    s = 1- epsilon + s;
    return s;
}


//Return random values for scale factors.
```

```
//Factors can be greater than or less than 1
private void cylindricalScaleFactors()
{
    float epsilon = 0.5f;
    scaleFactorW = randomFactor(epsilon);
    scaleFactorH =  scaleFactorW;
    scaleFactorL = randomFactor(epsilon);
}


//Returns th pivot of bone i:
//the middle x-value, the middle y-value and
//the minimal z-value
private static Point3D pivot(int i)
{
    double x = (coordBoxes[i][0] + coordBoxes[i][3])/2;
    double y = (coordBoxes[i][1] + coordBoxes[i][4])/2;
    double z = coordBoxes[i][5];
    Point3D p = new Point3D(x,y,z);
    return p;
}

private static Point3D pivot2(int i)
{
    double x = (coordBoxes[i][0] + coordBoxes[i][3])/2;
    double y = (coordBoxes[i][1] + coordBoxes[i][4])/2;
    double z = coordBoxes[i][2];
    Point3D p = new Point3D(x,y,z);
    return p;
}

private static void    defineGroups()
{
    //Pivots of groups
    Point3D finger0Ph2GroupPivot = new Point3D(0,0,0);
    Point3D finger0Ph21GroupPivot  = new Point3D(0,0,0);
    Point3D finger0Ph210GroupPivot = new Point3D(0,0,0);
    Point3D finger1Ph2GroupPivot = new Point3D(0,0,0);
```

```
Point3D finger1Ph21GroupPivot = new Point3D(0,0,0);
Point3D finger1Ph210GroupPivot = new Point3D(0,0,0);
Point3D finger2Ph2GroupPivot = new Point3D(0,0,0);
Point3D finger2Ph21GroupPivot = new Point3D(0,0,0);
Point3D finger2Ph210GroupPivot = new Point3D(0,0,0);
Point3D finger3Ph2GroupPivot = new Point3D(0,0,0);
Point3D finger3Ph21GroupPivot = new Point3D(0,0,0);
Point3D finger3Ph210GroupPivot = new Point3D(0,0,0);
Point3D thumbPh2GroupPivot = new Point3D(0,0,0);
Point3D thumbPh21GroupPivot = new Point3D(0,0,0);
Point3D thumbPh210GroupPivot = new Point3D(0,0,0);
Point3D wristGroupPivot = new Point3D(0,0,0);



//Natural groups are instantiated

for(int i = 0; i < counterOfBones; i++)
{
    String name = names2[i];
    System.out.println("i = " + i + "  " +name);


    //Finger 0 = little and its subgroups
    if (name.contains("little"))
    {
        if (name.contains("Distal"))
        {
            finger0Ph2Group
                .getChildren().add(bonesViewArray[i]);
            finger0Ph2GroupPivot = pivot(i);

        }
        if (name.contains("Middle"))
        {
            finger0Ph21Group
                .getChildren()
```

```
                .addAll(finger0Ph2Group, bonesViewArray[i]);
            finger0Ph21GroupPivot = pivot(i);
        }
        if (name.contains("Proximal"))
        {
            finger0Ph210Group
                .getChildren()
                .addAll(finger0Ph21Group, bonesViewArray[i])
            finger0Ph210GroupPivot = pivot(i);
        }
    }

    //Finger 1 = ring and its subgroups
    else if (name.contains("ring"))
    {
        if (name.contains("Distal"))
        {
            finger1Ph2Group
                .getChildren().add(bonesViewArray[i]);
            finger1Ph2GroupPivot = pivot(i);
        }
        if (name.contains("Middle"))
        {
            finger1Ph21Group
                .getChildren()
                .addAll(finger1Ph2Group, bonesViewArray[i]);
            finger1Ph21GroupPivot = pivot(i);
        }
        if (name.contains("Proximal"))
        {
            finger1Ph210Group
                .getChildren()
                .addAll(finger1Ph21Group, bonesViewArray[i])
            finger1Ph210GroupPivot = pivot(i);
        }
    }

    //Finger 2 = middle and its subgroups
```

```
else if (name.contains("middle"))
{
    if (name.contains("Distal"))
    {
        finger2Ph2Group
        .getChildren().add(bonesViewArray[i]);
        finger2Ph2GroupPivot = pivot(i);
    }
    if (name.contains("Middle"))
    {
        finger2Ph21Group
            .getChildren()
            .addAll(finger2Ph2Group, bonesViewArray[i]);
        finger2Ph21GroupPivot = pivot(i);
    }
    if (name.contains("Proximal"))
    {
        finger2Ph210Group
            .getChildren()
            .addAll(finger2Ph21Group, bonesViewArray[i]);
        finger2Ph210GroupPivot = pivot(i);
    }
}

//Finger 3 = index and its subgroups
else if (name.contains("index"))
{
    if (name.contains("Distal"))
    {
        finger3Ph2Group
            .getChildren().add(bonesViewArray[i]);
        finger3Ph2GroupPivot = pivot(i);
    }
    if (name.contains("Middle"))
    {
        finger3Ph21Group
            .getChildren()
            .addAll(finger3Ph2Group, bonesViewArray[i]);
```

```
            finger3Ph21GroupPivot = pivot(i);
        }
        if (name.contains("Proximal"))
        {
            finger3Ph210Group
                .getChildren()
                .addAll(finger3Ph21Group, bonesViewArray[i])
            finger3Ph210GroupPivot = pivot(i);
        }
    }

    //Thumb and its subgroups
    else if (name.contains("thumb"))
    {
        if (name.contains("Distal"))
        {
            thumbPh2Group
                .getChildren().add(bonesViewArray[i]);
                thumbPh2GroupPivot = pivot(i);
        }
        if (name.contains("Proximal"))
        {
            thumbPh21Group
                .getChildren()
                .addAll(thumbPh2Group, bonesViewArray[i]);
            thumbPh21GroupPivot = pivot(i);
        }
    }

    //Thumb and its subgroups
    else if (name.contains("first metacarpal"))
    {
        thumbPh210Group
            .getChildren()
            .addAll(thumbPh21Group, bonesViewArray[i]);
        //thumbPh210GroupPivot = pivot(i);
    }
```

```
        //Thumb and its subgroups
        else if (name.contains("trapezium"))
        {
           thumbPh210GroupPivot = pivot(i);
           wristGroup.getChildren().add(bonesViewArray[i]);
        }
        else

        {
            wristGroup.getChildren().add(bonesViewArray[i]);
            if (name.contains("lunate"))
                wristGroupPivot = pivot(i);
        }
}


    //Fingers are attached to the palm

        palmGroup.getChildren().addAll(wristGroup
            , finger0Ph210Group
            , finger1Ph210Group
            , finger2Ph210Group
            , finger3Ph210Group
            , thumbPh210Group);

        world3DXform.getChildren().addAll(palmGroup);


    //========OPENING/CL0SING OF THE HAND============

    NumberBinding closeHandPropertyHalf
            = closeHandProperty.multiply(0.5);

    //The slider that closes the hand extends its
    //control to the palm:
    //it must rotate around
    //the end of the arm that is parallel to the X_AXIS
    //This is the pivot of rotation
```

```
palmGroup.rwpx.pivotZProperty()
        .setValue(wristGroupPivot.getZ());
palmGroup.rwpx.pivotYProperty()
        .setValue(wristGroupPivot.getY());
//The angle of rotation is controlled by a slider
palmGroup.rwpx.angleProperty()
        .bind(closeHandPropertyHalf);



//Specifications of movement for subgroups of finger 0
finger0Ph2Group.rwpx.pivotZProperty()
        .setValue(finger0Ph2GroupPivot.getZ());
finger0Ph2Group.rwpx.pivotYProperty()
        .setValue(finger0Ph2GroupPivot.getY());
finger0Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger0Ph21Group.rwpx.pivotZProperty()
        .setValue(finger0Ph21GroupPivot.getZ());
finger0Ph21Group.rwpx.pivotYProperty()
        .setValue(finger0Ph21GroupPivot.getY());
finger0Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger0Ph210Group.rwpx.pivotZProperty()
        .setValue(finger0Ph210GroupPivot.getZ());
finger0Ph210Group.rwpx.pivotYProperty()
        .setValue(finger0Ph210GroupPivot.getY());
finger0Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

//Specifications of movement for subgroups of finger 1
finger1Ph2Group.rwpx.pivotZProperty()
        .setValue(finger1Ph2GroupPivot.getZ());
finger1Ph2Group.rwpx.pivotYProperty()
        .setValue(finger1Ph2GroupPivot.getY());
finger1Ph2Group.rwpx.angleProperty()
```

```
        .bindBidirectional(closeHandProperty);

finger1Ph21Group.rwpx.pivotZProperty()
        .setValue(finger1Ph21GroupPivot.getZ());
finger1Ph21Group.rwpx.pivotYProperty()
        .setValue(finger1Ph21GroupPivot.getY());
finger1Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger1Ph210Group.rwpx.pivotZProperty()
        .setValue(finger1Ph210GroupPivot.getZ());
finger1Ph210Group.rwpx.pivotYProperty()
        .setValue(finger1Ph210GroupPivot.getY());
finger1Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

//Specifications of movement for subgroups of finger 2
finger2Ph2Group.rwpx.pivotZProperty()
        .setValue(finger2Ph2GroupPivot.getZ());
finger2Ph2Group.rwpx.pivotYProperty()
        .setValue(finger2Ph2GroupPivot.getY());
finger2Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger2Ph21Group.rwpx.pivotZProperty()
        .setValue(finger2Ph21GroupPivot.getZ());
finger2Ph21Group.rwpx.pivotYProperty()
        .setValue(finger2Ph21GroupPivot.getY());
finger2Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger2Ph210Group.rwpx.pivotZProperty()
        .setValue(finger2Ph210GroupPivot.getZ());
finger2Ph210Group.rwpx.pivotYProperty()
        .setValue(finger2Ph210GroupPivot.getY());
finger2Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);
```

```
//Specifications of movement for subgroups of finger 3
finger3Ph2Group.rwpx.pivotZProperty()
        .setValue(finger3Ph2GroupPivot.getZ());
finger3Ph2Group.rwpx.pivotYProperty()
        .setValue(finger3Ph2GroupPivot.getY());
finger3Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger3Ph21Group.rwpx.pivotZProperty()
        .setValue(finger3Ph21GroupPivot.getZ());
finger3Ph21Group.rwpx.pivotYProperty()
        .setValue(finger3Ph21GroupPivot.getY());
finger3Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger3Ph210Group.rwpx.pivotZProperty()
        .setValue(finger3Ph210GroupPivot.getZ());
finger3Ph210Group.rwpx.pivotYProperty()
        .setValue(finger3Ph210GroupPivot.getY());
finger3Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);



//======OPENING/CLOSING OF FINGERS========

//Fingers that are at the border must
//open twice faster as those
//near the center.
NumberBinding openFingersPropertyTwo
        = openFingersProperty.multiply(2);
//Specifications of movement for subgroups of finger 0
finger0Ph210Group.rwpy.pivotXProperty()
        .setValue(finger0Ph210GroupPivot.getX());
finger0Ph210Group.rwpy.pivotYProperty()
        .setValue(finger0Ph210GroupPivot.getY());
finger0Ph210Group.rwpy.pivotZProperty()
        .setValue(finger0Ph210GroupPivot.getZ());
```

```
finger0Ph210Group.rwpy.angleProperty()
        .bind(openFingersPropertyTwo);


//Specifications of movement for subgroups of finger 1
finger1Ph210Group.rwpy.pivotXProperty()
        .setValue(finger1Ph210GroupPivot.getX());
finger1Ph210Group.rwpy.pivotYProperty()
        .setValue(finger1Ph210GroupPivot.getY());
finger1Ph210Group.rwpy.pivotZProperty()
        .setValue(finger1Ph210GroupPivot.getZ());

finger1Ph210Group.rwpy.angleProperty()
        .bind(openFingersProperty);

//Finger two remains invariant under opening of fingers

//Opening in the contrary direction of fingers 3 and 4
NumberBinding openFingersPropertyMinus
        = openFingersProperty.multiply(-1);

//Specifications of movement for finger 3
finger3Ph210Group.rwpy.pivotXProperty()
        .setValue(finger3Ph210GroupPivot.getX());
finger3Ph210Group.rwpy.pivotYProperty()
        .setValue(finger3Ph210GroupPivot.getY());
finger3Ph210Group.rwpy.pivotZProperty()
        .setValue(finger3Ph210GroupPivot.getZ());
finger3Ph210Group.rwpy.angleProperty()
        .bind(openFingersPropertyMinus);

NumberBinding openFingersPropertyMinusTwo
        = openFingersProperty.multiply(-2);



//====== The thumb ========
//Specifications of movement for the thumb
```

```
NumberBinding rotateThumbPropertyDown
        = rotateThumbProperty.multiply(-0.4);

NumberBinding rotateThumbPropertyUp
        = rotateThumbProperty.multiply(-2);


NumberBinding closeThumbPropertyMinus
        = closeThumbProperty.multiply(-1);


//Traversal movement
thumbPh210Group.rwpy.pivotXProperty()
        .setValue(thumbPh210GroupPivot.getX());
thumbPh210Group.rwpy.pivotZProperty()
        .setValue(thumbPh210GroupPivot.getZ());
thumbPh210Group.rwpy.angleProperty()
        .bind(closeThumbProperty);

/*
//Rotations of phalanges 2 and 1 to make the grip.
//Rotations around X_AXIS and Y_AXIS at the same time
//Phalanx 2
thumbPh2Group.rwpx.pivotXProperty()
        .setValue(thumbPh2GroupPivot.getX());
thumbPh2Group.rwpx.pivotYProperty()
        .setValue(thumbPh2GroupPivot.getY());
thumbPh2Group.rwpx.pivotZProperty()
        .setValue(thumbPh2GroupPivot.getZ());
thumbPh2Group.rwpx.angleProperty()
        .bind(rotateThumbPropertyDown);

//Phalanges 2 and 1
thumbPh21Group.rwpx.pivotXProperty()
        .setValue(thumbPh21GroupPivot.getX());
thumbPh21Group.rwpx.pivotYProperty()
```

```
        .setValue(thumbPh21GroupPivot.getY());
thumbPh21Group.rwpx.pivotZProperty()
        .setValue(thumbPh21GroupPivot.getZ());
thumbPh21Group.rwpx.angleProperty()
        .bind(rotateThumbPropertyDown);

//Phalanx 2
thumbPh2Group.rwpy.pivotXProperty()
        .setValue(thumbPh2GroupPivot.getX());
thumbPh2Group.rwpy.pivotYProperty()
        .setValue(thumbPh2GroupPivot.getY());
thumbPh2Group.rwpy.pivotZProperty()
        .setValue(thumbPh2GroupPivot.getZ());
thumbPh2Group.rwpy.angleProperty()
        .bind(rotateThumbPropertyDown);
//Phalanges 2 and 1
thumbPh21Group.rwpy.pivotXProperty()
        .setValue(thumbPh21GroupPivot.getX());
thumbPh21Group.rwpy.pivotYProperty()
        .setValue(thumbPh21GroupPivot.getY());
thumbPh21Group.rwpy.pivotZProperty()
        .setValue(thumbPh21GroupPivot.getZ());
thumbPh21Group.rwpy.angleProperty()
        .bind(rotateThumbPropertyDown);
*/

//Rotation of the whole thumb
//to get opposite to other fingers
thumbPh210Group.rwpz.pivotXProperty()
        .setValue(thumbPh210GroupPivot.getX());
thumbPh210Group.rwpz.pivotYProperty()
        .setValue(thumbPh210GroupPivot.getY());
thumbPh210Group.rwpz.pivotZProperty()
        .setValue(thumbPh210GroupPivot.getZ());
thumbPh210Group.rwpz.angleProperty()
        .bind(rotateThumbProperty.multiply(-2));
```

```
    //===========Animation=========

    double endTime1 = 10000.0; ////1000 = 1 second
    double endTime2 = 2 * endTime1;
    double endTime3 = 3 * endTime1;
    double endTime4 = 4 * endTime1;
    double endTime5 = 5 * endTime1;
    double endTime6 = 6 * endTime1;
    double endTime7 = 7 * endTime1;
    double endTime8 = 8 * endTime1;
    double endTime9 = 9 * endTime1;




    animation.getKeyFrames().addAll
    (

//The hand gets closed
        //closeHandProperty: initial position at initial time
        new KeyFrame(Duration.ZERO,
            //Permanent condition
            new KeyValue(closeThumbProperty, 0)),
        //closeHandProperty: final position at final time
        new KeyFrame(new Duration(endTime1),
            new KeyValue(closeThumbProperty, -50)),

        //The hand gets closed
        //closeHandProperty: initial position at initial time
        new KeyFrame(new Duration(endTime1),
            //Permanent condition
            new KeyValue(closeHandProperty, 0)),
        //closeHandProperty: final position at final time
        new KeyFrame(new Duration(endTime2),
            new KeyValue(closeHandProperty, -90))
            ,
```

```
//The thumb closes the grip
//rotateThumbProperty: initial position at initial time
new KeyFrame(new Duration(endTime2),
    //Permanent condition
    new KeyValue(rotateThumbProperty, 0)),
//rotateThumbProperty: final position at final time
new KeyFrame(new Duration(endTime3),
    new KeyValue(rotateThumbProperty, 37)),

//The thumb gets reopened
//rotateThumbProperty: initial position at initial time
new KeyFrame(new Duration(endTime3),
    //Permanent condition
    new KeyValue(rotateThumbProperty, 37)),
//rotateThumbProperty: final position at final time
new KeyFrame(new Duration(endTime4),
    new KeyValue(rotateThumbProperty, 0)),

//The hand get reopened
//closeHandProperty: initial position at initial time
new KeyFrame(new Duration(endTime4),
    //Permanent condition
    new KeyValue(closeHandProperty, -90)),
//closeHandProperty: final position at final time
new KeyFrame(new Duration(endTime5),
    new KeyValue(closeHandProperty, 0)),

//Fingers get reopened
//openFingersProperty: initial position at initial time
new KeyFrame(new Duration(endTime5),
//Permanent condition
new KeyValue(openFingersProperty, 0)),
//openFingersProperty: final position at final time
new KeyFrame(new Duration(endTime6),
    new KeyValue(openFingersProperty, 15)),

//Fingers get closed
```

```
        //openFingersProperty: initial position at initial time
        new KeyFrame(new Duration(endTime6),
        //Permanent condition
        new KeyValue(openFingersProperty, 15)),
        //openFingersProperty: final position at final time
        new KeyFrame(new Duration(endTime7),
            new KeyValue(openFingersProperty, 0)),

        //The thumb gets closed
        //closeThumbProperty: initial position at initial time
        new KeyFrame(new Duration(endTime7),
            //Permanent condition
            new KeyValue(closeThumbProperty, -55)),
        //closeThumbProperty: final position at final time
        new KeyFrame(new Duration(endTime8),
            new KeyValue(closeThumbProperty, 0)),

        //The thumb gets reopen
        //closeThumbProperty: initial position at initial time
        new KeyFrame(new Duration(endTime8),
            //Permanent condition
            new KeyValue(closeThumbProperty, 0)),
        //closeThumbProperty: final position at final time
        new KeyFrame(new Duration(endTime9),
            new KeyValue(closeThumbProperty, -25))

    );




animation.play();

}

private static void printMatrix(double m[][], String title)
{
```

```
    System.out.println("This is the matrix " + title);
    for(int i=0;i<3;i++)
        System.out.println(m[i][0] + " " + m[i][1]+ " "
                                                + m[i][2]);
    System.out.println();
}


//The multiplication of a matrix times a vector
  private static  double[] matrixTimesVector(double m[][],
                                             double v[])
  {
    double[]  helpV = new double[3];
    for(int i=0; i<3; i++)
        helpV[i] =0;
    for(int i=0; i<3; i++)
        for(int j=0; j<3; j++)
            helpV[i]  =helpV[i]  + m[i][j]*v[j];
    return helpV;
  }


//Normalized crux product of vectors v and w
  private static  double dotProduct(double[] v,
                                    double[] w)
  {
    return v[0]*w[0] + v[1]*w[1] + v[2]*w[2];

  }


public void play()
{

}

@Override public void stop()
{
```

```
}

public static void main(String[] args)
{

    launch(args);
}


}//End of main class N45 Cylindricalmutations
```

47, page 211. A hand with cylindrical mutations and with bones more or less accommodated is shown by the next code.

```
/*
    Program N47 Cylindricalmutations3
    This program shows how to accommodate
    the phalanges of the  fingers of the hand
    after  small cylindrical mutations.
    A cylindrical mutation over a cylinder
    may enlarge its radius or its height.
    PCA (Principal Component Analysis) allows to fit
    the best cylinder over a bone,
    which is then  mutated cylindrically.
    An attempt has been made to recover
    the natural pattern of the hand.

    Tested only with some  files,
    particularly  with those of the BodyParts3D project.
    http://lifesciencedb.jp/bp3d/
    Those files are licensed:

    The license for this database is specified in the
    Creative Commons Attribution-Share Alike 2.1 Japan.
    If you use data from this database, please be sure
    attribute this database as follows: "BodyParts3D, (c)
```

```
    The Database Center for Life Science licensed under
    CC Attribution-Share Alike 2.1 Japan".
    http://dbguidehive.biosciencedbc.jp/en/bodyparts3d/lic.html

 */

package Programs;

//Files in the hosting project Programs
import Jama.EigenvalueDecomposition;
import Jama.Matrix;
import Tools.Axes;
import Tools.Cameras;
import Tools.Xform;
import Tools.XformWithPivot;


import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.List;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.binding.NumberBinding;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.geometry.Point3D;
import javafx.scene.Group;
```

```
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.Slider;
import javafx.scene.control.TextArea;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.MeshView;;
import javafx.scene.shape.TriangleMesh;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import javafx.util.Duration;



public class CylindricalMutations3 extends Application
{
    //Visibility of the mesh
    //might depend on the correct choice
    //of this factor. Move it in within 1 and 500.
    private static final float scaleFactor = 5;

    //Scale factors along width, height and length
    static private float scaleFactorW = 1;
    static private float scaleFactorH = 1;
    static private float scaleFactorL = 1;

    //Printing for debugging
    final static boolean printReadFile = false;
```

```
private static final String title
        =  "Importer of meshes in format ";

//Tool to read files
BufferedReader br ;
private static  String textOfFile = "";

private static int counterOfFaces = 0;
private static int counterOfVertexes = 0;
private static int counterOfNormals = 0;

private static int[] destFaces ;

private static final TriangleMesh[] bonesMeshesArray
        = new TriangleMesh[100];

private static final MeshView[] bonesViewArray
        = new MeshView[100];
private static final String[] names1  = new String[100];
private static final String[] names2  = new String[100];

//Counter
private static int counterOfBones = 0;

//*****************

final static Group root = new Group();

final static Group axisGroup = new Group();


final static Xform cameraXform = new Xform();
final static Xform cameraXform2 = new Xform();
final static Xform cameraXform3 = new Xform();

final static double cameraDistance = 0; //1450;//450
final static Xform world3DXform = new Xform();
final static Xform bonesXform = new Xform();
```

```
    //Axes : ctrl + A
    private static final Axes axes = new Axes();
    private static final Cameras camera = new Cameras();

    boolean timelinePlaying = false;
    double ONE_FRAME = 1.0/24.0;
    double DELTA_MULTIPLIER = 200.0;
    double CONTROL_MULTIPLIER = 0.1;
    double SHIFT_MULTIPLIER = 0.1;
    double ALT_MULTIPLIER = 0.5;

    double mousePosX;
    double mousePosY;
    double mouseOldX;
    double mouseOldY;
    double mouseDeltaX;
    double mouseDeltaY;




   //Menubar declaration and initialization
    MenuBar menuBar = new MenuBar();


//This is the outer pane to place tabs
private static final BorderPane outerPane = new BorderPane();
private static final Pane paneHand = new Pane();

//Slider for zooming
    //private  static Slider zSlider;
    //Position of graphic along the z-axis,
    //normal to the screen.
    //It is a double property because
    //it will get involved into a binding
    //private  static final DoubleProperty zPos
    //        = new SimpleDoubleProperty(200);
```

```
//Slider for angle of rotation of fingers
private  static Slider closeHandSlider;
//To close the hand,
//each phalange suffers a rotation.
//It is a double property because
//it will get involved into a binding
private  static final DoubleProperty closeHandProperty
        = new SimpleDoubleProperty(0);

//Slider for opening fingers thanks to
//their transversal rotation
private  static Slider openFingersSlider;
//To open the fingers,
//each phalange suffers a transversal rotation.
//It is a double property because
//it will get involved into a binding
private  static final DoubleProperty openFingersProperty
        = new SimpleDoubleProperty(0);

//Slider for opening/closing the thumb
private  static Slider closeThumbSlider;
//To close the thumb we need a rotation.
//It is a double property because
//it will get involved into a binding
private  static final DoubleProperty closeThumbProperty
        = new SimpleDoubleProperty(0);

//Slider for rotating the thumb to form the grip
private  static Slider rotateThumbSlider;
//To form the grip we need a rotation of the thumb.
//It is a double property because
//it will get involved into a binding
private  static final DoubleProperty rotateThumbProperty
        = new SimpleDoubleProperty(0);


//Nested definition of   nodes is implemented
```

```
//to have, say,  the palm dragging the fingers,
//and a phalanx dragging the rest of its the finger.


//Each finger generates various natural groups:
//Group with the extreme phalanx of finger0
//Symbolic initialization
private static final XformWithPivot finger0Ph2Group
        = new XformWithPivot();
//Group with the two extreme phalanges
private static final XformWithPivot finger0Ph21Group
        = new XformWithPivot();
//Group with all three phalanges. It is the finger
private static final XformWithPivot finger0Ph210Group
        = new XformWithPivot();
//Finger1
private static final XformWithPivot finger1Ph2Group
        = new XformWithPivot();
private static final XformWithPivot finger1Ph21Group
        = new XformWithPivot();
private static final XformWithPivot finger1Ph210Group
        = new XformWithPivot();
//Finger2
private static final XformWithPivot finger2Ph2Group
        = new XformWithPivot();
private static final XformWithPivot finger2Ph21Group
        = new XformWithPivot();
private static final XformWithPivot finger2Ph210Group
        = new XformWithPivot();
//Finger3
private static final XformWithPivot finger3Ph2Group
        = new XformWithPivot();
private static final XformWithPivot finger3Ph21Group
        = new XformWithPivot();
private static final XformWithPivot finger3Ph210Group
        = new XformWithPivot();
//Thumb
private static final XformWithPivot thumbPh2Group
```

```
            = new XformWithPivot();
    private static final XformWithPivot thumbPh21Group
            = new XformWithPivot();
    private static final XformWithPivot thumbPh210Group
            = new XformWithPivot();

     private static final XformWithPivot wristGroup
            = new XformWithPivot();
    //Group with wrist and fingers
    private static final XformWithPivot palmGroup
            = new XformWithPivot();

    private static final Timeline animation
            = new Timeline();

    private static final double[][] coordBoxes
            = new double[100][6];

    private static final BoneDescription[] bones
            = new BoneDescription[30];



    Random r = new Random();

@Override
public void start(Stage primaryStage)
{
    System.out.println("start");

    Scene scene = new Scene(root, 1200, 1000, true);

    paneHand.setPrefSize(500, 500);

    outerPane.setOpacity(0);
    //outerPane.setLayoutX(700);
    //outerPane.setLayoutY(700);
```

```
/*
   Multiple file loader
*/

//Content of files will be shown in textArea
final TextArea textArea = new TextArea();
textArea.prefWidth(400);
textArea.setWrapText(true);
textArea.setText("");
//Container  for textArea.
ScrollPane scrollPane = new ScrollPane();
scrollPane.getStyleClass().add("noborder-scroll-pane");
scrollPane.setContent(textArea);
scrollPane.setFitToWidth(true);
scrollPane.setPrefWidth(400);
scrollPane.setPrefHeight(180);
VBox vBox = new VBox();
vBox.visibleProperty().setValue(false);
vBox.getChildren().addAll(scrollPane);
root.getChildren().addAll(vBox);


// --- Menu files
Menu filesMenu = new Menu("Files");
//Menu to load multiple files
MenuItem loadFiles
        = new MenuItem("Load + process multiple files ");
//MenuItem added to menu
filesMenu.getItems().addAll(loadFiles);
menuBar.getMenus().add(filesMenu);
root.getChildren().add(menuBar);

//Listener
loadFiles.setOnAction((ActionEvent t) ->
{
    FileChooser fileChooser = new FileChooser();

    //Name of file with extension "
```

```
FileChooser.ExtensionFilter extFilter
        = new FileChooser
                .ExtensionFilter("TXT files (*.obj)"
                        , "*.obj");
fileChooser.getExtensionFilters().add(extFilter);

//Files will be kept in a list
List<File> files = new FileChooser()
        .showOpenMultipleDialog(primaryStage);



    //Delete possible repetitions
    //files = deleteRepetitions(files);
//counterOfBones = 0;



for(int j = 0; j < files.size(); j++)
{
    File file = files.get(j);
    String name = file.getName();
    names1[j] = name;
}
counterOfBones = 0;
//Each file is processed
for(int j = 0; j < files.size(); j++)
{

    String name = names1[j];
    boolean repeated = false;
    for (int i = 0; i <  j; i++)
    {
        if (name.equals(names1[i]))
        {
            repeated = true;

            //System.out.println("Repeated" + name);
```

```
                }
            }

            if (repeated);
            else
            try
            {
                File file = files.get(j);
                //Text of file is read and mesh is extracted
                textOfFile = readFile(file);
                //Text copied to textArea
                textArea.appendText(textOfFile);

            } catch (IOException ex)
                {
                    Logger.getLogger(AnimatedHandSkeleton
                            .class.getName())
                            .log(Level.SEVERE, null, ex);
                }



        }

        defineGroups();



    //To remedy a not understood bug
    //( = menuBar is added to the graphic )
    root.getChildren().remove(menuBar);

    vBox.visibleProperty().set(true);
    });

    //====== End of File loader
```

```
primaryStage.setTitle(title
        + ". Drag the mouse to rotate. "
    + " Ctrl + A = Axis visibility"
        );
scene.setFill(Color.BLUE);
primaryStage.setScene(scene);
primaryStage.show();
axes.buildAxes();
axes.handleKeyboard(scene);
//camera.handleMouse(scene);
camera.handleMouse(paneHand);

//Hide textArea and show graphic
Button hideTextArea = new Button();
hideTextArea.setText("View graphic");
//Listener
hideTextArea.setOnAction((ActionEvent t) ->
{
    vBox.visibleProperty().set(false);
    camera.buildCamera();


    world3DXform.getChildren().add(bonesXform);


    //world3DXform.setRx(180);
    //world3DXform.setRy(90);
    world3DXform.setScale(scaleFactor);
    world3DXform.setTranslate(-3500, -1400, 0);

    paneHand.getChildren().add(world3DXform);
    root.getChildren().add(paneHand);
    scene.setCamera(camera.camera);
});

vBox.getChildren().addAll(hideTextArea);
```

```
//This slider   controls the closing of the hand.
closeHandSlider = new Slider();
closeHandSlider.setMax(90);
closeHandSlider.setMin(0);
closeHandSlider.setValue(10);

//To close the hand, an angle of rotation is enabled.
//Bind angleProperty to the reading of the slider
closeHandSlider.valueProperty()
        .bindBidirectional(closeHandProperty);


//This slider controls the openning of fingers.
openFingersSlider = new Slider();
openFingersSlider.setMax(15);
openFingersSlider.setMin(0);
openFingersSlider.setValue(10);

//To open fingers, an angle of rotation is enabled.
//Bind openFingersProperty to the reading of the slider
openFingersSlider.valueProperty()
        .bindBidirectional(openFingersProperty);


//Slider to open  the thumb
closeThumbSlider = new Slider();
closeThumbSlider.setMax(55);
closeThumbSlider.setMin(0);
closeThumbSlider.setValue(-0);
closeThumbSlider.adjustValue(closeThumbSlider.getValue());

//To open the thumb, an angle of rotation is enabled.
//Bind closeThumbProperty to the reading of the slider
closeThumbSlider.valueProperty()
        .bindBidirectional(closeThumbProperty);
```

```
//Slider to rotate  the thumb
rotateThumbSlider = new Slider();
rotateThumbSlider.setMax(50);
rotateThumbSlider.setMin(0);
rotateThumbSlider.setValue(0);
rotateThumbSlider
        .adjustValue(rotateThumbSlider.getValue());

//To rotate the thumb, an angle of rotation is enabled.
//Bind closeThumbProperty to the reading of the slider
rotateThumbSlider.valueProperty()
        .bindBidirectional(rotateThumbProperty);



//Other JavaFX controls, labels and containers
GridPane gridPane = new GridPane();
gridPane.setVgap(4);
gridPane.setHgap(10);
gridPane.setPadding(new Insets(5, 5, 5, 5));
//Column 0, Row   0
gridPane.add(new Label("Zoom: "), 0, 0);
//gridPane.add(zSlider, 1, 0);
gridPane.add(new Label("Close hand: "), 2, 0);
gridPane.add(closeHandSlider, 3, 0);
gridPane.add(new Label("Open fingers: "), 0, 1);
gridPane.add(openFingersSlider, 1, 1);
gridPane.add(new Label("Close thumb: "), 0, 2);
gridPane.add(closeThumbSlider, 1, 2);
gridPane.add(new Label("Rotate thumb: "), 2, 2);
gridPane.add(rotateThumbSlider, 3,2);



SubScene subScene
        = new SubScene(gridPane, 880, 200);
subScene.setLayoutX(10);
subScene.setLayoutY(800);
```

```
    paneHand.getChildren().add(subScene);









    primaryStage.show();
}




//File is read and its content is returned as a string.
//The associated mesh is recovered
//and correct orientation of faces is provided.
//The mesh is a global variable.
private  String readFile(File file)
        throws FileNotFoundException, IOException
{
    //Name of file is printed
    String name = file.getName();
    //if (printReadFile)
        System.out.println(name);
    names2[counterOfBones] = name;
    BoneDescription b = new BoneDescription();
    b.name = name;
    bones[counterOfBones] = b;
```

```
    //Buffer to keep the content of file
    //StringBuilder stringBuffer = new StringBuilder();
    StringBuilder stringBuffer  = new StringBuilder();
    //Tool to read file
    BufferedReader bufferedReader = null;

    //A mesh is declared and instantiated
    TriangleMesh mesh  = new TriangleMesh();
    //The mesh has no predefined texture
    mesh.getTexCoords().addAll(0,0);
    //In the " format, counting begins with 1
    //but in Java, we begin with 0.
    //So, an additional inocuos point is added.
    mesh.getPoints().addAll(0,0,0);

    if (printReadFile)
        System.out.println("File content: ");
//File is read
try
{
//
bufferedReader = new BufferedReader(new FileReader(file));

String text;




while( (text = bufferedReader.readLine()) != null)
{
    //Text for the textArea
    stringBuffer.append(text);
    //Print each line
    //System.out.println(text);

    //Vertexes are read
    if (((text.length() > 0 ) && text.charAt(0) == 'v')
```

```
            && text.charAt(1) == ' ')
    {
        if (printReadFile)
            System.out.println("Vertex  = "
                    + counterOfVertexes + " = "+text);
        counterOfVertexes++;
        text = text.substring(2);
        int j = 0;
        //Three vertexes are read
        for(int i = 0; i < 3; i++)
        {
            double d ;
            String s = "";
            Boolean go = true;

            while (go)
            {
                char thisChar = text.charAt(j);
                s = s + thisChar;
                char nextChar = ' ';
                //++j means add one to j
                //and then evaluate expression with <
                if (++j < text.length())
                 nextChar = text.charAt(j);
                if ((nextChar == ' ') ||
                        (j >= text.length()))
                        go = false;

            }
            //string to double
            d = Double.parseDouble(s);
            float df = (float) d;
            if (printReadFile)
                System.out.println("Float number = " + df);
            mesh.getPoints().addAll(df);

        }
    }
```

```
//Normal vectors are counted and displayed
if ((text.length() > 0 ) && (text.charAt(0) == 'v')
        && (text.charAt(1) == 'n'))
{

    if (printReadFile)
        System.out.println("Normal " + counterOfNormals
            + " = " + text);
    counterOfNormals++;
}


//Face is read, one face per line.
//A face has 3 or more vertexes. These are extracted.
//Additional information is ignored
if ( (text.length() > 0 ) && (text.charAt(0) == 'f'))
{

    if (printReadFile)
        System.out.println("Face  = " + counterOfFaces
            + " = "+text);
    counterOfFaces++;
    text = text.substring(2);

    int vertex[] = new int[4];
    int j = 0;

    Boolean isNotBlank ;
    //Three vertexes are extracted
    for(int i = 0; i < 3; i++)
    {

        double d ;
        String s = "";
        Boolean go1 = true;
```

```
while (go1)
{
    if (j >= text.length()-2)
        break;

    char thisChar = text.charAt(j);
    s = s + thisChar;
    /*
    System.out.println(" j = "
            + j + "    s = " + s);
            */
    char nextChar = ' ';
    j++;
    if (j < text.length())
     nextChar = text.charAt(j);
    char nextNextChar;
    isNotBlank = true;
    if (nextChar == '/' )
    {
        go1 = false;
        //go until the following blank
        while ((isNotBlank)
                && (j < text.length()-1))
        {
            j++;
            nextNextChar = text.charAt(j);
            if (nextNextChar == ' ' )
                isNotBlank = false;

        }
    }

}

d = Double.parseDouble(s);
int v = (int) d;
vertex[i] = v;
```

```
    if (printReadFile)
        System.out.println("Int number = " + v);
}//End of cycle for 3 vertexes

//A face with 3 vertexes is added to mesh
mesh.getFaces().addAll(vertex[0], 0, vertex[1], 0,
    vertex[2], 0);

if (printReadFile)
{
    for(int i = 0; i < 3; i++)
     System.out.print(vertex[i] + " . ");
    System.out.println();
}

//Is there a four vertex in the face?
if   ( (text.length() - j )> 3)
{
    String s = "";
    Boolean go1 = true;
    while (go1)
    {


        char thisChar = text.charAt(j);
        s = s + thisChar;
        j++;
        char nextChar = ' ';
        if (j < text.length())
         nextChar = text.charAt(j);
        if (nextChar == '/' ) go1 = false;
    }
    if (printReadFile)
        System.out.println("4th = " + s);
    double d = Double.parseDouble(s);
    int v = (int) d;
    vertex[3] = v;
```

```
            //A polyface with 4 vertex, say, a,b,c,d,
            //can be covered by 2 triangles:
            //Triangle 1: a,b,c (Already added to mesh)
            //and triangle 2: a,c,d.
            //Triangle 2 with 3 vertexes is added to mesh
            mesh.getFaces().addAll(vertex[0], 0, vertex[2],
                    0, vertex[3], 0);
        }
    }//End of face cycle

    //Bounds of containing box are read and displayed
    if ( (text.length() > 0 )
            && (text.contains("# Bounds(mm):")))
    {
        //System.out.println(text);
        System.out.print("Bounds =   ");
        String a = "# Bounds(mm): (";
        int m = a.length();
        text = text.substring(m);
        text = text.replace(")-(",   ",   ");
        text = text.replace(")",   ",   ");
        System.out.println(text);

        int j = 0;


        //Six vertexes are extracted
        for(int i = 0; i < 6; i++)
        {

            double d ;
            String s = "";
            Boolean go1 = true;


            while (go1)
            {
                if (j >= text.length()-2)
```

```
                    break;

                char thisChar = text.charAt(j);
                s = s + thisChar;
                /*
                System.out.println(" j = "
                        + j + "    s = " + s);
                */
                char nextChar = ' ';
                j++;
                if (j < text.length())
                 nextChar = text.charAt(j);
                if (nextChar == ',' )
                {
                    go1 = false;
                   if ( j < text.length() ) j++;
                }

            }

            d = Double.parseDouble(s);

            coordBoxes[counterOfBones][i] = d;
           if (printReadFile)
                System.out.println("Coordinate = "
                        + i + "= " + d);
        }//End of cycle for 3 vertexes
    }

}

} //End of  try.
    //Exception management
    catch (FileNotFoundException ex)
    {
        Logger.getLogger(AnimatedHandSkeleton
        .class.getName()).log(Level.SEVERE, null, ex);
    }
```

```
catch (IOException ex)
{
    Logger.getLogger(AnimatedHandSkeleton
    .class.getName()).log(Level.SEVERE, null, ex);
}
finally
{
    try
    {
        bufferedReader.close();
    } catch (IOException ex)
    {
        Logger.getLogger(AnimatedHandSkeleton
        .class.getName()).log(Level.SEVERE, null, ex);
    }
}



if (printReadFile)
{
    System.out.println("End of file content. ");
    //Other indicators are reported
    report(mesh);
}


//======MeshWork=======

PCA(mesh);

MeshView boneMesh =
        new MeshView(mesh);

PhongMaterial goldenMaterial
        = new PhongMaterial(Color.GOLDENROD);
boneMesh.setMaterial(goldenMaterial);
Group boneGroup = new Group(boneMesh);
```

```
    bonesViewArray[counterOfBones] = boneMesh;

    Xform boneXform = new Xform();
    boneXform.getChildren().add(boneGroup);

    bonesXform.getChildren().add(boneXform);
    counterOfBones++;

    return stringBuffer.toString();
}


//Indicators of retrieved data are printed
private void report(TriangleMesh mesh)
{
    float[] destPoints = new float[mesh.getPoints().size()];
    mesh.getPoints().toArray(destPoints);
    System.out.println("POINTS");
    for(int j = 0; j <  destPoints.length/3; j++)
    {
        System.out.print(j + "   " );
        for(int k = 0; k < 3; k++)
        {
            System.out.print( destPoints[j*3 + k] + "   " );
        }
        System.out.println();
    }

    destFaces = new int[mesh.getFaces().size()];
    //Faces go to a matrix
    mesh.getFaces().toArray(destFaces);
    System.out.println("FACES WITH TEXTURE COORDINATES");
    for(int j = 0; j <  destFaces.length/6; j++)
    {
        System.out.print(j + "   " );
        for(int k = 0; k < 6; k++)
        {
            System.out.print( destFaces[j*6 + k] + "   " );
```

```
        }
        System.out.println();
    }

    System.out.println("counterOfPoints = "
            + counterOfVertexes );
    System.out.println("counterOfFaces = "
            + counterOfFaces );
    System.out.println("counterOfNormals = "
            + counterOfNormals);
    System.out.println("mesh.getPoints().size() = "
            + mesh.getPoints().size() );
    System.out.println("mesh.getFaces().size() = "
            + mesh.getFaces().size() );

    System.out.println("getPointElementSize = "
            + mesh.getPointElementSize() );
    System.out.println("getfaceElementSize = "
            + mesh.getFaceElementSize() );

}



public static String fixedWidthDoubletoString
        (double x, int w, int d) {
   java.text.DecimalFormat fmt
           = new java.text.DecimalFormat();
   fmt.setMaximumFractionDigits(d);
   fmt.setMinimumFractionDigits(d);
   fmt.setGroupingUsed(false);
   String s = fmt.format(x);
   while (s.length() < w) {
      s = " " + s;
   }
   return s;
}
```

```
//Principal component analysis of a mesh
private void PCA(TriangleMesh mesh)
{
    //Point are declared as a matrix of coordinates
    int n = mesh.getPoints().size();
        float nPoints = n;
    nPoints = nPoints/3;
    float[][] CoordinatesFloat = new float[(int) nPoints][3];


    /*
    //Test: adapted from
    //Stat Trek: Variance-Covariance Matrix
    //http://stattrek.com/matrix-algebra/covariance-matrix.aspx

    float[] data = {90,  60,  90,
                    90,  90,  30,
                    60,  60,  60,
                    60,  60,  90,
                    30,  30,  30 };


        n = 15;
        nPoints = 5;
    int counter = 0;
    for(int i = 0; i < n; i = i+3)
    {
        //Reading
        float x =  data[i];
        float y =  data[i+1] ;
        float z =  data[i+2] ;
        CoordinatesFloat[counter][0] = x;
        CoordinatesFloat[counter][1] = y;
        CoordinatesFloat[counter][2] = z;
        counter++;
    }

    //Mean is found
```

```
float[] mean = new float[3];

for(int i = 0; i < 3; i++)
{
    mean[i] = 0;
    for (int j = 0; j <  nPoints; j++)
        mean[i] = mean[i] + CoordinatesFloat[j][i];
    mean[i] = mean[i]/(nPoints);
}

System.out.println("meanX = " + mean[0]);
System.out.println("meanY = " + mean[1]);
System.out.println("meanZ = " + mean[2]);




//Calculate covariance matrix
double[][] Cov = new double[3][3];
for(int i = 0; i < 3; i++)
{
    for (int j = 0; j <  3; j++)
    {
        Cov[i][j] = 0;
        for (int k = 0; k <  nPoints; k++)
            Cov[i][j] = Cov[i][j] +
                    (CoordinatesFloat[k][i] - mean[i])
                    * (CoordinatesFloat[k][j] - mean[j]);

        Cov[i][j] = Cov[i][j]/nPoints;

    }

}

printMatrix(Cov, "Cov");

//Answer: cov matrix must be:
```

```
//504  360  180
//360  360  0
//180  0  720
*/

//=============

int counter = 0;
for(int i = 0; i < n; i = i+3)
{
    //Reading
    float x =  mesh.getPoints().get(i);
    float y =  mesh.getPoints().get(i+1);
    float z =  mesh.getPoints().get(i+2);
    CoordinatesFloat[counter][0] = x;
    CoordinatesFloat[counter][1] = y;
    CoordinatesFloat[counter][2] = z;
    counter++;
    //Warning: the zeroth point is artificial,
    //filled with zeros.
    if (i == 0)
        System.out.println("Zeroth point: x = " + x
                + "   y =   " + y + "   z =  "  + z);
}

//Mean is found. It is vector with 3 coordinates
float[] mean = new float[3];

for(int i = 0; i < 3; i++)
{
    mean[i] = 0;
    //Zeroth point does not count
    for (int j = 1; j <  nPoints; j++)
        mean[i] = mean[i] + CoordinatesFloat[j][i];
    mean[i] = mean[i]/(nPoints -1);
}

System.out.println("meanX = " + mean[0]);
```

```
System.out.println("meanY = " + mean[1]);
System.out.println("meanZ = " + mean[2]);
Point3D meanPoint = new Point3D(mean[0], mean[1], mean[2]);
bones[counterOfBones].mean = meanPoint;



//Calculate covariance matrix
double[][] Cov = new double[3][3];
for(int i = 0; i < 3; i++)
{
    for (int j = 0; j <  3; j++)
    {
        Cov[i][j] = 0;
        for (int k = 1; k <  nPoints; k++)
            Cov[i][j] = Cov[i][j] +
                    (CoordinatesFloat[k][i] - mean[i])
                    * (CoordinatesFloat[k][j] - mean[j]);

        Cov[i][j] = Cov[i][j]/(nPoints-1);

    }

}

printMatrix(Cov, "Cov");

//=========

//Eigenvalues of Cov matrix are found
Matrix M = new Matrix(Cov);
EigenvalueDecomposition E =
     new EigenvalueDecomposition(M);
double[] d = E.getRealEigenvalues();

for(int i = 0; i < 3; i++)
    System.out.println("Eigenvalue " + i + " : " + d[i]);
double[][] EV = E.getV().getArray();
```

```
for(int i = 0; i < 3; i++)
{
    System.out.println("Eigenvector " + i + " : " );
    for (int j = 0; j <  3; j++)
    {
        System.out.println(EV[i][j]);
    }
    System.out.println();
}

//Test: eigenvectors of a real symmetric matrix
//are orthogonal one to another: dot product must be zero
System.out.println("Orthogonality test "
        + ": matrix I is expected");
double[][] Ort = new double[3][3];
for(int i = 0; i < 3; i++)
{

    for (int j = 0; j <  3; j++)
    {

        //sum = sum + EV[i][j]*EV[i+1][j];
        Ort[i][j] = dotProduct(EV[i], EV[j]);

    }
    //System.out.println(sum);

}

printMatrix(Ort, "Ort test");

//Determinant must be 1.
System.out.println("Orientation test: "
        + "+1 is expected. Det E = "
        + E.getV().det() + " \n");


 //Mesh is centered
```

```
float [] pointsFloat = new float[n];
for(int i = 0; i < n; i = i+3)
{
    //Reading
    float x =  mesh.getPoints().get(i);
    float y =  mesh.getPoints().get(i+1);
    float z =  mesh.getPoints().get(i+2);

    //Mesh is centered
    pointsFloat[i]    =  (x - mean[0]);
    pointsFloat[i+1]  =  (y - mean[1]);
    pointsFloat[i+2]  =  (z - mean[2]);
}
//Resetting of points of mesh
mesh.getPoints().setAll(pointsFloat, 0, n);

axes.axesGroup.setVisible(true);

//Scale factors are composed
Point3D P =  cylindricalScaleFactors();
//Memorization
bones[counterOfBones].scaleFactors = P;

/*
//Test: slight magnification
scaleFactorW = 1.1f;
scaleFactorH = 1.1f;
scaleFactorL = 1.1f;
*/

//Scale factors as a diagonal matrix
double[][] scaleMatrix = {{scaleFactorW, 0,0},
                          {0, scaleFactorH, 0},
                          {0,0, scaleFactorL}};



//1D array for recording
```

```
    float[]newPoints = new float[n];
    //Mesh is aligned with the new basis
    //given by the PCA eigenvetors
    for(int i = 0; i < n; i = i+3)
    {
         //Reading
        float x =  mesh.getPoints().get(i);
        float y =  mesh.getPoints().get(i+1);
        float z =  mesh.getPoints().get(i+2);
        //Three coordinates define a point
        double[] w = {x,y,z};
        //Coordinates of point w are transformed
        //to the new basis
        w = matrixTimesVector(E.getV()
                .transpose().getArray(),w);
        //PCA scaling is applied
        w = matrixTimesVector(scaleMatrix, w);
        //Coordinates of point w are transformed
        //to the old, natural basis
        w = matrixTimesVector(E.getV().getArray(),w);

        //Initial center is restored.
        //Point is added to the  1-dimensional array
        newPoints[i ] = (float) ( w[0] + mean[0]);
        newPoints[i + 1] = (float) ( w[1]  + mean[1] );
        newPoints[i + 2] = (float) ( w[2] + mean[2]);
    }
    //Points are reset
    mesh.getPoints().setAll(newPoints, 0, n);

}


//Returns a random float number
//uniformly distributed in the interval
//(1-epsilon, 1 + epsilon)
private float randomFactor(float epsilon)
{
```

```
    float s = 2 * epsilon * r.nextFloat();
    s = 1- epsilon + s;
    return s;
}


//Return random values for scale factors.
//Factors can be greater than or less than 1
//Tests are included.
private Point3D cylindricalScaleFactors()
{
    float epsilon = 0.5f;
    scaleFactorW = randomFactor(epsilon);
    scaleFactorH =  scaleFactorW;
    scaleFactorL = randomFactor(epsilon);
    //Test: uniform shrinking
    /*
    scaleFactorW = 0.5f;
    scaleFactorH = 0.5f;
    scaleFactorL = 0.5f;
    */
    //Test: homogeneous enlargement
    /*
    scaleFactorW = 3f;
    scaleFactorH = 3f;
    scaleFactorL = 3f;
    */
    return new Point3D(scaleFactorW, scaleFactorH,
            scaleFactorL);

}


//Returns th pivot of bone i:
//the middle x-value, the middle y-value and
//the minimal z-value
private static Point3D pivot(int i)
{
```

```
    double x = (coordBoxes[i][0] + coordBoxes[i][3])/2;
    double y = (coordBoxes[i][1] + coordBoxes[i][4])/2;
    double z = coordBoxes[i][5];
    Point3D p = new Point3D(x,y,z);
    return p;
}


private static Point3D initial(int i)
{
    double x = (coordBoxes[i][0] + coordBoxes[i][3])/2;
    double y = (coordBoxes[i][1] + coordBoxes[i][4])/2;
    double z = coordBoxes[i][5];
    Point3D p = new Point3D(x,y,z);
    return p;
}

private static Point3D end(int i)
{
    double x = (coordBoxes[i][0] + coordBoxes[i][3])/2;
    double y = (coordBoxes[i][1] + coordBoxes[i][4])/2;
    double z = coordBoxes[i][2];
    Point3D p = new Point3D(x,y,z);
    return p;
}

private static void    defineGroups()
{
    //Pivots of groups
    Point3D finger0Ph2GroupPivot ;
    Point3D finger0Ph21GroupPivot  ;
    Point3D finger0Ph210GroupPivot ;
    Point3D finger1Ph2GroupPivot ;
    Point3D finger1Ph21GroupPivot ;
    Point3D finger1Ph210GroupPivot ;
    Point3D finger2Ph2GroupPivot ;
    Point3D finger2Ph21GroupPivot ;
    Point3D finger2Ph210GroupPivot ;
```

```
Point3D finger3Ph2GroupPivot ;
Point3D finger3Ph21GroupPivot ;
Point3D finger3Ph210GroupPivot ;

Point3D thumbPh21GroupPivot = new Point3D(0,0,0);
Point3D thumbPh210GroupPivot ;
Point3D wristGroupPivot = new Point3D(0,0,0);



//Natural groups are instantiated

for(int i = 0; i < counterOfBones; i++)
{
    String name = names2[i];
    System.out.println("i = " + i + "  " +name);


    //Finger 0 = little and its subgroups
    if (name.contains("little"))
    {
        if (name.contains("Distal"))
        {
            finger0Ph2Group
                .getChildren().add(bonesViewArray[i]);
            bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "Middle phalanx of"
                    + " left little finger";
            bones[i].next = "";

        }
        if (name.contains("Middle"))
        {
            finger0Ph21Group
                .getChildren()
                .addAll(finger0Ph2Group
```

```
                    , bonesViewArray[i]);
            bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "Proximal phalanx "
                    + "of left little finger";
            bones[i].next = "Distal phalanx of "
                    + "left little finger";

        }
        if (name.contains("Proximal"))
        {
            finger0Ph210Group
                .getChildren()
                .addAll(finger0Ph21Group
                        , bonesViewArray[i]);
            bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "Left first "
                    + "metacarpal bone";
            bones[i].next = "Middle phalanx "
                    + "of left little finger";
        }
    }

    //Finger 1 = ring and its subgroups
    else if (name.contains("ring"))
    {
        if (name.contains("Distal"))
        {
            finger1Ph2Group
                .getChildren().add(bonesViewArray[i]);
            bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "Middle phalanx "
                    + "of left ring finger";
            bones[i].next = "";
        }
        if (name.contains("Middle"))
```

```
        {
            finger1Ph21Group
                .getChildren()
                .addAll(finger1Ph2Group, bonesViewArray[i]);
            bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "Proximal phalanx "
                    + "of left ring finger";
            bones[i].next = "Distal phalanx "
                    + "of left ring finger";
        }
        if (name.contains("Proximal"))
        {
            finger1Ph210Group
                .getChildren()
                .addAll(finger1Ph21Group, bonesViewArray[i])
            bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "Left second "
                    + "metacarpal bone";
            bones[i].next = "Middle phalanx "
                    + "of left ring finger";
        }
    }

    //Finger 2 = middle and its subgroups
    else if (name.contains("middle"))
    {
        if (name.contains("Distal"))
        {
            finger2Ph2Group
            .getChildren().add(bonesViewArray[i]);
            bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "Middle phalanx "
                    + "of left middle finger";
            bones[i].next = "";
        }
```

```
    if (name.contains("Middle"))
    {
        finger2Ph21Group
            .getChildren()
            .addAll(finger2Ph2Group, bonesViewArray[i]);
        bones[i].start = initial(i);
        bones[i].end = end(i);
        bones[i].previous = "Proximal phalanx "
                + "of left middle finger";
        bones[i].next = "Distal phalanx "
                + "of left middle finger";
    }
    if (name.contains("Proximal"))
    {
        finger2Ph210Group
            .getChildren()
            .addAll(finger2Ph21Group, bonesViewArray[i]);
        bones[i].start = initial(i);
        bones[i].end = end(i);
        bones[i].previous = "Left third "
                + "metacarpal bone";
        bones[i].next = "Middle phalanx "
                + "of left middle finger";
    }
}

//Finger 3 = index and its subgroups
else if (name.contains("index"))
{
    if (name.contains("Distal"))
    {
        finger3Ph2Group
            .getChildren().add(bonesViewArray[i]);
        bones[i].start = initial(i);
        bones[i].end = end(i);
        bones[i].previous = "Middle phalanx "
                + "of left index finger";
        bones[i].next = "";
```

```
        }
        if (name.contains("Middle"))
        {
            finger3Ph21Group
                .getChildren()
                .addAll(finger3Ph2Group, bonesViewArray[i]);
            bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "Proximal phalanx "
                    + "of left index finger";
            bones[i].next = "Distal phalanx "
                    + "of left middle finger";
        }
        if (name.contains("Proximal"))
        {
            finger3Ph210Group
                .getChildren()
                .addAll(finger3Ph21Group
                        , bonesViewArray[i]);
            bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "Left fourth "
                    + "metacarpal bone";
            bones[i].next = "Middle phalanx "
                    + "of left index finger";
        }
    }

    //Thumb and its subgroups
    else if (name.contains("thumb"))
    {
        if (name.contains("Distal"))
        {
            thumbPh2Group
                .getChildren().add(bonesViewArray[i]);
               bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "Proximal phalanx "
```

```
                + "of left thumb";
            bones[i].next = "";
        }
        if (name.contains("Proximal"))
        {
            thumbPh21Group
                .getChildren()
                .addAll(thumbPh2Group
                        , bonesViewArray[i]);
            thumbPh21GroupPivot = pivot(i);
            bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "";
            bones[i].next = "Distal phalanx "
                    + "of left thumb";
        }
    }

    //Thumb and its subgroups
    else if (name.contains("first metacarpal"))
    {
        thumbPh210Group
            .getChildren()
            .addAll(thumbPh21Group, bonesViewArray[i]);
        //thumbPh210GroupPivot = pivot(i);
        bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "";
            bones[i].next = "";
    }
    //Other bones
    else if (name.contains("second metacarpal"))
    {
        wristGroup.getChildren().add(bonesViewArray[i]);
        bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].previous = "";
            bones[i].next = "Proximal phalanx "
```

```
                + "of left index finger";
        }

        else if (name.contains("third metacarpal"))
        {
            wristGroup.getChildren().add(bonesViewArray[i]);
            bones[i].start = initial(i);
                bones[i].end = end(i);
                bones[i].previous = "";
                bones[i].next = "Proximal phalanx "
                        + "of left middle finger";
        }

        else if (name.contains("fourth metacarpal"))
        {
            wristGroup.getChildren().add(bonesViewArray[i]);
            bones[i].start = initial(i);
                bones[i].end = end(i);
                bones[i].previous = "";
                bones[i].next = "Proximal phalanx "
                        + "of left ring finger";
        }

        else if (name.contains("fifth metacarpal"))
        {
            wristGroup.getChildren().add(bonesViewArray[i]);
            bones[i].start = initial(i);
                bones[i].end = end(i);
                bones[i].previous = "";
                bones[i].next = "Proximal phalanx "
                        + "of left little finger";
        }

        //Thumb and its subgroups
        else if (name.contains("trapezium"))
        {
           wristGroup.getChildren().add(bonesViewArray[i]);
           bones[i].start = initial(i);
```

```
                bones[i].end = end(i);
                bones[i].previous = "";
                bones[i].next = "";
        }
        else

        {
            wristGroup.getChildren().add(bonesViewArray[i]);
            if (name.contains("lunate"))
                wristGroupPivot = pivot(i);
            bones[i].start = initial(i);
                bones[i].end = end(i);
                bones[i].previous = "";
                bones[i].next = "";
        }
}


    //Fingers are attached to the palm

        palmGroup.getChildren().addAll(wristGroup
            , finger0Ph210Group
            , finger1Ph210Group
            , finger2Ph210Group
            , finger3Ph210Group
            , thumbPh210Group);

        world3DXform.getChildren().addAll(palmGroup);

    Point3D I0 = new Point3D(0,0,0);
    Point3D I1 = new Point3D(0,0,0);
    Point3D I2 = new Point3D(0,0,0);
    Point3D I3 = new Point3D(0,0,0);
    Point3D M0 = new Point3D(0,0,0);
    Point3D M1 = new Point3D(0,0,0);
    Point3D M2 = new Point3D(0,0,0);
    Point3D M3 = new Point3D(0,0,0);
    Point3D E0 = new Point3D(0,0,0);
```

```
Point3D E1 = new Point3D(0,0,0);
Point3D E2 = new Point3D(0,0,0);
Point3D E3 = new Point3D(0,0,0);
Point3D S0 = new Point3D(0,0,0);
Point3D S1 = new Point3D(0,0,0);
Point3D S2 = new Point3D(0,0,0);
Point3D S3 = new Point3D(0,0,0);
Point3D C0 ;
Point3D C1 ;
Point3D C2;



//Bone groups of index finger get reaccommodated
for(int i = 0; i < counterOfBones ; i++)
{
    if (bones[i].name.contains("index"))
    {
        if (bones[i].name.contains("Distal"))
        {
           I0 = bones[i].start;
           M0 = bones[i].mean;
           E0 = bones[i].end;
           S0 = bones[i].scaleFactors;
        }

        if (bones[i].name.contains("Middle"))
        {
           I1 = bones[i].start;
           M1 = bones[i].mean;
           E1 = bones[i].end;
           S1 = bones[i].scaleFactors;
        }

        if (bones[i].name.contains("Proximal"))
        {
           I2 = bones[i].start;
           M2 = bones[i].mean;
```

```
            E2 = bones[i].end;
            S2 = bones[i].scaleFactors;
         }


    }
}



//The index finger is connected to
//the second metacarpal bone
//Bug: this piece of code appeared
//into the previous loop. In that way it was innocuous.
for(int i = 0; i < counterOfBones ; i++)
{


         if (bones[i].name.contains("Left second "
                + "metacarpal bone"))
         {
            I3 = bones[i].start;
            M3 = bones[i].mean;
            E3 = bones[i].end;
            //Bug: S2 was there when S3 was needed
            S3 = bones[i].scaleFactors;
           System.out.println("Left second "
                + "metacarpal bone");
         }

}


C0 = (I0.add(E1)).multiply(0.5);
double tz2 = (1-S0.getZ()) * ((C0.subtract(M0)).getZ())
            + (1-S1.getZ() )* ((M1.subtract(C0)).getZ()) ;
```

```
finger3Ph2Group.setTranslateZ(tz2);
finger3Ph2GroupPivot = new Point3D(C0.getX()
        ,C0.getY(), C0.getZ()-tz2);


C1 = (I1.add(E2)).multiply(0.5);
double tz21 = (1-S1.getZ()) * ((C1.subtract(M1)).getZ())
            + (1-S2.getZ() )* ((M2.subtract(C1)).getZ()) ;
finger3Ph21Group.setTranslateZ(tz21);
finger3Ph21GroupPivot = new Point3D(C1.getX()
        ,C1.getY(), C1.getZ()-tz21);


C2 = (I2.add(E3)).multiply(0.5);
double tz210 = (1-S2.getZ()) * ((C2.subtract(M2)).getZ())
            + (1-S3.getZ() )* ((M3.subtract(C2)).getZ()) ;
finger3Ph210Group.setTranslateZ(tz210);
finger3Ph210GroupPivot = new Point3D(C2.getX()
        ,C2.getY(), C2.getZ()-tz210);


System.out.println("Data for bones of the index finger");
System.out.println(" E0.toString()" + E0.toString());
System.out.println(" M0.toString()" + M0.toString());
System.out.println(" I0.toString()" + I0.toString());
System.out.println(" C0.toString()" + C0.toString());
System.out.println(" S0.toString()" + S0.toString());

System.out.println(" E1.toString()" + E1.toString());
System.out.println(" M1.toString()" + M1.toString());
System.out.println(" I1.toString()" + I1.toString());
System.out.println(" C1.toString()" + C0.toString());
System.out.println(" S1.toString()" + S1.toString());
System.out.println(" E2.toString()" + E2.toString());
System.out.println(" M2.toString()" + M2.toString());
System.out.println(" I2.toString()" + I2.toString());
```

```
System.out.println(" C2.toString()" + C0.toString());
System.out.println(" S2.toString()" + S2.toString());
System.out.println(" E3.toString()" + E3.toString());
System.out.println(" M3.toString()" + M3.toString());
System.out.println(" I3.toString()" + I3.toString());
System.out.println(" C3.toString()" + C0.toString());
System.out.println(" S3.toString()" + S3.toString());


//Bone groups of middle finger get reaccommodated
for(int i = 0; i < counterOfBones ; i++)
{
    if (bones[i].name.contains("middle"))
    {
        if (bones[i].name.contains("Distal"))
        {
           I0 = bones[i].start;
           M0 = bones[i].mean;
           S0 = bones[i].scaleFactors;
        }

        if (bones[i].name.contains("Middle"))
        {
           I1 = bones[i].start;
           M1 = bones[i].mean;
           E1 = bones[i].end;
           S1 = bones[i].scaleFactors;
        }

        if (bones[i].name.contains("Proximal"))
        {
           I2 = bones[i].start;
           M2 = bones[i].mean;
           E2 = bones[i].end;
           S2 = bones[i].scaleFactors;
        }
```

```
    }
}


for(int i = 0; i < counterOfBones ; i++)
{


        if (bones[i].name.contains("Left third "
                + "metacarpal bone"))
        {
           M3 = bones[i].mean;
           E3 = bones[i].end;
           //Bug: S2 was there when S3 was needed
           S3 = bones[i].scaleFactors;
           System.out.println("Left third "
                + "metacarpal bone");
        }

}


C0 = (I0.add(E1)).multiply(0.5);
tz2 = (1-S0.getZ()) * ((C0.subtract(M0)).getZ())
            + (1-S1.getZ() )* ((M1.subtract(C0)).getZ()) ;
finger2Ph2Group.setTranslateZ(tz2);
finger2Ph2GroupPivot = new Point3D(C0.getX()
        ,C0.getY(), C0.getZ()-tz2);


C1 = (I1.add(E2)).multiply(0.5);
tz21 = (1-S1.getZ()) * ((C1.subtract(M1)).getZ())
            + (1-S2.getZ() )* ((M2.subtract(C1)).getZ()) ;
finger2Ph21Group.setTranslateZ(tz21);
finger2Ph21GroupPivot = new Point3D(C1.getX()
        ,C1.getY(), C1.getZ()-tz21);
```

```
C2 = (I2.add(E3)).multiply(0.5);
tz210 = (1-S2.getZ()) * ((C2.subtract(M2)).getZ())
            + (1-S3.getZ() )* ((M3.subtract(C2)).getZ()) ;
finger2Ph210Group.setTranslateZ(tz210);
finger2Ph210GroupPivot = new Point3D(C2.getX()
        ,C2.getY(), C2.getZ()-tz210);


//Bone groups of ring finger get reaccommodated
for(int i = 0; i < counterOfBones ; i++)
{
    if (bones[i].name.contains("ring"))
    {
        if (bones[i].name.contains("Distal"))
        {
           I0 = bones[i].start;
           M0 = bones[i].mean;
           S0 = bones[i].scaleFactors;
        }

        if (bones[i].name.contains("Middle"))
        {
           I1 = bones[i].start;
           M1 = bones[i].mean;
           E1 = bones[i].end;
           S1 = bones[i].scaleFactors;
        }

        if (bones[i].name.contains("Proximal"))
        {
           I2 = bones[i].start;
           M2 = bones[i].mean;
           E2 = bones[i].end;
           S2 = bones[i].scaleFactors;
        }


    }
```

```
}

for(int i = 0; i < counterOfBones ; i++)
{


    if (bones[i].name.contains("Left fourth "
            + "metacarpal bone"))
    {
       M3 = bones[i].mean;
       E3 = bones[i].end;
       //Bug: S2 was there when S3 was needed
       S3 = bones[i].scaleFactors;
       System.out.println("Left fourth "
            + "metacarpal bone");
    }

}


C0 = (I0.add(E1)).multiply(0.5);
tz2 = (1-S0.getZ()) * ((C0.subtract(M0)).getZ())
            + (1-S1.getZ() )* ((M1.subtract(C0)).getZ()) ;
finger1Ph2Group.setTranslateZ(tz2);
finger1Ph2GroupPivot = new Point3D(C0.getX()
        ,C0.getY(), C0.getZ()-tz2);


C1 = (I1.add(E2)).multiply(0.5);
tz21 = (1-S1.getZ()) * ((C1.subtract(M1)).getZ())
            + (1-S2.getZ() )* ((M2.subtract(C1)).getZ()) ;
finger1Ph21Group.setTranslateZ(tz21);
finger1Ph21GroupPivot = new Point3D(C1.getX()
        ,C1.getY(), C1.getZ()-tz21);


C2 = (I2.add(E3)).multiply(0.5);
tz210 = (1-S2.getZ()) * ((C2.subtract(M2)).getZ())
```

```
            + (1-S3.getZ() )* ((M3.subtract(C2)).getZ()) ;
finger1Ph210Group.setTranslateZ(tz210);
finger1Ph210GroupPivot = new Point3D(C2.getX()
        ,C2.getY(), C2.getZ()-tz210);


//Bone groups of little finger get reaccommodated
for(int i = 0; i < counterOfBones ; i++)
{
    if (bones[i].name.contains("little"))
    {
        if (bones[i].name.contains("Distal"))
        {
           I0 = bones[i].start;
           M0 = bones[i].mean;
           S0 = bones[i].scaleFactors;
        }

        if (bones[i].name.contains("Middle"))
        {
           I1 = bones[i].start;
           M1 = bones[i].mean;
           E1 = bones[i].end;
           S1 = bones[i].scaleFactors;
        }

        if (bones[i].name.contains("Proximal"))
        {
           I2 = bones[i].start;
           M2 = bones[i].mean;
           E2 = bones[i].end;
           S2 = bones[i].scaleFactors;
        }


    }
}
```

```
for(int i = 0; i < counterOfBones ; i++)
{


        if (bones[i].name.contains("Left fifth "
                + "metacarpal bone"))
        {
           M3 = bones[i].mean;
           E3 = bones[i].end;
           //Bug: S2 was there when S3 was needed
           S3 = bones[i].scaleFactors;
           System.out.println("Left fifth "
                + "metacarpal bone");

        }

}


C0 = (I0.add(E1)).multiply(0.5);
tz2 = (1-S0.getZ()) * ((C0.subtract(M0)).getZ())
            + (1-S1.getZ() )* ((M1.subtract(C0)).getZ()) ;
finger0Ph2Group.setTranslateZ(tz2);
finger0Ph2GroupPivot = new Point3D(C0.getX()
        ,C0.getY(), C0.getZ()-tz2);


C1 = (I1.add(E2)).multiply(0.5);
tz21 = (1-S1.getZ()) * ((C1.subtract(M1)).getZ())
            + (1-S2.getZ() )* ((M2.subtract(C1)).getZ()) ;
finger0Ph21Group.setTranslateZ(tz21);
finger0Ph21GroupPivot = new Point3D(C1.getX()
        ,C1.getY(), C1.getZ()-tz21);


C2 = (I2.add(E3)).multiply(0.5);
tz210 = (1-S2.getZ()) * ((C2.subtract(M2)).getZ())
```

```
            + (1-S3.getZ() )* ((M3.subtract(C2)).getZ()) ;
finger0Ph210Group.setTranslateZ(tz210);
finger0Ph210GroupPivot = new Point3D(C2.getX()
        ,C2.getY(), C2.getZ()-tz210);


//Bone groups of thumb finger get reaccommodated
for(int i = 0; i < counterOfBones ; i++)
{
    if (bones[i].name.contains("thumb"))
    {
        if (bones[i].name.contains("Distal"))
        {
           I0 = bones[i].start;
           M0 = bones[i].mean;
           S0 = bones[i].scaleFactors;
        }



        if (bones[i].name.contains("Proximal"))
        {
           I1 = bones[i].start;
           M1 = bones[i].mean;
           E1 = bones[i].end;
           S1 = bones[i].scaleFactors;
        }


    }
}

//The thumb goes on top of the first metacarpal
for(int i = 0; i < counterOfBones ; i++)
{
    if (bones[i].name.contains("Left first "
                + "metacarpal bone"))
        {
```

```
            I2 = bones[i].start;
            M2 = bones[i].mean;
            E2 = bones[i].end;
            S2 = bones[i].scaleFactors;
            System.out.println("Left first "
                 + "metacarpal bone");
         }
 }

 //The thumb + metacarpal spins over the trapazium
 for(int i = 0; i < counterOfBones ; i++)
 {
     if (bones[i].name.contains("trapezium"))
         {
            M3 = bones[i].mean;
            E3 = bones[i].end;
            S3 = bones[i].scaleFactors;
            System.out.println("trapezium");
         }
 }
```

```
C0 = (I0.add(E1)).multiply(0.5);
 tz2 = (1-S0.getZ()) * ((C0.subtract(M0)).getZ())
             + (1-S1.getZ() )* ((M1.subtract(C0)).getZ()) ;
 thumbPh2Group.setTranslateZ(tz2);
```

```
 C1 = (I1.add(E2)).multiply(0.5);
 tz21 = (1-S1.getZ()) * ((C1.subtract(M1)).getZ())
             + (1-S2.getZ() )* ((M2.subtract(C1)).getZ()) ;
 thumbPh21Group.setTranslateZ(tz21);
 thumbPh21GroupPivot = new Point3D(C1.getX()
         ,C1.getY(), C1.getZ()-tz21);
```

```
C2 = (I2.add(E3)).multiply(0.5);
tz210 = (1-S2.getZ()) * ((C2.subtract(M2)).getZ())
            + (1-S3.getZ() )* ((M3.subtract(C2)).getZ()) ;
thumbPh210Group.setTranslateZ(tz210);
thumbPh210GroupPivot = new Point3D(C2.getX()
        ,C2.getY(), C2.getZ()-tz210);



//========OPENING/CL0SING OF THE HAND===========

NumberBinding closeHandPropertyHalf
        = closeHandProperty.multiply(0.5);

//The slider that closes the hand extends its
//control to the palm:
//it must rotate around
//the end of the arm that is parallel to the X_AXIS
//This is the pivot of rotation
palmGroup.rwpx.pivotZProperty()
        .setValue(wristGroupPivot.getZ());
palmGroup.rwpx.pivotYProperty()
        .setValue(wristGroupPivot.getY());
//The angle of rotation is controlled by a slider
palmGroup.rwpx.angleProperty()
        .bind(closeHandPropertyHalf);



//Specifications of movement for subgroups of finger 0
finger0Ph2Group.rwpx.pivotZProperty()
        .setValue(finger0Ph2GroupPivot.getZ());
finger0Ph2Group.rwpx.pivotYProperty()
        .setValue(finger0Ph2GroupPivot.getY());
finger0Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);
```

```
finger0Ph21Group.rwpx.pivotZProperty()
        .setValue(finger0Ph21GroupPivot.getZ());
finger0Ph21Group.rwpx.pivotYProperty()
        .setValue(finger0Ph21GroupPivot.getY());
finger0Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger0Ph210Group.rwpx.pivotZProperty()
        .setValue(finger0Ph210GroupPivot.getZ());
finger0Ph210Group.rwpx.pivotYProperty()
        .setValue(finger0Ph210GroupPivot.getY());
finger0Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

//Specifications of movement for subgroups of finger 1
finger1Ph2Group.rwpx.pivotZProperty()
        .setValue(finger1Ph2GroupPivot.getZ());
finger1Ph2Group.rwpx.pivotYProperty()
        .setValue(finger1Ph2GroupPivot.getY());
finger1Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger1Ph21Group.rwpx.pivotZProperty()
        .setValue(finger1Ph21GroupPivot.getZ());
finger1Ph21Group.rwpx.pivotYProperty()
        .setValue(finger1Ph21GroupPivot.getY());
finger1Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger1Ph210Group.rwpx.pivotZProperty()
        .setValue(finger1Ph210GroupPivot.getZ());
finger1Ph210Group.rwpx.pivotYProperty()
        .setValue(finger1Ph210GroupPivot.getY());
finger1Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

//Specifications of movement for subgroups of finger 2
finger2Ph2Group.rwpx.pivotZProperty()
```

```
        .setValue(finger2Ph2GroupPivot.getZ());
finger2Ph2Group.rwpx.pivotYProperty()
        .setValue(finger2Ph2GroupPivot.getY());
finger2Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger2Ph21Group.rwpx.pivotZProperty()
        .setValue(finger2Ph21GroupPivot.getZ());
finger2Ph21Group.rwpx.pivotYProperty()
        .setValue(finger2Ph21GroupPivot.getY());
finger2Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger2Ph210Group.rwpx.pivotZProperty()
        .setValue(finger2Ph210GroupPivot.getZ());
finger2Ph210Group.rwpx.pivotYProperty()
        .setValue(finger2Ph210GroupPivot.getY());
finger2Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

//Specifications of movement for subgroups of finger 3
finger3Ph2Group.rwpx.pivotZProperty()
        .setValue(finger3Ph2GroupPivot.getZ());
finger3Ph2Group.rwpx.pivotYProperty()
        .setValue(finger3Ph2GroupPivot.getY());
finger3Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger3Ph21Group.rwpx.pivotZProperty()
        .setValue(finger3Ph21GroupPivot.getZ());
finger3Ph21Group.rwpx.pivotYProperty()
        .setValue(finger3Ph21GroupPivot.getY());
finger3Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger3Ph210Group.rwpx.pivotZProperty()
        .setValue(finger3Ph210GroupPivot.getZ());
finger3Ph210Group.rwpx.pivotYProperty()
```

```
        .setValue(finger3Ph210GroupPivot.getY());
finger3Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);



//======OPENING/CLOSING OF FINGERS========

//Fingers that are at the border must
//open twice faster as those
//near the center.
NumberBinding openFingersPropertyTwo
        = openFingersProperty.multiply(2);
//Specifications of movement for subgroups of finger 0
finger0Ph210Group.rwpy.pivotXProperty()
        .setValue(finger0Ph210GroupPivot.getX());
finger0Ph210Group.rwpy.pivotYProperty()
        .setValue(finger0Ph210GroupPivot.getY());
finger0Ph210Group.rwpy.pivotZProperty()
        .setValue(finger0Ph210GroupPivot.getZ());
finger0Ph210Group.rwpy.angleProperty()
        .bind(openFingersPropertyTwo);


//Specifications of movement for subgroups of finger 1
finger1Ph210Group.rwpy.pivotXProperty()
        .setValue(finger1Ph210GroupPivot.getX());
finger1Ph210Group.rwpy.pivotYProperty()
        .setValue(finger1Ph210GroupPivot.getY());
finger1Ph210Group.rwpy.pivotZProperty()
        .setValue(finger1Ph210GroupPivot.getZ());

finger1Ph210Group.rwpy.angleProperty()
        .bind(openFingersProperty);

//Finger two remains invariant under opening of fingers

//Opening in the contrary direction of fingers 3 and 4
```

```
NumberBinding openFingersPropertyMinus
        = openFingersProperty.multiply(-1);

//Specifications of movement for finger 3
finger3Ph210Group.rwpy.pivotXProperty()
        .setValue(finger3Ph210GroupPivot.getX());
finger3Ph210Group.rwpy.pivotYProperty()
        .setValue(finger3Ph210GroupPivot.getY());
finger3Ph210Group.rwpy.pivotZProperty()
        .setValue(finger3Ph210GroupPivot.getZ());
finger3Ph210Group.rwpy.angleProperty()
        .bind(openFingersPropertyMinus);

NumberBinding openFingersPropertyMinusTwo
        = openFingersProperty.multiply(-2);



//====== The thumb ========
//Specifications of movement for the thumb


NumberBinding rotateThumbPropertyDown
        = rotateThumbProperty.multiply(-0.4);

NumberBinding rotateThumbPropertyUp
        = rotateThumbProperty.multiply(-2);


NumberBinding closeThumbPropertyMinus
        = closeThumbProperty.multiply(-1);


//Traversal movement
thumbPh210Group.rwpy.pivotXProperty()
        .setValue(thumbPh210GroupPivot.getX());
thumbPh210Group.rwpy.pivotZProperty()
        .setValue(thumbPh210GroupPivot.getZ());
```

```
thumbPh210Group.rwpy.angleProperty()
        .bind(closeThumbProperty);

/*
//Rotations of phalanges 2 and 1 to make the grip.
//Rotations around X_AXIS and Y_AXIS at the same time
//Phalanx 2
thumbPh2Group.rwpx.pivotXProperty()
        .setValue(thumbPh2GroupPivot.getX());
thumbPh2Group.rwpx.pivotYProperty()
        .setValue(thumbPh2GroupPivot.getY());
thumbPh2Group.rwpx.pivotZProperty()
        .setValue(thumbPh2GroupPivot.getZ());
thumbPh2Group.rwpx.angleProperty()
        .bind(rotateThumbPropertyDown);

//Phalanges 2 and 1
thumbPh21Group.rwpx.pivotXProperty()
        .setValue(thumbPh21GroupPivot.getX());
thumbPh21Group.rwpx.pivotYProperty()
        .setValue(thumbPh21GroupPivot.getY());
thumbPh21Group.rwpx.pivotZProperty()
        .setValue(thumbPh21GroupPivot.getZ());
thumbPh21Group.rwpx.angleProperty()
        .bind(rotateThumbPropertyDown);

//Phalanx 2
thumbPh2Group.rwpy.pivotXProperty()
        .setValue(thumbPh2GroupPivot.getX());
thumbPh2Group.rwpy.pivotYProperty()
        .setValue(thumbPh2GroupPivot.getY());
thumbPh2Group.rwpy.pivotZProperty()
        .setValue(thumbPh2GroupPivot.getZ());
thumbPh2Group.rwpy.angleProperty()
        .bind(rotateThumbPropertyDown);
//Phalanges 2 and 1
thumbPh21Group.rwpy.pivotXProperty()
        .setValue(thumbPh21GroupPivot.getX());
```

```
thumbPh21Group.rwpy.pivotYProperty()
        .setValue(thumbPh21GroupPivot.getY());
thumbPh21Group.rwpy.pivotZProperty()
        .setValue(thumbPh21GroupPivot.getZ());
thumbPh21Group.rwpy.angleProperty()
        .bind(rotateThumbPropertyDown);
*/

//Rotation of the whole thumb
//to get opposite to other fingers
thumbPh210Group.rwpz.pivotXProperty()
        .setValue(thumbPh210GroupPivot.getX());
thumbPh210Group.rwpz.pivotYProperty()
        .setValue(thumbPh210GroupPivot.getY());
thumbPh210Group.rwpz.pivotZProperty()
        .setValue(thumbPh210GroupPivot.getZ());
thumbPh210Group.rwpz.angleProperty()
        .bind(rotateThumbProperty.multiply(-2));



//===========Animation=========

double endTime1 = 10000.0; ////1000 = 1 second
double endTime2 = 2 * endTime1;
double endTime3 = 3 * endTime1;
double endTime4 = 4 * endTime1;
double endTime5 = 5 * endTime1;
double endTime6 = 6 * endTime1;
double endTime7 = 7 * endTime1;
double endTime8 = 8 * endTime1;
double endTime9 = 9 * endTime1;




animation.getKeyFrames().addAll
(
```

```
//The hand gets closed
        //closeHandProperty: initial position at initial time
        new KeyFrame(Duration.ZERO,
            //Permanent condition
            new KeyValue(closeThumbProperty, 0)),
        //closeHandProperty: final position at final time
        new KeyFrame(new Duration(endTime1),
            new KeyValue(closeThumbProperty, -50)),

        //The hand gets closed
        //closeHandProperty: initial position at initial time
        new KeyFrame(new Duration(endTime1),
            //Permanent condition
            new KeyValue(closeHandProperty, 0)),
        //closeHandProperty: final position at final time
        new KeyFrame(new Duration(endTime2),
            new KeyValue(closeHandProperty, -90))
            ,


        //The thumb closes the grip
        //rotateThumbProperty: initial position at initial time
        new KeyFrame(new Duration(endTime2),
            //Permanent condition
            new KeyValue(rotateThumbProperty, 0)),
        //rotateThumbProperty: final position at final time
        new KeyFrame(new Duration(endTime3),
            new KeyValue(rotateThumbProperty, 37)),

        //The thumb gets reopened
        //rotateThumbProperty: initial position at initial time
        new KeyFrame(new Duration(endTime3),
            //Permanent condition
            new KeyValue(rotateThumbProperty, 37)),
        //rotateThumbProperty: final position at final time
        new KeyFrame(new Duration(endTime4),
            new KeyValue(rotateThumbProperty, 0)),
```

```
//The hand get reopened
//closeHandProperty: initial position at initial time
new KeyFrame(new Duration(endTime4),
    //Permanent condition
    new KeyValue(closeHandProperty, -90)),
//closeHandProperty: final position at final time
new KeyFrame(new Duration(endTime5),
    new KeyValue(closeHandProperty, 0)),

//Fingers get reopened
//openFingersProperty: initial position at initial time
new KeyFrame(new Duration(endTime5),
//Permanent condition
new KeyValue(openFingersProperty, 0)),
//openFingersProperty: final position at final time
new KeyFrame(new Duration(endTime6),
    new KeyValue(openFingersProperty, 15)),

//Fingers get closed
//openFingersProperty: initial position at initial time
new KeyFrame(new Duration(endTime6),
//Permanent condition
new KeyValue(openFingersProperty, 15)),
//openFingersProperty: final position at final time
new KeyFrame(new Duration(endTime7),
    new KeyValue(openFingersProperty, 0)),

//The thumb gets closed
//closeThumbProperty: initial position at initial time
new KeyFrame(new Duration(endTime7),
    //Permanent condition
    new KeyValue(closeThumbProperty, -55)),
//closeThumbProperty: final position at final time
new KeyFrame(new Duration(endTime8),
    new KeyValue(closeThumbProperty, 0)),

//The thumb gets reopen
```

```
        //closeThumbProperty: initial position at initial time
        new KeyFrame(new Duration(endTime8),
            //Permanent condition
            new KeyValue(closeThumbProperty, 0)),
        //closeThumbProperty: final position at final time
        new KeyFrame(new Duration(endTime9),
            new KeyValue(closeThumbProperty, -25))

    );




animation.play();

}

private static void printMatrix(double m[][], String title)
{
    System.out.println("This is the matrix " + title);
    for(int i=0;i<3;i++)
        System.out.println(m[i][0] + " " + m[i][1]+ " "
                                              + m[i][2]);
    System.out.println();
}


//The multiplication of a matrix times a vector
  private static  double[] matrixTimesVector(double m[][],
                                             double v[])
  {
    double[]  helpV = new double[3];
    for(int i=0; i<3; i++)
        helpV[i] =0;
    for(int i=0; i<3; i++)
        for(int j=0; j<3; j++)
            helpV[i]  =helpV[i]  + m[i][j]*v[j];
```

```
    return helpV;
  }


//Normalized crux product of vectors v and w
  private static  double dotProduct(double[] v,
                                    double[] w)
  {
    return v[0]*w[0] + v[1]*w[1] + v[2]*w[2];

  }


public void play()
{

}

@Override public void stop()
{

}

static class BoneDescription
{
    String name = "";
    Point3D start ;
    Point3D mean ;
    Point3D end ;
    String previous ;
    String next ;
    Point3D scaleFactors ;

    BoneDescription()
    {
        start = new Point3D(0,0,0);
        mean = new Point3D(0,0,0);
        end = new Point3D(0,0,0);
```

```
        previous = "";
        next = "";
        scaleFactors = new Point3D(1f,1f,1f);
    }
}


public static void main(String[] args)
{

    launch(args);
}


}//End of main class N47 Cylindricalmutations3
```

48, page 211. For your considertion, we present a development that is based on the following observation in regard with the organization of bones:

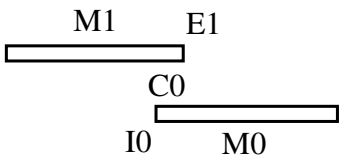


Figure 6.0. Bones are not blocks. Instead, they have cavities and protuberances that fit one another. That is why they look as overlapping. This conception is ensued by the following change: the pivot for bone 0 was I0 but now it is C0, the average of E1 and I0.

The corresponding code follows in which some bugs were corrected:

```
/*
    Program N48 Cylindricalmutations4
    This program shows how to accommodate
    the phalanges of the  fingers of the hand
    after  small cylindrical mutations.
    A cylindrical mutation over a cylinder
```

```
may enlarge its radius or its height.
PCA (Principal Component Analysis) allows to fit
the best cylinder over a bone,
which is then  mutated cylindrically.

Tested only with some  files,
particularly  with those of the BodyParts3D project.
http://lifesciencedb.jp/bp3d/
Those files are licensed:

The license for this database is specified in the
Creative Commons Attribution-Share Alike 2.1 Japan.
If you use data from this database, please be sure
attribute this database as follows: "BodyParts3D, (c)
The Database Center for Life Science licensed under
CC Attribution-Share Alike 2.1 Japan".
http://dbguidehive.biosciencedbc.jp/en/bodyparts3d/lic.html

The standard numeration of metacarpals
has been adopted for the fingers:
thumb = 1, index = 2, middle = 3, ring = 4, little = 5.
Numeration for phalanges and corresponding metacarpal:
0 = distal, 1 = middle, 2 = proximal, 4 = metacarpal.

Structure of graphic synthesis:
1. Files are read and interpreted.
    One file = one bone = one mesh.
2. The mesh is dressed in a meshView, boneMesh.
3. BoneMesh is located into an array, bonesViewArray.
4. Bones in BonesViewArray are distributed among
    the natural groups of the hand, say fingers,
    that are all united into the palmGroup.
5. PalmGroup is added to an intermediary item, handXform
6. HandXform is added to a global item, world3DXform.
7. World3DXform is added to a pane, paneHand.
8. PaneHand is added to root, which is a JavaFX group.
9. Root is added to the Scene,
    which is the final container.
```

```
 */


package Programs;

//Files in the hosting project Programs
import Jama.EigenvalueDecomposition;
import Jama.Matrix;
import Tools.Axes;
import Tools.Cameras;
import Tools.XformWithPivot;

//Java files
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.List;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.binding.NumberBinding;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.event.ActionEvent;
import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
```

```
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.TextArea;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.MeshView;
import javafx.scene.shape.Sphere;
import javafx.scene.shape.TriangleMesh;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import javafx.util.Duration;



public class CylindricalMutations4 extends Application
{
    //Visibility of the mesh
    //might depend on the correct choice
    //of this factor.
    private static final float meshScaleFactor = 3f;

    //This parameter regulates the size of random
    //cylindrycal mutations.
    private static final float epsilon = 0.5f;
    //Scale factors for mutation along width, height and length
    static private float scaleFactorW = 1;
    static private float scaleFactorH = 1;
    static private float scaleFactorL = 1;

    //Printing for debugging
    final static boolean printReadFile = false;
    private static final String title
            =  "Importer of meshes in format .obj";
```

```
//Tool to read files
BufferedReader br ;
private static  String textOfFile = "";

private static int counterOfFaces = 0;
private static int counterOfVertexes = 0;
private static int counterOfNormals = 0;

private static int[] destFaces ;


private static final MeshView[] bonesViewArray
        = new MeshView[100];
private static final String[] names1  = new String[100];
private static final String[] names2  = new String[100];
//Group of Xforms
//??????private static final Xform bonesXforms  = new Xform(
//Counter
private static int counterOfBones = 0;

//*****************

final static Group root = new Group();

final static Group axisGroup = new Group();


final static XformWithPivot cameraXform = new XformWithPivot
final static XformWithPivot cameraXform2 = new XformWithPivo
final static XformWithPivot cameraXform3 = new XformWithPivo

final static double cameraDistance = 2450; //1450;//450
final static XformWithPivot world3DXform = new XformWithPivo
final static XformWithPivot handXform = new XformWithPivot()

//Axes : ctrl + A
private static final Axes axes = new Axes();
private static final Cameras camera = new Cameras();
```

```
   //Menubar declaration and initialization
    MenuBar menuBar = new MenuBar();


private static final Pane paneHand = new Pane();


    //To close the hand,
    //each phalange suffers a rotation.
    //It is a double property because
    //it will get involved into a binding
    private  static final DoubleProperty closeHandProperty
            = new SimpleDoubleProperty(0);


    //To open the fingers,
    //each phalange suffers a transversal rotation.
    //It is a double property because
    //it will get involved into a binding
    private  static final DoubleProperty openFingersProperty
            = new SimpleDoubleProperty(0);


    //To close the thumb we need a rotation.
    //It is a double property because
    //it will get involved into a binding
    private  static final DoubleProperty closeThumbProperty
            = new SimpleDoubleProperty(0);


    //To form the grip we need a rotation of the thumb.
    //It is a double property because
    //it will get involved into a binding
    private  static final DoubleProperty rotateThumbProperty
            = new SimpleDoubleProperty(0);
```

```
//Group with finger 1 (thumb)
//and its corresponding metacarpal
private static final XformWithPivot metaFinger1Group
        = new XformWithPivot();

//Finger2 = index
//Group with the extreme phalanx of finger0
private static final XformWithPivot finger2Ph0Group
        = new XformWithPivot();
//Group with the two extreme phalanges
private static final XformWithPivot finger2Ph01Group
        = new XformWithPivot();
//Group with all three phalanges. It is the finger 2
private static final XformWithPivot finger2Ph012Group
        = new XformWithPivot();
//Group with finger and its corresponding metacarpal
private static final XformWithPivot metaFinger2Group
        = new XformWithPivot();
//Finger3
private static final XformWithPivot finger3Ph0Group
        = new XformWithPivot();
private static final XformWithPivot finger3Ph01Group
        = new XformWithPivot();
private static final XformWithPivot finger3Ph012Group
        = new XformWithPivot();
//Group with finger and its corresponding metacarpal
private static final XformWithPivot metaFinger3Group
        = new XformWithPivot();
//Finger4
private static final XformWithPivot finger4Ph0Group
        = new XformWithPivot();
private static final XformWithPivot finger4Ph01Group
        = new XformWithPivot();
private static final XformWithPivot finger4Ph012Group
        = new XformWithPivot();
```

```
//Group with finger and its corresponding metacarpal
private static final XformWithPivot metaFinger4Group
        = new XformWithPivot();
//Finger5
private static final XformWithPivot finger5Ph0Group
        = new XformWithPivot();
private static final XformWithPivot finger5Ph01Group
        = new XformWithPivot();
private static final XformWithPivot finger5Ph012Group
        = new XformWithPivot();
//Group with finger and its corresponding metacarpal
private static final XformWithPivot metaFinger5Group
        = new XformWithPivot();
//Thumb
private static final XformWithPivot thumbPh0Group
        = new XformWithPivot();
private static final XformWithPivot thumbPh01Group
        = new XformWithPivot();


 private static final XformWithPivot wristGroup
        = new XformWithPivot();
//Group with wrist and fingers
private static final XformWithPivot palmGroup
        = new XformWithPivot();

private static final Timeline animation
        = new Timeline();

private static final double[][] coordBoxes
        = new double[100][6];

private static final BoneDescription[] bones
        = new BoneDescription[30];

private static Point3D MeanOfHand
        = new Point3D(0,0,0);
```

```
    Random r = new Random();

@Override
public void start(Stage primaryStage)
{
    System.out.println("start");


    Scene scene = new Scene(root, 1200, 1000, true);

    paneHand.setPrefSize(500, 500);


    /*
       Multiple file loader
    */

    //Content of files will be shown in textArea
    final TextArea textArea = new TextArea();
    textArea.prefWidth(400);
    textArea.setWrapText(true);
    textArea.setText("");
    //Container  for textArea.
    ScrollPane scrollPane = new ScrollPane();
    scrollPane.getStyleClass().add("noborder-scroll-pane");
    scrollPane.setContent(textArea);
    scrollPane.setFitToWidth(true);
    scrollPane.setPrefWidth(400);
    scrollPane.setPrefHeight(180);
    VBox vBox = new VBox();
    vBox.visibleProperty().setValue(false);
    vBox.getChildren().addAll(scrollPane);
    root.getChildren().addAll(vBox);


    Sphere s = new Sphere(20);
    //root.getChildren().add(s);
```

```
// --- Menu files
Menu filesMenu = new Menu("Files");
//Menu to load multiple files
MenuItem loadFiles
        = new MenuItem("Load + process multiple files ");
//MenuItem added to menu
filesMenu.getItems().addAll(loadFiles);
menuBar.getMenus().add(filesMenu);
root.getChildren().add(menuBar);

//Listener
loadFiles.setOnAction((ActionEvent t) ->
{
    FileChooser fileChooser = new FileChooser();

    //Name of file with extension "
    FileChooser.ExtensionFilter extFilter
            = new FileChooser
                    .ExtensionFilter("TXT files (*.obj)"
                            , "*.obj");
    fileChooser.getExtensionFilters().add(extFilter);

    //Files will be kept in a list
    List<File> files = new FileChooser()
            .showOpenMultipleDialog(primaryStage);



        //Delete possible repetitions
        //files = deleteRepetitions(files);
    //counterOfBones = 0;



    for(int j = 0; j < files.size(); j++)
    {
        File file = files.get(j);
```

```
            String name = file.getName();
            names1[j] = name;
        }
        counterOfBones = 0;
        //Each file is processed
        for(int j = 0; j < files.size(); j++)
        {

            String name = names1[j];
            boolean repeated = false;
            for (int i = 0; i <  j; i++)
            {
                if (name.equals(names1[i]))
                {
                    repeated = true;

                    //System.out.println("Repeated" + name);
                }
            }

            if (repeated);
            else
            try
            {
                File file = files.get(j);
                //Text of file is read and mesh is extracted
                textOfFile = readFile(file);
                //Text copied to textArea
                textArea.appendText(textOfFile);

            } catch (IOException ex)
                {
                    Logger.getLogger(AnimatedHandSkeleton
                            .class.getName())
                            .log(Level.SEVERE, null, ex);
                }
```

```
    }

    defineGroups();



//To remedy a not understood bug
root.getChildren().remove(menuBar);


vBox.visibleProperty().set(true);
});

//====== End of File loader

primaryStage.setTitle(title
        + ". Drag the mouse to rotate. "
    + " Ctrl + A = Axis visibility"
        );
scene.setFill(Color.BLUE);
primaryStage.setScene(scene);
primaryStage.show();



//Hide textArea and show graphic
Button hideTextArea = new Button();
hideTextArea.setText("View graphic");
//Listener
hideTextArea.setOnAction((ActionEvent t) ->
{
    vBox.visibleProperty().set(false);
    //Clean the scene
    root.getChildren().removeAll();
    //Axes
    axes.buildAxes();
```

```
        axes.axesGroup.setVisible(true);
        axes.handleKeyboard(scene);

        //Camera
        camera.handleMouse(paneHand);
        camera.cameraDistance = cameraDistance;
        camera.buildCamera();
        //The hand is added to the world3D
        world3DXform.getChildren().addAll(axes.axesXform, handXf


        //Booking
        paneHand.getChildren().add(world3DXform);
        root.getChildren().add(paneHand);
        scene.setCamera(camera.camera);
    });

    vBox.getChildren().addAll(hideTextArea);


    primaryStage.show();
}




//File is read and its content is returned as a string.
//The associated mesh is recovered
//and correct orientation of faces is provided.
//Our correspondence is the following:
//one bone <-> one mesh <-> one file.
//PCA is used to implement cylindrical mutations.
private  String readFile(File file)
        throws FileNotFoundException, IOException
{
    //Name of file is printed
    String name = file.getName();
```

```
    //if (printReadFile)
        System.out.println(name);
    names2[counterOfBones] = name;
    BoneDescription b = new BoneDescription();
    b.name = name;
    bones[counterOfBones] = b;

    //Buffer to keep the content of file
    //StringBuilder stringBuffer = new StringBuilder();
    StringBuilder stringBuffer  = new StringBuilder();
    //Tool to read file
    BufferedReader bufferedReader = null;

    //A mesh is declared and instantiated
    TriangleMesh mesh  = new TriangleMesh();
    //The mesh has no predefined texture
    mesh.getTexCoords().addAll(0,0);
    //In the .obj format, counting begins with 1
    //but in Java, we begin with 0.
    //So, an additional inocuos point is added.
    mesh.getPoints().addAll(0,0,0);

    if (printReadFile)
        System.out.println("File content: ");
//File is read
try
{
//
bufferedReader = new BufferedReader(new FileReader(file));

String text;



while( (text = bufferedReader.readLine()) != null)
{
    //Text for the textArea
```

```
stringBuffer.append(text);
//Print each line
//System.out.println(text);

//Vertexes are read
if (((text.length() > 0 ) && text.charAt(0) == 'v')
        && text.charAt(1) == ' ')
{
    if (printReadFile)
        System.out.println("Vertex  = "
                + counterOfVertexes + " = "+text);
    counterOfVertexes++;
    text = text.substring(2);
    int j = 0;
    //Three vertexes are read
    for(int i = 0; i < 3; i++)
    {
        double d ;
        String s = "";
        Boolean go = true;

        while (go)
        {
            char thisChar = text.charAt(j);
            s = s + thisChar;
            char nextChar = ' ';
            if (++j < text.length())
             nextChar = text.charAt(j);
            if ((nextChar == ' ') ||
                    (j >= text.length()))
                    go = false;

        }
        //string to double
        d = Double.parseDouble(s);
        float df = (float) d;
        if (printReadFile)
            System.out.println("Float number = " + df);
```

```
        mesh.getPoints().addAll(df);

    }
}


//Normal vectors are counted and displayed
if ((text.length() > 0 ) && (text.charAt(0) == 'v')
        && (text.charAt(1) == 'n'))
{

    if (printReadFile)
        System.out.println("Normal " + counterOfNormals
            + " = " + text);
    counterOfNormals++;
}


//Face is read, one face per line.
//A face has 3 or more vertexes. These are extracted.
//Additional information is ignored
if ( (text.length() > 0 ) && (text.charAt(0) == 'f'))
{

    if (printReadFile)
        System.out.println("Face  = " + counterOfFaces
            + " = "+text);
    counterOfFaces++;
    text = text.substring(2);

    int vertex[] = new int[4];
    int j = 0;

    Boolean isNotBlank ;
    //Three vertexes are extracted
    for(int i = 0; i < 3; i++)
    {
```

```
double d ;
String s = "";
Boolean go1 = true;


while (go1)
{
    if (j >= text.length()-2)
        break;

    char thisChar = text.charAt(j);
    s = s + thisChar;
    /*
    System.out.println(" j = "
            + j + "    s = " + s);
            */
    char nextChar = ' ';
    j++;
    if (j < text.length())
     nextChar = text.charAt(j);
    char nextNextChar;
    isNotBlank = true;
    if (nextChar == '/' )
    {
        go1 = false;
        //go until the following blank
        while ((isNotBlank)
                && (j < text.length()-1))
        {
            j++;
            nextNextChar = text.charAt(j);
            if (nextNextChar == ' ' )
                isNotBlank = false;

        }
    }

}
```

```
    d = Double.parseDouble(s);
    int v = (int) d;
    vertex[i] = v;
    if (printReadFile)
        System.out.println("Int number = " + v);
}//End of cycle for 3 vertexes

//A face with 3 vertexes is added to mesh
mesh.getFaces().addAll(vertex[0], 0, vertex[1], 0,
    vertex[2], 0);

if (printReadFile)
{
    for(int i = 0; i < 3; i++)
     System.out.print(vertex[i] + " . ");
    System.out.println();
}

//Is there a four vertex in the face?
if   ( (text.length() - j )> 3)
{
    String s = "";
    Boolean go1 = true;
    while (go1)
    {


        char thisChar = text.charAt(j);
        s = s + thisChar;
        j++;
        char nextChar = ' ';
        if (j < text.length())
         nextChar = text.charAt(j);
        if (nextChar == '/' ) go1 = false;
    }
    if (printReadFile)
        System.out.println("4th = " + s);
```

```
            double d = Double.parseDouble(s);
            int v = (int) d;
            vertex[3] = v;

            //A polyface with 4 vertex, say, a,b,c,d,
            //can be covered by 2 triangles:
            //Triangle 1: a,b,c (Already added to mesh)
            //and triangle 2: a,c,d.
            //Triangle 2 with 3 vertexes is added to mesh
            mesh.getFaces().addAll(vertex[0], 0, vertex[2],
                    0, vertex[3], 0);
        }
    }//End of face cycle

    //Bounds of containing box are read and displayed
    if ( (text.length() > 0 )
            && (text.contains("# Bounds(mm):")))
    {
        //System.out.println(text);
        System.out.print("Bounds =   ");
        String a = "# Bounds(mm): (";
        int m = a.length();
        text = text.substring(m);
        text = text.replace(")-(",   ",   ");
        text = text.replace(")",   ",   ");
        System.out.println(text);

        int j = 0;


        //Six vertexes are extracted
        for(int i = 0; i < 6; i++)
        {

            double d ;
            String s = "";
            Boolean go1 = true;
```

```
            while (go1)
            {
                if (j >= text.length()-2)
                    break;

                char thisChar = text.charAt(j);
                s = s + thisChar;
                /*
                System.out.println(" j = "
                        + j + "    s = " + s);
                */
                char nextChar = ' ';
                j++;
                if (j < text.length())
                 nextChar = text.charAt(j);
                if (nextChar == ',' )
                {
                    go1 = false;
                   if ( j < text.length() ) j++;
                }

            }

            d = Double.parseDouble(s);
            //Vertex i of the box that
            //contains bone with number counterOfBones
            coordBoxes[counterOfBones][i] = d;
           if (printReadFile)
                System.out.println("Coordinate = "
                        + i + "= " + d);
        }//End of cycle for 6 vertexes
    }

}

} //End of  try.
    //Exception management
```

```
catch (FileNotFoundException ex)
{
    Logger.getLogger(AnimatedHandSkeleton
    .class.getName()).log(Level.SEVERE, null, ex);
}
catch (IOException ex)
{
    Logger.getLogger(AnimatedHandSkeleton
    .class.getName()).log(Level.SEVERE, null, ex);
}
finally
{
    if (bufferedReader != null)  bufferedReader.close();

}



if (printReadFile)
{
    System.out.println("End of file content. ");
    //Other indicators are reported
    report(mesh);
}


//======MeshWork=======

PCA(mesh);

MeshView boneMesh =
        new MeshView(mesh);

PhongMaterial goldenMaterial
        = new PhongMaterial(Color.GOLDENROD);
boneMesh.setMaterial(goldenMaterial);
bonesViewArray[counterOfBones] = boneMesh;
counterOfBones++;
```

```
    return stringBuffer.toString();
}


//Indicators of retrieved data are printed
private void report(TriangleMesh mesh)
{
    float[] destPoints = new float[mesh.getPoints().size()];
    mesh.getPoints().toArray(destPoints);
    System.out.println("POINTS");
    for(int j = 0; j <  destPoints.length/3; j++)
    {
        System.out.print(j + "   " );
        for(int k = 0; k < 3; k++)
        {
            System.out.print( destPoints[j*3 + k] + "   " );
        }
        System.out.println();
    }

    destFaces = new int[mesh.getFaces().size()];
    //Faces go to a matrix
    mesh.getFaces().toArray(destFaces);
    System.out.println("FACES WITH TEXTURE COORDINATES");
    for(int j = 0; j <  destFaces.length/6; j++)
    {
        System.out.print(j + "   " );
        for(int k = 0; k < 6; k++)
        {
            System.out.print( destFaces[j*6 + k] + "   " );
        }
        System.out.println();
    }

    System.out.println("counterOfPoints = "
            + counterOfVertexes );
    System.out.println("counterOfFaces = "
```

```
            + counterOfFaces );
    System.out.println("counterOfNormals = "
            + counterOfNormals);
    System.out.println("mesh.getPoints().size() = "
            + mesh.getPoints().size() );
    System.out.println("mesh.getFaces().size() = "
            + mesh.getFaces().size() );

    System.out.println("getPointElementSize = "
            + mesh.getPointElementSize() );
    System.out.println("getfaceElementSize = "
            + mesh.getFaceElementSize() );

}



public static String fixedWidthDoubletoString
        (double x, int w, int d) {
   java.text.DecimalFormat fmt
           = new java.text.DecimalFormat();
   fmt.setMaximumFractionDigits(d);
   fmt.setMinimumFractionDigits(d);
   fmt.setGroupingUsed(false);
   String s = fmt.format(x);
   while (s.length() < w) {
      s = " " + s;
   }
   return s;
}

//Principal component analysis of a mesh
private void PCA(TriangleMesh mesh)
{
    //Point are declared as a matrix of coordinates
    int n = mesh.getPoints().size();
        float nPoints = n;
    nPoints = nPoints/3;
```

```
float[][] CoordinatesFloat = new float[(int) nPoints][3];


/*
//Test: adapted from
//Stat Trek: Variance-Covariance Matrix
//http://stattrek.com/matrix-algebra/covariance-matrix.aspx

float[] data = {90,  60,  90,
                90,  90,  30,
                60,  60,  60,
                60,  60,  90,
                30,  30,  30 };


    n = 15;
    nPoints = 5;
int counter = 0;
for(int i = 0; i < n; i = i+3)
{
    //Reading
    float x =  data[i];
    float y =  data[i+1] ;
    float z =  data[i+2] ;
    CoordinatesFloat[counter][0] = x;
    CoordinatesFloat[counter][1] = y;
    CoordinatesFloat[counter][2] = z;
    counter++;
}

//Mean is found
float[] mean = new float[3];

for(int i = 0; i < 3; i++)
{
    mean[i] = 0;
    for (int j = 0; j <  nPoints; j++)
        mean[i] = mean[i] + CoordinatesFloat[j][i];
```

```
    mean[i] = mean[i]/(nPoints);
}

System.out.println("meanX = " + mean[0]);
System.out.println("meanY = " + mean[1]);
System.out.println("meanZ = " + mean[2]);



//Calculate covariance matrix
double[][] Cov = new double[3][3];
for(int i = 0; i < 3; i++)
{
    for (int j = 0; j <  3; j++)
    {
        Cov[i][j] = 0;
        for (int k = 0; k <  nPoints; k++)
            Cov[i][j] = Cov[i][j] +
                    (CoordinatesFloat[k][i] - mean[i])
                    * (CoordinatesFloat[k][j] - mean[j]);

        Cov[i][j] = Cov[i][j]/nPoints;

    }

}

printMatrix(Cov, "Cov");

//Answer: cov matrix must be:
//504  360  180
//360  360  0
//180  0  720
*/

//=============
```

```
int counter = 0;
for(int i = 0; i < n; i = i+3)
{
    //Reading
    float x =  mesh.getPoints().get(i);
    float y =  mesh.getPoints().get(i+1);
    float z =  mesh.getPoints().get(i+2);
    CoordinatesFloat[counter][0] = x;
    CoordinatesFloat[counter][1] = y;
    CoordinatesFloat[counter][2] = z;
    counter++;
    //Warning: the zeroth point is artificial,
    //filled with zeros.
    if (i == 0)
        System.out.println("Zeroth point: x = " + x
                + "   y =   " + y + "   z =  "  + z);
}

//Mean is found. It is vector with 3 coordinates
float[] mean = new float[3];

for(int i = 0; i < 3; i++)
{
    mean[i] = 0;
    //Zeroth point does not count
    for (int j = 1; j <  nPoints; j++)
        mean[i] = mean[i] + CoordinatesFloat[j][i];
    mean[i] = mean[i]/(nPoints -1);
}

System.out.println("meanX = " + mean[0]);
System.out.println("meanY = " + mean[1]);
System.out.println("meanZ = " + mean[2]);
Point3D meanPoint = new Point3D(mean[0], mean[1], mean[2]);
bones[counterOfBones].mean = meanPoint;
```

```
//Calculate covariance matrix
double[][] Cov = new double[3][3];
for(int i = 0; i < 3; i++)
{
    for (int j = 0; j <  3; j++)
    {
        Cov[i][j] = 0;
        for (int k = 1; k <  nPoints; k++)
            Cov[i][j] = Cov[i][j] +
                    (CoordinatesFloat[k][i] - mean[i])
                    * (CoordinatesFloat[k][j] - mean[j]);

        Cov[i][j] = Cov[i][j]/(nPoints-1);

    }

}

printMatrix(Cov, "Cov");

//=========

//Eigenvalues of Cov matrix are found
Matrix M = new Matrix(Cov);
EigenvalueDecomposition E =
     new EigenvalueDecomposition(M);
double[] d = E.getRealEigenvalues();

for(int i = 0; i < 3; i++)
    System.out.println("Eigenvalue " + i + " : " + d[i]);
double[][] EV = E.getV().getArray();
for(int i = 0; i < 3; i++)
{
    System.out.println("Eigenvector " + i + " : " );
    for (int j = 0; j <  3; j++)
    {
        System.out.println(EV[i][j]);
    }
```

```
    System.out.println();
}

//Test: eigenvectors of a real symmetric matrix
//are orthogonal one to another: dot product must be zero
System.out.println("Orthogonality test "
        + ": matrix I is expected");
double[][] Ort = new double[3][3];
for(int i = 0; i < 3; i++)
{

    for (int j = 0; j <  3; j++)
    {

        //sum = sum + EV[i][j]*EV[i+1][j];
        Ort[i][j] = dotProduct(EV[i], EV[j]);

    }
    //System.out.println(sum);

}

printMatrix(Ort, "Ort test");

//Determinant must be 1.
System.out.println("Orientation test: "
        + "+1 is expected. Det E = "
        + E.getV().det() + " \n");


 //Mesh is centered
float [] pointsFloat = new float[n];
for(int i = 0; i < n; i = i+3)
{
    //Reading
    float x =  mesh.getPoints().get(i);
    float y =  mesh.getPoints().get(i+1);
    float z =  mesh.getPoints().get(i+2);
```

```
    //Mesh is centered
    pointsFloat[i]    =  (x - mean[0]);
    pointsFloat[i+1]  =  (y - mean[1]);
    pointsFloat[i+2]  =  (z - mean[2]);
}
//Resetting of points of mesh
mesh.getPoints().setAll(pointsFloat, 0, n);


//=============CYLINDRICAL MUTATION=============

//Scale factors for cylindrical mutations are composed:
//scaleFactorW, scaleFactorH, scaleFactorL.
Point3D P =  cylindricalMutationFactors();
//Memorization
bones[counterOfBones].scaleFactors
        = P.multiply(meshScaleFactor);


//Scale factors as a diagonal matrix
double[][] scaleMatrix
        = { {scaleFactorW*meshScaleFactor, 0, 0},
            {0, scaleFactorH*meshScaleFactor, 0},
            {0, 0, scaleFactorL*meshScaleFactor}};



//1D array for recording
float[]newPoints = new float[n];
//Mesh is aligned with the new basis
//given by the PCA eigenvetors
for(int i = 0; i < n; i = i+3)
{
     //Reading
    float x =  mesh.getPoints().get(i);
    float y =  mesh.getPoints().get(i+1);
    float z =  mesh.getPoints().get(i+2);
```

```
        //Three coordinates define a point
        double[] w = {x,y,z};
        //Coordinates of point w are transformed
        //to the new basis
        w = matrixTimesVector(E.getV()
                .transpose().getArray(),w);
        //PCA scaling is applied
        w = matrixTimesVector(scaleMatrix, w);
        //Coordinates of point w are transformed
        //to the old, natural basis
        w = matrixTimesVector(E.getV().getArray(),w);

        //Initial center is restored.
        //Middle point is added to the  1-dimensional array

        newPoints[i ] = (float) ( w[0] + mean[0]);
        newPoints[i + 1] = (float) ( w[1]  + mean[1] );
        newPoints[i + 2] = (float) ( w[2] + mean[2]);


    }
    //Points of mesh are  reset
    mesh.getPoints().setAll(newPoints, 0, n);

}


//Returns a random float number
//uniformly distributed over an interval circa 1
private float randomFactor(float epsilon)
{
    float s, t;
    //Interval = (1, 1 + epsilon), magnifying.
    //s= epsilon * r.nextFloat();
    //t = 1 + s;

    //Interval = (1-epsilon, 1 + epsilon),
    //magnifying and shrinking
```

```
    s = 2 * epsilon * r.nextFloat();
    t = 1- epsilon + s;
    return t;
}


//Return random values for scale factors.
//Factors can be greater than or less than 1
//Tests are included.
private Point3D cylindricalMutationFactors()
{
    scaleFactorW = randomFactor(epsilon);
    scaleFactorH =  scaleFactorW;
    scaleFactorL = randomFactor(epsilon);

    //Test: uniform shrinking
     /*
    scaleFactorW = 0.5f;
    scaleFactorH = 0.5f;
    scaleFactorL = 0.5f;
    */
    //Test: homogeneous enlargement
    /*
    scaleFactorW = 3f;
    scaleFactorH = 3f;
    scaleFactorL = 3f;
    */
    return new Point3D(scaleFactorW, scaleFactorH,
            scaleFactorL);

}


//Returns the pivot of bone i:
//the middle x-value, the middle y-value and
//the minimal z-value
private static Point3D pivot(int i)
{
```

```
    double x = (coordBoxes[i][0] + coordBoxes[i][3])/2;
    double y = (coordBoxes[i][1] + coordBoxes[i][4])/2;
    double z = coordBoxes[i][5];
    Point3D p = new Point3D(x,y,z);
    return p;
}

////Returns the initial point of bone i:
//the middle x-value, the middle y-value and
//the minimal z-value
private static Point3D initial(int i)
{
    double x = (coordBoxes[i][0] + coordBoxes[i][3])/2;
    double y = (coordBoxes[i][1] + coordBoxes[i][4])/2;
    double z = coordBoxes[i][5];
    Point3D p = new Point3D(x,y,z);
    return p;
}

////Returns the final point of bone i:
//the middle x-value, the middle y-value and
//the maximal z-value
private static Point3D end(int i)
{
    double x = (coordBoxes[i][0] + coordBoxes[i][3])/2;
    double y = (coordBoxes[i][1] + coordBoxes[i][4])/2;
    double z = coordBoxes[i][2];
    Point3D p = new Point3D(x,y,z);
    return p;
}

////Returns the middle point of bone i:
//the average point of its initial and final points.
private static Point3D mean(int i)
{

    Point3D p = initial(i).midpoint(end(i));
    return p;
```

```
}


//Bones move in groups. Example:
//distal phalange must be moved
//when the middle one moves.
private static void    defineGroups()
{
    //PIVOTS OF GROUPS
    Point3D finger1Ph0GroupPivot;
    Point3D finger1Ph01GroupPivot;
    Point3D finger1Ph012GroupPivot;
    Point3D finger2Ph0GroupPivot ;
    Point3D finger2Ph01GroupPivot ;
    Point3D finger2Ph012GroupPivot ;
    Point3D finger3Ph0GroupPivot;
    Point3D finger3Ph01GroupPivot;
    Point3D finger3Ph012GroupPivot;
    Point3D finger4Ph0GroupPivot;
    Point3D finger4Ph01GroupPivot;
    Point3D finger4Ph012GroupPivot;
    Point3D finger5Ph0GroupPivot;
    Point3D finger5Ph01GroupPivot;
    Point3D finger5Ph012GroupPivot;
    Point3D thumbPh0GroupPivot = new Point3D(0,0,0);
    Point3D thumbPh01GroupPivot = new Point3D(0,0,0);
    Point3D metaFinger1GroupPivot;
    Point3D wristGroupPivot = new Point3D(0,0,0);



    //NATURAL GROUPS ARE INSTANTIATED

    for(int i = 0; i < counterOfBones; i++)
    {
        String name = names2[i];
        System.out.println("i = " + i + "  " +name);
```

```
//Finger 5 = little finger, and its subgroups
if (name.contains("little"))
{
    if (name.contains("Distal"))
    {
        finger5Ph0Group
            .getChildren().add(bonesViewArray[i]);
        bones[i].start = initial(i);
        bones[i].end = end(i);
        bones[i].mean = mean(i);
        bones[i].previous = "Middle phalanx of"
                + " left little finger";
        bones[i].next = "";

    }
    if (name.contains("Middle"))
    {
        finger5Ph01Group
            .getChildren()
            .addAll(finger5Ph0Group
                    , bonesViewArray[i]);
        bones[i].start = initial(i);
        bones[i].end = end(i);
        bones[i].mean = mean(i);
        bones[i].previous = "Proximal phalanx "
                + "of left little finger";
        bones[i].next = "Distal phalanx of "
                + "left little finger";

    }
    if (name.contains("Proximal"))
    {
        finger5Ph012Group
            .getChildren()
            .addAll(finger5Ph01Group
                    , bonesViewArray[i]);
```

```
                bones[i].start = initial(i);
                bones[i].end = end(i);
                bones[i].mean = mean(i);
                bones[i].previous = "Left first "
                        + "metacarpal bone";
                bones[i].next = "Middle phalanx "
                        + "of left little finger";
            }
        }

        //Finger 4 = ring and its subgroups
        else if (name.contains("ring"))
        {
            if (name.contains("Distal"))
            {
                finger4Ph0Group
                    .getChildren().add(bonesViewArray[i]);
                bones[i].start = initial(i);
                bones[i].end = end(i);
                bones[i].mean = mean(i);
                bones[i].previous = "Middle phalanx "
                        + "of left ring finger";
                bones[i].next = "";
            }
            if (name.contains("Middle"))
            {
                finger4Ph01Group
                    .getChildren()
                    .addAll(finger4Ph0Group, bonesViewArray[i]);
                bones[i].start = initial(i);
                bones[i].end = end(i);
                bones[i].mean = mean(i);
                bones[i].previous = "Proximal phalanx "
                        + "of left ring finger";
                bones[i].next = "Distal phalanx "
                        + "of left ring finger";
            }
            if (name.contains("Proximal"))
```

```
    {
        finger4Ph012Group
            .getChildren()
            .addAll(finger4Ph01Group, bonesViewArray[i]);
        bones[i].start = initial(i);
        bones[i].end = end(i);
        bones[i].mean = mean(i);
        bones[i].previous = "Left second "
                + "metacarpal bone";
        bones[i].next = "Middle phalanx "
                + "of left ring finger";
    }
}

//Finger 3 = middle and its subgroups
else if (name.contains("middle"))
{
    if (name.contains("Distal"))
    {
        finger3Ph0Group
        .getChildren().add(bonesViewArray[i]);
        bones[i].start = initial(i);
        bones[i].end = end(i);
        bones[i].mean = mean(i);
        bones[i].previous = "Middle phalanx "
                + "of left middle finger";
        bones[i].next = "";
    }
    if (name.contains("Middle"))
    {
        finger3Ph01Group
            .getChildren()
            .addAll(finger3Ph0Group, bonesViewArray[i]);
        bones[i].start = initial(i);
        bones[i].end = end(i);
        bones[i].mean = mean(i);
        bones[i].previous = "Proximal phalanx "
                + "of left middle finger";
```

```
                bones[i].next = "Distal phalanx "
                        + "of left middle finger";
            }
            if (name.contains("Proximal"))
            {
                finger3Ph012Group
                    .getChildren()
                    .addAll(finger3Ph01Group, bonesViewArray[i])
                bones[i].start = initial(i);
                bones[i].end = end(i);
                bones[i].mean = mean(i);
                bones[i].previous = "Left third "
                        + "metacarpal bone";
                bones[i].next = "Middle phalanx "
                        + "of left middle finger";
            }
        }

        //Finger 2 = index and its subgroups
        else if (name.contains("index"))
        {
            if (name.contains("Distal"))
            {
                finger2Ph0Group
                    .getChildren().add(bonesViewArray[i]);
                bones[i].start = initial(i);
                bones[i].end = end(i);
                bones[i].mean = mean(i);
                bones[i].previous = "Middle phalanx "
                        + "of left index finger";
                bones[i].next = "";
            }
            if (name.contains("Middle"))
            {
                finger2Ph01Group
                    .getChildren()
                    .addAll(finger2Ph0Group, bonesViewArray[i]);
                bones[i].start = initial(i);
```

```
            bones[i].end = end(i);
            bones[i].mean = mean(i);
            bones[i].previous = "Proximal phalanx "
                    + "of left index finger";
            bones[i].next = "Distal phalanx "
                    + "of left middle finger";
        }
        if (name.contains("Proximal"))
        {
            finger2Ph012Group
                .getChildren()
                .addAll(finger2Ph01Group
                        , bonesViewArray[i]);
            bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].mean = mean(i);
            bones[i].previous = "Left fourth "
                    + "metacarpal bone";
            bones[i].next = "Middle phalanx "
                    + "of left index finger";
        }
    }

    //Thumb and its subgroups
    else if (name.contains("thumb"))
    {
        if (name.contains("Distal"))
        {
            thumbPh0Group
                .getChildren().add(bonesViewArray[i]);
                thumbPh0GroupPivot = pivot(i);
                bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].mean = mean(i);
            bones[i].previous = "Proximal phalanx "
                    + "of left thumb";
            bones[i].next = "";
        }
```

```
        if (name.contains("Proximal"))
        {
            thumbPh01Group
                .getChildren()
                .addAll(thumbPh0Group
                        , bonesViewArray[i]);
            thumbPh01GroupPivot = pivot(i);
            bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].mean = mean(i);
            bones[i].previous = "";
            bones[i].next = "Distal phalanx "
                    + "of left thumb";
        }
    }

    //First metacarpal  and thumb are
    //added to the metaFinger1Group
    else if (name.contains("first metacarpal"))
    {
        metaFinger1Group
            .getChildren()
            .addAll(thumbPh01Group, bonesViewArray[i]);
        bones[i].start = initial(i);
        bones[i].end = end(i);
        bones[i].mean = mean(i);
        bones[i].previous = "";
        bones[i].next = "";
    }
    //Second metacarpal  is added to the metaFinger2Group
    else if (name.contains("second metacarpal"))
    {
        metaFinger2Group.getChildren()
                .addAll(finger2Ph012Group, bonesViewArray[i]
        bones[i].start = initial(i);
        bones[i].end = end(i);
        bones[i].mean = mean(i);
        bones[i].previous = "";
```

```
    bones[i].next = "Proximal phalanx "
            + "of left index finger";
}
//Third metacarpal is part of metaFinger3Group
else if (name.contains("third metacarpal"))
{
    metaFinger3Group.getChildren()
            .addAll(finger3Ph012Group, bonesViewArray[i]);
    bones[i].start = initial(i);
    bones[i].end = end(i);
    bones[i].mean = mean(i);
    bones[i].previous = "";
    bones[i].next = "Proximal phalanx "
            + "of left middle finger";
}
//Fourth metacarpal is part of metaFinger4Group
else if (name.contains("fourth metacarpal"))
{
    metaFinger4Group.getChildren()
            .addAll(finger4Ph012Group, bonesViewArray[i]);
    bones[i].start = initial(i);
    bones[i].end = end(i);
    bones[i].mean = mean(i);
    bones[i].previous = "";
    bones[i].next = "Proximal phalanx "
            + "of left ring finger";
}
//Fifth metacarpal is part of metaFinger5Group
else if (name.contains("fifth metacarpal"))
{
    metaFinger5Group.getChildren()
            .addAll(finger5Ph012Group, bonesViewArray[i]);
    bones[i].start = initial(i);
    bones[i].end = end(i);
    bones[i].mean = mean(i);
    bones[i].previous = "";
    bones[i].next = "Proximal phalanx "
            + "of left little finger";
```

```
        }

        //Trapezium is  part of the wrist
        else if (name.contains("trapezium"))
        {
           wristGroup.getChildren().add(bonesViewArray[i]);
           bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].mean = mean(i);
            bones[i].previous = "";
            bones[i].next = "";
        }
        else if (name.contains("lunate"))
        //Lunate is also part of the wrist
        {
            wristGroup.getChildren().add(bonesViewArray[i]);

            wristGroupPivot = pivot(i);
            bones[i].start = initial(i);
            bones[i].end = end(i);
            bones[i].mean = mean(i);
            bones[i].previous = "";
            bones[i].next = "";
            MeanOfHand = end(i);
               System.out.println("MeanOfHand = " + MeanOfHand);
        }
}


    //MetaFingers and wrist are attached to the palm

    palmGroup.getChildren().addAll(wristGroup
        , metaFinger1Group
        , metaFinger2Group
        , metaFinger3Group
        , metaFinger4Group
        , metaFinger5Group);
```

```
//=============CENTERING AND BOOKING ============
handXform.setTranslate(MeanOfHand.multiply(-1));
handXform.getChildren().add(palmGroup);



//====REACCOMMODATION OF BONES AFTER SIZE MUTATION======

//Helping special points of a bone :
//I = Initial point; M = Middle point;
//E = End point; S = Scale factors in mutation;
//C = middle point in within I and E of the next bone.
//Numeration of phalanges and corresponding metacarpal:
//0 = distal phalange, 1 = middle, 2 = proximal,
//3 = metacarpal.
Point3D I0 = new Point3D(0,0,0);
Point3D I1 = new Point3D(0,0,0);
Point3D I2 = new Point3D(0,0,0);
Point3D I3 = new Point3D(0,0,0);
Point3D M0 = new Point3D(0,0,0);
Point3D M1 = new Point3D(0,0,0);
Point3D M2 = new Point3D(0,0,0);
Point3D M3 = new Point3D(0,0,0);
Point3D E0 = new Point3D(0,0,0);
Point3D E1 = new Point3D(0,0,0);
Point3D E2 = new Point3D(0,0,0);
Point3D E3 = new Point3D(0,0,0);
Point3D S0 = new Point3D(0,0,0);
Point3D S1 = new Point3D(0,0,0);
Point3D S2 = new Point3D(0,0,0);
Point3D S3 = new Point3D(0,0,0);
Point3D C0;
Point3D C1;
Point3D C2;
Point3D C3 = new Point3D(0,0,0);
```

```
//Translations to recover a seamless, soft pattern
double  tx0, ty0, tz0;//Phalanx 0, distal
double tx01, ty01, tz01;//Phalanges 0 and 1, distal and midd
double tx012, ty012, tz012;//All three phalanges
double tx012m, ty012m, tz012m;//Finger and metacarpal


//Helped points are retrieved for the index = finger 2
for(int i = 0; i < counterOfBones ; i++)
{
    if (bones[i].name.contains("index"))
    {
        if (bones[i].name.contains("Distal"))
        {
           I0 = bones[i].start;
           M0 = bones[i].mean;
           E0 = bones[i].end;
           S0 = bones[i].scaleFactors;
        }

        if (bones[i].name.contains("Middle"))
        {
           I1 = bones[i].start;
           M1 = bones[i].mean;
           E1 = bones[i].end;
           S1 = bones[i].scaleFactors;
        }

        if (bones[i].name.contains("Proximal"))
        {
           I2 = bones[i].start;
           M2 = bones[i].mean;
           E2 = bones[i].end;
           S2 = bones[i].scaleFactors;
        }
    }
}
```

```
//The second metacarpal bone is connected
//to the index finger = finger 2
for(int i = 0; i < counterOfBones ; i++)
{
        if (bones[i].name.contains("Left second "
                + "metacarpal bone"))
        {
           I3 = bones[i].start;
           M3 = bones[i].mean;
           E3 = bones[i].end;
           S3 = bones[i].scaleFactors;
          System.out.println("Left second "
                + "metacarpal bone");
        }

}



//Bone groups of index finger get reaccommodated.
//Step 1: The metacarpal, bone 3,  gets reaccommodated.
//and the whole group is dragged.
tx012m = (1-S3.getX() )* ((M3.subtract(E3)).getX());
metaFinger2Group.setTranslateX(tx012m);
ty012m = (1-S3.getY() )* ((M3.subtract(E3)).getY());
metaFinger2Group.setTranslateY(ty012m);
tz012m = (1-S3.getZ() )* ((M3.subtract(E3)).getZ());
metaFinger2Group.setTranslateZ(tz012m);

//Step 2: Proximal phalanx 2 and its group 012 get into place
C2 = (I3.add(E2)).multiply(0.5);
tx012 = (1-S2.getX()) * ((C2.subtract(M2)).getX())
            + (1-S3.getX() )* ((M3.subtract(C2)).getX()) ;
finger2Ph012Group.setTranslateX(tx012);
ty012 = (1-S2.getY()) * ((C2.subtract(M2)).getY())
```

```
            + (1-S3.getY() )* ((M3.subtract(C2)).getY()) ;
finger2Ph012Group.setTranslateY(ty012);
tz012 = (1-S2.getZ()) * ((C2.subtract(M2)).getZ())
            + (1-S3.getZ() )* ((M3.subtract(C2)).getZ()) ;
finger2Ph012Group.setTranslateZ(tz012);

finger2Ph012GroupPivot = M3.add((E3.subtract(M3)).multiply(0

//Step 3: Middle phalanx 1 and its group 01 get into place.
//Moves found by trial and error with the purpose of
//creating the illusion of a natural bending.
C1 = (I2.add(E1)).multiply(0.5);
tx01 = (1-S1.getX()) * ((C1.subtract(M1)).getX())
            + (1-S2.getX() )* ((M2.subtract(C1)).getX()) ;
finger2Ph01Group.setTranslateX(tx01-3);
ty01 = (1-S1.getY()) * ((C1.subtract(M1)).getY())
            + (1-S2.getY() )* ((M2.subtract(C1)).getY()) ;
finger2Ph01Group.setTranslateY(ty01+3);
tz01 = (1-S1.getZ()) * ((C1.subtract(M1)).getZ())
            + (1-S2.getZ() )* ((M2.subtract(C1)).getZ()) ;
finger2Ph01Group.setTranslateZ(tz01+3);
finger2Ph01GroupPivot = M2.add((E2.subtract(M2)).multiply(0.

//Step 4: Distal phalanx 0 and its group 0 get into place
C0 = (I1.add(E0)).multiply(0.5);
tx0 = (1-S0.getX()) * ((C0.subtract(M0)).getX())
            + (1-S1.getX() )* ((M1.subtract(C0)).getX()) ;
finger2Ph0Group.setTranslateX(tx0);
ty0 = (1-S0.getY()) * ((C0.subtract(M0)).getY())
            + (1-S1.getY() )* ((M1.subtract(C0)).getY()) ;
finger2Ph0Group.setTranslateY(ty0);
tz0 = (1-S0.getZ()) * ((C0.subtract(M0)).getZ())
            + (1-S1.getZ() )* ((M1.subtract(C0)).getZ()) ;
finger2Ph0Group.setTranslateZ(tz0);
finger2Ph0GroupPivot = M1.add((E1.subtract(M1)).multiply(0.0


System.out.println("Data for bones of the index finger");
```

```
System.out.println(" E0.toString()" + E0.toString());
System.out.println(" M0.toString()" + M0.toString());
System.out.println(" I0.toString()" + I0.toString());
System.out.println(" C0.toString()" + C0.toString());
System.out.println(" S0.toString()" + S0.toString());

System.out.println(" E1.toString()" + E1.toString());
System.out.println(" M1.toString()" + M1.toString());
System.out.println(" I1.toString()" + I1.toString());
System.out.println(" C1.toString()" + C0.toString());
System.out.println(" S1.toString()" + S1.toString());
System.out.println(" E2.toString()" + E2.toString());
System.out.println(" M2.toString()" + M2.toString());
System.out.println(" I2.toString()" + I2.toString());
System.out.println(" C2.toString()" + C0.toString());
System.out.println(" S2.toString()" + S2.toString());
System.out.println(" E3.toString()" + E3.toString());
System.out.println(" M3.toString()" + M3.toString());
System.out.println(" I3.toString()" + I3.toString());
System.out.println(" C3.toString()" + C0.toString());
System.out.println(" S3.toString()" + S3.toString());


//Bone groups of middle finger get reaccommodated
for(int i = 0; i < counterOfBones ; i++)
{
    if (bones[i].name.contains("middle"))
    {
        if (bones[i].name.contains("Distal"))
        {
           M0 = bones[i].mean;
           E0 = bones[i].end;
           S0 = bones[i].scaleFactors;
        }

        if (bones[i].name.contains("Middle"))
        {
           I1 = bones[i].start;
```

```
                M1 = bones[i].mean;
                E1 = bones[i].end;
                S1 = bones[i].scaleFactors;
            }

            if (bones[i].name.contains("Proximal"))
            {
                I2 = bones[i].start;
                M2 = bones[i].mean;
                E2 = bones[i].end;
                S2 = bones[i].scaleFactors;
            }


        }
    }


    for(int i = 0; i < counterOfBones ; i++)
    {


            if (bones[i].name.contains("Left third "
                    + "metacarpal bone"))
            {
                I3 = bones[i].start;
                M3 = bones[i].mean;
                E3 = bones[i].end;
                S3 = bones[i].scaleFactors;
                System.out.println("Left third "
                    + "metacarpal bone");

            }

    }

    //Bone groups of middle finger get reaccommodated.
    //Step 1: The metacarpal, bone 3,  gets reaccommodated.
```

```
//and the whole group is dragged.
tx012m = (1-S3.getX() )* ((M3.subtract(E3)).getX());
metaFinger3Group.setTranslateX(tx012m);
ty012m = (1-S3.getY() )* ((M3.subtract(E3)).getY());
metaFinger3Group.setTranslateY(ty012m);
tz012m = (1-S3.getZ() )* ((M3.subtract(E3)).getZ());
metaFinger3Group.setTranslateZ(tz012m);

//Step 2: Proximal phalanx 2 and its group 012 get into place
C2 = (I3.add(E2)).multiply(0.5);
tx012 = (1-S2.getX()) * ((C2.subtract(M2)).getX())
            + (1-S3.getX() )* ((M3.subtract(C2)).getX()) ;
finger3Ph012Group.setTranslateX(tx012);
ty012 = (1-S2.getY()) * ((C2.subtract(M2)).getY())
            + (1-S3.getY() )* ((M3.subtract(C2)).getY()) ;
finger3Ph012Group.setTranslateY(ty012);
tz012 = (1-S2.getZ()) * ((C2.subtract(M2)).getZ())
            + (1-S3.getZ() )* ((M3.subtract(C2)).getZ()) ;
finger3Ph012Group.setTranslateZ(tz012);

finger3Ph012GroupPivot = M3.add((E3.subtract(M3)).multiply(0.10) );

//Step 3: Middle phalanx 1 and its group 01 get into place.
//Moves found by trial and error with the purpose of
//creating the illusion of a natural bending.
C1 = (I2.add(E1)).multiply(0.5);
tx01 = (1-S1.getX()) * ((C1.subtract(M1)).getX())
            + (1-S2.getX() )* ((M2.subtract(C1)).getX()) ;
finger3Ph01Group.setTranslateX(tx01-3);
ty01 = (1-S1.getY()) * ((C1.subtract(M1)).getY())
            + (1-S2.getY() )* ((M2.subtract(C1)).getY()) ;
finger3Ph01Group.setTranslateY(ty01+3);
tz01 = (1-S1.getZ()) * ((C1.subtract(M1)).getZ())
            + (1-S2.getZ() )* ((M2.subtract(C1)).getZ()) ;
finger3Ph01Group.setTranslateZ(tz01+3);
finger3Ph01GroupPivot = M2.add((E2.subtract(M2)).multiply(0.08) );

//Step 4: Distal phalanx 0 and its group 0 get into place
```

```
C0 = (I1.add(E0)).multiply(0.5);
tx0 = (1-S0.getX()) * ((C0.subtract(M0)).getX())
            + (1-S1.getX() )* ((M1.subtract(C0)).getX()) ;
finger3Ph0Group.setTranslateX(tx0-4);
ty0 = (1-S0.getY()) * ((C0.subtract(M0)).getY())
            + (1-S1.getY() )* ((M1.subtract(C0)).getY()) ;
finger3Ph0Group.setTranslateY(ty0+4);
tz0 = (1-S0.getZ()) * ((C0.subtract(M0)).getZ())
            + (1-S1.getZ() )* ((M1.subtract(C0)).getZ()) ;
finger3Ph0Group.setTranslateZ(tz0+3);
finger3Ph0GroupPivot = M1.add((E1.subtract(M1)).multiply(0.0



//Bone groups of ring finger get reaccommodated
for(int i = 0; i < counterOfBones ; i++)
{
    if (bones[i].name.contains("ring"))
    {
        if (bones[i].name.contains("Distal"))
        {
           M0 = bones[i].mean ;
           E0 = bones[i].end;
           S0 = bones[i].scaleFactors;
        }

        if (bones[i].name.contains("Middle"))
        {
           I1 = bones[i].start;
           M1 = bones[i].mean;
           E1 = bones[i].end;
           S1 = bones[i].scaleFactors;
        }

        if (bones[i].name.contains("Proximal"))
        {
           I2 = bones[i].start;
```

```
            M2 = bones[i].mean;
            E2 = bones[i].end;
            S2 = bones[i].scaleFactors;
        }


    }
}

for(int i = 0; i < counterOfBones ; i++)
{


        if (bones[i].name.contains("Left fourth "
                + "metacarpal bone"))
        {
            I3 = bones[i].start;
            M3 = bones[i].mean;
            E3 = bones[i].end;
            //Bug: S2 was there when S3 was needed
            S3 = bones[i].scaleFactors;
            System.out.println("Left fourth "
                + "metacarpal bone");
        }

}


//Bone groups of ring finger get reaccommodated.
//Step 1: The metacarpal, bone 3,  gets reaccommodated.
//and the whole group is dragged.
tx012m = (1-S3.getX() )* ((M3.subtract(E3)).getX());
metaFinger4Group.setTranslateX(tx012m);
ty012m = (1-S3.getY() )* ((M3.subtract(E3)).getY());
metaFinger4Group.setTranslateY(ty012m);
tz012m = (1-S3.getZ() )* ((M3.subtract(E3)).getZ());
metaFinger4Group.setTranslateZ(tz012m);
```

```
//Step 2: Proximal phalanx 2 and its group 012 get into plac
C2 = (I3.add(E2)).multiply(0.5);
tx012 = (1-S2.getX()) * ((C2.subtract(M2)).getX())
            + (1-S3.getX() )* ((M3.subtract(C2)).getX()) ;
finger4Ph012Group.setTranslateX(tx012);
ty012 = (1-S2.getY()) * ((C2.subtract(M2)).getY())
            + (1-S3.getY() )* ((M3.subtract(C2)).getY()) ;
finger4Ph012Group.setTranslateY(ty012);
tz012 = (1-S2.getZ()) * ((C2.subtract(M2)).getZ())
            + (1-S3.getZ() )* ((M3.subtract(C2)).getZ()) ;
finger4Ph012Group.setTranslateZ(tz012);

finger4Ph012GroupPivot = M3.add((E3.subtract(M3)).multiply(0

//Step 3: Middle phalanx 1 and its group 01 get into place.
//Moves found by trial and error with the purpose of
//creating the illusion of a natural bending.
C1 = (I2.add(E1)).multiply(0.5);
tx01 = (1-S1.getX()) * ((C1.subtract(M1)).getX())
            + (1-S2.getX() )* ((M2.subtract(C1)).getX()) ;
finger4Ph01Group.setTranslateX(tx01);
ty01 = (1-S1.getY()) * ((C1.subtract(M1)).getY())
            + (1-S2.getY() )* ((M2.subtract(C1)).getY()) ;
finger4Ph01Group.setTranslateY(ty01);
tz01 = (1-S1.getZ()) * ((C1.subtract(M1)).getZ())
            + (1-S2.getZ() )* ((M2.subtract(C1)).getZ()) ;
finger4Ph01Group.setTranslateZ(tz01);
finger4Ph01GroupPivot = M2.add((E2.subtract(M2)).multiply(0.

//Step 4: Distal phalanx 0 and its group 0 get into place
C0 = (I1.add(E0)).multiply(0.5);
tx0 = (1-S0.getX()) * ((C0.subtract(M0)).getX())
            + (1-S1.getX() )* ((M1.subtract(C0)).getX()) ;
finger4Ph0Group.setTranslateX(tx0);
ty0 = (1-S0.getY()) * ((C0.subtract(M0)).getY())
            + (1-S1.getY() )* ((M1.subtract(C0)).getY()) ;
finger4Ph0Group.setTranslateY(ty0);
tz0 = (1-S0.getZ()) * ((C0.subtract(M0)).getZ())
```

```
            + (1-S1.getZ() )* ((M1.subtract(C0)).getZ()) ;
finger4Ph0Group.setTranslateZ(tz0);
finger4Ph0GroupPivot = M1.add((E1.subtract(M1)).multiply(0.07));



//Bone groups of little finger  = 5 get reaccommodated
for(int i = 0; i < counterOfBones ; i++)
{
    if (bones[i].name.contains("little"))
    {
        if (bones[i].name.contains("Distal"))
        {
           M0 = bones[i].mean;
           E0 = bones[i].end;
           S0 = bones[i].scaleFactors;
        }

        if (bones[i].name.contains("Middle"))
        {
           I1 = bones[i].start;
           M1 = bones[i].mean;
           E1 = bones[i].end;
           S1 = bones[i].scaleFactors;
        }

        if (bones[i].name.contains("Proximal"))
        {
           I2 = bones[i].start;
           M2 = bones[i].mean;
           E2 = bones[i].end;
           S2 = bones[i].scaleFactors;
        }


    }
}
```

```
for(int i = 0; i < counterOfBones ; i++)
{


        if (bones[i].name.contains("Left fifth "
                + "metacarpal bone"))
        {
            I3 = bones[i].start;
            M3 = bones[i].mean;
            E3 = bones[i].end;
            //Bug: S2 was there when S3 was needed
            S3 = bones[i].scaleFactors;
            System.out.println("Left fifth "
                 + "metacarpal bone");

        }

}

//Bone groups of little finger get reaccommodated.
//Step 1: The metacarpal, bone 3,  gets reaccommodated.
//and the whole group is dragged.
tx012m = (1-S3.getX() )* ((M3.subtract(E3)).getX());
metaFinger5Group.setTranslateX(tx012m);
ty012m = (1-S3.getY() )* ((M3.subtract(E3)).getY());
metaFinger5Group.setTranslateY(ty012m);
tz012m = (1-S3.getZ() )* ((M3.subtract(E3)).getZ());
metaFinger5Group.setTranslateZ(tz012m);

//Step 2: Proximal phalanx 2 and its group 012 get into plac
C2 = (I3.add(E2)).multiply(0.5);
tx012 = (1-S2.getX()) * ((C2.subtract(M2)).getX())
            + (1-S3.getX() )* ((M3.subtract(C2)).getX()) ;
finger5Ph012Group.setTranslateX(tx012);
ty012 = (1-S2.getY()) * ((C2.subtract(M2)).getY())
            + (1-S3.getY() )* ((M3.subtract(C2)).getY()) ;
```

```
finger5Ph012Group.setTranslateY(ty012);
tz012 = (1-S2.getZ()) * ((C2.subtract(M2)).getZ())
            + (1-S3.getZ() )* ((M3.subtract(C2)).getZ()) ;
finger5Ph012Group.setTranslateZ(tz012);

finger5Ph012GroupPivot = M3.add((E3.subtract(M3)).multiply(0.10) );

//Step 3: Middle phalanx 1 and its group 01 get into place.
//Moves found by trial and error with the purpose of
//creating the illusion of a natural bending.
C1 = (I2.add(E1)).multiply(0.5);
tx01 = (1-S1.getX()) * ((C1.subtract(M1)).getX())
            + (1-S2.getX() )* ((M2.subtract(C1)).getX()) ;
finger5Ph01Group.setTranslateX(tx01);
ty01 = (1-S1.getY()) * ((C1.subtract(M1)).getY())
            + (1-S2.getY() )* ((M2.subtract(C1)).getY()) ;
finger5Ph01Group.setTranslateY(ty01);
tz01 = (1-S1.getZ()) * ((C1.subtract(M1)).getZ())
            + (1-S2.getZ() )* ((M2.subtract(C1)).getZ()) ;
finger5Ph01Group.setTranslateZ(tz01);
finger5Ph01GroupPivot = M2.add((E2.subtract(M2)).multiply(0.08) );

//Step 4: Distal phalanx 0 and its group 0 get into place
C0 = (I1.add(E0)).multiply(0.5);
tx0 = (1-S0.getX()) * ((C0.subtract(M0)).getX())
            + (1-S1.getX() )* ((M1.subtract(C0)).getX()) ;
finger5Ph0Group.setTranslateX(tx0);
ty0 = (1-S0.getY()) * ((C0.subtract(M0)).getY())
            + (1-S1.getY() )* ((M1.subtract(C0)).getY()) ;
finger5Ph0Group.setTranslateY(ty0);
tz0 = (1-S0.getZ()) * ((C0.subtract(M0)).getZ())
            + (1-S1.getZ() )* ((M1.subtract(C0)).getZ()) ;
finger5Ph0Group.setTranslateZ(tz0);
finger5Ph0GroupPivot = M1.add((E1.subtract(M1)).multiply(0.07));


//Bone groups of thumb finger get reaccommodated
for(int i = 0; i < counterOfBones ; i++)
```

```
{
    if (bones[i].name.contains("thumb"))
    {
        if (bones[i].name.contains("Distal"))
        {
           M0 = bones[i].mean;
           E0 = bones[i].end;
           S0 = bones[i].scaleFactors;
        }



        if (bones[i].name.contains("Proximal"))
        {
           I1 = bones[i].start;
           M1 = bones[i].mean;
           E1 = bones[i].end;
           S1 = bones[i].scaleFactors;
        }


    }
}

//The thumb goes on top of the first metacarpal
for(int i = 0; i < counterOfBones ; i++)
{
    if (bones[i].name.contains("Left first "
                + "metacarpal bone"))
        {
           I2 = bones[i].start;
           M2 = bones[i].mean;
           E2 = bones[i].end;
           S2 = bones[i].scaleFactors;
           System.out.println("Left first "
                + "metacarpal bone");
        }
}
```

```
//The thumb + metacarpal spin over the trapazium
for(int i = 0; i < counterOfBones ; i++)
{
    if (bones[i].name.contains("trapezium"))
        {
           M3 = bones[i].mean;
           E3 = bones[i].end;
           S3 = bones[i].scaleFactors;
           System.out.println("trapezium");
        }
}


//Bone groups of the thumb get reaccommodated.
//Step 1: The metacarpal, bone 3,  gets reaccommodated.
//and the whole group is dragged.
tx012m = (1-S3.getX() )* ((M3.subtract(E3)).getX());
metaFinger1Group.setTranslateX(tx012m);
ty012m = (1-S3.getY() )* ((M3.subtract(E3)).getY());
metaFinger1Group.setTranslateY(ty012m);
tz012m = (1-S3.getZ() )* ((M3.subtract(E3)).getZ());
metaFinger1Group.setTranslateZ(tz012m);
metaFinger1GroupPivot = M3.add((E3.subtract(M3)).multiply(0.10) );

//Step 2: Proximal phalanx 2 and its group 012 get into place


//Step 3: Middle phalanx 1 and its group 01 get into place.
//Moves found by trial and error with the purpose of
//creating the illusion of a natural bending.
C1 = (I2.add(E1)).multiply(0.5);
tx01 = (1-S1.getX()) * ((C1.subtract(M1)).getX())
            + (1-S2.getX() )* ((M2.subtract(C1)).getX()) ;
thumbPh01Group.setTranslateX(tx01);
ty01 = (1-S1.getY()) * ((C1.subtract(M1)).getY())
            + (1-S2.getY() )* ((M2.subtract(C1)).getY()) ;
thumbPh01Group.setTranslateY(ty01);
```

```
tz01 = (1-S1.getZ()) * ((C1.subtract(M1)).getZ())
            + (1-S2.getZ() )* ((M2.subtract(C1)).getZ()) ;
thumbPh01Group.setTranslateZ(tz01);
thumbPh01GroupPivot = M2.add((E2.subtract(M2)).multiply(0.08

//Step 4: Distal phalanx 0 and its group 0 get into place
C0 = (I1.add(E0)).multiply(0.5);
tx0 = (1-S0.getX()) * ((C0.subtract(M0)).getX())
            + (1-S1.getX() )* ((M1.subtract(C0)).getX()) ;
thumbPh0Group.setTranslateX(tx0);
ty0 = (1-S0.getY()) * ((C0.subtract(M0)).getY())
            + (1-S1.getY() )* ((M1.subtract(C0)).getY()) ;
thumbPh0Group.setTranslateY(ty0);
tz0 = (1-S0.getZ()) * ((C0.subtract(M0)).getZ())
            + (1-S1.getZ() )* ((M1.subtract(C0)).getZ()) ;
thumbPh0Group.setTranslateZ(tz0);
thumbPh0GroupPivot = M1.add((E1.subtract(M1)).multiply(0.07)

/*

C0 = (I0.add(E1)).multiply(0.5);
tz0 = (1-S0.getZ()) * ((C0.subtract(M0)).getZ())
            + (1-S1.getZ() )* ((M1.subtract(C0)).getZ()) ;
thumbPh0Group.setTranslateZ(tz0);
thumbPh0GroupPivot = new Point3D(C0.getX()
        ,C0.getY(), C0.getZ()-tz0);


C1 = (I1.add(E2)).multiply(0.5);
tz01 = (1-S1.getZ()) * ((C1.subtract(M1)).getZ())
            + (1-S2.getZ() )* ((M2.subtract(C1)).getZ()) ;
thumbPh01Group.setTranslateZ(tz01);
thumbPh01GroupPivot = new Point3D(C1.getX()
        ,C1.getY(), C1.getZ()-tz01);


C2 = (I2.add(E3)).multiply(0.5);
tz012 = (1-S2.getZ()) * ((C2.subtract(M2)).getZ())
```

```
            + (1-S3.getZ() )* ((M3.subtract(C2)).getZ()) ;
    metaFinger1Group.setTranslateZ(tz012);
    metaFinger1GroupPivot = new Point3D(C2.getX()
            ,C2.getY(), C2.getZ()-tz012);

    */

    //========OPENING/CL0SING OF THE HAND===========

    NumberBinding closeHandPropertyHalf
            = closeHandProperty.multiply(0.5);

    //The slider that closes the hand extends its
    //control to the palm:
    //it must rotate around
    //the end of the arm that is parallel to the X_AXIS
    //This is the pivot of rotation
    palmGroup.rwpx.pivotZProperty()
            .setValue(wristGroupPivot.getZ());
    palmGroup.rwpx.pivotYProperty()
            .setValue(wristGroupPivot.getY());
    //The angle of rotation is controlled by a slider
    palmGroup.rwpx.angleProperty()
            .bind(closeHandPropertyHalf);


/*
    //Specifications of movement for subgroups of finger 0
    finger0Ph0Group.rwpx.pivotZProperty()
            .setValue(finger0Ph0GroupPivot.getZ());
    finger0Ph0Group.rwpx.pivotYProperty()
            .setValue(finger0Ph0GroupPivot.getY());
    finger0Ph0Group.rwpx.angleProperty()
            .bindBidirectional(closeHandProperty);

    finger0Ph01Group.rwpx.pivotZProperty()
            .setValue(finger0Ph01GroupPivot.getZ());
    finger0Ph01Group.rwpx.pivotYProperty()
```

```
            .setValue(finger0Ph01GroupPivot.getY());
    finger0Ph01Group.rwpx.angleProperty()
            .bindBidirectional(closeHandProperty);

    finger0Ph012Group.rwpx.pivotZProperty()
            .setValue(finger0Ph012GroupPivot.getZ());
    finger0Ph012Group.rwpx.pivotYProperty()
            .setValue(finger0Ph012GroupPivot.getY());
    finger0Ph012Group.rwpx.angleProperty()
            .bindBidirectional(closeHandProperty);


    //Specifications of movement for subgroups of finger 1
    finger1Ph0Group.rwpx.pivotZProperty()
            .setValue(finger1Ph0GroupPivot.getZ());
    finger1Ph0Group.rwpx.pivotYProperty()
            .setValue(finger1Ph0GroupPivot.getY());
    finger1Ph0Group.rwpx.angleProperty()
            .bindBidirectional(closeHandProperty);

    finger1Ph01Group.rwpx.pivotZProperty()
            .setValue(finger1Ph01GroupPivot.getZ());
    finger1Ph01Group.rwpx.pivotYProperty()
            .setValue(finger1Ph01GroupPivot.getY());
    finger1Ph01Group.rwpx.angleProperty()
            .bindBidirectional(closeHandProperty);

    finger1Ph012Group.rwpx.pivotZProperty()
            .setValue(finger1Ph012GroupPivot.getZ());
    finger1Ph012Group.rwpx.pivotYProperty()
            .setValue(finger1Ph012GroupPivot.getY());
    finger1Ph012Group.rwpx.angleProperty()
            .bindBidirectional(closeHandProperty);
*/

    //Specifications of movement for subgroups of finger 2
    finger2Ph0Group.rwpx.pivotZProperty()
            .setValue(finger2Ph0GroupPivot.getZ());
```

```
finger2Ph0Group.rwpx.pivotYProperty()
        .setValue(finger2Ph0GroupPivot.getY());
finger2Ph0Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger2Ph01Group.rwpx.pivotZProperty()
        .setValue(finger2Ph01GroupPivot.getZ());
finger2Ph01Group.rwpx.pivotYProperty()
        .setValue(finger2Ph01GroupPivot.getY());
finger2Ph01Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger2Ph012Group.rwpx.pivotZProperty()
        .setValue(finger2Ph012GroupPivot.getZ());
finger2Ph012Group.rwpx.pivotYProperty()
        .setValue(finger2Ph012GroupPivot.getY());
finger2Ph012Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

//Specifications of movement for subgroups of finger 3
finger3Ph0Group.rwpx.pivotZProperty()
        .setValue(finger3Ph0GroupPivot.getZ());
finger3Ph0Group.rwpx.pivotYProperty()
        .setValue(finger3Ph0GroupPivot.getY());
finger3Ph0Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger3Ph01Group.rwpx.pivotZProperty()
        .setValue(finger3Ph01GroupPivot.getZ());
finger3Ph01Group.rwpx.pivotYProperty()
        .setValue(finger3Ph01GroupPivot.getY());
finger3Ph01Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger3Ph012Group.rwpx.pivotZProperty()
        .setValue(finger3Ph012GroupPivot.getZ());
finger3Ph012Group.rwpx.pivotYProperty()
        .setValue(finger3Ph012GroupPivot.getY());
```

```
finger3Ph012Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

//Specifications of movement for subgroups of finger 4
finger4Ph0Group.rwpx.pivotZProperty()
        .setValue(finger4Ph0GroupPivot.getZ());
finger4Ph0Group.rwpx.pivotYProperty()
        .setValue(finger4Ph0GroupPivot.getY());
finger4Ph0Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger4Ph01Group.rwpx.pivotZProperty()
        .setValue(finger4Ph01GroupPivot.getZ());
finger4Ph01Group.rwpx.pivotYProperty()
        .setValue(finger4Ph01GroupPivot.getY());
finger4Ph01Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger4Ph012Group.rwpx.pivotZProperty()
        .setValue(finger4Ph012GroupPivot.getZ());
finger4Ph012Group.rwpx.pivotYProperty()
        .setValue(finger4Ph012GroupPivot.getY());
finger4Ph012Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

//Specifications of movement for subgroups of finger 5
finger5Ph0Group.rwpx.pivotZProperty()
        .setValue(finger5Ph0GroupPivot.getZ());
finger5Ph0Group.rwpx.pivotYProperty()
        .setValue(finger5Ph0GroupPivot.getY());
finger5Ph0Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger5Ph01Group.rwpx.pivotZProperty()
        .setValue(finger5Ph01GroupPivot.getZ());
finger5Ph01Group.rwpx.pivotYProperty()
        .setValue(finger5Ph01GroupPivot.getY());
finger5Ph01Group.rwpx.angleProperty()
```

```
        .bindBidirectional(closeHandProperty);

finger5Ph012Group.rwpx.pivotZProperty()
        .setValue(finger5Ph012GroupPivot.getZ());
finger5Ph012Group.rwpx.pivotYProperty()
        .setValue(finger5Ph012GroupPivot.getY());
finger5Ph012Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);



//======OPENING/CLOSING OF FINGERS========

//Fingers that are at the border must
//open twice faster as those
//near the center.
NumberBinding openFingersPropertyTwo
        = openFingersProperty.multiply(2);
/*
//Specifications of movement for subgroups of finger 0
finger0Ph012Group.rwpy.pivotXProperty()
        .setValue(finger0Ph012GroupPivot.getX());
finger0Ph012Group.rwpy.pivotYProperty()
        .setValue(finger0Ph012GroupPivot.getY());
finger0Ph012Group.rwpy.pivotZProperty()
        .setValue(finger0Ph012GroupPivot.getZ());
finger0Ph012Group.rwpy.angleProperty()
        .bind(openFingersPropertyTwo);


//Specifications of movement for subgroups of finger 1
finger1Ph012Group.rwpy.pivotXProperty()
        .setValue(finger1Ph012GroupPivot.getX());
finger1Ph012Group.rwpy.pivotYProperty()
        .setValue(finger1Ph012GroupPivot.getY());
finger1Ph012Group.rwpy.pivotZProperty()
        .setValue(finger1Ph012GroupPivot.getZ());
```

```
finger1Ph012Group.rwpy.angleProperty()
        .bind(openFingersProperty);

*/


//Opening in the contrary direction of fingers 3 and 4
NumberBinding openFingersPropertyMinus
        = openFingersProperty.multiply(-1);

//Specifications of movement for finger 2
finger2Ph012Group.rwpy.pivotXProperty()
        .setValue(finger2Ph012GroupPivot.getX());
finger2Ph012Group.rwpy.pivotYProperty()
        .setValue(finger2Ph012GroupPivot.getY());
finger2Ph012Group.rwpy.pivotZProperty()
        .setValue(finger2Ph012GroupPivot.getZ());
finger2Ph012Group.rwpy.angleProperty()
        .bind(openFingersPropertyMinus);

//Finger three remains invariant under opening of fingers


//Specifications of movement for finger 4
finger4Ph012Group.rwpy.pivotXProperty()
        .setValue(finger4Ph012GroupPivot.getX());
finger4Ph012Group.rwpy.pivotYProperty()
        .setValue(finger4Ph012GroupPivot.getY());
finger4Ph012Group.rwpy.pivotZProperty()
        .setValue(finger4Ph012GroupPivot.getZ());
finger4Ph012Group.rwpy.angleProperty()
        .bind(openFingersProperty);

//Specifications of movement for finger 5
finger5Ph012Group.rwpy.pivotXProperty()
        .setValue(finger5Ph012GroupPivot.getX());
finger5Ph012Group.rwpy.pivotYProperty()
        .setValue(finger5Ph012GroupPivot.getY());
```

```
finger5Ph012Group.rwpy.pivotZProperty()
        .setValue(finger5Ph012GroupPivot.getZ());
finger5Ph012Group.rwpy.angleProperty()
        .bind(openFingersPropertyTwo);


//====== The thumb ========
//Specifications of movement for the thumb


NumberBinding rotateThumbPropertyDown
        = rotateThumbProperty.multiply(-0.4);

NumberBinding rotateThumbPropertyUp
        = rotateThumbProperty.multiply(-2);


NumberBinding closeThumbPropertyMinus
        = closeThumbProperty.multiply(-1);


//Traversal movement
metaFinger1Group.rwpy.pivotXProperty()
        .setValue(metaFinger1GroupPivot.getX());
metaFinger1Group.rwpy.pivotZProperty()
        .setValue(metaFinger1GroupPivot.getZ());
metaFinger1Group.rwpy.angleProperty()
        .bind(closeThumbProperty);

/*
//Rotations of phalanges 2 and 1 to make the grip.
//Rotations around X_AXIS and Y_AXIS at the same time
//Phalanx 2
thumbPh0Group.rwpx.pivotXProperty()
        .setValue(thumbPh0GroupPivot.getX());
thumbPh0Group.rwpx.pivotYProperty()
        .setValue(thumbPh0GroupPivot.getY());
thumbPh0Group.rwpx.pivotZProperty()
```

```
        .setValue(thumbPh0GroupPivot.getZ());
thumbPh0Group.rwpx.angleProperty()
        .bind(rotateThumbPropertyDown);

//Phalanges 2 and 1
thumbPh01Group.rwpx.pivotXProperty()
        .setValue(thumbPh01GroupPivot.getX());
thumbPh01Group.rwpx.pivotYProperty()
        .setValue(thumbPh01GroupPivot.getY());
thumbPh01Group.rwpx.pivotZProperty()
        .setValue(thumbPh01GroupPivot.getZ());
thumbPh01Group.rwpx.angleProperty()
        .bind(rotateThumbPropertyDown);

//Phalanx 2
thumbPh0Group.rwpy.pivotXProperty()
        .setValue(thumbPh0GroupPivot.getX());
thumbPh0Group.rwpy.pivotYProperty()
        .setValue(thumbPh0GroupPivot.getY());
thumbPh0Group.rwpy.pivotZProperty()
        .setValue(thumbPh0GroupPivot.getZ());
thumbPh0Group.rwpy.angleProperty()
        .bind(rotateThumbPropertyDown);
//Phalanges 2 and 1
thumbPh01Group.rwpy.pivotXProperty()
        .setValue(thumbPh01GroupPivot.getX());
thumbPh01Group.rwpy.pivotYProperty()
        .setValue(thumbPh01GroupPivot.getY());
thumbPh01Group.rwpy.pivotZProperty()
        .setValue(thumbPh01GroupPivot.getZ());
thumbPh01Group.rwpy.angleProperty()
        .bind(rotateThumbPropertyDown);
*/

//Rotation of the whole thumb
//to get opposite to other fingers
metaFinger1Group.rwpz.pivotXProperty()
        .setValue(metaFinger1GroupPivot.getX());
```

```
metaFinger1Group.rwpz.pivotYProperty()
        .setValue(metaFinger1GroupPivot.getY());
metaFinger1Group.rwpz.pivotZProperty()
        .setValue(metaFinger1GroupPivot.getZ());
metaFinger1Group.rwpz.angleProperty()
        .bind(rotateThumbProperty.multiply(-2));



//===========Animation=========

double endTime1 = 10000.0; ////1000 = 1 second
double endTime2 = 2 * endTime1;
double endTime3 = 3 * endTime1;
double endTime4 = 4 * endTime1;
double endTime5 = 5 * endTime1;
double endTime6 = 6 * endTime1;
double endTime7 = 7 * endTime1;
double endTime8 = 8 * endTime1;
double endTime9 = 9 * endTime1;




animation.getKeyFrames().addAll
(

    //The hand gets closed
    //closeHandProperty: initial position at initial time
    new KeyFrame(Duration.ZERO,
        //Permanent condition
        new KeyValue(closeThumbProperty, 0)),
    //closeThumbProperty: final position at final time
    new KeyFrame(new Duration(endTime1),
        new KeyValue(closeThumbProperty, -50)),

    //The hand gets closed
    //closeHandProperty: initial position at initial time
```

```
new KeyFrame(new Duration(endTime1),
    //Permanent condition
    new KeyValue(closeHandProperty, 0)),
//closeHandProperty: final position at final time
new KeyFrame(new Duration(endTime2),
    new KeyValue(closeHandProperty, -90))
    ,


//The thumb closes the grip
//rotateThumbProperty: initial position at initial time
new KeyFrame(new Duration(endTime2),
    //Permanent condition
    new KeyValue(rotateThumbProperty, 0)),
//rotateThumbProperty: final position at final time
new KeyFrame(new Duration(endTime3),
    new KeyValue(rotateThumbProperty, 37)),

//The thumb gets reopened
//rotateThumbProperty: initial position at initial time
new KeyFrame(new Duration(endTime3),
    //Permanent condition
    new KeyValue(rotateThumbProperty, 37)),
//rotateThumbProperty: final position at final time
new KeyFrame(new Duration(endTime4),
    new KeyValue(rotateThumbProperty, 0)),

//The hand get reopened
//closeHandProperty: initial position at initial time
new KeyFrame(new Duration(endTime4),
    //Permanent condition
    new KeyValue(closeHandProperty, -90)),
//closeHandProperty: final position at final time
new KeyFrame(new Duration(endTime5),
    new KeyValue(closeHandProperty, 0)),

//Fingers get reopened
//openFingersProperty: initial position at initial time
```

```
        new KeyFrame(new Duration(endTime5),
        //Permanent condition
        new KeyValue(openFingersProperty, 0)),
        //openFingersProperty: final position at final time
        new KeyFrame(new Duration(endTime6),
            new KeyValue(openFingersProperty, 15)),

        //Fingers get closed
        //openFingersProperty: initial position at initial time
        new KeyFrame(new Duration(endTime6),
        //Permanent condition
        new KeyValue(openFingersProperty, 15)),
        //openFingersProperty: final position at final time
        new KeyFrame(new Duration(endTime7),
            new KeyValue(openFingersProperty, 0)),

        //The thumb gets closed
        //closeThumbProperty: initial position at initial time
        new KeyFrame(new Duration(endTime7),
            //Permanent condition
            new KeyValue(closeThumbProperty, -55)),
        //closeThumbProperty: final position at final time
        new KeyFrame(new Duration(endTime8),
            new KeyValue(closeThumbProperty, 0)),

        //The thumb gets reopen
        //closeThumbProperty: initial position at initial time
        new KeyFrame(new Duration(endTime8),
            //Permanent condition
            new KeyValue(closeThumbProperty, 0)),
        //closeThumbProperty: final position at final time
        new KeyFrame(new Duration(endTime9),
            new KeyValue(closeThumbProperty, -25))

);
```

```
animation.play();

}

private static void printMatrix(double m[][], String title)
{
    System.out.println("This is the matrix " + title);
    for(int i=0;i<3;i++)
        System.out.println(m[i][0] + " " + m[i][1]+ " "
                                              + m[i][2]);
    System.out.println();
}


//The multiplication of a matrix times a vector
  private static  double[] matrixTimesVector(double m[][],
                                             double v[])
  {
    double[]  helpV = new double[3];
    for(int i=0; i<3; i++)
        helpV[i] =0;
    for(int i=0; i<3; i++)
        for(int j=0; j<3; j++)
            helpV[i]  =helpV[i]  + m[i][j]*v[j];
    return helpV;
  }


//Normalized crux product of vectors v and w
  private static  double dotProduct(double[] v,
                                    double[] w)
  {
    return v[0]*w[0] + v[1]*w[1] + v[2]*w[2];

  }
```

```
public void play()
{

}

@Override public void stop()
{

}

static class BoneDescription
{
    String name = "";
    Point3D start ;
    Point3D mean ;
    Point3D end ;
    String previous ;
    String next ;
    Point3D scaleFactors ;

    BoneDescription()
    {
        start = new Point3D(0,0,0);
        mean = new Point3D(0,0,0);
        end = new Point3D(0,0,0);
        previous = "";
        next = "";
        scaleFactors = new Point3D(1f,1f,1f);
    }
}


public static void main(String[] args)
{

    launch(args);
}
```

```
}//End of main class N48 Cylindricalmutations4
```

Problems of Chapter 4

54, page 221. Code to study the capability of recombination to battle recessive lethal mutations. The appreciation of the Author is that this program is a fiasco: the expected healing power of recombination is not clearly seen.

```
/*
/*
    Program N54 Recombination2.
    An organism is modeled just as a red circle and
    a gene as a string that encodes for
    the measure of its radius. Our genes have thee chars.
    A viable gene reads 123 or 256 and so on while
    an inviable one has a mutation that makes it
    to lost its meaning, say, 1a3 or 2h6 and so on.
    Our organisms are diploid,
    with two alleles for each locus. Both alleles
    appear in the same chromosome or string.
    If an  embryo has two normal alleles, say 123 and 256,
    that appear encoded as the string 123256,
    it will develop both genes and produce two superposed
    red circles that will look like just one,
    the greatest, i.e., 256. If one allele is normal, say
    123, and the other is a nonsensical mutant, say, 2h6,
    the embryo will look like the red circle
    encoded by the normal gene, 123.
    If both alleles are nonsensical mutants, say,
    1a3 and 2h6, no embryo  will be produced.
    So, our lethal mutations are recessive.
    Nevertheless and for the sake of clarity,
    an inviable allele produces a line.
```

```
    It is expected that recombination heals
    the population of lethal mutations when the
    mutation rate is small and the population size
    is great enough to almost securely overpass
    bottlenecks and the ensuing
    extinction caused by sampling of double mutants.
    This expectation was not verified.
 */


package Programs;




import java.util.Random;

import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.concurrent.Service;
import javafx.concurrent.Task;
import javafx.concurrent.WorkerStateEvent;
import javafx.event.ActionEvent;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Line;
import javafx.stage.Stage;



public class Recombination2 extends Application
{
```

```
    private static final String title
        = "Populations with and without recombination";

    //withRecombination must be turn on and off
    //to assess the healing power of recombination
    //against lethal recessive mutations.
    //Done automatically.
    private static boolean withRecombination;

    //Number of generations
    private static final int nGenerations = 5;
    //Actual generation
    private static int gen = 0;
    //Individuals are numbered according to an array  v x h
    private static  final int h = 15;
    private static  final int v = 3;

    //Chromosomes (treated as individuals)
    private static final String[][] Pop = new String[v][h] ;
    private static final  String[][] Pop2 = new String[v][h] ;

    private static final  String[][][][] Memory
            = new String[2][nGenerations][v][h] ;

    //Number of chars per allele
    private static final int numberOfChars = 3;



    private static final double pointMutationRate  = 0.8;
    //If a char must be changed, it can be replaced
    //with proportion this by a number and
    //by proportion  1-this  by a char
    private static final double propMutationToNumber = 0.2;
```

```
private static final double scaleFactor = 0.005 ;


private static final Group root = new Group();

//Boolean flags for printing



private static boolean runTests;
private static boolean printDrawPopulation;
private static boolean printDecode ;
private static boolean printLongTaskFinish ;
private static boolean printRec ;
private static boolean printMut ;
private static boolean printReproduction ;
private static boolean printPopulation  ;
private static boolean printNonDoubleMutant;

//The virtual simulation is declared and instantiated
private static final LongTask longTask
        = new LongTask();
//Click over this button to cancel simulation
private static final Button cancelTaskBt
        = new Button ("CancelSimulation");




//Turn on of the random generator
private static final Random r  = new Random();

private static int counterOfAbortions = 0;
private static int[] MemoryOfAbortions =  new int[2];
private static int counterOfSingleMutants = 0;
private static int counterOfDoubleMutants = 0;
private static int numberOfNoMutants = 0;
```

```
    private static int counterOfRepeats = 0;


    @Override
public void start(Stage primaryStage)
{
    primaryStage.setResizable(true);
    //Dimensions of the screen
    primaryStage.setScene(new Scene(root, 500, 500));

    primaryStage.setTitle(title);

    //Test
    runTests = false;
    if (runTests) runTests();
    else
    {
        //A long task can be cancelled:
        //Listener of cancelTaskBt button
        cancelTaskBt.setOnAction
            (
                (ActionEvent e) ->
                {
                    cancelTaskBt.setOpacity(0);
                    //Simulation ends
                    longTask.cancel();
                }
            );
        root.getChildren().add(cancelTaskBt);

        printingSetting();


        //Simulation begins
        longTask.start();
        //What to do if longTask is accomplished?
        longTask.setOnSucceeded((WorkerStateEvent t) ->
```

```
        {
             longTaskFinish();
        });
    }

    //The scene is shown
    primaryStage.show();
}


//Setting for the simulation
private static void printingSetting()
{
    printDrawPopulation = false;
    printDecode = false;
    printLongTaskFinish = true;
    printRec = false;
    printMut = false;
    printNonDoubleMutant = false;
    printReproduction = false;
    printPopulation  = false;
}


private  static  void runTests()
{
    System.out.println( "Running tests ");
    printingSetting();
    initialization();
    //Rec test:
    /*
    System.out.println( "Rec test ");
    printRec = true;
    recombination();
    */

    //Mut test:
    /*
```

```
System.out.println( "Mut test ");
printMut = true;
mutation();
*/

//Repro test:
//System.out.println( "Repro test ");
/*
for(int i = 0; i < h; i++)
{Pop[0][i] = "123abc";
Pop[1][i] = "123abc";}
reproduction();*/

//Draw test

System.out.println( "Draw test: "
        + "look at the graphic ");
//Test: nonsensical redefinition
Pop[2][0] = "123abc";
Pop[2][2] = "abc123";
Pop[2][4] = "abcdef";
drawPopulation(0); //end of draw test


//Classifer tests

System.out.println( "123456 is singleMutant: "
            + isSingleMutant("123456"));
System.out.println( "1a3456 is singleMutant: "
            + isSingleMutant("1a3456"));
System.out.println( "1234b6 is singleMutant: "
            + isSingleMutant("1234b6"));
System.out.println( "1a34b6 is singleMutant: "
            + isSingleMutant("1a34b6"));
System.out.println( "123456 is doubleMutant: "
            + isDoubleMutant("123456"));
System.out.println( "1a3456 is doubleMutant: "
            + isDoubleMutant("1a3456"));
```

```
    System.out.println( "1234b6 is doubleMutant: "
                + isDoubleMutant("1234b6"));
    System.out.println( "1a34b6 is doubleMutant: "
                + isDoubleMutant("1a34b6"));
    System.out.println( );


}


//The circle is drawn at row j, column i
private static Circle  createCircle(int j, int i
                                    , int radius )
{
    //
    int x = 20 + i* 20 ;
    int y = 20 + j * 20 ;
    int rho = (int) Math.ceil(scaleFactor * radius);
    Circle circle = new Circle(x,y,rho);

    //A color is prepared
    Color c = Color.RED;
    //The circle is given a color
    circle.setFill(c);

    return circle;
}

// This method generates random  chars
// that represent digits
private  static  Character  randomDigit()
{
    return  ( char ) ( r.nextInt( 10 ) + 48  );
}


//This method generates random nonnumeric chars
private  static  Character  randomChar()
```

```
{
    return  ( char ) ( r.nextInt( 26 ) + 65  );
}


//Population is initialized.
//All individuals are viable:
//they encode for 2 numbers,
//both in the range 0..999
private static void initialization( )
{
    System.out.println("Initialization");
    for(int j = 0; j <  v; j++)
    {
        for(int i = 0; i <  h; i++)
        {
            //Initialization in blank.
            Pop[j][i] = "";
            //A diploid individual is assembled char by char
            for(int k = 0; k < 2*numberOfChars; k++)
            {
                // We generate a random numeric char
                Character c =  randomDigit();
                //System.out.println( "char = " + c);
                // Char c is converted into a string s
                String s = "" + c;
                // s is concatenated to Pop[i]
                Pop[j][i] = Pop[j][i] + s;

            }
        }
    }
}


private static void printPopulation( )
{
    if (printPopulation)
```

```
    {
        System.out.println("Population at gen = " + gen);
        for(int j = 0; j <  v; j++)
        {
            for(int i = 0; i <  h; i++)
            {
                String s = Pop[j][i];
                System.out.println(j + "   "
                + i + "   string = " + s);

            }
        }
    }

}



//Strings are decoded into int numbers
//if only that is possible. Otherwise,
//an exception is handled.
//Viable alleles give rise to red circles (phenotypes).
//l=0: with rec.
//l = 1: wihout rec.
private static void drawPopulation(int l)
{
    if (printDrawPopulation)
    {
        System.out.println("drawPopulation");
    }


    //Rows
    for(int j = 0; j < v; j++)
    {

        //Columns
        for(int i = 0; i < h; i++)
```

```
{
    String ind  = Pop[j][i];
    if (printDrawPopulation)
        {
            System.out.println(
                " String  =  " + ind);
        }
    //Two alleles are  scanned
    for(int k = 0; k < 2; k++)
    {
        String s ;
        if (k == 0)  s = ind
                .substring(0,3);
        else s = ind.substring(3);
    int number;
    //Number is retrieved, if that is possible
    try
    {
         number = Integer.parseInt(s);
        if (printDrawPopulation)
        {
            System.out.println(
                "    number " + k + " = " + number);
        }


        //Create a circle
        Circle circle = createCircle(j,i, number);
        circle.setTranslateY(l * (v +2) * 20);
        //circle is attached to the root
        root.getChildren().add(circle);

    }
    //inviable mutants are caught here
    catch(NumberFormatException e1)
    {
        Line line;
        int y = 20 + j * 20 + l * (v +2) * 20;
```

```
                if (k == 0)
                    line = new Line(20 + i * 20 -10
                            , 20 + j * 20 -10
                            , 20 +i * 20 + 10
                            , 20 + j * 20 + 10);
                else
                    line = new Line(20 + i * 20 -10
                            , 20 + j * 20 + 10
                            , 20 +i * 20 + 10
                            , 20 + j * 20 - 10);

                line.setFill(Color.BLACK);
                line.setTranslateY(l * (v +2) * 20);
                //line is booked
                root.getChildren().add(line);
                if (printDrawPopulation)
                {
                    System.out.print("    NO NUMBER " );
                    System.out.println("Complaint = "
                            + e1.getMessage());
                }

            }

            }
        }

    }

}


//Draw the populations of last generation.
//Up: with recombination, Down: without recombination
private static void drawPopulations(int gen)
{
    for(int l = 0; l <  2; l++)
    {
```

```
        for(int j = 0; j <  v; j++)
        {
            for(int i = 0; i <  h; i++)
            {
                Pop[j][i] = Memory[l][gen] [j][i];
            }
        }
        //Draw population l=0: with rec; l = 1: without rec
        drawPopulation(l);
    }




}

//Inner class that contains the long task
//with a virtual,  non visual, simulation.
//This class is NOT compatible with JavaFX.
//JavaFX tasks, if any, are done after finishing.
private static class LongTask extends Service<String>
{

    @Override //createTask() must return a task
    protected Task<String> createTask()
    {
        return new Task<String>()
    {

    @Override //If start is commanded
    protected String call()
    {
        System.out.println(">>>>>>>>Simulation"
                + "\n=====Class LongTask: "
            + "\nnGenerations = " + nGenerations
        + "\nwith recombination = " + withRecombination);
```

```
    String s = "Long running task done";
    withRecombination = false;
    for(int l = 0; l < 2; l++)
    {
        counterOfRepeats = l;
        counterOfAbortions = 0;
        //Initial population.
        initialization( );
        if (l == 0) withRecombination = true;
        if (l == 1) withRecombination = false;
        //Loop for nGenerations
        for(int i = 0; i < nGenerations; ++i)
        {
            gen = i;

            oneGeneration();
            //If cancel button is clicked?????
            if (isCancelled())
            {


                s = "Long running task   cancelled";
               break;
            }

        }




    }
    System.out.println(s);
    return s;
}
```

```
    };
    }
}//End of inner class LongTask



//When longTask is finished, print the report
private static void longTaskFinish()
{
    if(printLongTaskFinish)
    {
        System.out.println("longTaskFinish()\n\nReport:");

    }
    cancelTaskBt.setOpacity(0);

    for(int l = 0; l < 2; l++)
    {
        if (l == 0) withRecombination = true;
        if (l == 1) withRecombination = false;
        System.out.println("\nwith Recombination = "
                + withRecombination);
        System.out.println("gen + propOfNoMutants "
              + "+ propOfSingleMutants "
            + "+ propOfDoubleMutants "
      + "+ test: sum is 1.0" );
        counterOfRepeats = l;


        for(int t = 0; t < nGenerations; t++)
        {
            for(int j = 0; j <  v; j++)
            {
                System.arraycopy(Memory[l][t][j]
                        , 0, Pop[j], 0, h);
            }
            gen = t;
            if(printPopulation) printPopulation();
```

```
            analizePopulation();
        }
        System.out.println("Total number of abortions = " +
            MemoryOfAbortions[l]);
        //Last population is drawn

    }
    //Draw population of last generation
    gen = nGenerations -1;
    drawPopulations(gen);
}


private static void analizePopulation()
{

    counterOfSingleMutants = 0;
    counterOfDoubleMutants = 0;
    int populationSize = h * v;
    for(int j = 0; j <  v; j++)
    {
        for(int i = 0; i <  h; i++)
        {
            if (isSingleMutant(Pop[j][i]))
                    counterOfSingleMutants++;
            if (isDoubleMutant(Pop[j][i]))
                    counterOfDoubleMutants++;
        }
    }
    numberOfNoMutants = populationSize -
        counterOfSingleMutants - counterOfDoubleMutants;
    double propOfSingleMutants = 1.0 * counterOfSingleMutants
            / populationSize ;
    double propOfDoubleMutants = 1.0 * counterOfDoubleMutants
            / populationSize ;
    double propOfNoMutants = 1.0 * numberOfNoMutants
            / populationSize ;
    double sum = propOfNoMutants + propOfSingleMutants
```

```
            + propOfDoubleMutants;


    //Table is printed
    System.out.printf(
            "%d    %f     %f     %f      % f%n"
            ,  gen, propOfNoMutants, propOfSingleMutants
            , propOfDoubleMutants, sum
            );

}


//=====================================
//============Evolutionary part========
//=====================================


//One generation = expression + assessement
//+ (for evolution) reproduction,
//recombination and mutation.
private static void oneGeneration()
{
    System.out.println("Generation: " + gen);
    if (withRecombination) recombination();
    mutation();
    reproduction();
    memorizePopulation( );
}


private static void memorizePopulation( )
{
    System.out.println("repeat: " + counterOfRepeats);
    MemoryOfAbortions[counterOfRepeats]
            = counterOfAbortions;
    //Transfer to Memory
    for(int j = 0; j <  v; j++)
```

```
    {
        System.arraycopy(Pop[j], 0
            , Memory[counterOfRepeats][gen][j], 0, h);
    }

}



//Recombination is enabled
private static String rec(int a, int b)
{
    String s = Pop[a][b];
    int m= s.length();
    String  s1= "";
    //Place of recombination
     int m1 = 0;
    if (m>0)
    {
        m1 = r.nextInt(m) ;
        s1= s.substring(0,m1);
    }

    int c = r.nextInt(v);
    int d = r.nextInt(h);
    String t = Pop[c][d];
    String   t2= t.substring(m1);

    String rec = s1 + t2;
    if (printRec)
        System.out.println(
            "Input 1 = " + s
                + " Input 2 = " + t
                + " Recombinant = " + rec);

    //Recombinant
    return rec;
}
```

```
//The definitive population  is a recombinant one.
private static void recombination()
{
    //Recombinants are saved to Pop2[][]
    for(int j = 0; j <  v; j++)
    {
        for(int i = 0; i <  h; i++)
        {
            //
            Pop2[j][i] = rec(j,i);

        }
    }

    //Transfer to Pop[][]
    for(int j = 0; j <  v; j++)
    {
        System.arraycopy(Pop2[j], 0, Pop[j], 0, h);
    }
}

//Mutation of individual Pop[a][b]
private static String mut(int a, int b)
{
    String s = Pop[a][b];
    String mutant = "";
    for(int i = 0; i <  s.length(); i++)
    {
        char c = s.charAt(i);
        double p = r.nextDouble();
        if (p < pointMutationRate)
        {
```

```
            double q = r.nextDouble();
            if (q < propMutationToNumber)
            {
                c = randomDigit();

            }
            else
            {
                c = randomChar();
            }


        }

  mutant = s.substring(0, i) + c
                        + s.substring(i+1);
    }
    if (printMut)
            {
                System.out.println("Input = " + s

                + " mutant = " + mutant);

            }
    return mutant;
}


private static void mutation()
{
    //Mutants are saved to Pop2[][]
    for(int j = 0; j <  v; j++)
    {
        for(int i = 0; i <  h; i++)
        {
            //
            Pop2[j][i] = mut(j,i);
```

```
        }
    }

    //Transfer to Pop[][]
    for(int j = 0; j <  v; j++)
    {
        System.arraycopy(Pop2[j], 0, Pop[j], 0, h);
    }
}


private static boolean isSingleMutant(String s)
{
    String s1 = s.substring(0,3);
    String s2 = s.substring(3);
    boolean b1 = isMutant(s1);
    boolean b2 = isMutant(s2);
    boolean output = ((b1 || b2) & (!( b1 & b2)));
    return output;

}

private static boolean isDoubleMutant(String s)
{
    String s1 = s.substring(0,3);
    String s2 = s.substring(3);
    boolean b1 = isMutant(s1);
    boolean b2 = isMutant(s2);
    return (b1 & b2) ;

}

private static boolean isMutant(String s)
{

    int number;
        //Number is retrieved, if that is possible
        try
```

```
        {
             number = Integer.parseInt(s);
            if (printDecode)
            {
                System.out.println(
                    "    number = " + number);
            }

            return false;
        }
        //inviable mutants are caught here
        catch(NumberFormatException e1)
        {

            if (printDecode)
            {
                System.out.print("notNUMBER " );
                System.out.println("Complaint = "
                        + e1.getMessage());
            }
            return true;
        }

}


//A non double lethal mutant is chosen at random
private static String nonDoubleMutant()
{
    boolean go = true;
    String s = "";
    while (go)
    {
        int a = r.nextInt(v);
        int b = r.nextInt(h);
        s = Pop[a][b];
        if ( printNonDoubleMutant)
            System.out.println("Try : individual [ "
```

```
                    + a + "]["+ b + "] = " + s);
        //A double mutant will produce an abortion
        if (  isDoubleMutant(s) ) counterOfAbortions++;
        else go = false;
    }
    if ( printNonDoubleMutant)
        System.out.println("Non double mutant = "  + s);
    return s;
}



//From two individuals, one is formed
//Parthenogenesis is allowed.
private static String offspring()
{
    String s = nonDoubleMutant();
    String o1 = s.substring(0, 3);
    int i = r.nextInt(2);
    if (  i == 1 ) o1 = s.substring(3);
    String t = nonDoubleMutant();
    String o2 = s.substring(0, 3);
    int j = r.nextInt(2);
    if (  j == 1 ) o2 = t.substring(3);
    return o1 + o2;
}




//Reproduction by random sampling:
//Two individuals of the old generation are sampled
//at random with a uniform distribution
//to generate an individual of the next generation.
//Double lethal mutants are rejected as fathers.
//Population size is maintained constant.
```

```
//Parthenogenesis is allowed.
private static void reproduction()
{
    if (printReproduction)
        {
            System.out.println("\nReproduction");
        }
    //Offspring is  saved to Pop2[][]
    for(int j = 0; j <  v; j++)
    {
        for(int i = 0; i <  h; i++)
        {

            Pop2[j][i] = offspring();

        }
    }

    //Transfer to Pop[][]
    for(int j = 0; j <  v; j++)
    {
        System.arraycopy(Pop2[j], 0, Pop[j], 0, h);
    }
}


public static void main(String[] args)
{
    launch(args);
}



}//End of application  N54 Recombination2
```

55, page 221. The following code allows to study the role of role of haploidy, diploidy, recombination and cloning to battle recessive lethal mutations. The appreciation of the Author is that the cost in abortions for

haploid populations is higher than for diploid ones. Therefore, diploid populations can support higher mutation rates to lethal recessive mutations than haploid ones. Additionally, recombination is an important weapon to battle recessive lethal mutations when the population size is 2, i.e., a male and a female. This implies that recombination was behind the rapid spread of population through great areas thanks to migration. On one side, recombination was responsible for variability and from another it exercises a small curative effect against considered type of mutation. The code follows:

```
/*
/*
    Program N55 Recombination3.

    This programs allows to investigate the role of
    haploidy, diploidy, recombination and cloning
    to battle recessive lethal mutations.

    You can run two settings of parameters,
    marked as 0 and 1.
    If these settings differ by the value
    of just one variable, as it is recommended,
    the usual  experimental vs control situation
    is recovered.

    An organism is modeled just as a red circle and
    a gene as a string that encodes for
    the measure of its radius. Our genes have thee chars.
    A viable gene reads 123 or 256 and so on while
    an inviable one has a mutation that makes it
    to lost its meaning, say, 1a3 or 2h6 and so on.
    Our organisms can be haploid, with one allele per locus
    or diploid, with two versions or alleles per locus.
    Both alleles appear in the same chromosome or string.
    If an  embryo has two normal alleles, say 123 and 256,
    that appear encoded as the string 123256,
    it will develop both genes and produce two superposed
    red circles that will look like just one,
    the greatest, i.e., 256. If one allele is normal, say
    123, and the other is a nonsensical mutant, say, 2h6,
```

```
    the embryo will look like the red circle
    encoded by the normal gene, 123.
    If both alleles are nonsensical mutants, say,
    1a3 and 2h6, no embryo  will be produced.
    So, our lethal mutations are recessive.
    Nevertheless and for the sake of clarity,
    an inviable allele produces a line.

    Cloning + mutation is  the only mean of reproduction
    of haploid species,  such as bacteria. Nevertheless,
    they also enjoy recombination which is enabled
    by conjugation  and plasmid exchange.
    In parthenogenesis, diploid individuals
    may reproduce  asexually by cloning + recombinant processes.
    Cloning of diploid species is simulated
    when the same individual is sampled at random  twice
    for mating. So, cloning corresponds with
    sampling with replacement and non-cloning
    with sampling without replacement.


 */


package Programs;




import java.util.Random;

import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.concurrent.Service;
import javafx.concurrent.Task;
import javafx.concurrent.WorkerStateEvent;
import javafx.event.ActionEvent;
import javafx.scene.Group;
```

```
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Line;
import javafx.stage.Stage;



public class Recombination3 extends Application
{


    private static final String title
        = "Battling recessive lethal mutations";

    //A population can be haploid else diploid.
    private static boolean withDiploidy0;
    private static boolean withDiploidy1;
    //local variable
    private static boolean withDiploidy;
    //withRecombination must be turn on and off
    //to assess the healing power of recombination
    //against lethal recessive mutations.
    //Done automatically.
    private static boolean withRecombination0;
    private static boolean withRecombination1;
    private static boolean withRecombination;

    //Allow crossing of one diploid individual with itself
    //else make sure to cross it with a different one
    private static boolean withReplacement0;
    private static boolean withReplacement1;
    private static boolean withReplacement;

    //Values of parameters are kept here
    private static final boolean[][] Parameters
            = new boolean[2][3] ;
```

```
//Number of generations
private static final int nGenerations = 3;
//Actual generation
private static int gen = 0;
//Individuals are numbered according to an array  v x h
private static  final int h = 2;
private static  final int v = 1;

//Chromosomes (treated as individuals)
private static final String[][] Pop = new String[v][h];
private static final  String[][] Pop2 = new String[v][h];

private static final  String[][][][] Recording
        = new String[2][nGenerations][v][h] ;

//Number of chars per allele
private static final int numberOfChars = 3;




private static final double pointMutationRate  = 0.5;
//If a char must be changed, it can be replaced
//with proportion this by a number and
//by proportion  1-this  by a char
private static final double propMutationToNumber = 0.2;




private static final double scaleFactor = 0.005 ;


private static final Group root = new Group();

//Boolean flags for printing
private static boolean runTests;
```

```
    private static boolean printDrawPopulation;
    private static boolean printIsMutant ;
    private static boolean printRec ;
    private static boolean printMut ;
    private static boolean printReproduction ;
    private static boolean printPopulation  ;
    private static boolean printNonDoubleMutant;
    private static boolean printNonMutant;
    private static boolean printCloningDetected;

    //The virtual simulation is declared and instantiated
    private static final LongTask longTask
            = new LongTask();
    //Click over this button to cancel simulation
    private static final Button cancelTaskBt
            = new Button ("CancelSimulation");

    //Turn on of the random generator
    private static final Random r  = new Random();
    private static int counterOfAbortions;
    private static int cummulatedAbortions[]
                                        =  new int[2];
    private static int counterOfSingleMutants = 0;
    private static int counterOfDoubleMutants = 0;
    private static int counterOfNoMutants = 0;
    private static int expSetting;

    private static boolean shortReport;

    private static int row;
    private static int column;


//Two different settings are run, 0 and 1.
//It is recommended to compare runs when
//just one boolean variable suffers a change.
private static void parameterSetting()
{
```

```
    withDiploidy0  = true;
    withDiploidy1  = true;

    withRecombination0 = true;
    withRecombination1 = false;

    withReplacement0 = false;
    withReplacement1 = false;

    //!!!! Warning !!!!
    //Turn off recombination
    //and turn on parthenogenesis
    //when there is no diploidy.
    if (withDiploidy0); //do nothing
    else //in haploidy
    {
        withRecombination0 = false;
        withReplacement0 = true;
    }
    if (withDiploidy1);
    else
    {
        withRecombination1 = false;
        withReplacement1 = true;
    }



    Parameters[0][0] = withDiploidy0;
    Parameters[0][1] = withRecombination0;
    Parameters[0][2] = withReplacement0;

    Parameters[1][0] = withDiploidy1;
    Parameters[1][1] = withRecombination1;
    Parameters[1][2] = withReplacement1;

}
```

```
//Setting for the simulation
private static void printingSetting()
{
    printDrawPopulation = false;
    printIsMutant = false;
    printRec = false;
    printMut = false;
    printNonDoubleMutant = false;
    printNonMutant = false;
    printReproduction = false;
    printCloningDetected = false;
    printPopulation  = true;
    shortReport = true;

}


private static void printParameters()
{
    System.out.println(
        "\nExperimental setting  =  " + expSetting
        + "\nwithDiploidy = " + withDiploidy
        + "\nwithRecombination = " + withRecombination
        + "\nwithReplacement = " + withReplacement);
}


@Override
public void start(Stage primaryStage)
{
    primaryStage.setResizable(true);
    //Dimensions of the screen
    primaryStage.setScene(new Scene(root, 500, 500));

    primaryStage.setTitle(title);
```

```
    //Test
    runTests = false;
    if (runTests) runTests();
    else
    {
        //A long task can be cancelled:
        //Listener of cancelTaskBt button
        cancelTaskBt.setOnAction
            (
                (ActionEvent e) ->
                {
                    cancelTaskBt.setOpacity(0);
                    //Simulation ends
                    longTask.cancel();
                    System.out.println(
                            "Simulation cancelled!");
                    System.exit(0);
                }
            );
        root.getChildren().add(cancelTaskBt);
        //Initialization
        cummulatedAbortions[0] = 0;
        cummulatedAbortions[1] = 0;
        printingSetting();
        //Simulation begins
        longTask.start();
        //What to do if longTask is accomplished?
        longTask.setOnSucceeded((WorkerStateEvent t) ->
        {
             longTaskFinish();
        });
    }

    //The scene is shown
    primaryStage.show();
}
```

```
private  static  void runTests()
{
    System.out.println( "Running tests ");
    printingSetting();
    initialization(0);
    //Rec test:
    /*
    System.out.println( "Rec test ");
    printRec = true;
    recombination();
    */

    //Mut test:
    /*
    System.out.println( "Mut test ");
    printMut = true;
    mutation();
    */

    //Repro test:
    //System.out.println( "Repro test ");
    /*
    for(int i = 0; i < h; i++)
    {Pop[0][i] = "123abc";
    Pop[1][i] = "123abc";}
    reproduction();*/

    //Draw test

    System.out.println( "Draw test: "
            + "look at the graphic ");
    //Test: nonsensical redefinition
    Pop[2][0] = "123abc";
    Pop[2][2] = "abc123";
    Pop[2][4] = "abcdef";
    drawPopulation(0); //end of draw test
```

```
    //Classifier tests

    System.out.println( "123 is noMutant: "
                + isNoMutant("123"));
    System.out.println( "123 is singleMutant: "
                + isSingleMutant("123"));
    System.out.println( "1a3 is singleMutant: "
                + isSingleMutant("1a3"));
    System.out.println( "123 is doubleMutant: "
                + isDoubleMutant("123"));
    System.out.println( "1a3 is doubleMutant: "
                + isDoubleMutant("1a3"));

    System.out.println( "123456 is noMutant: "
                + isNoMutant("123456"));
    System.out.println( "123456 is singleMutant: "
                + isSingleMutant("123456"));
    System.out.println( "1a3456 is singleMutant: "
                + isSingleMutant("1a3456"));
    System.out.println( "1234b6 is singleMutant: "
                + isSingleMutant("1234b6"));
    System.out.println( "1a34b6 is singleMutant: "
                + isSingleMutant("1a34b6"));
    System.out.println( "123456 is doubleMutant: "
                + isDoubleMutant("123456"));
    System.out.println( "1a3456 is doubleMutant: "
                + isDoubleMutant("1a3456"));
    System.out.println( "1234b6 is doubleMutant: "
                + isDoubleMutant("1234b6"));
    System.out.println( "1a34b6 is doubleMutant: "
                + isDoubleMutant("1a34b6"));
    System.out.println( );


}
```

```
//The circle is drawn at row j, column i
private static Circle  createCircle(int j, int i
                                    , int radius )
{
    //
    int x = 20 + i* 20 ;
    int y = 20 + j * 20 ;
    int rho = (int) Math.ceil(scaleFactor * radius);
    Circle circle = new Circle(x,y,rho);

    //A color is prepared
    Color c = Color.RED;
    //The circle is given a color
    circle.setFill(c);

    return circle;
}

// This method generates random  chars
// that represent digits
private  static  Character  randomDigit()
{
    return  ( char ) ( r.nextInt( 10 ) + 48  );
}


//This method generates random nonnumeric chars
private  static  Character  randomChar()
{
    return  ( char ) ( r.nextInt( 26 ) + 65  );
}


//Population is initialized.
//All individuals are viable.
//Chromosomes of diploid populations
//encode for 2 numbers,
```

```
//both in the range 0..999.
//Haploid for one.
//Parameters turn into global variables.
private static void initialization(int l )
{
    int nChars;
    if (withDiploidy) nChars = 2*numberOfChars;
        else nChars = numberOfChars;

    System.out.println("\nInitialization");
    for(int j = 0; j <  v; j++)
    {
        for(int i = 0; i <  h; i++)
        {
            //Initialization in blank.
            Pop[j][i] = "";
            //A diploid individual is assembled char by char
            for(int k = 0; k < nChars; k++)
            {
                // We generate a random numeric char
                Character c =  randomDigit();
                //System.out.println( "char = " + c);
                // Char c is converted into a string s
                String s = "" + c;
                // s is concatenated to Pop[i]
                Pop[j][i] = Pop[j][i] + s;
            }
        }
    }
}

//Parameters turn into global variables.
private static void workingParameters(int l )
{
    withDiploidy = Parameters[l][0];
    withRecombination = Parameters[l][1];
    withReplacement = Parameters[l][2];
}
```

```
private static void printPopulation( )
{

        System.out.println("Population at gen = " + gen);

        for(int j = 0; j <  v; j++)
        {
            for(int i = 0; i <  h; i++)
            {
                String s = Pop[j][i];
                System.out.println(j + "   "
                + i + "   string = " + s);

            }
        }


}



//Strings are decoded into int numbers
//if only that is possible. Otherwise,
//an exception is handled.
//Viable alleles give rise to red circles (phenotypes).
//l=0 = experimental setting 0;
//l = 1 = experimental setting 1.
private static void drawPopulation(int l)
{
```

```
workingParameters( l );
int nAlleles;
if (withDiploidy) nAlleles = 2;
else nAlleles = 1;
if (printDrawPopulation)
{
    System.out.println("drawPopulation");
    System.out.println("nAlleles = " + nAlleles);
}
//if (printPopulation)  printPopulation();
//Rows
for(int j = 0; j < v; j++)
{
    //Columns
    for(int i = 0; i < h; i++)
    {
        String ind  = Pop[j][i];
        if (printDrawPopulation)
        {
            System.out.println(
                " String  =  " + ind);
        }
        //nAlleles alleles are  scanned
        for(int k = 0; k < nAlleles; k++)
        {
            String s ;
            if (k == 0)  s = ind
                    .substring(0,3);
            else s = ind.substring(3);
            int number;
            //Number is retrieved, if that is possible
            try
            {
                 number = Integer.parseInt(s);
                if (printDrawPopulation)
                {
                    System.out.println(
                    "    number " + k + " = " + number);
```

```
        }


        //Create a circle
        Circle circle
                = createCircle(j,i, number);
        circle.setTranslateY(l * (v +2) * 20);
        //circle is attached to the root
        root.getChildren().add(circle);

    }
    //inviable mutants are caught here
    catch(NumberFormatException e1)
    {
        Line line;
        int y = 20 + j * 20 + l * (v +2) * 20;
        if (k == 0)
            line = new Line(20 + i * 20 -10
                    , 20 + j * 20 -10
                    , 20 +i * 20 + 10
                    , 20 + j * 20 + 10);
        else
            line = new Line(20 + i * 20 -10
                    , 20 + j * 20 + 10
                    , 20 +i * 20 + 10
                    , 20 + j * 20 - 10);

        line.setFill(Color.BLACK);
        line.setTranslateY( l * (v +2) * 20 );
        //line is booked
        root.getChildren().add(line);
        if (printDrawPopulation)
        {
            System.out.print(
                    "    NO NUMBER " );
            System.out.println("Complaint = "
                    + e1.getMessage());
        }
```

```
                }

            }
        }

    }

}


//Draw the populations of last generation.
//Up: setting 0, Down: setting 1
private static void drawPopulations(int gen)
{

    for(int l = 0; l <  2; l++)
    {
        for(int j = 0; j <  v; j++)
        {
            for(int i = 0; i <  h; i++)
            {
                Pop[j][i] = Recording[l][gen] [j][i];
            }
        }
        drawPopulation(l);
    }


}

//Inner class that contains the long task
//with a virtual,  non visual, simulation.
//This class is NOT compatible with JavaFX.
//JavaFX tasks, if any, are done after finishing.
private static class LongTask extends Service<String>
{
```

```
@Override //createTask() must return a task
protected Task<String> createTask()
{
    return new Task<String>()
{

@Override //If start is commanded
protected String call()
{
    System.out.println(">>>>>>>>Simulation"
            + "\n=====Class LongTask: "
        + "\nnGenerations = " + nGenerations);

    parameterSetting();

    String s = "Long running task done";
    withRecombination = false;
    for(int l = 0; l < 2; l++)
    {
        expSetting = l;
        //Used variables are initialized
        workingParameters( l );
        //Initial population.
        initialization(l );
        printParameters();


        //Loop for nGenerations
        for(int t = 0; t < nGenerations; ++t)
        {
            gen = t;

            oneGeneration();
            //If cancel button is clicked?????
            if (isCancelled())
            {
                s = "Long running task   cancelled";
               break;
```

```
                }

            }
        }
        System.out.println(s);
        return s;
    }
    };
    }
}//End of inner class LongTask



//When longTask is finished, print a report
private static void longTaskFinish()
{

    System.out.println("\nlongTaskFinish()\n\n"
                + "\">>>>>>>>Report:");

    cancelTaskBt.setOpacity(0);
    parameterSetting();

    if (shortReport) shortReport();
    else longReport();

    System.out.println("\nCummulated number of abortions"
        + "\nfor setting 0 = " + cummulatedAbortions[0]
        + "\nfor setting 1 = " + cummulatedAbortions[1]);
    //Draw populations of last generation
    gen = nGenerations -1;
    drawPopulations(gen);

}


private static void shortReport()
{
```

```
    for(int l = 0; l < 2; l++)
    {
        expSetting = l;
        workingParameters( l );
        printParameters();
        System.out.println("gen + propOfNoMutants "
                + "+ propOfSingleMutants "
                + "+ propOfDoubleMutants "
                + "+ test: sum is 1.0" );
        for(int t = 0; t < nGenerations; t++)
        {
            gen = t;
            //Recording is read
            for(int j = 0; j <  v; j++)
            {
                for(int i = 0; i <  h; i++)
                {
                    Pop[j][i]
                        = Recording[expSetting][gen][j][i]  ;
                }
            }
            analizePopulation();

        }
    }
}


private static void longReport()
{
    for(int t = 0; t < nGenerations; t++)
    {
        gen = t;
        for(int l = 0; l < 2; l++)
        {
            expSetting = l;
            workingParameters( l );
            printParameters();
```

```
            //Recording is read
            for(int j = 0; j <  v; j++)
            {
                for(int i = 0; i <  h; i++)
                {
                    Pop[j][i]
                        = Recording[expSetting][gen][j][i]  ;
                }
            }


            if(printPopulation) printPopulation();
            analizePopulation();
        }
    }
}



private static void analizePopulation()
{
    counterOfNoMutants = 0;
    counterOfSingleMutants = 0;
    counterOfDoubleMutants = 0;
    int populationSize = h * v;
    for(int j = 0; j <  v; j++)
    {
        for(int i = 0; i <  h; i++)
        {
            if (isNoMutant(Pop[j][i]))
                    counterOfNoMutants++;
            if (isSingleMutant(Pop[j][i]))
                    counterOfSingleMutants++;
            if (isDoubleMutant(Pop[j][i]))
                    counterOfDoubleMutants++;
        }
```

```
    }

    if (shortReport) ;
    else
        System.out.println(
            "\ncounterOfNoMutants = " + counterOfNoMutants
            + "\ncounterOfSingleMutants = "
                    + counterOfSingleMutants
            + "\ncounterOfDoubleMutants = "
                    + counterOfDoubleMutants);

    double propOfNoMutants = counterOfNoMutants;
    propOfNoMutants =  propOfNoMutants
            / populationSize ;
    double propOfSingleMutants = counterOfSingleMutants ;
    propOfSingleMutants = 1.0 * propOfSingleMutants
            / populationSize ;
    double propOfDoubleMutants = counterOfDoubleMutants ;
    propOfDoubleMutants = propOfDoubleMutants
            / populationSize ;
    double sum = propOfNoMutants + propOfSingleMutants
            + propOfDoubleMutants;
    if (shortReport) ;
    else
    System.out.println("gen + propOfNoMutants "
                + "+ propOfSingleMutants "
                + "+ propOfDoubleMutants "
                + "+ test: sum is 1.0" );
    //Table is printed
    System.out.printf(
            "%d    %f     %f     %f      % f%n"
            ,  gen, propOfNoMutants, propOfSingleMutants
            , propOfDoubleMutants, sum
            );

}
```

```
//=====================================
//============Evolutionary part========
//=====================================


//One generation = expression + assessement
//+ (for evolution) reproduction,
//recombination and mutation.
private static void oneGeneration()
{
    counterOfAbortions = 0;
    //Warning: the order as these processes appear
    //may strongly change the output.
    System.out.println("Generation: " + gen);
    mutation();
    if (withRecombination) recombination();
    reproduction();

    memorizePopulation( );

}



private static void memorizePopulation( )
{
    //Transfer from Pop to Recording
    for(int j = 0; j <  v; j++)
    {
        for(int i = 0; i <  h; i++)
            {
                Recording[expSetting][gen][j][i]
                        = Pop[j][i];
            }

    }
    cummulatedAbortions[expSetting] =
```

```
        cummulatedAbortions[expSetting]
            + counterOfAbortions;

}



//Recombination is enabled in within the alleles
//of one individual Pop[a][b].
//Bug is corrected:
//two diploid individuals were involved
//in the recombination method of the previous version.
private static String rec(int a, int b)
{
    String s = Pop[a][b];
    int m = s.length();
    int l = m/2;
    //Two alleles are retrieved
    String a1 = s.substring(0,l);
    String a2 = s.substring(l);
    //Two incoming alleles, a1 and a2,
    //produce two alleles o1 and o2;
    String  s1 ;
    String  s2 ;
    String  t1 ;
    String  t2 ;
    String  o1 ;
    String  o2 ;

    //Place of recombination
    int m1 = 0;
    if (l>0) m1 = r.nextInt(l) ;
    //Recombinant parts
    s1= a1.substring(0,m1);
    s2= a1.substring(m1);
    t1= a2.substring(0,m1);
    t2= a2.substring(m1);
    //Products of homologous recombination:
```

```
    o1 = s1 + t2;
    o2 = t1 + s2;
    //Two alleles anre encoded in one string
    String rec = o1 + o2;
    if (printRec)
        System.out.println(
                "Input  = " + s +
            " Allele1 = " + a1
                + " Allele2 = " + a2
                        + "\nRecombinant parts: "
                        + s1 + "  " +  s2 + "   "
                        + t1 + "  " + t2 + "    "
                + " Recombinant product = " + rec);

    //Recombinant
    return rec;
}




//The definitive population  is a recombinant one.
private static void recombination()
{
    //Recombinants are saved to Pop2[][]
    for(int j = 0; j <  v; j++)
    {
        for(int i = 0; i <  h; i++)
        {
            //
            Pop2[j][i] = rec(j,i);

        }
    }

    //Transfer of Pop2[][] to Pop[][]
    for(int j = 0; j <  v; j++)
```

```
    {
        for(int i = 0; i <  h; i++)
        {
            //
            Pop[j][i] = Pop2[j][i];

        }
    }
}

//Mutation of individual Pop[a][b]
private static String mut(int a, int b)
{
    String s = Pop[a][b];
    String mutant = "";
    for(int i = 0; i <  s.length(); i++)
    {
        char c = s.charAt(i);
        double p = r.nextDouble();
        if (p < pointMutationRate)
        {

            double q = r.nextDouble();
            if (q < propMutationToNumber)
            {
                c = randomDigit();

            }
            else
            {
                c = randomChar();
            }


        }

  mutant = s.substring(0, i) + c
                       + s.substring(i+1);
```

```
    }
    if (printMut)
        {
            System.out.println("Input = " + s

            + " mutant = " + mutant);

        }
    return mutant;
}


//The population undergoes mutation
private static void mutation()
{
    //Mutants are saved to Pop2[][]
    for(int j = 0; j <  v; j++)
    {
        for(int i = 0; i <  h; i++)
        {
            //
            Pop2[j][i] = mut(j,i);

        }
    }

    //Transfer to Pop[][]
    for(int j = 0; j <  v; j++)
    {
        System.arraycopy(Pop2[j], 0, Pop[j], 0, h);
    }

    //Lethal mutations can cause extinction
    if (isExtinct())
    {
        System.out.println("Population got extinct");
        System.exit(0);
    }
```

```
}


private static boolean isExtinct()
{
  boolean isExtinct = true;

    for(int j = 0; j <  v; j++)
    {
        for(int i = 0; i <  h; i++)
        {
            if (withDiploidy)
            {
                if (isDoubleMutant(Pop[j][i]));
                else //one viable detected
                {
                    isExtinct = false;
                    System.out.println("Viable = "
                            + Pop[j][i]);
                    break;
                }
            }
            else
            {
                if (isMutant(Pop[j][i]));
                else //one viable detected
                {
                    isExtinct = false;
                    System.out.println("Viable = "
                            + Pop[j][i]);
                    break;
                }
            }

        }
    }
    return isExtinct;
```

```
}

private static boolean isNoMutant(String s)
{

    boolean b = !(isMutant(s));

    return b;

}

private static boolean isSingleMutant(String s)
{
    boolean output ;
    int n = s.length();

    String s1 = s.substring(0,3);

    boolean b1 = isMutant(s1);
    if ( n == numberOfChars) output = b1;
    else
    {
        String s2 = s.substring(3);
        boolean b2 = isMutant(s2);
        //Return b1 or b2 but not both, b1 and b2
        output = ((b1 || b2) & (!( b1 & b2)));
    }
    return output;

}

private static boolean isDoubleMutant(String s)
{
    boolean output ;
    int n = s.length();
    String s1 = s.substring(0,3);
    boolean b1 = isMutant(s1);
    boolean b2 = false;
```

```
    if ( n == numberOfChars) output = b1;
     else
    {
        String s2 = s.substring(3);
        b2 = isMutant(s2);
    }
    //Return b1 and b2
    output = b1 & b2;
    return output ;

}

private static boolean isMutant(String s)
{

    int number;
        //Number is retrieved, if that is possible
        try
        {
             number = Integer.parseInt(s);
            if (printIsMutant)
            {
                System.out.println(
                    "    number = " + number);
            }

            return false;
        }
        //inviable mutants are caught here
        catch(NumberFormatException e1)
        {

            if (printIsMutant)
            {
                System.out.print("notNUMBER " );
                System.out.println("Complaint = "
                        + e1.getMessage());
            }
```

```
            return true;
        }

}


//A non double lethal mutant is chosen at random
//
private static String nonDoubleMutant()
{
    boolean go = true;
    String s = "";
    while (go)
    {
        int a = r.nextInt(v);
        int b = r.nextInt(h);
        row = a;
        column = b;
        s = Pop[a][b];
        if ( printNonDoubleMutant)
            System.out.println("Try : individual [ "
                    + a + "]["+ b + "] = " + s);
        //A double mutant will produce an abortion
        if (  isDoubleMutant(s) ) counterOfAbortions++;
        else go = false;
    }
    if ( printNonDoubleMutant)
        System.out.println("Non double mutant = "  + s
        + "\ncounterOfAbortions = "
                + counterOfAbortions);
    return s;
}


//For haploid reproduction
//a non-lethal-mutant is chosen at random
private static String nonMutant()
{
```

```
    boolean go = true;
    String s = "";
    while (go)
    {
        int a = r.nextInt(v);
        int b = r.nextInt(h);
        s = Pop[a][b];

        //A mutant will produce an abortion
        if (  isMutant(s) ) counterOfAbortions++;
        else go = false;
        if ( printNonMutant)
            System.out.println("Try : individual [ "
                    + a + "]["+ b + "] = " + s
            + "\ncounterOfAbortions = "
                    + counterOfAbortions);
    }
    if ( printNonMutant)
        System.out.println("Non  mutant = "  + s);
    return s;
}


//A population consisting of one individual or less
//cannot undergo sexual reproduction
private static boolean oneOrLess()
{
  boolean oneOrLess = true;
  int numberOfViables = 0;

    for(int j = 0; j <  v; j++)
    {
        for(int i = 0; i <  h; i++)
        {
            if (withDiploidy)
            {
                if (isDoubleMutant(Pop[j][i]));
                else //one viable detected
```

```
                {
                    numberOfViables++;
                    if (numberOfViables > 1)
                    {
                        oneOrLess = false;

                    System.out.println("Viable = "
                            + Pop[j][i]);
                    break;
                    }
                }
            }
            else
            {
                if (isMutant(Pop[j][i]));
                else //one viable detected
                {
                    numberOfViables++;
                    if (numberOfViables > 1)
                    {
                        oneOrLess = false;

                        System.out.println("Viable = "
                                + Pop[j][i]);
                        break;
                    }
                }
            }

        }
    }


    return oneOrLess;

}

//Diploid reproduction by random sampling:
```

```
//Two individuals of the old generation are sampled
//at random with a uniform distribution.
//Double lethal mutants are rejected as fathers.
//Each individual produces two gametes,
//one is taken at random.
//Two selected gametes conform a new individual.
private static String offspringDiploid()
{
    if ((withDiploidy) & ( (oneOrLess())
            & (!withReplacement)))
    {
        System.out.println(
                "Sexual reproduction is impossible");
        System.exit(0);
    }
    //First individual
    String s = nonDoubleMutant();
    int rowS = row;
    int columnS = column;
    //Gamete 1 is chosen
    String g1 = s.substring(0, 3);
    //Replace gamete 1 by 2?
    int i = r.nextInt(2);
    if (  i == 1 ) g1 = s.substring(3);


    //Second individual
    String t = "";

    //WithReplacement = mating with itself = cloning
    //Without replacement = mating with another one.
    boolean go = true;
    while (go)
    {
        t = nonDoubleMutant();
        int rowT = row;
    int columnT = column;
        if (withReplacement) break; //job is done
```

```
        else//try again
            if ( (rowS == rowT) & (columnS == columnT))
            {
                if (printCloningDetected)
                    System.out.println(
                        "Cloning detected: Input1  = " + s
                        + " Input2 = " + t);
            }
            else
            {
                go = false;
                if (printCloningDetected)
                    System.out.println(
                        "No cloning : Input1  = " + s
                        + " Input2 = " + t);
            }//job is done
    }


    //Gamete 3 is chosen
    String g3 = t.substring(0, 3);
    //Replace gamete 3 by 4?
    int j = r.nextInt(2);
    if (  j == 1 ) g3 = t.substring(3);
    String output = g1 + g3;
    if (printReproduction)
        System.out.println(
            "Input1  = " + s
            + "  first  gamete = " + g1
            + " Input2 = " + t
            + "   second gamete = " + g3
                        + "  Output = " + output);

    return output;
}
```

```
//Offspring for haploid population.
//One viable individual is chosen at random and cloned.
private static String offspringHaploid()
{
    String s = nonMutant();
    return s;
}


//Offspring for haploid population.
//One viable individual is chosen at random and cloned.
private static String offspring()
{
    String s;
    if (withDiploidy)   s = offspringDiploid();
    else s = offspringHaploid();
    return s;
}


//Population size is maintained constant.
//Parthenogenesis is allowed.
private static void reproduction()
{
    if (printReproduction)
        {
            System.out.println("\nReproduction");
        }
    //Offspring is  saved to Pop2[][]
    for(int j = 0; j <  v; j++)
    {
        for(int i = 0; i <  h; i++)
        {

            Pop2[j][i] = offspring();

        }
    }
```

```
    //Transfer of Pop2[][] to Pop[][]
    for(int j = 0; j <  v; j++)
    {
        for(int i = 0; i <  h; i++)
        {
            //
            Pop[j][i] = Pop2[j][i];

        }
    }
}




public static void main(String[] args)
{
    launch(args);
}



}//End of application  N55 Recombination3
```

70, page 244. The code for a random binary tree follows:

```
//Program N70 TreeViewSample7
//A JavaFX random binary tree
package Programs;



import javafx.application.Application;
```

```
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.concurrent.Task;
import javafx.concurrent.WorkerStateEvent;
import javafx.scene.Scene;
import javafx.scene.control.TreeItem;
import javafx.scene.control.TreeView;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

import java.security.SecureRandom;
import java.util.Random;
import static javafx.application.Application.launch;



public class TreeViewSample7 extends Application
{

public static void main(String[] args)
{
    launch(args);
}

@Override
public void start(Stage stage) throws Exception
{
    TreeView<String> treeView
            = new TreeView<>(createNode("root"));

    HBox hBox = new HBox();
    hBox.getChildren().addAll(treeView);
    Scene scene = new Scene(hBox);
    stage.setScene(scene);
    stage.show();
}
```

```
Random r = new SecureRandom();



private TreeItem<String> createNode(final String b)
{
    return new TreeItem<String>(b)
    {
        private boolean isLeaf;
        private boolean isFirstTimeChildren = true;
        private boolean isFirstTimeLeaf = true;



        @Override
        public ObservableList<TreeItem<String>> getChildren()
        {
            if (isFirstTimeChildren)
            {
              isFirstTimeChildren = false;
              buildChildren(super.getChildren());
            }
            return super.getChildren();
        }



        @Override
        public boolean isLeaf()
        {
            if (isFirstTimeLeaf)
            {
              isFirstTimeLeaf = false;
              //A leaf is terminal node.
              //Recursive construction of tree continues
              //until a leaf is found.
```

```
            //A node is defined as leaf
            //with probability (1/2)^1
            //isLeaf = r.nextBoolean();
            //Probability of terminating branch
            //= (1/2)^6: longer tree
            isLeaf = r.nextBoolean() && r.nextBoolean()
                    && r.nextBoolean() && r.nextBoolean()
                    && r.nextBoolean() && r.nextBoolean();

        }
        return isLeaf;
    }



    private void buildChildren(
      final ObservableList<TreeItem<String>> children)
    {

        final ObservableList<TreeItem<String>> taskChildren
                = FXCollections.observableArrayList();


        if (!this.isLeaf())
        {
            Task<Integer> task = new Task<Integer>()
            {
                @Override
                protected Integer call() throws Exception
                {
                    Integer i;
                    int max = r.nextInt(2);
                    for (i = 0; i < max; i++)
                    {

                        //Original: runs but generates warni
                        /*
                        taskChildren.addAll(new TreeItem[]
```

```
                                {createNode("#" + r.nextInt())});
                            */
                            //Modification that generates no warning:
                            //String is specified inside TreeItem
                            TreeItem<String> c
                                    = createNode("#" + r.nextInt())  ;
                            taskChildren.add(c);
                            //Expand tree after creating node c
                            c.setExpanded(true);

                        }
                    return i;
                }
            };


            task.setOnSucceeded(
                    (WorkerStateEvent workerStateEvent) ->
            {
                children.setAll(taskChildren);
                System.out.println("State "
                        + "of WorkerStateEvent = "
                        + workerStateEvent.toString());
            });
            new Thread(task).start();
        }
    }
  };
}

}//End of program N68 TreeViewSample7
```

74, page 252. We use our encoding of binary trees that was developed in Vol X and use JavaFX to draw them.

```
//Program N74  BinaryBushJavaFX
/*
```

```
 * The representation of binary trees must be
 * as simple as the trees themselves.
 * This program deals with the following notation:
 * each node has an address given by
 * the binary full expansion of its position in the tree,
 * beginning from 0 and 1, the sons of the root,
 * and ending with 1....1,
 * the second son of the last father.
 *
 * The program receives a complete tree
 * in this binary notation
 * and outputs a drawing of the tree using
 * our own methods.
 * The input must be ordered.
 *
 * The present code
 * accepts an input with
 * incomplete content. So, the complete tree
 * is drawn but some nodes may lack
 * corresponding strings.
 *
 * We rely on a natural interpretation of addresses:
 * Address in the tree 0110 means:
 * go  up, down, down and up, 4  generations at all.
 *
 */
package Programs;

import javafx.application.Application;
import javafx.geometry.Point2D;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.ScrollPane;
import javafx.scene.layout.Priority;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Line;
import javafx.scene.text.Text;
```

```
import javafx.stage.Stage;

public class BinaryBushJavaFX extends Application
{

    //This is data for a tree in our binary notation:
    //Even entries are the addresses in the tree of the nodes,
    //odd ones are the names or contents.
    private static final String[] Input
            = {
                "0", "a",
                "1", "b",
                "00", "aa",
                "01", "ab",
                //"10", "ba",
                "11", "bb",
                //"000", "aaa",
                //"001", "aab",
                "010", "aba",
                //"011", "abb",
                //"100", "baa",
                "101", "bab",
                "110", "bba",
                "111", "bbb"
            };

    private static final int InputLength = Input.length;

    private static final int numberOfGenerations = 5;
    //The terminal  leaves define the complete tree
    private static final int numberOfLeaves
            = (int) Math.pow(2, numberOfGenerations);
    //Binary paths to leaves
    private static final String[] pathsToLeaves
            = new String[numberOfLeaves];

    //Relative length of branches of the first U
    private static final int horizontalLength = 2;
```

```
    private static final int verticalLength = 8;

    //The scene
    private static Group root = new Group();

    private static ScrollPane pane = new ScrollPane();
    //Vertical  size of the panel
    private static final int verticalSize =
            (int) (5 * Math.pow(2, numberOfGenerations));
    private static final int horizontalSize = 800;
    //Scale must be a power of 2
    private static final int scale = 16;

    final VBox vb = new VBox();

    private static boolean print = false;

//private static final StackPane root = new StackPane();
//Paths to leaves are constructed as binary strings
private void createPathsToLeaves(int numberOfGenerations)
{
    String s;
    for (Integer i = 0; i < numberOfLeaves; i++)
    {
        s = Integer.toBinaryString(i);
        int l = s.length();
        if (l < numberOfGenerations) {
            for (int k = l; k < numberOfGenerations; k++)
            {
                s = "0" + s;
            }
        }
        pathsToLeaves[i] = s;
        if (print)
            System.out.println(pathsToLeaves[i]);
    }
}
```

```
//Drawing machinery
private void drawTree()
{

    createPathsToLeaves(numberOfGenerations);

    //Symbolic initialization
    int centerX = 1;
    int centerY = 1;

    //Draw stem
    int x00 = scale * centerX;
    int y00 = scale * centerY + numberOfLeaves
            * verticalLength/2;
    int x11 = x00 + scale * horizontalLength;
    int y11 = y00;
    Line l = new Line(x00, verticalSize - y00,
            x11, verticalSize - y11);
    root.getChildren().add(l);

    //Horizontal and basic vertical strokes
    int H = scale * horizontalLength;
    int V;

    String path;
    char direction;

    //The tree is drawn:
    //A path from the origin of the tree to
    //each leaf is drawn
    for (int i = 0; i < numberOfLeaves; i++)
    {
        //Addresses will be used as paths to  be drawn
        path = pathsToLeaves[i];

        //Reinitialization
        Point2D center = new Point2D(x11, y11);
```

```
V = scale * verticalLength;
//Path is interpreted
for (int j = 0; j < path.length(); j++)
{
    direction = path.charAt(j);
    if (direction == '0') //go up
    {
        int x0 = (int) center.getX();
        int y0 = (int) center.getY();
        int x1 = x0;
        int y1 = y0 + V;
        //Draw vertical stroke
        Line l2 = new Line(x0, verticalSize - y0, x1,
                verticalSize - y1);
        root.getChildren().add(l2);
        //Draw horizontal stroke
        Line l3 = new Line(x1, verticalSize - y1,
                x1 + H, verticalSize - y1);
        root.getChildren().add(l3);
        center = new Point2D(center.getX()
                , center.getY() + V);
        //center.add(new Point2D(0,V));
        if (print)
        {
            System.out.println("V = " + V);
            System.out.println("Center = "
                    + (int) center.getX() + "  "
                    + (int) center.getY());
        }

    } else // if direction == '1'  =  go down
    {
        int x0 = (int) center.getX();
        int y0 = (int) center.getY();
        int x1 = x0;

        int y1 = y0 - V;
        //Draw vertical stroke
```

```
            Line l4 = new Line(x0, verticalSize - y0,
                    x1, verticalSize - y1);
            root.getChildren().add(l4);
            //Draw horizontal stroke
            Line l5 = new Line(x1, verticalSize - y1,
                    x1 + H, verticalSize - y1);
            root.getChildren().add(l5);
            center = new Point2D(center.getX()
                    , center.getY() - V);

        } //End of if + else

        center = new Point2D(center.getX() + H
                    , center.getY());

        V = V / 2;
    }//End of for j
}  //End of for: numberOfLeaves

//Strings are drawn:
for (int i = 0; i < InputLength; i = i + 2)
{
    //Addresses will be used as paths to the places
    //where strings must be drawn
    path = Input[i];
    String s = Input[i + 1];
    //Reinitialization
    Point2D center = new Point2D(x00, y00);
    Point2D target = center;
    V = scale * verticalLength;
    //Path is interpreted
    for (int j = 0; j < path.length(); j++) {
        direction = path.charAt(j);
        if (direction == '0') //go up
        {
            target = new Point2D(target.getX()
                    , target.getY() + V);
        } else // if direction == '1'  =  go down
```

```
            {
                target = new Point2D(target.getX()
                        , target.getY() - V);
            }

            target = new Point2D(target.getX() + H
                    , target.getY());

            V = V / 2;
        }//End of for j
        if (print)
            System.out.println("Target = "
                    + (int) target.getX() + "  "
                + (int) target.getY());
        //String is drawn
        Text text = new Text(s);
        text.setX(target.getX() + 1);

        text.setY(verticalSize - target.getY() - 1);
        text.setFill(Color.RED);

        root.getChildren().add(text);
        pane.setContent(root);
        pane.setHbarPolicy(ScrollPane
                .ScrollBarPolicy.ALWAYS);
        pane.setVbarPolicy(ScrollPane
                .ScrollBarPolicy.ALWAYS);
    }
}


@Override
public void start(Stage primaryStage)
{
    VBox box = new VBox();
    Scene scene = new Scene(box
            , horizontalSize, verticalSize);
    primaryStage.setTitle("Binary tree");
```

```
    primaryStage.setScene(scene);
    primaryStage.show();
    drawTree();
    box.getChildren().add(pane);
    //Automatic growth of container
    VBox.setVgrow(pane, Priority.ALWAYS);
}


public static void main(String[] args)
{
    launch(args);
}

}//End of Program  N74 BinaryBushJavaFX
```

75, page 252. A random binary tree is generated by the following program:

```
//Program N75 BinaryBushJavaFX2
/*
 * A random binary tree.
 * Nodes in the tree have an address. Say,
 * Address   0110 means:
 * go  up, down, down and up, 4  generations at all.
 * Each node has a content that is a string.
 * Information is kept in an array, an address followed
 * by the corresponding content.

 * Example:

        private static final String[] TreeInfo
            = {
                "0", "a",
                "1", "b",
                "00", "aa",
```

```
                "01", "ab",
                //"10", "ba",
                "11", "bb",
                //"000", "aaa",
                //"001", "aab",
                "010", "aba",
                //"011", "abb",
                //"100", "baa",
                "101", "bab",
                "110", "bba",
                "111", "bbb"
            };*/

package Programs;


import java.util.Random;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point2D;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.ScrollPane;
import javafx.scene.layout.Priority;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Line;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class BinaryBushJavaFX2 extends Application
{

    //This is data for a tree in our binary notation:
    //Even entries are the addresses in the tree of the nodes,
    //odd ones are the names or contents.
    private static   String[] TreeInfo;
```

```
    private static  int treeInfoLength;

    private static final int numberOfGenerations = 5;
    //The terminal  leaves define the complete tree
    private static final int numberOfLeaves
            = (int) Math.pow(2, numberOfGenerations);
    //Binary paths to leaves
    private static final String[] pathsToLeaves
            = new String[numberOfLeaves];

    //Relative length of branches of the first U
    private static final int horizontalLength = 2;
    private static final int verticalLength = 8;

    //The scene
    private static final Group root = new Group();

    private static final ScrollPane pane = new ScrollPane();
    //Vertical  size of the panel
    private static final int verticalSize =
            (int) (5 * Math.pow(2, numberOfGenerations));
    private static final int horizontalSize = 800;
    //Scale must be a power of 2
    private static final int scale = 16;

    final VBox vb = new VBox();

    private static final boolean print = false;
    private static final Random r = new Random();


//  A string of stringLength characters is generated
//  The alphabet is {0,1}.
private static String binaryAddress(int stringLength)
{
    String individual = "";
    //An individual is assembled char by char
    for(int j = 0; j < stringLength; j++)
```

```
    {
        // We generate a random binary number as Character c
        Integer binary = r.nextInt(2);
        // System.out.println( "binary string = "+ s );
        // s is concatenated to Pop1[i]
        individual  = individual +binary.toString();
    }
    //System.out.println( individual);
    return individual;
}


//A random binary tree is created:
//addresses and contents
private static void createTree()
{
    int n = (int) ( Math.pow(2, numberOfGenerations));
    n = r.nextInt(n);
    TreeInfo = new String[2*n];
    treeInfoLength = TreeInfo.length;

    for(int i = 0; i < 2*n; i= i+2)
    {
       TreeInfo[i] = binaryAddress(5);
       TreeInfo[i+1] = TreeInfo[i];
       TreeInfo[i+1] = TreeInfo[i+1].replace('0', '+');
       TreeInfo[i+1] = TreeInfo[i+1].replace('1', '-');

       System.out.println(i + "   "
               + TreeInfo[i] + "   " + TreeInfo[i+1]);
    }

}

//private static final StackPane root = new StackPane();
//Paths to leaves are constructed as binary strings
private static void
        createPathsToLeaves(int numberOfGenerations)
```

```
{
    String s;
    for (Integer i = 0; i < numberOfLeaves; i++)
    {
        s = Integer.toBinaryString(i);
        int l = s.length();
        if (l < numberOfGenerations) {
            for (int k = l; k < numberOfGenerations; k++)
            {
                s = "0" + s;
            }
        }
        pathsToLeaves[i] = s;
        if (print)
            System.out.println(pathsToLeaves[i]);
    }
}

//Drawing machinery
private static void drawTree()
{

    createTree();
    createPathsToLeaves(numberOfGenerations);

    //Symbolic initialization
    int centerX = 1;
    int centerY = 1;

    //Draw stem
    int x00 = scale * centerX;
    int y00 = scale * centerY + numberOfLeaves
            * verticalLength/2;
    int x11 = x00 + scale * horizontalLength;
    int y11 = y00;
    Line l = new Line(x00, verticalSize - y00,
            x11, verticalSize - y11);
    root.getChildren().add(l);
```

```
//Horizontal and basic vertical strokes
int H = scale * horizontalLength;
int V;

String path;
char direction;

//The tree is drawn:
//A path from the origin of the tree to
//each leaf is drawn
for (int i = 0; i < numberOfLeaves; i++)
{
    //Addresses will be used as paths to  be drawn
    path = pathsToLeaves[i];

    //Reinitialization
    Point2D center = new Point2D(x11, y11);

    V = scale * verticalLength;
    //Path is interpreted
    for (int j = 0; j < path.length(); j++)
    {
        direction = path.charAt(j);
        if (direction == '0') //go up
        {
            int x0 = (int) center.getX();
            int y0 = (int) center.getY();
            int x1 = x0;
            int y1 = y0 + V;
            //Draw vertical stroke
            Line l2 = new Line(x0, verticalSize - y0, x1,
                    verticalSize - y1);
            root.getChildren().add(l2);
            //Draw horizontal stroke
            Line l3 = new Line(x1, verticalSize - y1,
                    x1 + H, verticalSize - y1);
            root.getChildren().add(l3);
```

```
            center = new Point2D(center.getX()
                    , center.getY() + V);
            //center.add(new Point2D(0,V));
            if (print)
            {
                System.out.println("V = " + V);
                System.out.println("Center = "
                        + (int) center.getX() + "  "
                        + (int) center.getY());
            }

        } else // if direction == '1'  =  go down
        {
            int x0 = (int) center.getX();
            int y0 = (int) center.getY();
            int x1 = x0;

            int y1 = y0 - V;
            //Draw vertical stroke
            Line l4 = new Line(x0, verticalSize - y0,
                    x1, verticalSize - y1);
            root.getChildren().add(l4);
            //Draw horizontal stroke
            Line l5 = new Line(x1, verticalSize - y1,
                    x1 + H, verticalSize - y1);
            root.getChildren().add(l5);
            center = new Point2D(center.getX()
                    , center.getY() - V);

        } //End of if + else

        center = new Point2D(center.getX() + H
                    , center.getY());

        V = V / 2;
    }//End of for j
}  //End of for: numberOfLeaves
```

```
//Strings are drawn:
for (int i = 0; i < treeInfoLength; i = i + 2)
{
    //Addresses will be used as paths to the places
    //where strings must be drawn
    path = TreeInfo[i];
    String s = TreeInfo[i + 1];
    //Reinitialization
    Point2D center = new Point2D(x00, y00);
    Point2D target = center;
    V = scale * verticalLength;
    //Path is interpreted
    for (int j = 0; j < path.length(); j++) {
        direction = path.charAt(j);
        if (direction == '0') //go up
        {
            target = new Point2D(target.getX()
                    , target.getY() + V);
        } else // if direction == '1'  =  go down
        {
            target = new Point2D(target.getX()
                    , target.getY() - V);
        }

        target = new Point2D(target.getX() + H
                , target.getY());

        V = V / 2;
    }//End of for j
    if (print)
        System.out.println("Target = "
                + (int) target.getX() + "  "
            + (int) target.getY());
    //String is drawn
    Text text = new Text(s);
    text.setX(target.getX() + 1);

    text.setY(verticalSize - target.getY() - 1);
```

```
        text.setFill(Color.RED);

        root.getChildren().add(text);
        pane.setContent(root);
        pane.setHbarPolicy(ScrollPane
                .ScrollBarPolicy.ALWAYS);
        pane.setVbarPolicy(ScrollPane
                .ScrollBarPolicy.ALWAYS);
    }
}


@Override
public void start(Stage primaryStage)
{
    VBox box = new VBox();
    Scene scene = new Scene(box
            , horizontalSize, verticalSize);
    primaryStage.setTitle("Binary tree");
    primaryStage.setScene(scene);
    primaryStage.show();
    drawTree();
    box.getChildren().add(pane);
    //Automatic growth of container
    VBox.setVgrow(pane, Priority.ALWAYS);
}


public static void main(String[] args)
{
    launch(args);
}

}//End of Program  N75 BinaryBushJavaFX2
```

76, page 252. A random binary bush (tree suitable to include the depiction of extinction events) is generated by the following program:

```
//Program N76 BinaryBushJavaFX3
/*
 * A random binary bush (tree that includes extinction events).
 * Nodes in the tree have an address. Say,
 * Address   0110 means:
 * go  up, down, down and up, 4  generations at all.
 * Each node has a content that is a string.
 * Information is kept in an array, an address followed
 * by the corresponding content.

 * Example:

        private static final String[] TreeInfo
            = {
                "0", "a",
                "1", "b",
                "00", "aa",
                "01", "ab",
                //"10", "ba",
                "11", "bb",
                //"000", "aaa",
                //"001", "aab",
                "010", "aba",
                //"011", "abb",
                //"100", "baa",
                "101", "bab",
                "110", "bba",
                "111", "bbb"
            };*/

package Programs;


import java.util.Random;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point2D;
import javafx.scene.Group;
```

```
import javafx.scene.Scene;
import javafx.scene.control.ScrollPane;
import javafx.scene.layout.Priority;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Line;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class BinaryBushJavaFX3 extends Application
{

    //This is data for a tree in our binary notation:
    //Even entries are the addresses in the tree of the nodes,
    //odd ones are the names or contents.
    private static   String[] TreeInfo;

    private static  int treeInfoLength;

    private static final int numberOfGenerations = 5;
    //The terminal  leaves define the complete tree
    private static final int numberOfLeaves
            = (int) Math.pow(2, numberOfGenerations);
    //Binary paths to leaves
    private static final String[] pathsToLeaves
            = new String[numberOfLeaves];

    //Relative length of branches of the first U
    private static final int horizontalLength = 3;
    private static final int verticalLength = 8;

    //The scene
    private static final Group root = new Group();

    private static final ScrollPane pane = new ScrollPane();
    //Vertical  size of the panel
    private static final int verticalSize =
            (int) (5 * Math.pow(2, numberOfGenerations));
```

```
    private static final int horizontalSize = 800;
    //Scale must be a power of 2
    private static final int scale = 16;

    final VBox vb = new VBox();

    private static final boolean print = false;
    private static final Random r = new Random();


//  A string of stringLength characters is generated
//  The alphabet is {0,1}.
//  Random length with a mximal value of maxLength.
private static String binaryAddress(int maxLength)
{
    String individual = "";
    int stringLength = r.nextInt(maxLength);
    //Test
    //stringLength = maxLength;
    //An individual is assembled char by char
    for(int j = 0; j < stringLength; j++)
    {
        // We generate a random binary number as Character c
        Integer binary = r.nextInt(2);
        // System.out.println( "binary string = "+ s );
        // s is concatenated to Pop1[i]
        individual  = individual +binary.toString();
    }
    //System.out.println( individual);
    return individual;
}


//A random binary tree is created:
//addresses and contents
private static void createTree()
{
```

```
    int n = r.nextInt(numberOfLeaves);
    //Test
    //n = numberOfLeaves;
    TreeInfo = new String[2*n];
    treeInfoLength = TreeInfo.length;

    for(int i = 0; i < 2*n; i= i+2)
    {
       TreeInfo[i] = binaryAddress(5);
       TreeInfo[i+1] = TreeInfo[i];
       TreeInfo[i+1] = TreeInfo[i+1].replace('0', '+');
       TreeInfo[i+1] = TreeInfo[i+1].replace('1', '-');

       System.out.println(i + "   "
              + TreeInfo[i] + "   " + TreeInfo[i+1]);
    }

}


//All paths to leaves are constructed as binary strings
private static void
        createPathsToLeaves(int numberOfGenerations)
{
    String s;
    for (Integer i = 0; i < numberOfLeaves; i++)
    {
        s = Integer.toBinaryString(i);
        int l = s.length();
        if (l < numberOfGenerations) {
            for (int k = l; k < numberOfGenerations; k++)
            {
                s = "0" + s;
            }
        }
        pathsToLeaves[i] = s;
        if (print)
            System.out.println(pathsToLeaves[i]);
```

```
    }
}

//Drawing machinery
private static void drawTree()
{

    createTree();
    createPathsToLeaves(numberOfGenerations);

    //Symbolic initialization
    int centerX = 1;
    int centerY = 1;

    //Draw stem
    int x00 = scale * centerX;
    int y00 = scale * centerY + numberOfLeaves
            * verticalLength/2;
    int x11 = x00 + scale * horizontalLength;
    int y11 = y00;
    Line l = new Line(x00, verticalSize - y00,
            x11, verticalSize - y11);
    root.getChildren().add(l);

    //Horizontal and basic vertical strokes
    int H = scale * horizontalLength;
    int V;

    String path;
    char direction;

    //The tree is drawn:
    //A path from the origin of the tree to
    //each leaf is drawn
    for (int i = 0; i < numberOfLeaves; i++)
    {
        //Addresses will be used as paths to  be drawn
        path = pathsToLeaves[i];
```

```
//Reinitialization
Point2D center = new Point2D(x11, y11);

V = scale * verticalLength;
//Path is interpreted
for (int j = 0; j < path.length(); j++)
{
    direction = path.charAt(j);
    if (direction == '0') //go up
    {
        int x0 = (int) center.getX();
        int y0 = (int) center.getY();
        int x1 = x0;
        int y1 = y0 + V;
        //Draw vertical stroke
        Line l2 = new Line(x0, verticalSize - y0, x1,
                verticalSize - y1);
        root.getChildren().add(l2);
        //Draw horizontal stroke
        Line l3 = new Line(x1, verticalSize - y1,
                x1 + H, verticalSize - y1);
        root.getChildren().add(l3);
        center = new Point2D(center.getX()
                , center.getY() + V);
        //center.add(new Point2D(0,V));
        if (print)
        {
            System.out.println("V = " + V);
            System.out.println("Center = "
                    + (int) center.getX() + "  "
                    + (int) center.getY());
        }

    } else // if direction == '1'  =  go down
    {
        int x0 = (int) center.getX();
        int y0 = (int) center.getY();
```

```
                int x1 = x0;

                int y1 = y0 - V;
                //Draw vertical stroke
                Line l4 = new Line(x0, verticalSize - y0,
                        x1, verticalSize - y1);
                root.getChildren().add(l4);
                //Draw horizontal stroke
                Line l5 = new Line(x1, verticalSize - y1,
                        x1 + H, verticalSize - y1);
                root.getChildren().add(l5);
                center = new Point2D(center.getX()
                        , center.getY() - V);

            } //End of if + else

            center = new Point2D(center.getX() + H
                        , center.getY());

            V = V / 2;
        }//End of for j
    }  //End of for: numberOfLeaves

    //Strings are drawn:
    for (int i = 0; i < treeInfoLength; i = i + 2)
    {
        //Addresses will be used as paths to the places
        //where strings must be drawn
        path = TreeInfo[i];
        String s = TreeInfo[i + 1];
        //Reinitialization
        Point2D center = new Point2D(x00, y00);
        Point2D target = center;
        V = scale * verticalLength;
        //Path is interpreted
        for (int j = 0; j < path.length(); j++) {
            direction = path.charAt(j);
            if (direction == '0') //go up
```

```
            {
                target = new Point2D(target.getX()
                        , target.getY() + V);
            } else // if direction == '1'  =  go down
            {
                target = new Point2D(target.getX()
                        , target.getY() - V);
            }

            target = new Point2D(target.getX() + H
                    , target.getY());

            V = V / 2;
        }//End of for j
        if (print)
            System.out.println("Target = "
                    + (int) target.getX() + "  "
                + (int) target.getY());
        //String is drawn
        Text text = new Text(s);
        text.setX(target.getX() + 1);

        text.setY(verticalSize - target.getY() - 1);
        text.setFill(Color.RED);

        root.getChildren().add(text);
        pane.setContent(root);
        pane.setHbarPolicy(ScrollPane
                .ScrollBarPolicy.ALWAYS);
        pane.setVbarPolicy(ScrollPane
                .ScrollBarPolicy.ALWAYS);
    }
}


@Override
public void start(Stage primaryStage)
{
```

```
    VBox box = new VBox();
    Scene scene = new Scene(box
            , horizontalSize, verticalSize);
    primaryStage.setTitle("Binary tree");
    primaryStage.setScene(scene);
    primaryStage.show();
    drawTree();
    box.getChildren().add(pane);
    //Automatic growth of container
    VBox.setVgrow(pane, Priority.ALWAYS);
}


public static void main(String[] args)
{
    launch(args);
}

}//End of Program  N76 BinaryBushJavaFX3
```

79, page 260. A fullbush is implemented over lists to make the way for deletion:

```
//Program N79 BinaryBushJavaFX5
/*
 * A random binary fullbush
 * (with all leaves and all internal nodes).
 * Nodes in the tree have an address. Say,
 * Address   0110 means:
 * go  up, down, down and up, 4  generations at all.
 * Each node has a content that is a string.
 * Information is kept in an array, an address followed
 * by the corresponding content.

 * Example:
```

```
        private static final String[] TreeInfo
            = {
                "0", "a",
                "1", "b",
                "00", "aa",
                "01", "ab",
                //"10", "ba",
                "11", "bb",
                //"000", "aaa",
                //"001", "aab",
                "010", "aba",
                //"011", "abb",
                //"100", "baa",
                "101", "bab",
                "110", "bba",
                "111", "bbb"
            };
 * To generate addresses we generate
 * random integers that are translated to binary strings.

*/
package Programs;


import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point2D;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.ScrollPane;
import javafx.scene.layout.Priority;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Line;
import javafx.scene.text.Text;
```

```
import javafx.stage.Stage;

public class BinaryBushJavaFX5 extends Application
{

    //This is data for a tree in our binary notation:
    //Even entries are the addresses in the tree of the nodes,
    //odd ones are the names or contents.
    private static   List<String> TreeInfo;

    private static  int treeInfoLength;

    private static final int numberOfGenerations = 5;
    //The terminal  leaves define the complete tree
    private static final int numberOfLeaves
            = (int) Math.pow(2, numberOfGenerations);
    //Binary paths to leaves
    private static final String[] pathsToLeaves
            = new String[numberOfLeaves];

    //Relative length of branches of the first U
    private static final int horizontalLength = 3;
    private static final int verticalLength = 8;

    //The scene
    private static final Group root = new Group();

    private static final ScrollPane pane = new ScrollPane();
    //Vertical  size of the panel
    private static final int verticalSize =
            (int) (5 * Math.pow(2, numberOfGenerations));
    private static final int horizontalSize = 800;
    //Scale must be a power of 2
    private static final int scale = 16;

    final VBox vb = new VBox();

    private static final boolean print = false;
```

```
    private static final Random r = new Random();


//A random binary tree is created:
//addresses and contents
private static void createTree()
{

    //int n = r.nextInt(numberOfLeaves);
    //Test
    int n = numberOfLeaves;
    TreeInfo = new  ArrayList<>(2*n);


    for(int i = 0; i < 2*n; i= i+2)
    {
       TreeInfo.add(Integer.toBinaryString(i/2));



        String s = TreeInfo.get(i).replace('0', '+');
        s = s.replace('1', '-');
        TreeInfo.add(  s);


        System.out.println(i/2 + "   "
            + TreeInfo.get(i) + "   " + TreeInfo.get(i+1));
    }

    for(int i = 2*n; i < 4*n; i= i+2)
    {
        int k = i -2*n;
        String s = Integer.toBinaryString(k/2);
        s = s.replace('0', 'a');
        s = s.replace('1', '0');
        s = s.replace('a', '1');
        TreeInfo.add(s);
```

```
            s = TreeInfo.get(i).replace('0', '+');
            s = s.replace('1', '-');
            TreeInfo.add(  s);

            System.out.println(i/2 + "   "
                    + TreeInfo.get(i) + "   "
                    + TreeInfo.get(i+1));
        }
        treeInfoLength = TreeInfo.size();
    }


    //All paths to leaves are constructed as binary strings
    private static void
            createPathsToLeaves(int numberOfGenerations)
    {
        String s;
        for (Integer i = 0; i < numberOfLeaves; i++)
        {
            s = Integer.toBinaryString(i);
            int l = s.length();
            if (l < numberOfGenerations) {
                for (int k = l; k < numberOfGenerations; k++)
                {
                    s = "0" + s;
                }
            }
            pathsToLeaves[i] = s;
            if (print)
                System.out.println(pathsToLeaves[i]);
        }
    }

    //Drawing machinery
    private static void drawTree()
    {

        createTree();
```

```
createPathsToLeaves(numberOfGenerations);

//Symbolic initialization
int centerX = 1;
int centerY = 1;

//Draw stem
int x00 = scale * centerX;
int y00 = scale * centerY + numberOfLeaves
        * verticalLength/2;
int x11 = x00 + scale * horizontalLength;
int y11 = y00;
Line l = new Line(x00, verticalSize - y00,
        x11, verticalSize - y11);
root.getChildren().add(l);

//Horizontal and basic vertical strokes
int H = scale * horizontalLength;
int V;

String path;
char direction;

//The tree is drawn:
//A path from the origin of the tree to
//each leaf is drawn
for (int i = 0; i < numberOfLeaves; i++)
{
    //Addresses will be used as paths to  be drawn
    path = pathsToLeaves[i];

    //Reinitialization
    Point2D center = new Point2D(x11, y11);

    V = scale * verticalLength;
    //Path is interpreted
    for (int j = 0; j < path.length(); j++)
    {
```

```
direction = path.charAt(j);
if (direction == '0') //go up
{
    int x0 = (int) center.getX();
    int y0 = (int) center.getY();
    int x1 = x0;
    int y1 = y0 + V;
    //Draw vertical stroke
    Line l2 = new Line(x0, verticalSize - y0, x1,
            verticalSize - y1);
    root.getChildren().add(l2);
    //Draw horizontal stroke
    Line l3 = new Line(x1, verticalSize - y1,
            x1 + H, verticalSize - y1);
    root.getChildren().add(l3);
    center = new Point2D(center.getX()
            , center.getY() + V);
    //center.add(new Point2D(0,V));
    if (print)
    {
        System.out.println("V = " + V);
        System.out.println("Center = "
                + (int) center.getX() + "  "
                + (int) center.getY());
    }

} else // if direction == '1'  =  go down
{
    int x0 = (int) center.getX();
    int y0 = (int) center.getY();
    int x1 = x0;

    int y1 = y0 - V;
    //Draw vertical stroke
    Line l4 = new Line(x0, verticalSize - y0,
            x1, verticalSize - y1);
    root.getChildren().add(l4);
    //Draw horizontal stroke
```

```
            Line l5 = new Line(x1, verticalSize - y1,
                    x1 + H, verticalSize - y1);
            root.getChildren().add(l5);
            center = new Point2D(center.getX()
                    , center.getY() - V);

        } //End of if + else

        center = new Point2D(center.getX() + H
                    , center.getY());

        V = V / 2;
    }//End of for j
}  //End of for: numberOfLeaves

//Strings are drawn:
for (int i = 0; i < treeInfoLength; i = i + 2)
{
    //Addresses will be used as paths to the places
    //where strings must be drawn
    path = TreeInfo.get(i);
    String s = TreeInfo.get(i+1);
    //Reinitialization
    Point2D center = new Point2D(x00, y00);
    Point2D target = center;
    V = scale * verticalLength;
    //Path is interpreted
    for (int j = 0; j < path.length(); j++)
    {
        direction = path.charAt(j);
        if (direction == '0') //go up
        {
            target = new Point2D(target.getX()
                    , target.getY() + V);
        } else // if direction == '1'  =  go down
        {
            target = new Point2D(target.getX()
                    , target.getY() - V);
```

```
            }

            target = new Point2D(target.getX() + H
                    , target.getY());

            V = V / 2;
        }//End of for j
        if (print)
            System.out.println("Target = "
                    + (int) target.getX() + "  "
                + (int) target.getY());
        //String is drawn
        Text text = new Text(s);
        text.setX(target.getX() + 1);

        text.setY(verticalSize - target.getY() - 1);
        text.setFill(Color.RED);

        root.getChildren().add(text);
        pane.setContent(root);
        pane.setHbarPolicy(ScrollPane
                .ScrollBarPolicy.ALWAYS);
        pane.setVbarPolicy(ScrollPane
                .ScrollBarPolicy.ALWAYS);
    }
}


@Override
public void start(Stage primaryStage)
{
    VBox box = new VBox();
    Scene scene = new Scene(box
            , horizontalSize, verticalSize);
    primaryStage.setTitle("Binary tree");
    primaryStage.setScene(scene);
    primaryStage.show();
    drawTree();
```

```
    box.getChildren().add(pane);
    //Automatic growth of container
    VBox.setVgrow(pane, Priority.ALWAYS);
}


public static void main(String[] args)
{
    launch(args);
}

}//End of Program  N79 BinaryBushJavaFX5
```

80, page 260. A fullbush is implemented over lists and the drawing method gets subordinated to the information of the tree.

```
//Program N80 BinaryBushJavaFX6
/*
 * A random binary fullbush
 * (with all leaves and all internal nodes).
 * Nodes in the tree have an address. Say,
 * Address   0110 means:
 * go  up, down, down and up, 4  generations at all.
 * Each node has a content that is a string.
 * Information is kept in an array, an address followed
 * by the corresponding content.

 * Example:

        private static final String[] TreeInfo
            = {
                "0", "a",
                "1", "b",
                "00", "aa",
                "01", "ab",
                //"10", "ba",
                "11", "bb",
```

```
                //"000", "aaa",
                //"001", "aab",
                "010", "aba",
                //"011", "abb",
                //"100", "baa",
                "101", "bab",
                "110", "bba",
                "111", "bbb"
            };
 * To generate addresses we generate
 * random integers that are translated to binary strings.

*/
package Programs;


import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point2D;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.ScrollPane;
import javafx.scene.layout.Priority;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Line;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class BinaryBushJavaFX6 extends Application
{

    //This is data for a tree in our binary notation:
    //Even entries are the addresses in the tree of the nodes,
    //odd ones are the names or contents.
```

```
    private static   List<String> TreeInfo;

    private static  int treeInfoLength;

    private static final int numberOfGenerations = 5;
    //The terminal  leaves define the complete tree
    private static final int numberOfLeaves
            = (int) Math.pow(2, numberOfGenerations);

    //Relative length of branches of the first U
    private static final int horizontalLength = 3;
    private static final int verticalLength = 8;

    //The scene
    private static final Group root = new Group();

    private static final ScrollPane pane = new ScrollPane();
    //Vertical  size of the panel
    private static final int verticalSize =
            (int) (5 * Math.pow(2, numberOfGenerations));
    private static final int horizontalSize = 800;
    //Scale must be a power of 2
    private static final int scale = 16;

    final VBox vb = new VBox();

    private static final boolean print = false;
    private static final Random r = new Random();


//A random binary tree is created:
//addresses and contents
private static void createTree()
{

    //int n = r.nextInt(numberOfLeaves);
    //Test
    int n = numberOfLeaves;
```

```
    TreeInfo = new  ArrayList<>(2*n);

    //Generation of the lower part of the tree
    System.out.println("Lower part");
    for(int i = 0; i < 2*n; i= i+2)
    {

        TreeInfo.add(Integer.toBinaryString(i/2));
        String s = TreeInfo.get(i).replace('0', '+');
        s = s.replace('1', '-');
        TreeInfo.add(  s);
        System.out.println(i/2 + "   "
            + TreeInfo.get(i) + "   " + TreeInfo.get(i+1));
    }
    //Generation of the upper part of the tree
    System.out.println("Upper part");
    //1 and 0  must not be repeated.
    for(int i = 4; i < 2*n; i= i+2)
    {

        String s = Integer.toBinaryString(i/2);
        s = s.replace('0', 'a');
        s = s.replace('1', '0');
        s = s.replace('a', '1');
        TreeInfo.add(s);

        s = s.replace('0', '+');
        s = s.replace('1', '-');
        TreeInfo.add(  s);

        System.out.println((2*n + i-4)/2 + "   "
                + TreeInfo.get(2*n + i-4) + "   "
                + TreeInfo.get(2*n + i-4+1));
    }
    treeInfoLength = TreeInfo.size();
}
```

```
//Drawing machinery
private static void drawTree()
{

    createTree();


    //Symbolic initialization
    int centerX = 1;
    int centerY = 1;

    //Draw stem
    int x00 = scale * centerX;
    int y00 = scale * centerY + numberOfLeaves
            * verticalLength/2;
    int x11 = x00 + scale * horizontalLength;
    int y11 = y00;
    Line l = new Line(x00, verticalSize - y00,
            x11, verticalSize - y11);
    root.getChildren().add(l);

    //Horizontal and basic vertical strokes
    int H = scale * horizontalLength;
    int V;

    String path;
    char direction;

    //The tree is drawn:
    //A path from the origin of the tree to
    //each node is drawn
    for (int i = 0; i < TreeInfo.size(); i= i+2)
    {
        //Addresses will be used as paths to  be drawn
        path = TreeInfo.get(i);
```

```
//Reinitialization
Point2D center = new Point2D(x11, y11);

V = scale * verticalLength;
//Path is interpreted
for (int j = 0; j < path.length(); j++)
{
    direction = path.charAt(j);
    if (direction == '0') //go up
    {
        int x0 = (int) center.getX();
        int y0 = (int) center.getY();
        int x1 = x0;
        int y1 = y0 + V;
        //Draw vertical stroke
        Line l2 = new Line(x0, verticalSize - y0, x1,
                verticalSize - y1);
        root.getChildren().add(l2);
        //Draw horizontal stroke
        Line l3 = new Line(x1, verticalSize - y1,
                x1 + H, verticalSize - y1);
        root.getChildren().add(l3);
        center = new Point2D(center.getX()
                , center.getY() + V);
        //center.add(new Point2D(0,V));
        if (print)
        {
            System.out.println("V = " + V);
            System.out.println("Center = "
                    + (int) center.getX() + "  "
                    + (int) center.getY());
        }

    } else // if direction == '1'  =  go down
    {
        int x0 = (int) center.getX();
        int y0 = (int) center.getY();
```

```
            int x1 = x0;

            int y1 = y0 - V;
            //Draw vertical stroke
            Line l4 = new Line(x0, verticalSize - y0,
                    x1, verticalSize - y1);
            root.getChildren().add(l4);
            //Draw horizontal stroke
            Line l5 = new Line(x1, verticalSize - y1,
                    x1 + H, verticalSize - y1);
            root.getChildren().add(l5);
            center = new Point2D(center.getX()
                    , center.getY() - V);

        } //End of if + else

        center = new Point2D(center.getX() + H
                    , center.getY());

        V = V / 2;
    }//End of for j
}  //End of for: numberOfLeaves

//Strings are drawn:
for (int i = 0; i < treeInfoLength; i = i + 2)
{
    //Addresses will be used as paths to the places
    //where strings must be drawn
    path = TreeInfo.get(i);
    String s = TreeInfo.get(i+1);
    //Reinitialization
    Point2D center = new Point2D(x00, y00);
    Point2D target = center;
    V = scale * verticalLength;
    //Path is interpreted
    for (int j = 0; j < path.length(); j++)
    {
        direction = path.charAt(j);
```

```
            if (direction == '0') //go up
            {
                target = new Point2D(target.getX()
                        , target.getY() + V);
            } else // if direction == '1'  =  go down
            {
                target = new Point2D(target.getX()
                        , target.getY() - V);
            }

            target = new Point2D(target.getX() + H
                    , target.getY());

            V = V / 2;
        }//End of for j
        if (print)
            System.out.println("Target = "
                    + (int) target.getX() + "  "
                + (int) target.getY());
        //String is drawn
        Text text = new Text(s);
        text.setX(target.getX() + 1);

        text.setY(verticalSize - target.getY() - 1);
        text.setFill(Color.RED);

        root.getChildren().add(text);
        pane.setContent(root);
        pane.setHbarPolicy(ScrollPane
                .ScrollBarPolicy.ALWAYS);
        pane.setVbarPolicy(ScrollPane
                .ScrollBarPolicy.ALWAYS);
    }
}


@Override
public void start(Stage primaryStage)
```

```
{
    VBox box = new VBox();
    Scene scene = new Scene(box
            , horizontalSize, verticalSize);
    primaryStage.setTitle("Binary tree");
    primaryStage.setScene(scene);
    primaryStage.show();
    drawTree();
    box.getChildren().add(pane);
    //Automatic growth of container
    VBox.setVgrow(pane, Priority.ALWAYS);
}


public static void main(String[] args)
{
    launch(args);
}

}//End of Program  N80 BinaryBushJavaFX6
```

81, page 260. The next code presents a bush in which branches can have any length up to a given maximum. Nevertheless, if a node exists then all ancestors must exist too. Or equivalently, if a node is deleted, then its forward subtree is also deleted.

```
//Program N81 BinaryBushJavaFX7
/*
 * A random binary  bush.
 * A fullbush (a tree with all leaves and all internal nodes)
 * is constructed from which some subtrees are removed.
 * This simulates evolution when extinction is possible.

 * Nodes in the tree have an address. Say,
 * Address   0110 means:
 * go  up, down, down and up, 4  generations at all.
 * Each node has a content that is a string.
```

```
 * Information is kept in an array, an address followed
 * by the corresponding content.

 * Example:

        private static final String[] TreeInfo
            = {
                "0", "a",
                "1", "b",
                "00", "aa",
                "01", "ab",
                //"10", "ba",
                "11", "bb",
                //"000", "aaa",
                //"001", "aab",
                "010", "aba",
                //"011", "abb",
                //"100", "baa",
                "101", "bab",
                "110", "bba",
                "111", "bbb"
            };
 * To generate addresses we generate
 * random integers that are translated to binary strings.

*/
package Programs;


import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point2D;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.ScrollPane;
```

```
import javafx.scene.layout.Priority;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Line;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class BinaryBushJavaFX7 extends Application
{

    //This is data for a tree in our binary notation:
    //Even entries are the addresses in the tree of the nodes,
    //odd ones are the names or contents.
    private static   List<String> TreeInfo;

    private static  int treeInfoLength;

    private static final int numberOfGenerations = 5;
    //The terminal  leaves define the complete tree
    private static final int numberOfLeaves
            = (int) Math.pow(2, numberOfGenerations);

    //Relative length of branches of the first U
    private static final int horizontalLength = 3;
    private static final int verticalLength = 8;

    //The scene
    private static final Group root = new Group();

    private static final ScrollPane pane = new ScrollPane();
    //Vertical  size of the panel
    private static final int verticalSize =
            (int) (5 * Math.pow(2, numberOfGenerations));
    private static final int horizontalSize = 800;
    //Scale must be a power of 2
    private static final int scale = 16;

    final VBox vb = new VBox();
```

```
    private static final boolean print = false;
    private static final Random r = new Random();

    //Number of listed nodes
    private static int nNodes;

//A random binary tree is created:
//addresses and contents
private static void createTree()
{

    //int n = r.nextInt(numberOfLeaves);
    //Test
    int n = numberOfLeaves;
    TreeInfo = new  ArrayList<>(2*n);

    //Generation of the lower part of the tree
    System.out.println("Lower part");
    for(int i = 0; i < 2*n; i= i+2)
    {

        TreeInfo.add(Integer.toBinaryString(i/2));
        String s = TreeInfo.get(i).replace('0', '+');
        s = s.replace('1', '-');
        TreeInfo.add(  s);
        System.out.println(i/2 + "   "
            + TreeInfo.get(i) + "   " + TreeInfo.get(i+1));
    }
    //Generation of the upper part of the tree
    System.out.println("Upper part");
    //1 and 0  must not be repeated.
    for(int i = 4; i < 2*n; i= i+2)
    {

        String s = Integer.toBinaryString(i/2);
        s = s.replace('0', 'a');
        s = s.replace('1', '0');
```

```
        s = s.replace('a', '1');
        TreeInfo.add(s);

        s = s.replace('0', '+');
        s = s.replace('1', '-');
        TreeInfo.add(  s);

        System.out.println((2*n + i-4)/2 + "   "
                + TreeInfo.get(2*n + i-4) + "   "
                + TreeInfo.get(2*n + i-4+1));
    }

    extinction();

    treeInfoLength = TreeInfo.size();
}

//If a node listed at place l must be removed,
//the forward subtree is also deleted
private static void removeSubtree(int l)
{
    String address = TreeInfo.get(l);
    int size0 = TreeInfo.size();
    //All nodes are backwards scanned for removing
    for(int i = size0-2; i > -1;  i = i-2)
    {
        //If a node is 10001,
        //a descendant has address 10001+something
        if (TreeInfo.get(i).contains(address))
        {
            //Address is removed
            TreeInfo.remove(i);
            //Content is removed
            TreeInfo.remove(i);
        }
    }
}
```

```
//All listed nodes of the tree are subjected to
//deletion with probability e, the same for every node.
private static void extinction()
{

    //Probability of extinction
    //uniformly distributed over all listed nodes
    double e = r.nextDouble();
    //Number of deletions of subtrees
    int nDeletions = 10;
    Integer place;

    for(int i = 0; i < nDeletions; i++)
    {
        nNodes = TreeInfo.size()/2;
        if (nNodes > 0)
        {
            place = 2*r.nextInt(nNodes);
            removeSubtree(place);
        }
    }
}


//Drawing machinery
private static void drawTree()
{

    createTree();


    //Symbolic initialization
    int centerX = 1;
    int centerY = 1;

    //Draw stem
    int x00 = scale * centerX;
```

```
int y00 = scale * centerY + numberOfLeaves
        * verticalLength/2;
int x11 = x00 + scale * horizontalLength;
int y11 = y00;
Line l = new Line(x00, verticalSize - y00,
        x11, verticalSize - y11);
root.getChildren().add(l);

//Horizontal and basic vertical strokes
int H = scale * horizontalLength;
int V;

String path;
char direction;

//The tree is drawn:
//A path from the origin of the tree to
//each node is drawn
for (int i = 0; i < TreeInfo.size(); i= i+2)
{
    //Addresses will be used as paths to  be drawn
    path = TreeInfo.get(i);


    //Reinitialization
    Point2D center = new Point2D(x11, y11);

    V = scale * verticalLength;
    //Path is interpreted
    for (int j = 0; j < path.length(); j++)
    {
        direction = path.charAt(j);
        if (direction == '0') //go up
        {
            int x0 = (int) center.getX();
            int y0 = (int) center.getY();
            int x1 = x0;
            int y1 = y0 + V;
```

```
            //Draw vertical stroke
            Line l2 = new Line(x0, verticalSize - y0, x1,
                    verticalSize - y1);
            root.getChildren().add(l2);
            //Draw horizontal stroke
            Line l3 = new Line(x1, verticalSize - y1,
                    x1 + H, verticalSize - y1);
            root.getChildren().add(l3);
            center = new Point2D(center.getX()
                    , center.getY() + V);
            //center.add(new Point2D(0,V));
            if (print)
            {
                System.out.println("V = " + V);
                System.out.println("Center = "
                        + (int) center.getX() + "  "
                        + (int) center.getY());
            }

        } else // if direction == '1'  =  go down
        {
            int x0 = (int) center.getX();
            int y0 = (int) center.getY();
            int x1 = x0;

            int y1 = y0 - V;
            //Draw vertical stroke
            Line l4 = new Line(x0, verticalSize - y0,
                    x1, verticalSize - y1);
            root.getChildren().add(l4);
            //Draw horizontal stroke
            Line l5 = new Line(x1, verticalSize - y1,
                    x1 + H, verticalSize - y1);
            root.getChildren().add(l5);
            center = new Point2D(center.getX()
                    , center.getY() - V);

        } //End of if + else
```

```
        center = new Point2D(center.getX() + H
                    , center.getY());

        V = V / 2;
    }//End of for j
}  //End of for: numberOfLeaves

//Strings are drawn:
for (int i = 0; i < treeInfoLength; i = i + 2)
{
    //Addresses will be used as paths to the places
    //where strings must be drawn
    path = TreeInfo.get(i);
    String s = TreeInfo.get(i+1);
    //Reinitialization
    Point2D center = new Point2D(x00, y00);
    Point2D target = center;
    V = scale * verticalLength;
    //Path is interpreted
    for (int j = 0; j < path.length(); j++)
    {
        direction = path.charAt(j);
        if (direction == '0') //go up
        {
            target = new Point2D(target.getX()
                    , target.getY() + V);
        } else // if direction == '1'  =  go down
        {
            target = new Point2D(target.getX()
                    , target.getY() - V);
        }

        target = new Point2D(target.getX() + H
                , target.getY());

        V = V / 2;
    }//End of for j
```

```
        if (print)
            System.out.println("Target = "
                    + (int) target.getX() + "  "
                + (int) target.getY());
        //String is drawn
        Text text = new Text(s);
        text.setX(target.getX() + 1);

        text.setY(verticalSize - target.getY() - 1);
        text.setFill(Color.RED);

        root.getChildren().add(text);

    }
}


@Override
public void start(Stage primaryStage)
{
    VBox box = new VBox();
    Scene scene = new Scene(box
            , horizontalSize, verticalSize);
    primaryStage.setTitle("Binary tree");
    primaryStage.setScene(scene);
    primaryStage.show();
    drawTree();
    pane.setContent(root);
    pane.setHbarPolicy(ScrollPane
            .ScrollBarPolicy.ALWAYS);
    pane.setVbarPolicy(ScrollPane
                .ScrollBarPolicy.ALWAYS);
    pane.setPrefSize(horizontalSize, verticalSize);
    box.getChildren().add(pane);
    //Automatic growth of container
    VBox.setVgrow(pane, Priority.ALWAYS);
}
```

```
public static void main(String[] args)
{
    launch(args);
}

}//End of Program  N81 BinaryBushJavaFX7
```

82, page 261. A 1 to 1 correspondence of binary bushes and binary strings is exhibited by the following code:

```
/*
 * Program N82 BinaryBushJavaFX8
 * A bush is completely defined by  a binary string.
 * This program generates a bush, encodes it
 * as a binary string,  recalculates and draws the tree
 * associated to that string.
 * One can know at once if the program works by comparing
 * the two drawings: they must coincide.
 *
 * Explanation:
 * Nodes in the tree have an address. Say,
 * Address   0110 means:
 * go  up, down, down and up, 4  generations at all.
 * Each node has a content that is a string.
 * Information is kept in an array, an address followed
 * by the corresponding content.

 * This system of encoding allows to represent
 * the whole set of binary addresses by a binary string

 * Example:

        private static final String[] TreeInfo
            = {
                "0", "a",
                "1", "b",
                "00", "aa",
```

```
                "01", "ab",
                //"10", "ba",
                "11", "bb",
                //"000", "aaa",
                //"001", "aab",
                "010", "aba",
                //"011", "abb",
                //"100", "baa",
                "101", "bab",
                "110", "bba",
                "111", "bbb"
            };
 * Each address is a binary number that is in 1 to 1
 *corresondence with ordered integers beginning from
 * 0 up to the number of nodes.
 * So, in a former program, to generate addresses
 * we generated  random integers that were translated
 * to binary strings.

 * Integers are naturally numbered.
 * If a node is listed, the correspondent integer also.
 * This allows to define a tree by a binary string:
 * The null string represents the null tree.
 * If node I exists in a tree, then place I in the string
 * is set to 1.

*/
package Programs;




import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point2D;
```

```
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.ScrollPane;
import javafx.scene.layout.Priority;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Line;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class BinaryBushJavaFX8 extends Application
{

    //This is data for a tree in our binary notation:
    //Even entries are the addresses in the tree of the nodes,
    //odd ones are the names or contents.
    private static   List<String> TreeInfo;

    private static  int treeInfoLength;

    private static final int numberOfGenerations = 5;
    //The terminal  leaves define the complete tree
    private static final int numberOfLeaves
            = (int) Math.pow(2, numberOfGenerations);

    //Relative length of branches of the first U
    private static final int horizontalLength = 3;
    private static final int verticalLength = 8;

    //The scene
    private static final Group root = new Group();

    private static final ScrollPane pane = new ScrollPane();
    //Vertical  size of the panel
    private static final int verticalSize =
            (int) (5 * Math.pow(2, numberOfGenerations));
    private static final int horizontalSize = 800;
    //Scale must be a power of 2
```

```
    private static final int scale = 16;

    final VBox vb = new VBox();

    private static final boolean print = false;
    private static final boolean print2 = false;
    private static final boolean print3 = false;
    private static final boolean print4 = false;
    private static final boolean print5 = false;
    private static final Random r = new Random();

    //Number of listed nodes
    private static int nNodes;
    //Symbolic initialization
    private static int centerX = 1;
    private static final int centerY = 1;


    private static void printTreeInfo()
{
    System.out.println("TreeInfo");
    int l = TreeInfo.size();
    for(int i = 0; i < l; i = i+2)
    {
        System.out.println(i/2 + "   " + TreeInfo.get(i)
        + "   " + TreeInfo.get(i+1));
    }

}


//A random binary tree is created:
//addresses and contents
private static void createTree()
{

    //int n = r.nextInt(numberOfLeaves);
    //Test
```

```
int n = numberOfLeaves;
TreeInfo = new  ArrayList<>(2*n);

//Generation of the lower part of the tree
if (print)
    System.out.println("Lower part of full tree");
for(int i = 0; i < 2*n; i= i+2)
{

    TreeInfo.add(Integer.toBinaryString(i/2));
    String s = TreeInfo.get(i).replace('0', '+');
    s = s.replace('1', '-');
    TreeInfo.add(  s);
    if (print)
        System.out.println(i/2 + "   "
        + TreeInfo.get(i) + "   " + TreeInfo.get(i+1));
}
//Generation of the upper part of the tree
if (print)
    System.out.println("Upper part of full tree");
//1 and 0  must not be repeated.
for(int i = 4; i < 2*n; i= i+2)
{

    String s = Integer.toBinaryString(i/2);
    s = s.replace('0', 'a');
    s = s.replace('1', '0');
    s = s.replace('a', '1');
    TreeInfo.add(s);

    s = s.replace('0', '+');
    s = s.replace('1', '-');
    TreeInfo.add(  s);
    if (print)
        System.out.println((2*n + i-4)/2 + "   "
            + TreeInfo.get(2*n + i-4) + "   "
            + TreeInfo.get(2*n + i-4+1));
}
```

```
    extinction();

    treeInfoLength = TreeInfo.size();
}

//If a node listed at place l must be removed,
//the forward subtree is also deleted
private static void removeSubtree(int l)
{
    String address = TreeInfo.get(l);
    int size0 = TreeInfo.size();
    //All nodes are backwards scanned for removing
    for(int i = size0-2; i > -1;  i = i-2)
    {
        //If a node is 10001,
        //a descendant has address 10001+something
        if (TreeInfo.get(i).contains(address))
        {
            //Address is removed
            TreeInfo.remove(i);
            //Content is removed
            TreeInfo.remove(i);
        }
    }
}


//All listed nodes of the tree are subjected to
//deletion with probability e, the same for every node.
private static void extinction()
{

    //Probability of extinction
    //uniformly distributed over all listed nodes
    double e = r.nextDouble();
    //Number of deletions of subtrees
    int nDeletions = 5;
```

```
    Integer place;

    for(int i = 0; i < nDeletions; i++)
    {
        nNodes = TreeInfo.size()/2;
        if (nNodes > 0)
        {
            place = 2*r.nextInt(nNodes);
            removeSubtree(place);
        }
    }
}


//Drawing machinery
private static void drawTree()
{
    //Draw stem
    int x00 = scale * centerX;
    int y00 = scale * centerY + numberOfLeaves
            * verticalLength/2;
    int x11 = x00 + scale * horizontalLength;
    int y11 = y00;
    Line l = new Line(x00, verticalSize - y00,
            x11, verticalSize - y11);
    root.getChildren().add(l);

    //Horizontal and basic vertical strokes
    int H = scale * horizontalLength;
    int V;

    String path;
    char direction;

    //The tree is drawn:
    //A path from the origin of the tree to
    //each node is drawn
    for (int i = 0; i < TreeInfo.size(); i= i+2)
```

```
{
    //Addresses will be used as paths to  be drawn
    path = TreeInfo.get(i);


    //Reinitialization
    Point2D center = new Point2D(x11, y11);

    V = scale * verticalLength;
    //Path is interpreted
    for (int j = 0; j < path.length(); j++)
    {
        direction = path.charAt(j);
        if (direction == '0') //go up
        {
            int x0 = (int) center.getX();
            int y0 = (int) center.getY();
            int x1 = x0;
            int y1 = y0 + V;
            //Draw vertical stroke
            Line l2 = new Line(x0, verticalSize - y0, x1,
                    verticalSize - y1);
            root.getChildren().add(l2);
            //Draw horizontal stroke
            Line l3 = new Line(x1, verticalSize - y1,
                    x1 + H, verticalSize - y1);
            root.getChildren().add(l3);
            center = new Point2D(center.getX()
                    , center.getY() + V);
            //center.add(new Point2D(0,V));
            if (print2)
            {
                System.out.println("V = " + V);
                System.out.println("Center = "
                        + (int) center.getX() + "  "
                        + (int) center.getY());
            }
```

```
        } else // if direction == '1'  =  go down
        {
            int x0 = (int) center.getX();
            int y0 = (int) center.getY();
            int x1 = x0;

            int y1 = y0 - V;
            //Draw vertical stroke
            Line l4 = new Line(x0, verticalSize - y0,
                    x1, verticalSize - y1);
            root.getChildren().add(l4);
            //Draw horizontal stroke
            Line l5 = new Line(x1, verticalSize - y1,
                    x1 + H, verticalSize - y1);
            root.getChildren().add(l5);
            center = new Point2D(center.getX()
                    , center.getY() - V);

        } //End of if + else

        center = new Point2D(center.getX() + H
                    , center.getY());

        V = V / 2;
    }//End of for j
}  //End of for: numberOfLeaves

//Strings are drawn:
for (int i = 0; i < treeInfoLength; i = i + 2)
{
    //Addresses will be used as paths to the places
    //where strings must be drawn
    path = TreeInfo.get(i);
    String s = TreeInfo.get(i+1);
    //Reinitialization
    Point2D center = new Point2D(x00, y00);
    Point2D target = center;
    V = scale * verticalLength;
```

```
        //Path is interpreted
        for (int j = 0; j < path.length(); j++)
        {
            direction = path.charAt(j);
            if (direction == '0') //go up
            {
                target = new Point2D(target.getX()
                        , target.getY() + V);
            } else // if direction == '1'  =  go down
            {
                target = new Point2D(target.getX()
                        , target.getY() - V);
            }

            target = new Point2D(target.getX() + H
                    , target.getY());

            V = V / 2;
        }//End of for j
        if (print2)
            System.out.println("Target = "
                    + (int) target.getX() + "  "
                + (int) target.getY());
        //String is drawn
        Text text = new Text(s);
        text.setX(target.getX() + 1);

        text.setY(verticalSize - target.getY() - 1);
        text.setFill(Color.RED);

        root.getChildren().add(text);

    }
}

//Binary string s is decoded into an int
private static int stringToNumber(String s)
{
```

```
    int n = 0;
    int l = s.length();

    for(int j=0;j  < l;j++ )
    {
        char c  =   s.charAt(j);
        int  v = Character.getNumericValue(c);
        int q = j+1;
        int number1 = (int) (v * Math.pow(2,l-q));
        n  = n + number1;
    }
    return n;
}


private static int stringToInt(String address)
{
    int output = 0;
    //System.out.print("String = " + address );
    char c  = address.charAt(0);
    if  (c == '1') output = stringToNumber(address);
    if  (c == '0')
    {
        if (address.length() == 1) output = 0;
            else
        {
        address = address.replace('0', 'a');
        address = address.replace('1', '0');
        address = address.replace('a', '1');
        output = stringToNumber(address) + numberOfLeaves -2;
        }
    }
    //System.out.println("  int =  " + output);
    return output;
}


//A binary string is associated to the tree
```

```
//in a 1 to 1 fashion.
private static String treeToBinaryString()
{
    //This string is filled in zeros:
    //It represents the null tree
    String bitTree = "";
    for (int i = 0; i < 2*numberOfLeaves-2; i++)
        bitTree = bitTree + "0";
    if (print3)
        System.out.println(bitTree);
    String address;
    //Each listed address  corresponds
    //with an integer. The corresponding bit
    //in bitTree is set to 1.
    for (int i = 0; i < TreeInfo.size(); i= i+2)
    {

        address = TreeInfo.get(i);
        int n = stringToInt(address);
        if (print3)
            System.out.println("String = " + address
                    + " int =  " + n);
        if (n < bitTree.length()-1)
         bitTree = bitTree.substring(0, n)
                + "1" + bitTree.substring(n+1);
        else bitTree = bitTree.substring(0, n)
                + "1" ;
        if (print3)
            System.out.println(bitTree);

    }

    return bitTree;
}


//The tree encoded by binary string b is
//decoded and drawn
```

```
private static void redrawTree(String b)
{
    TreeInfo.clear();
    int l = b.length();
    char c ;
    String s ;
    for (Integer i = 0; i < l; i++)
    {
        c = b.charAt(i);
        if (c == '1')
        {
            if ( i < numberOfLeaves)
            {
                s = Integer.toBinaryString(i);
                TreeInfo.add(s);
                s = s.replace('0', '+');
                s = s.replace('1', '-');
                TreeInfo.add(s);
            }

            else
            {
                s = Integer.toBinaryString(i
                        - numberOfLeaves +2);
                s = s.replace('0', 'a');
                s = s.replace('1', '0');
                s = s.replace('a', '1');
                TreeInfo.add(s);

                s = s.replace('0', '+');
                s = s.replace('1', '-');
                TreeInfo.add(  s);
            }

        }
    }
    treeInfoLength = TreeInfo.size();
    centerX = 40;
```

```
    drawTree();
}


@Override
public void start(Stage primaryStage)
{
    VBox box = new VBox();
    Scene scene = new Scene(box
            , horizontalSize, verticalSize);
    primaryStage.setTitle("Binary tree");
    primaryStage.setScene(scene);
    primaryStage.show();
    createTree();
    centerX = 10;
    System.out.println("After extinction");
    printTreeInfo();
    drawTree();

    String treeToBinaryString = treeToBinaryString();
    System.out.println("treeToBinaryString = \n"
                + treeToBinaryString);
    redrawTree(treeToBinaryString);

    pane.setContent(root);
    pane.setHbarPolicy(ScrollPane
            .ScrollBarPolicy.ALWAYS);
    pane.setVbarPolicy(ScrollPane
                .ScrollBarPolicy.ALWAYS);
    pane.setPrefSize(horizontalSize, verticalSize);
    box.getChildren().add(pane);
    //Automatic growth of container
    VBox.setVgrow(pane, Priority.ALWAYS);
}


public static void main(String[] args)
{
```

```
    launch(args);
}

}//End of Program  N82 BinaryBushJavaFX8
```

91, page 290. The program presents a 2D random binary bush that is inserted in 3D space. If a node appears, all its ancestors too.

```
/*
   Program N91  TreeToShape3
   The program presents
   a 2D random binary bush
   that is inserted in 3D space.
   If a node appears, all its ancestors too.
*/

package Programs;



//Files in the hosting project
import Tools.Axes;
import Tools.Cameras;
import Tools.Xform;
//Java files

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point2D;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Line;
```

```
import javafx.scene.text.Text;
import javafx.stage.Stage;



public class TreeToShape3 extends Application
{
    private static final int numberOfGenerations = 5;
    //The terminal  leaves define the complete tree
    private static final int numberOfLeaves
            = (int) Math.pow(2, numberOfGenerations);

    //This is data for a tree in our binary notation:
    //Even entries are the addresses in the tree of the nodes,
    //odd ones are the names or contents.
    private static final  List<String> TreeInfo
            = new  ArrayList<>(2*numberOfLeaves-1);

    private static int treeInfoLength;
    //Binary string that says whether or not  node i in bit i
    //is present
    private static String bitTree = "";



    private static final String title
            =  "2D bush inserted into 3D space";


    final Group root = new Group();
    private static final Cameras camera = new Cameras();
    final double cameraDistance = 5450;//450
    //Axes : ctrl + A
    private static final Axes axes = new Axes();
    final Xform world3DXform = new Xform();
    final Xform bonesXform = new Xform();

    //Generator of pseudo random numbers
```

```
    private static final Random r = new Random();

    private static final boolean print = false;
    private static final boolean print2 = false;
    private static final boolean print3 = false;

    //Vertical  size of the panel
    private static final int verticalSize =
            (int) (5 * Math.pow(2, numberOfGenerations));

    //Scale must be a power of 2
    private static final int scale = 16;
    //Symbolic initialization
    private static final int centerX = 1;
    private static final int centerY = 1;

    //Relative length of branches of the first U
    private static final int horizontalLength = 3;
    private static final int verticalLength = 8;

    final VBox vb = new VBox();



@Override
public void start(Stage primaryStage)
{
    primaryStage.setTitle(title);
    Scene scene = new Scene(root, 1200, 1000, true);

    scene.setFill(Color.AQUAMARINE);
    primaryStage.setScene(scene);
    primaryStage.show();
    axes.buildAxes();
    axes.handleKeyboard(scene);
    world3DXform.getChildren().addAll( axes.axesXform);

    camera.handleMouse(scene);
```

```
    camera.buildCamera();
    scene.setCamera(camera.camera);

    //Creation of a tree in two forms:
    //As a binary string, bitTree, that contains
    //the information of which nodes are present and
    //as a list with binary addresses to nodes in the tree
    //followed by their content.
    //To deal with strings of the form 00001,
    //the tree itself is inserted to the lower part
    //of a bigger tree with one more generation.
    createTree(  );
    printTreeInfo();

    Group parent = new Group(create3dContent(TreeInfo));
        world3DXform.getChildren().addAll(parent);
        root.getChildren().add(world3DXform);
    primaryStage.show();
    play();
}

public void play()
{

}

@Override public void stop()
{

}


private Node create3dContent(List<String> TreeInfo)
{
    Project3D  p =  new Project3D(TreeInfo);
    return new Group(p);
}
```

```
private static void printTreeInfo()
{
    System.out.println("TreeInfo");

    int l = TreeInfo.size();
    for(int i = 0; i < l; i = i+2)
    {
        System.out.println(i/2 + "   " + TreeInfo.get(i)
        + "   " + TreeInfo.get(i+1));
    }
}


//A binary string with 2*numberOfLeaves-1 bits
private static String randomBinaryString()
{

    for (int i = 0; i < 2*numberOfLeaves-1; i++)
    {
        Integer j = r.nextInt(2);
        bitTree = bitTree + j.toString();

    }

    if (print3)
        System.out.println(bitTree);
    return bitTree;
}

//Binary string s is decoded into an int
private static int stringToNumber(String s)
{
    int n = 0;
    int l = s.length();
```

```
    for(int j=0;j  < l;j++ )
    {
        char c  =   s.charAt(j);
        int  v = Character.getNumericValue(c);
        int q = j+1;
        int number1 = (int) (v * Math.pow(2,l-q));
        n  = n + number1;
    }
    return n;
}


private static int stringToInt(String address)
{
    int output = 0;
    //System.out.print("String = " + address );
    char c  = address.charAt(0);
    if  (c == '1') output = stringToNumber(address);
        else System.out.println("Bad encoding");
    //System.out.println("  int =  " + output);
    return output;
}

//All ancestors of node m are added
private  void  addAncestors(int m)
{
    String address = TreeInfo.get(m);
    int a = address.length();
    //If the address of  a node is 10001,
    //an ancestor has as address a left substring
    //of 0001, say 0, 00 or 000.
    //All of them must be added to TreeInfo

    for(int i = 0; i < a;  i++)
    {
        String leftSubstring
                = address.substring(0, i);
        leftSubstring = "1" + leftSubstring;
```

```
        int k = stringToInt(leftSubstring);
        //Updating of tree information
        if (k < bitTree.length()-1)
            bitTree = bitTree.substring(0, k)
                + "1" + bitTree.substring(k+1);
        else bitTree = bitTree.substring(0, k)
                + "1" ;
    }
    //Tree information is re-created from the bitTree
    TreeInfo.clear();
    createTree(bitTree);
}

/*

//If a node listed at place l must be removed,
//the forward subtree is also deleted
private static void removeSubtree(int l)
{

}



*/
//The tree encoded by binary string b is
//decoded into an array with binary addresses and contents.
//The tree corresponds with the lower
//part of a tree with one more generation.
private  void  createTree(String b )
{


    int l = b.length();
    char c ;
    String s ;
    //First two bits do not matter
    for (Integer i = 2; i < l; i++)
```

```
    {
        c = b.charAt(i);
        if (c == '1')
        {
            if ( i < 2*numberOfLeaves-1)
            {
                s = Integer.toBinaryString(i);
                //prefix "1" is deleted
                s = s.substring(1);
                TreeInfo.add(s);
                s = s.replace('0', '+');
                s = s.replace('1', '-');
                TreeInfo.add(s);
            }
        }
    }



}

private  void  createTree(  )
{
    //TreeInfo is filled in:
    //A binary address to each node of the tree
    //is following by a content
    bitTree = randomBinaryString();
    createTree(bitTree);
    //If a node exists, all its ancestors too
    for(int m = 0; m < TreeInfo.size()-1;   m = m+2)
        addAncestors(m);

    createTree(bitTree);
    treeInfoLength = TreeInfo.size();

}
```

```
//=======================================================
//======= Beginning of Inner class Project3D ===========
//=======================================================



//The drawing is synthesized
private class Project3D extends Group
{

    //Constructor
    Project3D(List<String> TreeInfo)
    {
        System.out.println("List<String> TreeInfo");

        //Tree is drawn to expand a 3D bush
        Group g = drawTree(TreeInfo);
        getChildren().addAll(g );
    }

   private   Group drawTree(List<String> TreeInfo)
{
    Group myGroup  = new Group();
        //Draw stem
    int x00 = scale * centerX;
    int y00 = scale * centerY + numberOfLeaves
            * verticalLength/2;
    int x11 = x00 + scale * horizontalLength;
    int y11 = y00;
    Line l = new Line(x00, verticalSize - y00,
            x11, verticalSize - y11);
    myGroup.getChildren().add(l);

    //Horizontal and basic vertical strokes
    int H = scale * horizontalLength;
    int V;

    String path;
```

```
    char direction;

    //The tree is drawn:
    //A path from the origin of the tree to
    //each node is drawn
    for (int i = 0; i < TreeInfo.size(); i= i+2)
    {
        //Addresses will be used as paths to  be drawn
        path = TreeInfo.get(i);


        //Reinitialization
        Point2D center = new Point2D(x11, y11);

        V = scale * verticalLength;
        //Path is interpreted
        for (int j = 0; j < path.length(); j++)
        {
            direction = path.charAt(j);
            if (direction == '0') //go up
            {
                int x0 = (int) center.getX();
                int y0 = (int) center.getY();
                int x1 = x0;
                int y1 = y0 + V;
                //Draw vertical stroke
                Line l2 = new Line(x0, verticalSize - y0, x1,
                        verticalSize - y1);
                myGroup.getChildren().add(l2);
                //Draw horizontal stroke
                Line l3 = new Line(x1, verticalSize - y1,
                        x1 + H, verticalSize - y1);
                myGroup.getChildren().add(l3);
                center = new Point2D(center.getX()
                        , center.getY() + V);
                //center.add(new Point2D(0,V));
                if (print2)
                {
```

```
                System.out.println("V = " + V);
                System.out.println("Center = "
                        + (int) center.getX() + "  "
                        + (int) center.getY());
            }

        } else // if direction == '1'  =  go down
        {
            int x0 = (int) center.getX();
            int y0 = (int) center.getY();
            int x1 = x0;

            int y1 = y0 - V;
            //Draw vertical stroke
            Line l4 = new Line(x0, verticalSize - y0,
                    x1, verticalSize - y1);
            myGroup.getChildren().add(l4);
            //Draw horizontal stroke
            Line l5 = new Line(x1, verticalSize - y1,
                    x1 + H, verticalSize - y1);
            myGroup.getChildren().add(l5);
            center = new Point2D(center.getX()
                    , center.getY() - V);

        } //End of if + else

        center = new Point2D(center.getX() + H
                    , center.getY());

        V = V / 2;
    }//End of for j
}  //End of for: numberOfLeaves

//Strings are drawn:
for (int i = 0; i < treeInfoLength; i = i + 2)
{
    //Addresses will be used as paths to the places
    //where strings must be drawn
```

```
path = TreeInfo.get(i);
String s = TreeInfo.get(i+1);
//Reinitialization
Point2D center = new Point2D(x00, y00);
Point2D target = center;
V = scale * verticalLength;
//Path is interpreted
for (int j = 0; j < path.length(); j++)
{
    direction = path.charAt(j);
    if (direction == '0') //go up
    {
        target = new Point2D(target.getX()
                , target.getY() + V);
    } else // if direction == '1'  =  go down
    {
        target = new Point2D(target.getX()
                , target.getY() - V);
    }

    target = new Point2D(target.getX() + H
            , target.getY());

    V = V / 2;
}//End of for j
if (print2)
    System.out.println("Target = "
            + (int) target.getX() + "  "
        + (int) target.getY());
//String is drawn
Text text = new Text(s);
text.setX(target.getX() + 1);

text.setY(verticalSize - target.getY() - 1);
text.setFill(Color.RED);

root.getChildren().add(text);
```

```
    }
    return myGroup;
}

}//End of inner class Project3D


public static void main(String[] args)
{
    launch(args);
}

}//End of programN91  TreeToShape3
```

100, page 330. Let us see why N96 TreeToShape5 is evolvable while N94 TreeToShape4 is not. In first place, the code of program N94 TreeToShape4 have repetitions that were cleaned in version N96 TreeToShape5. Evolve a code is overburden by repetitions because one needs to repeat changes but taking care of interference from other parts of the code. In second place, the simplest manner of drawing a bush given the form as these are encoded is by drawing a path to a given node. Now, a path consists of a concatenation of linear strokes. Concatenation is possible by both programs but it is natural only in program N96 TreeToShape5: one step is naturally followed by another thanks to the *lineTo(thisPoint)* method. By contrast, concatenation in the other program results from the independent drawing of different strokes. This works but is not natural and besides a repetition of points also appears augmenting the work to evolve it.

101, page 331. The following code presents a 3D self-intersecting rectangular bundle of paths whose strokes are parallel to one of the natural axes.

```
/*
    Program N101  TreeToShape7
    A random 3D self intersecting
    rectangular bundle of paths.
*/
```

```
package Programs;


//Files in the hosting project
import Tools.Axes;
import Tools.Cameras;
import Tools.Line3D;
import Tools.Xform;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Sphere;
import javafx.stage.Stage;



public class TreeToShape7 extends Application
{

    //This is data for a tree in our binary notation:
    //Even entries are the addresses in the tree of the nodes,
    //odd ones are the names or contents.
    private static   List<String> TreeInfo;



    private static final int numberOfGenerations = 5;
    //The terminal  leaves define the complete tree
    private static final int numberOfLeaves
            = (int) Math.pow(2, numberOfGenerations);
```

```
//Printing for debugging
final static boolean printReadFile = false;
private static final String title
        =  "3D bush";

//Binary string that says whether or not  node i in bit i
//is present
private static String bitTree = "";

final Group root = new Group();
private static final Cameras camera = new Cameras();
final double cameraDistance = 5450;//450
//Axes : ctrl + A
private static final Axes axes = new Axes();
final Xform world3DXform = new Xform();
final Xform bonesXform = new Xform();

//Generator of pseudo random numbers
private static final Random r = new Random();


private static final boolean print = false;

//Vertical  size of the panel
private static final int verticalSize =
        (int) (5 * Math.pow(2, numberOfGenerations));
//Scale must be a power of 2
private static final int scale = 16;
//Symbolic initialization
private static final int centerX = 1;
private static final int centerY = 1;

//Relative length of branches of the first U
private static final int horizontalLength = 3;
private static final int verticalLength = 8;

private static Point3D lastPoint;
```

```
    private static final int delta = 50;

    private static int direction;




@Override
public void start(Stage primaryStage)
{

    primaryStage.setTitle(title);
    Scene scene = new Scene(root, 1200, 1000, true);

    scene.setFill(Color.AQUAMARINE);
    primaryStage.setScene(scene);
    primaryStage.show();
    axes.buildAxes();
    axes.handleKeyboard(scene);
    world3DXform.getChildren().addAll( axes.axesXform);

    camera.handleMouse(scene);
    camera.buildCamera();
    scene.setCamera(camera.camera);

    //TreeInfo is filled in:
    //A binary address to each node of the tree
    //is following by a content
    createTree();
    printTreeInfo();
    //Tree is drawn
    Group parent = new Group(create3dContent(TreeInfo));
        world3DXform.getChildren().addAll(parent);
        root.getChildren().add(world3DXform);


    primaryStage.show();
```

```
    play();


}

public void play()
{

}

@Override public void stop()
{

}




private Node create3dContent(List<String> TreeInfo)
{
    Project3D  p =  new Project3D(TreeInfo);
    return new Group(p);
}







private static void printTreeInfo()
{
    System.out.println("TreeInfo");

    int l = TreeInfo.size();
    for(int i = 0; i < l; i = i+2)
    {
```

```
        System.out.println(i/2 + "   " + TreeInfo.get(i)
        + "   " + TreeInfo.get(i+1));
    }

}



//A binary string with 2*numberOfLeaves-1 bits
private static String randomBinaryString()
{
    //A random binary string
    bitTree = "";
    for (int i = 0; i < 2*numberOfLeaves-1; i++)
    {
        Integer j = r.nextInt(2);
        bitTree = bitTree + j.toString();

    }

    if (print)
        System.out.println(bitTree);
    return bitTree;
}

//Binary string s is decoded into an int
private static int stringToNumber(String s)
{
    int n = 0;
    int l = s.length();

    for(int j=0;j  < l;j++ )
    {
        char c  =   s.charAt(j);
        int  v = Character.getNumericValue(c);
        int q = j+1;
        int number1 = (int) (v * Math.pow(2,l-q));
        n  = n + number1;
```

```
    }
    return n;
}


private static int stringToInt(String address)
{
    int output = 0;
    //System.out.print("String = " + address );
    char c  = address.charAt(0);
    if  (c == '1') output = stringToNumber(address);
        else System.out.println("Bad encoding");
    //System.out.println("  int =  " + output);
    return output;
}


//All ancestors of node m are added
private  void  addAncestors(int m)
{
    String address = TreeInfo.get(m);
    int a = address.length();
    //If the address of  a node is 10001,
    //an ancestor has as address a left substring
    //of 0001, say 0, 00 or 000.
    //All of them must be added to TreeInfo

    for(int i = 0; i < a;  i++)
    {
        String leftSubstring
                = address.substring(0, i);
        leftSubstring = "1" + leftSubstring;
        int k = stringToInt(leftSubstring);
        //Updating of tree information
        if (k < bitTree.length()-1)
            bitTree = bitTree.substring(0, k)
                + "1" + bitTree.substring(k+1);
        else bitTree = bitTree.substring(0, k)
```

```
                + "1" ;
    }

}

//All ancestors of node m are deleted
private  void  deleteAncestors(int m)
{
    String address = TreeInfo.get(m);
    int a = address.length();
    //If the address of  a node is 10001,
    //an ancestor has as address a left substring
    //of 0001, say 0, 00 or 000.
    //All of them must be deleted from TreeInfo

    for(int i = 0; i < a;  i++)
    {
        String leftSubstring
                = address.substring(0, i);
        leftSubstring = "1" + leftSubstring;
        int k = stringToInt(leftSubstring);
        //Updating of tree information
        if (k < bitTree.length()-1)
            bitTree = bitTree.substring(0, k)
                + "0" + bitTree.substring(k+1);
        else bitTree = bitTree.substring(0, k)
                + "0" ;
    }

}

//The tree encoded by binary string b is
//decoded into an array with binary addresses and contents.
//The tree corresponds with the lower
//part of a tree with one more generation.
private  void  createTree(String b )
{
```

```
    int l = b.length();
    char c ;
    String s ;
    //First two bits do not matter
    for (Integer i = 2; i < l; i++)
    {
        c = b.charAt(i);
        if (c == '1')
        {
            if ( i < 2*numberOfLeaves)
            {
                s = Integer.toBinaryString(i);
                //prefix "1" is deleted
                s = s.substring(1);
                TreeInfo.add(s);
                s = s.replace('0', '+');
                s = s.replace('1', '-');
                TreeInfo.add(s);
            }
        }
    }
}


//The tree encoded by binary string b is
//decoded and drawn.
//The tree corresponds with the lower
//part of a tree with one more generation.
private  void  createTree( )
{
    int n = numberOfLeaves;
    TreeInfo = new  ArrayList<>(2*n-1);
    //A random binary string:
    //If bit I is 1, the I node is added to the tree
    bitTree = randomBinaryString();
    createTree(bitTree) ;
```

```
    /*
    //If a node exists, all its ancestors too
    for(int m = 0; m < TreeInfo.size()-1;   m = m+2)
        addAncestors(m);
    */

    //Simplify tree:
    //If a node exists, all its ancestors are erased
    for(int m = 0; m < TreeInfo.size()-1;   m = m+2)
        deleteAncestors(m);

    //Tree information is re-created from the bitTree
    TreeInfo.clear();
    /*
    //Test: full tree
    bitTree = "";
    for (int i = 0; i < 2*numberOfLeaves; i++)
    {

        bitTree = bitTree + "1";

    }
    */
    createTree(bitTree);

}


//Returns a 3D line that goes from lastPoint
//to thisPoint
private  Line3D  line3DTo(Point3D thisPoint)
{
    Line3D l = new Line3D(lastPoint, thisPoint);
    lastPoint = thisPoint;
    return l;
}

//=====================================================
```

```
//======= Beginning of Inner class Project3D ==========
//=====================================================



//The drawing is synthesized
private class Project3D extends Group
{

    //Constructor
    Project3D(List<String> TreeInfo)
    {
        System.out.println("List<String> TreeInfo");
        //Tree is drawn to expand a 3D bush
        Group g = drawTree(TreeInfo);
        getChildren().addAll(g );
    }

    private int distinctInt(int d)
    {
        int i = r.nextInt(6);
        //System.out.println(i + "  " + d);
        if (i == d) distinctInt(d);
        direction = i;
        return i;
    }

    /*
    There are 6 directions:
    0: +X, 1 = -X,  2 = +Y,  3 = -Y
    4 = +Z, 5 = -Z.
    One distinct than d is chosen
    */
    private  Point3D  randomAddition(int d)
    {
        int i = distinctInt(d);
        Point3D p = new Point3D(0,0,0);
        switch (i)
```

```
        {
            case 0: p = lastPoint.add( delta,0,0); break;
            case 1: p = lastPoint.add(-delta,0,0); break;
            case 2: p = lastPoint.add(0, delta,0); break;
            case 3: p = lastPoint.add(0,-delta,0); break;
            case 4: p = lastPoint.add(0,0, delta); break;
            case 5: p = lastPoint.add(0,0,-delta); break;
        }
        return p;
    }

   private   Group drawTree(List<String> TreeInfo)
{
    Group myGroup  = new Group();
    //Draw stem
    int x00 = scale * centerX;
    int y00 = verticalSize - scale * centerY - numberOfLeaves
            * verticalLength/2;
    //Horizontal and basic vertical strokes
    int H = scale * horizontalLength;

    int x11 = x00 + H;
    int y11 = y00;
    //The stalk is drawn
    Point3D P0=  new Point3D(x00, y00,0);
    Point3D   P1 = P0.add(H, 0,0); //+X direction
    /*
    There are 6 directions:
    0: +X, 1 = -X,  2 = +Y,  3 = -Y
    4 = +Z, 5 = -Z
    */
    direction = r.nextInt(6);

    String path;


    //Put a landmark at the root
    Sphere sphere = new Sphere();
```

```
    sphere.setRadius(10);
    sphere.setTranslateX((int) P1.getX());
    sphere.setTranslateY((int) P1.getY() );
    sphere.setTranslateZ((int) P1.getZ() );
    root.getChildren().add(sphere);


    //The tree is drawn:
    //A random path from the origin of the tree
    for (int i = 0; i < TreeInfo.size(); i= i+2)
    {
        //Addresses will be used as paths to  be drawn
        path = TreeInfo.get(i);
        //Reinitialization
        lastPoint = P1;
        Point3D thisPoint;
        //Path is created at random.
        //Two strokes at a time
        for (int j = 0; j < path.length(); j++)
        {
            thisPoint = randomAddition(direction);
            //Draw a stroke
            //from lastPoint to thisPoint
            Line3D l2 = line3DTo(thisPoint);
            thisPoint = randomAddition(direction);
            //Draw a seciond stroke
            Line3D l3 = line3DTo(thisPoint);
            myGroup.getChildren().addAll(l2.group, l3.group);
        }//End of for j
    }  //End of for: numberOfLeaves

    return myGroup;
}

}//End of inner class Project3D


public static void main(String[] args)
```

```
{
    launch(args);
}

}//End of program N101  TreeToShape7
```

102, page 331. The following is the code for a self-intersecting rectangular random binary tree whose arms extends randomly in 3D:

```
/*
    Program N102  TreeToShape8
    A rectangular random 3D self intersecting binary tree.
*/

package Programs;


//Files in the hosting project
import Tools.Axes;
import Tools.Cameras;
import Tools.Line3D;
import Tools.Xform;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Sphere;
import javafx.stage.Stage;
```

```
public class TreeToShape8 extends Application
{

    private static final int numberOfGenerations = 5;
    //The terminal  leaves define the complete tree
    private static final int numberOfLeaves
            = (int) Math.pow(2, numberOfGenerations);

    //This is data for a tree in our binary notation:
    //Even entries are the addresses
    //of the nodes in the tree,
    //odd ones are the names or contents.
    //An address could be like this 100010,
    //it gives the structure of ramification.
    //A content could be any string with a message.
    private static   List<String> TreeInfo
            = new  ArrayList<>(2*numberOfLeaves-1);

    /*
    There are 6 directions for a rectangular tree:
    0: +X, 1 = -X,  2 = +Y,  3 = -Y
    4 = +Z, 5 = -Z
    */
    private static int direction;

    //Each node is localized in space thanks
    //to a string of directions.
    private static final   List<String> Directions
            = new  ArrayList<>(2*numberOfLeaves-1);

    //Printing for debugging
    final static boolean printReadFile = false;
    private static final String title
            =  "3D bush";

    //Binary string that says whether or not  node i in bit i
    //is present
```

```
    private static String bitTree = "";

    final Group root = new Group();
    private static final Cameras camera = new Cameras();
    final double cameraDistance = 5450;//450
    //Axes : ctrl + A
    private static final Axes axes = new Axes();
    final Xform world3DXform = new Xform();
    final Xform bonesXform = new Xform();

    //Generator of pseudo random numbers
    private static final Random r = new Random();



    //Vertical  size of the panel
    private static final int verticalSize =
            (int) (5 * Math.pow(2, numberOfGenerations));
    //Scale must be a power of 2
    private static final int scale = 16;
    //Symbolic initialization
    private static final int centerX = 1;
    private static final int centerY = 1;

    //Relative length of branches of the first U
    private static final int horizontalLength = 3;
    private static final int verticalLength = 8;

    private static Point3D lastPoint;

    private static final int delta = 50;

    private static final boolean print = false;
    private static final boolean print1 = false;
    private static final boolean print2 = false;
    private static final boolean print3 = false;
```

```
@Override
public void start(Stage primaryStage)
{

    primaryStage.setTitle(title);
    Scene scene = new Scene(root, 1200, 1000, true);

    scene.setFill(Color.AQUAMARINE);
    primaryStage.setScene(scene);
    primaryStage.show();
    axes.buildAxes();
    axes.handleKeyboard(scene);
    world3DXform.getChildren().addAll( axes.axesXform);

    camera.handleMouse(scene);
    camera.buildCamera();
    scene.setCamera(camera.camera);

    //TreeInfo is filled in:
    //A binary address to each node of the tree
    //is following by a content
    createTree();
    printTreeInfo();
    //Tree is drawn
    Group parent = new Group(create3dContent(TreeInfo));
        world3DXform.getChildren().addAll(parent);
        root.getChildren().add(world3DXform);


    primaryStage.show();
    play();
```

```
}

public void play()
{

}

@Override public void stop()
{

}
```

```
private Node create3dContent(List<String> TreeInfo)
{
    Project3D  p =  new Project3D(TreeInfo);
    return new Group(p);
}
```

```
private static void printTreeInfo()
{
    if (print1) System.out.println("TreeInfo");

    int l = TreeInfo.size();
    for(int i = 0; i < l; i = i+2)
    {
        if (print1)
            System.out.println(i/2 + "   " + TreeInfo.get(i)
                + "   " + TreeInfo.get(i+1));
```

```
    }

}



//A binary string with 2*numberOfLeaves-1 bits
private static String randomBinaryString()
{
    //A random binary string
    bitTree = "";
    for (int i = 0; i < 2*numberOfLeaves-1; i++)
    {
        Integer j = r.nextInt(2);
        bitTree = bitTree + j.toString();

    }

    if (print)
        System.out.println(bitTree);
    return bitTree;
}

//Binary string s is decoded into an int
private static int stringToNumber(String s)
{
    int n = 0;
    int l = s.length();

    for(int j=0;j  < l;j++ )
    {
        char c  =   s.charAt(j);
        int  v = Character.getNumericValue(c);
        int q = j+1;
        int number1 = (int) (v * Math.pow(2,l-q));
        n  = n + number1;
    }
    return n;
```

```
}


private static int stringToInt(String address)
{
    int output = 0;
    //System.out.print("String = " + address );
    char c  = address.charAt(0);
    if  (c == '1') output = stringToNumber(address);
        else System.out.println("Bad encoding");
    //System.out.println("  int =  " + output);
    return output;
}


//All ancestors of node m are added
private  void  addAncestors(int m)
{
    String address = TreeInfo.get(m);
    int a = address.length();
    //If the address of  a node is 10001,
    //an ancestor has as address a left substring
    //of 0001, say 0, 00 or 000.
    //All of them must be added to TreeInfo

    for(int i = 0; i < a;  i++)
    {
        String leftSubstring
                = address.substring(0, i);
        leftSubstring = "1" + leftSubstring;
        int k = stringToInt(leftSubstring);
        //Updating of tree information
        if (k < bitTree.length()-1)
            bitTree = bitTree.substring(0, k)
                + "1" + bitTree.substring(k+1);
        else bitTree = bitTree.substring(0, k)
                + "1" ;
    }
```

```
}

//All ancestors of node m are deleted
private  void  deleteAncestors(int m)
{
    String address = TreeInfo.get(m);
    int a = address.length();
    //If the address of  a node is 10001,
    //an ancestor has as address a left substring
    //of 0001, say 0, 00 or 000.
    //All of them must be deleted from TreeInfo

    for(int i = 0; i < a;  i++)
    {
        String leftSubstring
                = address.substring(0, i);
        leftSubstring = "1" + leftSubstring;
        int k = stringToInt(leftSubstring);
        //Updating of tree information
        if (k < bitTree.length()-1)
            bitTree = bitTree.substring(0, k)
                + "0" + bitTree.substring(k+1);
        else bitTree = bitTree.substring(0, k)
                + "0" ;
    }

}

//The tree encoded by binary string b is
//decoded into an array with binary addresses and contents.
//The tree corresponds with the lower
//part of a tree with one more generation.
private  void  createTree(String b )
{


    int l = b.length();
```

```
    char c ;
    String s ;
    //First two bits do not matter
    for (Integer i = 2; i < l; i++)
    {
        c = b.charAt(i);
        if (c == '1')
        {
            if ( i < 2*numberOfLeaves)
            {
                s = Integer.toBinaryString(i);
                //prefix "1" is deleted
                s = s.substring(1);
                TreeInfo.add(s);
                s = s.replace('0', '+');
                s = s.replace('1', '-');
                TreeInfo.add(s);
            }
        }
    }
}


//The tree encoded by binary string b is
//decoded and drawn.
//The tree corresponds with the lower
//part of a tree with one more generation.
private  void  createTree( )
{
    int n = numberOfLeaves;
    TreeInfo = new  ArrayList<>(2*n-1);
    //A random binary string:
    //If bit I is 1, the I node is added to the tree
    bitTree = randomBinaryString();
    createTree(bitTree) ;

    /*
    //If a node exists, all its ancestors too
```

```
    for(int m = 0; m < TreeInfo.size()-1;   m = m+2)
        addAncestors(m);
    */

    //Simplify tree:
    //If a node exists, all its ancestors are erased
    for(int m = 0; m < TreeInfo.size()-1;   m = m+2)
        deleteAncestors(m);

    //Tree information is re-created from the bitTree
    TreeInfo.clear();
    /*
    //Test: full tree
    bitTree = "";
    for (int i = 0; i < 2*numberOfLeaves; i++)
    {

        bitTree = bitTree + "1";

    }
    */
    createTree(bitTree);

}


//Returns a 3D line that goes from lastPoint
//to thisPoint
private  Line3D  line3DTo(Point3D thisPoint)
{
    Line3D l = new Line3D(lastPoint, thisPoint);
    lastPoint = thisPoint;
    return l;
}


//=====================================================
//======= Beginning of Inner class Project3D ==========
```

```
//=======================================================



//The drawing is synthesized
private class Project3D extends Group
{

//Constructor
Project3D(List<String> TreeInfo)
{
    //Tree is drawn to expand a 3D bush
    Group g = drawTree(TreeInfo);
    getChildren().addAll(g );
}


/*
There are 6 directions:
0: +X, 1 = -X,  2 = +Y,  3 = -Y
4 = +Z, 5 = -Z.
One distinct than d is chosen
*/
private int distinctDirection(int d)
{
    int i = r.nextInt(6);
    //System.out.println(i + "  " + d);
    if (i == d) distinctDirection(d);
    return i;
}



private  Point3D  move3D(int d)
{

    Point3D p = new Point3D(0,0,0);
    switch (d)
```

```
    {
        case 0: p = lastPoint.add( delta,0,0); break;
        case 1: p = lastPoint.add(-delta,0,0); break;
        case 2: p = lastPoint.add(0, delta,0); break;
        case 3: p = lastPoint.add(0,-delta,0); break;
        case 4: p = lastPoint.add(0,0, delta); break;
        case 5: p = lastPoint.add(0,0,-delta); break;
    }
    return p;
}

//Common initial substring to strings s and t
private  String commonSubstring(String s, String t)
{
    int a = s.length();
    int b = t.length();
    int min = a;
    if (b < a ) min = b;
    String c = "";
    for(int i  = 0; i < min; i++)
    {
        if (s.charAt(i) == t.charAt(i))
            c = c + s.charAt(i);
        else break;
    }
    return c;
}

//TreeInfo in the tree
//gives the ramification structure but not
//the 3D presentation which arises from
//3D directions chosen at random   but
//if two addresses have an initial common segment,
//so do their initial 3D subpaths.
//Example: 00110 and 00101 have in common 001
//so, their 3D paths share
//an initial subpath with 3 strokes
private  void harmonizeTree()
```

```
{

    for (int i = Directions.size()-2;
            i > -1; i= i-2)
    {
        String s = TreeInfo.get(i);
        for (int j = i-2;   j > -1; j = j-2)
        {
            String t = TreeInfo.get(j);

            String common = commonSubstring(s,t);

            int l = common.length();
            if (l > 0)
            {

                int m = t.length();
                String c  = "";
                String ds = Directions.get(i);
                String dt = Directions.get(j);
                if (print2)
                    System.out.print(s + "  " + t + "  "
                            + common + "  "
                            + ds + "   " + dt);

                String prefix = ds.substring(0, l);
                String insertion1 = "";
                String insertion2 = "";
                String suffix = "";

                if (m > l)
                {
                    insertion2 = dt.substring(l, l+1);
                    if (m > l+1) suffix = dt.substring(l+1);
                }
                if ( ds.charAt(l) == dt.charAt(l))
                {
                    insertion2 = ""
```

```
                        + distinctDirection(t.charAt(l));
                }


                dt = prefix + insertion1
                        + insertion2 + suffix;

                if (print2)
                    System.out.println( "   " + dt);

                Directions.set(j, dt);
            }
        }

    }
}



//Information is created to realize tree into 3D space.
private  void treeTo3D()
{
    //Directions are concatenated at random
    for (int i = 0; i < TreeInfo.size(); i= i+2)
    {
        String s = "";
        for (int j = 0; j < TreeInfo.get(i).length(); j++)
        {
            //A new direction is chosen
            direction = distinctDirection(direction);
            s = s + direction;
        }//End of for j
        Directions.add(s);
        Directions.add(s);
    }
    //The tree structure is recovered
    harmonizeTree();
```

```
}


   private   Group drawTree(List<String> TreeInfo)
{
    Group myGroup  = new Group();
    //Draw stem
    int x00 = scale * centerX;
    int y00 = verticalSize - scale * centerY - numberOfLeaves
            * verticalLength/2;
    //Horizontal and basic vertical strokes
    int H = scale * horizontalLength;


    //The stalk is drawn
    Point3D P0=  new Point3D(x00, y00,0);
    Point3D   P1 = P0.add(H, 0,0); //+X direction
    /*
    There are 6 directions:
    0: +X, 1 = -X,  2 = +Y,  3 = -Y
    4 = +Z, 5 = -Z
    */
    direction = r.nextInt(6);




    //Put a landmark at the root
    Sphere sphere = new Sphere();
    sphere.setRadius(10);
    sphere.setTranslateX((int) P1.getX());
    sphere.setTranslateY((int) P1.getY() );
    sphere.setTranslateZ((int) P1.getZ() );
    root.getChildren().add(sphere);

    //Information is created to realize tree into 3D space
    treeTo3D();
    //The tree is drawn:
```

```
    for (int i = 0; i < TreeInfo.size(); i= i+2)
    {

        //Reinitialization
        lastPoint = P1;
        Point3D thisPoint;
        String d = Directions.get(i);
        if (print3)
            System.out.println( "Path = " + d);
        int length = d.length();

        for (int j = 0; j < length; j++)
        {
            //Direction at j is read
            direction = Integer.parseInt("" + d.charAt(j));
            if (print3)
                System.out.println(direction + "   ");
            //thisPoint moves  along the chosen direction
            thisPoint = move3D(direction);
            //Draw a stroke
            //from lastPoint to thisPoint
            Line3D l2 = line3DTo(thisPoint);
            myGroup.getChildren().add(l2.group);
        }//End of for j

    }  //End of for: numberOfLeaves

    return myGroup;
}

}//End of inner class Project3D


public static void main(String[] args)
{
    launch(args);
}
```

```
}//End of program N102  TreeToShape8
```

105, page 347. The following is the code for a non-self-intersecting rectangular random binary tree whose arms extends randomly in 3D:

```
/*
    Program N105  TreeToShapes10
    This program displays
    a rectangular random non-self-intersecting binary tree
    whose arms extend randomnly in 3D.

    When we think of a non-self-intersecting tree,
    we have in mind the property
    that two cells
    cannot occupy the same space.
    So, we demand from any point in space not to be
    occupied by more than one cell.

*/

package Programs;


//Files in the hosting project
import Tools.Axes;
import Tools.Cameras;
import Tools.Line3D;
import Tools.Xform;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.Node;
```

```
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Sphere;
import javafx.stage.Stage;



public class TreeToShapee10 extends Application
{

    private static final int numberOfGenerations = 6;
    //The terminal  leaves define the complete tree
    private static final int numberOfLeaves
            = (int) Math.pow(2, numberOfGenerations);

    //This is data for a tree in our binary notation:
    //Even entries are the addresses
    //of the nodes in the tree,
    //odd ones are the names or contents.
    //An address could be like this 100010,
    //it gives the structure of ramification.
    //A content could be any string with a message.
    private static final   List<String> TreeInfo
            = new  ArrayList<>(2*numberOfLeaves-1);

    //List to keep the occupied places in 3D space
    private static final   List<Point3D> Occupancy
            = new  ArrayList<>(2*numberOfLeaves-1);

    /*
    There are 6 directions for a rectangular tree:
    0: +X, 1 = -X,  2 = +Y,  3 = -Y
    4 = +Z, 5 = -Z
    */
    private static int direction;

    //Array with  strings of directions that define a path
    //to localize each node in 3D space.
```

```
    private static final   List<String> Paths3D
            = new  ArrayList<>(2*numberOfLeaves-1);

    //Printing for debugging
    final static boolean printReadFile = false;
    private static final String title
            =  "3D binary bush";

    //Binary string that says whether or not  node i
    //is present (bit i = 1)
    private static String bitTree = "";

    //Root of the   tree of the JavaFX application
    final Group root = new Group();
    private static final Cameras camera = new Cameras();
    final double cameraDistance = 5450;//450
    //Axes : ctrl + A
    private static final Axes axes = new Axes();
    final Xform world3DXform = new Xform();
    final Xform bonesXform = new Xform();

    //Generator of pseudo random numbers
    private static final Random r = new Random();


    //Hidden argument of method lineTo(thisPoint)
    //to call a line creator method:
    //Line3D(LastPoint, thisPoint)
    private static Point3D lastPoint;
    private static final Point3D root3D
            = new Point3D(0,0,0);

    //Length of strokes in 3D space
    private static final int delta = 50;

    //Boolean printing flags
    private static final boolean print = false;
    private static final boolean printInfo = true;
```

```
    private static final boolean printIsOccupied = false;
    private static final boolean printGetAncestralPath
            = false;
    private static final boolean printTryToGetPath = false;
    private static final boolean printDrawTree = false;
    private static final boolean printTreeTo3D = false;
    private static final boolean printIsPathRepeated
            = false;
    //Maximal number of trials to find
    //a path that is not repeated and that ends
    // in a point  that is not occupied
    private static final int limit = 50;
    private static  int counterOfTrials = 0;


@Override
public void start(Stage primaryStage)
{

    primaryStage.setTitle(title);
    Scene scene = new Scene(root, 1200, 1000, true);

    scene.setFill(Color.AQUAMARINE);
    primaryStage.setScene(scene);
    primaryStage.show();
    axes.buildAxes();
    axes.handleKeyboard(scene);
    world3DXform.getChildren().addAll( axes.axesXform);

    camera.handleMouse(scene);
    camera.buildCamera();
    scene.setCamera(camera.camera);

    //TreeInfo is filled in:
    //A binary string is composed:
    //if bit i is 1, node i is added to the tree.
    //A binary address to each node of the tree
    //is following by a content in TrreInfo
```

```
    createTree();

    //Tree is drawn
    Group parent = new Group(create3dContent());
        world3DXform.getChildren().addAll(parent);
        root.getChildren().add(world3DXform);

    if (printInfo) printInfo();
    primaryStage.show();
    play();


}

public void play()
{

}

@Override public void stop()
{

}




//Master of 3D graphics
private Node create3dContent( )
{
    Project3D  p =  new Project3D( );
    return new Group(p);
}



private static void printAddresses()
{
```

```
    System.out.println("Addresses");

    int l = TreeInfo.size();
    for(int i = 0; i < l; i = i+2)
    {
        System.out.println(i/2 + "   " + TreeInfo.get(i)
                + "   " + TreeInfo.get(i+1));
    }
}


private static void printOccupancy()
{
    System.out.println("Record of occupancy");

    int l = Occupancy.size();
    System.out.println("Occupancy");
    for(int i = 0; i < l; i++)
    {
        System.out.println(i
                + "Address = " + TreeInfo.get(2*i)
                + "   Path = " + Paths3D.get(i)
                + "   point =    "
                + Occupancy.get(i).toString());
        if (i > 0)
            for(int j = 0; j < i; j++)
            {
                if (Occupancy.get(i)
                        .equals(Occupancy.get(j)))
                    System.out.println(i
                            + " Repeated with " + j);
            }
    }
}


private static void printInfo()
{
```

```
    printAddresses();
    printOccupancy();

}


//A binary string with 2*numberOfLeaves-1 bits
private static String randomBinaryString()
{
    //A random binary string
    bitTree = "";
    for (int i = 0; i < 2*numberOfLeaves-1; i++)
    {
        Integer j = r.nextInt(2);
        bitTree = bitTree + j.toString();

    }

    if (print)
        System.out.println(bitTree);
    return bitTree;
}

//Binary string s is decoded into an int
private static int stringToNumber(String s)
{
    int n = 0;
    int l = s.length();

    for(int j=0;j  < l;j++ )
    {
        char c  =   s.charAt(j);
        int  v = Character.getNumericValue(c);
        int q = j+1;
        int number1 = (int) (v * Math.pow(2,l-q));
        n  = n + number1;
    }
    return n;
```

```
}


//String address is interpreted as a binary number
//that corresponds with an int
private static int stringToInt(String address)
{
    int output = 0;
    //System.out.print("String = " + address );
    char c  = address.charAt(0);
    if  (c == '1') output = stringToNumber(address);
        else System.out.println("Bad encoding");
    //System.out.println("  int =  " + output);
    return output;
}


//All ancestors of node m are added
private  void  addAncestors(int m)
{
    String address = TreeInfo.get(m);
    int a = address.length();
    //If the address of  a node is 10001,
    //an ancestor has as address a left substring
    //of 0001, say 0, 00 or 000.
    //All of them must be added to TreeInfo

    for(int i = 0; i < a;  i++)
    {
        String leftSubstring
                = address.substring(0, i);
        leftSubstring = "1" + leftSubstring;
        int k = stringToInt(leftSubstring);
        //Updating of tree information
        if (k < bitTree.length()-1)
            bitTree = bitTree.substring(0, k)
                + "1" + bitTree.substring(k+1);
        else bitTree = bitTree.substring(0, k)
```

```
                + "1" ;
    }

}



//The tree encoded by binary string b is
//decoded into an array with binary addresses and contents.
//The tree corresponds with the lower
//part of a tree with one more generation.
private  void  createTree(String b )
{


    int l = b.length();
    char c ;
    String s ;
    //First two bits do not matter
    for (Integer i = 2; i < l; i++)
    {
        c = b.charAt(i);
        if (c == '1')
        {
            if ( i < 2*numberOfLeaves)
            {
                s = Integer.toBinaryString(i);
                //prefix "1" is deleted
                s = s.substring(1);
                TreeInfo.add(s);
                s = s.replace('0', '+');
                s = s.replace('1', '-');
                TreeInfo.add(s);
            }
        }
    }
}
```

```
//The tree encoded by binary string b is
//decoded and drawn.
//The tree corresponds with the lower
//part of a tree with one more generation.
private  void  createTree( )
{
    int n = numberOfLeaves;

    //A random binary string:
    //If bit I is 1, the I node is added to the tree
    bitTree = randomBinaryString();
    createTree(bitTree) ;


    //If a node exists, all its ancestors too.
    //bitTree will be modified in consequence.
    for(int m = 0; m < TreeInfo.size()-1;   m = m+2)
        addAncestors(m);

    TreeInfo.clear();
    createTree(bitTree);

}




//=======================================================
//======= Beginning of Inner class Project3D ===========
//=======================================================



//The drawing is synthesized
private class Project3D extends Group
{
```

```
//Constructor
Project3D()
{
    //Tree is drawn to expand a 3D bush
    Group g = drawTree( );
    getChildren().addAll(g );
}

//Returns a 3D line that goes from lastPoint
//to thisPoint
private  Line3D  line3DTo(Point3D thisPoint)
{
    Line3D l = new Line3D(lastPoint, thisPoint);
    lastPoint = thisPoint;
    return l;
}

/*
There are 6 directions given by
//the natural basis.
One distinct than d is chosen
*/
private int distinctDirection(int d)
{
    int i = r.nextInt(6);
    //System.out.println(i + "  " + d);
    if (i == d) distinctDirection(d);
    return i;
}


//A direction distinct than  backwards is chosen
private int notBackwards(int d)
{
    int backwards  = 0;
    switch (d)
    {
```

```
        case 0: backwards = 1; break;
        case 1: backwards = 0 ; break;
        case 2: backwards = 3; break;
        case 3: backwards = 2; break;
        case 4: backwards = 5; break;
        case 5: backwards = 4; break;
    }
    int i = distinctDirection(backwards);
    return i;
}

//Direction d is realized as a move
//There are 6 directions:
//0: +X, 1 = -X,  2 = +Y,  3 = -Y
//4 = +Z, 5 = -Z.
private  Point3D  move3D(int d)
{

    Point3D p = new Point3D(0,0,0);
    switch (d)
    {
        case 0: p = new Point3D( delta,0,0); break;
        case 1: p = new Point3D(-delta,0,0); break;
        case 2: p = new Point3D(0, delta,0); break;
        case 3: p = new Point3D(0,-delta,0); break;
        case 4: p = new Point3D(0,0, delta); break;
        case 5: p = new Point3D(0,0,-delta); break;
    }
    return p;
}

//Common initial substring to strings s and t
private  String commonSubstring(String s, String t)
{
    int a = s.length();
    int b = t.length();
    int min = a;
    if (b < a ) min = b;
```

```
    String c = "";
    for(int i  = 0; i < min; i++)
    {
        if (s.charAt(i) == t.charAt(i))
            c = c + s.charAt(i);
        else break;
    }
    return c;
}
```

```
//Is thisPoint already occupied?
private  boolean isOccupied(Point3D thisPoint)
{
    //printOccupancy();
    int l = Occupancy.size();
    boolean occupied = false;
    if (printIsOccupied)
        System.out.println("isOccupied?");

    if (root3D.equals(thisPoint))
        {
            return true;
        }
    else
    {
        for(int i = 0; i < l; i++)
        {
            if (Occupancy.get(i).equals(thisPoint))
            {
                occupied = true;
                break;
            }
        }
```

```
        return occupied;
    }
}


//Is thisPoint already occupied?
private  boolean isPathRepeated(String path)
{
    int l = Paths3D.size();
    boolean repeated = false;
    if (printIsPathRepeated)
        System.out.println("isPathRepeated?");
    for(int i = 0; i < l; i++)
    {
        if (Paths3D.get(i).equals(path))
        {
            repeated = true;
            break;
        }
    }
    return repeated;
}


//A node has a path that must coincide with
//the paths of its ancestors.
//So, we look for the longest one
private String getAncestralPath(int i)
{
    String addressi = TreeInfo.get(i);

    String ancestralPath = "";
    String candidatePath;
    int f = Paths3D.size();
    List<String> Candidates
            = new  ArrayList<>(2*numberOfLeaves-1);
    for (int j = f-1; j > -1; j--)
    {
```

```
        String addressj = TreeInfo.get(2*j);
        if (printGetAncestralPath)
            System.out.print(" addressi =  " + addressi
                + " addressj = " + addressj);
        String commonAddress
                = commonSubstring(addressi, addressj);


        int k = commonAddress.length();
        if (k == 0)
        {
            if (printGetAncestralPath)
                System.out.println(" No common subaddress ");
        }
        else
        {
            String path = Paths3D.get(j);
            candidatePath = path.substring(0, k);
            Candidates.add(candidatePath);
            if (printGetAncestralPath)
                System.out.println(" Common address = "
                    + commonAddress
                    + " Pathj = " + path
                    + "  candidate path = "
                    + candidatePath);

        }

    }
    int l = Candidates.size();
    int max = 0;
    for(int k = 0; k < l; k++)
    {
        String s = Candidates.get(k);
        int p = s.length();
        if (p > max)
        {
```

```
            max = p;
            ancestralPath = s;
        }
    }
    if (printGetAncestralPath)
                System.out.println(" ancestral path = "
                    + ancestralPath);
    return ancestralPath;
}


//Node i in the list TreeInfo
//needs an exclusive location in 3D.
//If possible, one is found.
private  String tryToGetPath(int i)
{
    //This is the root of the tree
    Point3D   P1 = new Point3D(0, 0,0);
    //Reinitialization
    lastPoint = P1;
    Point3D thisPoint = P1;
    //A child must share subpaths with their ancestors

    String address = TreeInfo.get(i);
    int l = address.length();

    String ancestralPath = getAncestralPath(i);
    String path = ancestralPath;
    int lengthPath = path.length();
    for (int j = 0; j < lengthPath; j++)
    {
        //Direction at j is read
        direction = Integer.parseInt("" + path.charAt(j));

        //thisPoint moves  along the chosen direction
        thisPoint = thisPoint.add(move3D(direction));

    }
```

```
//New directions are added to complete path
if (counterOfTrials < limit) //Overflow is interdicted.
{
    //Directions are concatenated
    for (int j = lengthPath; j < l; j++)
    {
        //A non backwards new direction is chosen
        direction = notBackwards(direction);
        path = path + direction;
        //Chosen direction is executed in memory
        thisPoint = thisPoint.add( move3D(direction));
    }

    if (printTryToGetPath)
        System.out.println("\nAddress to try = "
            + address
             + "   Ancetralpath = " + ancestralPath
            + " counterOfTrials = "
            + counterOfTrials
            +   " \nThisPoint = " + thisPoint);

    if ( (isPathRepeated(path))
            || (isOccupied(thisPoint)) )
    {
        if (printTryToGetPath)
            System.out.println("    Path is repeated.");
        counterOfTrials++;
        tryToGetPath(i);
    }
    else
    {
        if (printTryToGetPath)
            System.out.println(
                    "    Path is NOT repeated.");
        Occupancy.add(thisPoint);
        Paths3D.add(path);
    }
```

```
    }
    if (counterOfTrials == limit)
    {
        if (printTryToGetPath)
            System.out.println(
                    "    Path not found at trial "
                + counterOfTrials);
        path =  "-1";
    }

    return path;
}


//Information is created to realize
//and draw tree into 3D space.
private  void treeTo3D()
{
    String path ;

    //Directions are proposed at random,
    //and concatenated if no double occupancy appears
    for (int i = 0; i < TreeInfo.size(); i= i+2)
    {
        if (printTreeTo3D)
            System.out.print("Node i = " + i
                + "  Address = " + TreeInfo.get(i));
        counterOfTrials = 0;
        path = tryToGetPath(i);
        if (printTreeTo3D)
            System.out.println("  Path at treeTo3D = "
            + path);
        if (path.contentEquals("-1"))
        {
            System.out.println("Truncation");
        }
    }
    if (printTreeTo3D) printOccupancy();
```

```
}



private   Group drawTree( )
{
    Group myGroup  = new Group();
    //This is the root of the tree
    Point3D   P1 = new Point3D(0, 0,0); //+X direction


    //Put a landmark at the root
    Sphere sphere = new Sphere();
    sphere.setRadius(10);
    sphere.setTranslateX((int) P1.getX());
    sphere.setTranslateY((int) P1.getY() );
    sphere.setTranslateZ((int) P1.getZ() );
    root.getChildren().add(sphere);

    //Information is created to realize
    //and draw tree into 3D space
    treeTo3D();
    Occupancy.clear();
    //The tree is drawn:
    for (String Paths3D1 : Paths3D)
    {
        //Reinitialization
        lastPoint = P1;
        Point3D thisPoint = P1;
        String path = Paths3D1;
        if (printDrawTree)
            System.out.println( "Path = " + path);
        int length = path.length();
        for (int j = 0; j < length; j++)
        {
            //Direction at j is read
            direction = Integer.parseInt("" + path.charAt(j));
```

```
            if (printDrawTree)
                System.out.print(direction + "   ");
            //thisPoint moves  along the chosen direction
            thisPoint = thisPoint.add(move3D(direction));
            //Draw a stroke
            //from lastPoint to thisPoint
            Line3D l = line3DTo(thisPoint);
            myGroup.getChildren().add(l.group);
        }
        if (printDrawTree)
            System.out.println();
        Occupancy.add(thisPoint);
    } //End of for: numberOfLeaves
    return myGroup;
}

}//End of inner class Project3D


public static void main(String[] args)
{
    launch(args);
}

}//End of program N105  TreeToShapes10
```

109, page 368. The following code allows to study the power of randomness to create a tree whose nodes populate the corners and lateral integral points of a cube of size 2. The decision of the Author is as follows: randomness is not suitable to fit a cubic shape of edge 2. Henceforth, it makes sense to study the power of evolution to solve this task and other more complicated, cubic shapes of edge of arbitrary length.

```
/*
    Program N109  TreeToShapes12
    A  cube of size 2 is fitted by a random
    3D binary tree.
```

```
    The cube has a corner at the origin
    and the others extend over the positive octant.
    The root of the tree is at the origin and it can
    extend its arms in within the cube only.
    The task assigned to randomness
    consists in populating the points with
    integer coordinates that reside at the
    walls of the cube.
    Variant: a predefined number of points
    with integer coordinates
    shall be visited by the nodes of the tree.

    Our   goal is to know whether or not
    randomness  can explain a cubic shape.

*/

package Programs;


//Files in the hosting project
import Tools.Axes;
import Tools.Cameras;
import Tools.Line3D;
import Tools.Xform;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Sphere;
```

```
import javafx.stage.Stage;


public class TreeToShapes12 extends Application
{

    private static int numberOfGenerations;
    //The terminal  leaves define the complete tree
    private static int numberOfLeaves;

    //This is data for a tree in our binary notation:
    //Even entries are the addresses
    //of the nodes in the tree,
    //odd ones are the names or contents.
    //An address could be like this 100010,
    //it gives the structure of ramification.
    //A content could be any string with a message.
    private static List<String> TreeInfo;

    //List to keep the occupied places in 3D space
    private static List<Point3D> Occupancy;

    /*
    There are 6 directions for a rectangular tree:
    0: +X, 1 = -X,  2 = +Y,  3 = -Y
    4 = +Z, 5 = -Z
    */
    private static int direction;

    //Array with  strings of directions that define a path
    //to localize each node in 3D space.
    private static List<String> Paths3D;

    //Printing for debugging
    final static boolean printReadFile = false;
    private static final String title
            =  "3D binary bush";
```

```
//Binary string that says whether or not  node i
//is present (bit i = 1)
private static String bitTree = "";

//Root of the   tree of the JavaFX application
final Group root = new Group();
private static final Cameras camera = new Cameras();
final double cameraDistance = 5450;//450
//Axes : ctrl + A
private static final Axes axes = new Axes();
final Xform world3DXform = new Xform();
final Xform bonesXform = new Xform();

//Generator of pseudo random numbers
private static final Random r = new Random();


//Hidden argument of method lineTo(thisPoint)
//to call a line creator method:
//Line3D(LastPoint, thisPoint)
private static Point3D lastPoint;
private static final Point3D root3D
        = new Point3D(0,0,0);

//Length of strokes in 3D space
private static final int delta = 50;

//Boolean printing flags
private static final boolean print = false;
private static final boolean printInfo = false;
private static final boolean printIsOccupied = false;
private static final boolean printGetAncestralPath
        = false;
private static final boolean printTryToGetPath = false;
private static final boolean printDrawTree = false;
private static final boolean printTreeTo3D = false;
private static final boolean printIsPathRepeated
```

```
            = false;
    //Maximal number of trials to find
    //a path that is not repeated and that ends
    // in a point  that is not occupied
    private static final int limit = 50;
    //Counter of trials to find a path
    private static  int counterOfTrials = 0;
    //Counter of trials to fit the unitary cube
    private static  int numberOfTrialsCube = 0;
    //An incomplete task is denoted by the required
    //number of visited points of the cube
    private static  int RequiredNumberOfEdges;




//Fundamental parameters are given a value
private void initialization()
{
    numberOfGenerations = r.nextInt(5);
    //The terminal  leaves define the complete tree
    numberOfLeaves
            = (int) Math.pow(2, numberOfGenerations);

    //This is data for a tree in our binary notation:
    //Even entries are the addresses
    //of the nodes in the tree,
    //odd ones are the names or contents.
    //An address could be like this 100010,
    //it gives the structure of ramification.
    //A content could be any string with a message.
    TreeInfo  = new  ArrayList<>(2*numberOfLeaves-1);


    //List to keep the occupied places in 3D space
    Occupancy = new  ArrayList<>(2*numberOfLeaves-1);
    Paths3D  = new  ArrayList<>(2*numberOfLeaves-1);
```

```
}


@Override
public void start(Stage primaryStage)
{


    primaryStage.setTitle(title);
    Scene scene = new Scene(root, 1200, 1000, true);

    scene.setFill(Color.AQUAMARINE);
    primaryStage.setScene(scene);
    primaryStage.show();
    axes.buildAxes();
    axes.handleKeyboard(scene);
    world3DXform.getChildren().addAll( axes.axesXform);

    camera.handleMouse(scene);
    camera.buildCamera();
    scene.setCamera(camera.camera);



    //Tree is drawn
    Group parent = new Group(create3dContent());
        world3DXform.getChildren().addAll(parent);
        root.getChildren().add(world3DXform);

    if (printInfo) printInfo();
    primaryStage.show();
    play();


}

public void play()
{
```

```
}

@Override public void stop()
{

}




//Master of 3D graphics
private Node create3dContent( )
{
    Project3D  p =  new Project3D( );
    return new Group(p);
}



private static void printAddresses()
{
    System.out.println("Addresses");

    int l = TreeInfo.size();
    for(int i = 0; i < l; i = i+2)
    {
        System.out.println(i/2 + "   " + TreeInfo.get(i)
                + "   " + TreeInfo.get(i+1));
    }
}


private static void printOccupancy()
{
    System.out.println("Record of occupancy");

    int l = Occupancy.size();
```

```
    System.out.println("Occupancy");
    for(int i = 0; i < l; i++)
    {
        System.out.println(i
                + "Address = " + TreeInfo.get(2*i)
                + "   Path = " + Paths3D.get(i)
                + "   point =    "
                + Occupancy.get(i).toString());
        if (i > 0)
            for(int j = 0; j < i; j++)
            {
                if (Occupancy.get(i)
                        .equals(Occupancy.get(j)))
                    System.out.println(i
                            + " Repeated with " + j);
            }
    }
}


private static void printInfo()
{
    printAddresses();
    printOccupancy();

}


//A binary string with 2*numberOfLeaves-1 bits
private static String randomBinaryString()
{
    //A random binary string
    bitTree = "";
    for (int i = 0; i < 2*numberOfLeaves-1; i++)
    {
        Integer j = r.nextInt(2);
        bitTree = bitTree + j.toString();
```

```
    }

    if (print)
        System.out.println(bitTree);
    return bitTree;
}

//Binary string s is decoded into an int
private static int stringToNumber(String s)
{
    int n = 0;
    int l = s.length();

    for(int j=0;j  < l;j++ )
    {
        char c  =   s.charAt(j);
        int  v = Character.getNumericValue(c);
        int q = j+1;
        int number1 = (int) (v * Math.pow(2,l-q));
        n  = n + number1;
    }
    return n;
}


//String address is interpreted as a binary number
//that corresponds with an int
private static int stringToInt(String address)
{
    int output = 0;
    //System.out.print("String = " + address );
    char c  = address.charAt(0);
    if  (c == '1') output = stringToNumber(address);
        else System.out.println("Bad encoding");
    //System.out.println("  int =  " + output);
    return output;
}
```

```
//All ancestors of node m are added
private  void  addAncestors(int m)
{
    String address = TreeInfo.get(m);
    int a = address.length();
    //If the address of  a node is 10001,
    //an ancestor has as address a left substring
    //of 0001, say 0, 00 or 000.
    //All of them must be added to TreeInfo

    for(int i = 0; i < a;  i++)
    {
        String leftSubstring
                = address.substring(0, i);
        leftSubstring = "1" + leftSubstring;
        int k = stringToInt(leftSubstring);
        //Updating of tree information
        if (k < bitTree.length()-1)
            bitTree = bitTree.substring(0, k)
                + "1" + bitTree.substring(k+1);
        else bitTree = bitTree.substring(0, k)
                + "1" ;
    }

}



//The tree encoded by binary string b is
//decoded into an array with binary addresses and contents.
//The tree corresponds with the lower
//part of a tree with one more generation.
private  void  createTree(String b )
{


    int l = b.length();
```

```
    char c ;
    String s ;
    //First two bits do not matter
    for (Integer i = 2; i < l; i++)
    {
        c = b.charAt(i);
        if (c == '1')
        {
            if ( i < 2*numberOfLeaves)
            {
                s = Integer.toBinaryString(i);
                //prefix "1" is deleted
                s = s.substring(1);
                TreeInfo.add(s);
                s = s.replace('0', '+');
                s = s.replace('1', '-');
                TreeInfo.add(s);
            }
        }
    }
}


//The tree encoded by binary string b is
//decoded and drawn.
//The tree corresponds with the lower
//part of a tree with one more generation.
private  void  createTree( )
{


    //A random binary string:
    //If bit I is 1, the I node is added to the tree
    bitTree = randomBinaryString();
    createTree(bitTree) ;


    //If a node exists, all its ancestors too.
```

```
    //bitTree will be modified in consequence.
    for(int m = 0; m < TreeInfo.size()-1;   m = m+2)
        addAncestors(m);

    TreeInfo.clear();
    createTree(bitTree);
}




//=========================================================
//======= Beginning of Inner class Project3D ===========
//=========================================================



//The drawing is synthesized
private class Project3D extends Group
{

//Constructor
Project3D()
{
    //Tree is drawn to expand a 3D bush
    Group g = masterTree( );
    getChildren().addAll(g );
}

//Returns a 3D line that goes from lastPoint
//to thisPoint
private  Line3D  line3DTo(Point3D thisPoint)
{
    Line3D l = new Line3D(lastPoint, thisPoint);
    lastPoint = thisPoint;
    return l;
}
```

```
/*
There are 6 directions given by
//the natural basis.
One distinct than d is chosen
*/
private int distinctDirection(int d)
{
    int i = r.nextInt(6);
    //System.out.println(i + "  " + d);
    if (i == d) distinctDirection(d);
    return i;
}


//A direction distinct than  backwards is chosen
private int notBackwards(int d)
{
    int backwards  = 0;
    switch (d)
    {
        case 0: backwards = 1; break;
        case 1: backwards = 0 ; break;
        case 2: backwards = 3; break;
        case 3: backwards = 2; break;
        case 4: backwards = 5; break;
        case 5: backwards = 4; break;
    }
    int i = distinctDirection(backwards);
    return i;
}

//Direction d is realized as a move
//There are 6 directions:
//0: +X, 1 = -X,  2 = +Y,  3 = -Y
//4 = +Z, 5 = -Z.
private  Point3D  move3D(int d)
{
```

```
    Point3D p = new Point3D(0,0,0);
    switch (d)
    {
        case 0: p = new Point3D( delta,0,0); break;
        case 1: p = new Point3D(-delta,0,0); break;
        case 2: p = new Point3D(0, delta,0); break;
        case 3: p = new Point3D(0,-delta,0); break;
        case 4: p = new Point3D(0,0, delta); break;
        case 5: p = new Point3D(0,0,-delta); break;
    }
    return p;
}

//Common initial substring to strings s and t
private  String commonSubstring(String s, String t)
{
    int a = s.length();
    int b = t.length();
    int min = a;
    if (b < a ) min = b;
    String c = "";
    for(int i  = 0; i < min; i++)
    {
        if (s.charAt(i) == t.charAt(i))
            c = c + s.charAt(i);
        else break;
    }
    return c;
}


//Geometric barriers are defined here
private  boolean isForbidden(Point3D thisPoint)
{
    boolean b = false;
    double x = thisPoint.getX();
    double y = thisPoint.getY();
```

```
    double z = thisPoint.getZ();
    //Codification of  cube of size 2
    //Unitary edge = 50
    if ((x < 0) || (x > 110)) b = true;
    if ((y < 0) || (y > 110)) b = true;
    if ((z < 0) || (z > 110)) b = true;
    return b;
}



//Is thisPoint already occupied?
private  boolean isOccupied(Point3D thisPoint)
{
    //printOccupancy();
    int l = Occupancy.size();
    boolean occupied = false;
    if (printIsOccupied)
        System.out.println("isOccupied?");
    if (isForbidden(thisPoint) )
    {
        if (printIsOccupied)
            System.out.println("Forbidden" + thisPoint);
        return true;
    }
    if (root3D.equals(thisPoint))
        {
            return true;
        }
    else
    {
        for(int i = 0; i < l; i++)
        {
            if (Occupancy.get(i).equals(thisPoint))
            {
                occupied = true;
                break;
```

```
            }
        }
        return occupied;
    }
}


//Is thisPoint already occupied?
private  boolean isPathRepeated(String path)
{
    int l = Paths3D.size();
    boolean repeated = false;
    if (printIsPathRepeated)
        System.out.println("isPathRepeated?");
    for(int i = 0; i < l; i++)
    {
        if (Paths3D.get(i).equals(path))
        {
            repeated = true;
            break;
        }
    }
    return repeated;
}


//A node has a path that must coincide with
//the paths of its ancestors.
//So, we look for the longest one
private String getAncestralPath(int i)
{
    String addressi = TreeInfo.get(i);

    String ancestralPath = "";
    String candidatePath;
    int f = Paths3D.size();
    List<String> Candidates
            = new  ArrayList<>(2*numberOfLeaves-1);
```

```
for (int j = f-1; j >= 0; j--)
{
    String addressj = TreeInfo.get(2*j);
    if (printGetAncestralPath)
        System.out.print(" addressi =  " + addressi
            + " addressj = " + addressj);
    String commonAddress
            = commonSubstring(addressi, addressj);


    int k = commonAddress.length();
    if (k == 0)
    {
        if (printGetAncestralPath)
            System.out.println(" No common subaddress ");
    }
    else
    {
        String path = Paths3D.get(j);
        candidatePath = path.substring(0, k);
        Candidates.add(candidatePath);
        if (printGetAncestralPath)
            System.out.println(" Common address = "
                + commonAddress
                + " Pathj = " + path
                + "  candidate path = "
                + candidatePath);

    }

}
int l = Candidates.size();
int max = 0;
for(int k = 0; k < l; k++)
{
    String s = Candidates.get(k);
    int p = s.length();
    if (p > max)
```

```
        {

            max = p;
            ancestralPath = s;
        }
    }
    if (printGetAncestralPath)
                System.out.println(" ancestral path = "
                    + ancestralPath);
    return ancestralPath;
}


//Node i in the list TreeInfo
//needs an exclusive location in 3D.
//If possible, one is found.
private  String tryToGetPath(int i)
{
    //This is the root of the tree
    Point3D   P1 = new Point3D(0, 0,0);
    //Reinitialization
    lastPoint = P1;
    Point3D thisPoint = P1;
    //A child must share subpaths with their ancestors

    String address = TreeInfo.get(i);
    int l = address.length();

    String ancestralPath = getAncestralPath(i);
    String path = ancestralPath;
    int lengthPath = path.length();
    for (int j = 0; j < lengthPath; j++)
    {
        //Direction at j is read
        direction = Integer.parseInt("" + path.charAt(j));

        //thisPoint moves  along the chosen direction
        thisPoint = thisPoint.add(move3D(direction));
```

```
}

//New directions are added to complete path
if (counterOfTrials < limit) //Overflow is interdicted.
{
    //Directions are concatenated
    for (int j = lengthPath; j < l; j++)
    {
        //A non backwards new direction is chosen
        direction = notBackwards(direction);
        path = path + direction;
        //Chosen direction is executed in memory
        thisPoint = thisPoint.add( move3D(direction));
    }

    if (printTryToGetPath)
        System.out.println("\nAddress to try = "
            + address
             + "   Ancetralpath = " + ancestralPath
            + " counterOfTrials = "
            + counterOfTrials
            +   " \nThisPoint = " + thisPoint);

    if ( (isPathRepeated(path))
            || (isOccupied(thisPoint)) )
    {
        if (printTryToGetPath)
            System.out.println("    Path is repeated.");
        counterOfTrials++;
        tryToGetPath(i);
    }
    else
    {
        if (printTryToGetPath)
            System.out.println(
                    "    Path is NOT repeated.");
        Occupancy.add(thisPoint);
```

```
            Paths3D.add(path);
        }
    }
    if (counterOfTrials == limit)
    {
        if (printTryToGetPath)
            System.out.println(
                    "    Path not found at trial "
                + counterOfTrials);
        path =  "-1";
    }

    return path;
}


//Information is created to realize
//and draw tree into 3D space.
private  void treeTo3D()
{
    String path ;

    //Directions are proposed at random,
    //and concatenated if no double occupancy appears
    for (int i = 0; i < TreeInfo.size(); i= i+2)
    {
        if (printTreeTo3D)
            System.out.print("Node i = " + i
                + "  Address = " + TreeInfo.get(i));
        counterOfTrials = 0;
        path = tryToGetPath(i);
        if (printTreeTo3D)
            System.out.println("  Path at treeTo3D = "
            + path);
        if (path.contentEquals("-1"))
        {
            if (printTreeTo3D)
                System.out.println("Truncation");
```

```
        }
    }
    if (printTreeTo3D)
        System.out.println("Paths3D size = "
                + Paths3D.size());
    if (printTreeTo3D) printOccupancy();
}



private   Group drawTree( )
{
    Group myGroup  = new Group();
    //This is the root of the tree
    Point3D   P1 = new Point3D(0, 0,0); //+X direction


    //Put a landmark at the root
    Sphere sphere = new Sphere();
    sphere.setRadius(10);
    sphere.setTranslateX((int) P1.getX());
    sphere.setTranslateY((int) P1.getY() );
    sphere.setTranslateZ((int) P1.getZ() );
    root.getChildren().add(sphere);

    //Information is created to realize
    //and draw tree into 3D space
    treeTo3D();
    Occupancy.clear();
    //The tree is drawn:
    for (String Paths3D1 : Paths3D)
    {
        //Reinitialization
        lastPoint = P1;
        Point3D thisPoint = P1;
        String path = Paths3D1;
        if (printDrawTree)
```

```
            System.out.println( "Path = " + path);
        int length = path.length();
        for (int j = 0; j < length; j++)
        {
            //Direction at j is read
            direction = Integer.parseInt("" + path.charAt(j));
            if (printDrawTree)
                System.out.print(direction + "   ");
            //thisPoint moves  along the chosen direction
            thisPoint = thisPoint.add(move3D(direction));
            //Draw a stroke
            //from lastPoint to thisPoint
            Line3D l = line3DTo(thisPoint);
            myGroup.getChildren().add(l.group);
        }
        if (printDrawTree)
            System.out.println();
        Occupancy.add(thisPoint);
    } //End of for: numberOfLeaves
    return myGroup;
}

private   void makeTree( )
{
    initialization();
    //TreeInfo is filled in:
    //A binary string is composed:
    //if bit i is 1, node i is added to the tree.
    //A binary address to each node of the tree
    //is following by a content in TrreInfo
    createTree();
    //Information is created to realize
    //and draw tree into 3D space
    treeTo3D();
}


//Is the task done?
```

```
//Task: populate lateral points
//with integer coordinates
//of a cube of edge 2
private   boolean weAreDone()
{
    boolean b = false;
    int n = Paths3D.size();
    System.out.println("Number of edges = " + n);
    if (n == RequiredNumberOfEdges) b = true;
    return b;
}


private   Group masterTree( )
{
    Group myGroup;
    numberOfTrialsCube = 0;
    int maxNumberOfTrials = 100;
    //For a cube of edge 2,
    //a fitting tree that fills lateral walls
    //has 26 edges or so.
    //To fill the inner space also,
    //one more edge is necessary, 27 at all.
    RequiredNumberOfEdges = 22;
    for(int i = 0; i < maxNumberOfTrials; i++)
    {
        makeTree();
        numberOfTrialsCube++;
        if (weAreDone())   break;
        if (numberOfTrialsCube < maxNumberOfTrials-1)
        {
            TreeInfo.clear();
            Occupancy.clear();
            Paths3D.clear();
        }
    }
    System.out.println("Number of done trials  = "
            + numberOfTrialsCube );
```

```
    myGroup  = drawTree( );
    return myGroup;
}


}//End of inner class Project3D



public static void main(String[] args)
{
    launch(args);
}

}//End of program N109  TreeToShapes12
```

111, page 390. To make the code more evolvable the tree has been encapsulated into a class so that many instances could be defined at once to generate a population.

```
/*
    Program N111  TreeToShapes14
    A  cube of size 2 is fitted by a random
    3D binary tree.

    More evolvable code: to that aim
    the tree has been encapsulated into a class so
    that many instances could be defined at once
    to generate a population.

    The cube has a corner at the origin
    and the others extend over the positive octant.
    The root of the tree is at the origin and it can
    extend its arms in within the cube only.
    The task assigned to randomness
    consists in visiting  with its nodes at least
    a predefined number of points of the cube
    with integer coordinates.
```

```
    Our   goal is to know whether or not
    randomness  can explain a cube of edge 2,
    a simple model of   biological shape.

*/


package Programs;


//Files in the hosting project
import Tools.Axes;
import Tools.Cameras;
import Tools.Line3D;
import Tools.Xform;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Sphere;
import javafx.stage.Stage;



public class TreeToShapes14 extends Application
{
    //A task is denoted by the required
    //number of visited points of the cube, less than 26.
    final  int
            RequiredNumberOfVisitedPoints = 22;
```

```
    //Maximal number of trials to get the task done
    final int maxNumberOfTrials = 100;

    //Root of the   tree of the JavaFX application
    final Group root = new Group();
    final Cameras camera = new Cameras();
    final double cameraDistance = 5450;//450
    //Axes : ctrl + A
    final Axes axes = new Axes();
    final Xform world3DXform = new Xform();
    final Xform bonesXform = new Xform();

    //Generator of pseudo random numbers
    final Random r = new Random();
    final String title
            =  "3D binary bush";
        //Hidden argument of method lineTo(thisPoint)
    //to call a line creator method:
    //Line3D(LastPoint, thisPoint)
    Point3D lastPoint;
    final boolean printDrawTree = false;

    /*
    There are 6 directions for a rectangular tree:
    0: +X, 1 = -X,  2 = +Y,  3 = -Y
    4 = +Z, 5 = -Z
    */
    int direction;



@Override
public void start(Stage primaryStage)
{


    primaryStage.setTitle(title);
    Scene scene = new Scene(root, 1200, 1000, true);
```

```
        scene.setFill(Color.AQUAMARINE);
        primaryStage.setScene(scene);
        primaryStage.show();
        axes.buildAxes();
        axes.handleKeyboard(scene);
        world3DXform.getChildren().addAll( axes.axesXform);

        camera.handleMouse(scene);
        camera.buildCamera();
        scene.setCamera(camera.camera);

        //Various random trees are tried out
        ABinaryTree t = variousTrees( );

        //The last tree is drawn
        Group parent = new Group(create3dContent(t));
            world3DXform.getChildren().addAll(parent);
            root.getChildren().add(world3DXform);


        primaryStage.show();
        play();


    }

    public void play()
    {

    }

    @Override public void stop()
    {

    }
```

```
//================================
//======= Tree as a class =========
//================================


//Encapsulation of the tree
private class ABinaryTree
{
       //Maximal number of trials to find
    //a path that is not repeated and that ends
    // in a point  that is not occupied
    final int limit = 50;


    //Counter of trials to find a path
    int counterOfTrialsToFindPath= 0;
    //Counter of trials to fit the unitary cube
    int counterOfTrialsCube = 0;
    //Maximal length or number of links along arms of trees
    int maxHeight;
    //The terminal  leaves define the complete tree
    int numberOfLeaves;

    //This is data for a tree in our binary notation:
    //Even entries are the addresses
    //of the nodes in the tree,
    //odd ones are the names or contents.
    //An address could be like this 100010,
    //it gives the structure of ramification.
    //A content could be any string with a message.
    List<String> TreeInfo;

    //List to keep the occupied places in 3D space
    List<Point3D> Occupancy;



    //Array with  strings of directions that define a path
```

```
//to localize each node in 3D space.
List<String> Paths3D;



//Binary string that says whether or not  node i
//is present (bit i = 1)
String bitTree = "";



final Point3D root3D
        = new Point3D(0,0,0);

//Length of strokes in 3D space
final int delta = 50;

//Boolean printing flags
final boolean print = false;
final boolean printInfo = false;
final boolean printIsOccupied = false;
final boolean printGetAncestralPath
        = false;
final boolean printTryToGetPath = false;

final boolean printTreeTo3D = false;
final boolean printIsPathRepeated
        = false;


//Constructor
ABinaryTree()
{
    makeTree();
}
```

```
//Fundamental parameters are given a value
private void initialization()
{
    maxHeight = r.nextInt(5);
    counterOfTrialsCube = 0;

    //The terminal  leaves define the complete tree
    numberOfLeaves
            = (int) Math.pow(2, maxHeight);

    //This is data for a tree in our binary notation:
    //Even entries are the addresses
    //of the nodes in the tree,
    //odd ones are the names or contents.
    //An address could be like this 100010,
    //it gives the structure of ramification.
    //A content could be any string with a message.
    TreeInfo  = new  ArrayList<>(2*numberOfLeaves-1);


    //List to keep the occupied places in 3D space
    Occupancy = new  ArrayList<>(2*numberOfLeaves-1);
    Paths3D  = new  ArrayList<>(2*numberOfLeaves-1);
}
```

```
void printAddresses()
{
    System.out.println("Addresses");

    int l = TreeInfo.size();
    for(int i = 0; i < l; i = i+2)
```

```
    {
        System.out.println(i/2 + "   " + TreeInfo.get(i)
                + "   " + TreeInfo.get(i+1));
    }
}


void printOccupancy()
{
    System.out.println("Record of occupancy");

    int l = Occupancy.size();
    System.out.println("Occupancy");
    for(int i = 0; i < l; i++)
    {
        System.out.println(i
                + "Address = " + TreeInfo.get(2*i)
                + "   Path = " + Paths3D.get(i)
                + "   point =    "
                + Occupancy.get(i).toString());
        if (i > 0)
            for(int j = 0; j < i; j++)
            {
                if (Occupancy.get(i)
                        .equals(Occupancy.get(j)))
                    System.out.println(i
                            + " Repeated with " + j);
            }
    }
}


void printInfo()
{
    printAddresses();
    printOccupancy();

}
```

```
//=========================================================
//========= Tree as a ramificating structure is built=====
//=========================================================

//A binary string with 2*numberOfLeaves-1 bits
String randomBinaryString()
{
    //A random binary string
    bitTree = "";
    for (int i = 0; i < 2*numberOfLeaves-1; i++)
    {
        Integer j = r.nextInt(2);
        bitTree = bitTree + j.toString();

    }

    if (print)
        System.out.println(bitTree);
    return bitTree;
}

//Binary string s is decoded into an int
int stringToNumber(String s)
{
    int n = 0;
    int l = s.length();

    for(int j=0;j  < l;j++ )
    {
        char c  =   s.charAt(j);
        int  v = Character.getNumericValue(c);
        int q = j+1;
        int number1 = (int) (v * Math.pow(2,l-q));
        n  = n + number1;
    }
    return n;
}
```

```
//String address is interpreted as a binary number
//that corresponds with an int
int stringToInt(String address)
{
    int output = 0;
    //System.out.print("String = " + address );
    char c  = address.charAt(0);
    if  (c == '1') output = stringToNumber(address);
        else System.out.println("Bad encoding");
    //System.out.println("  int =  " + output);
    return output;
}


//All ancestors of node m are added
private  void  addAncestors(int m)
{
    String address = TreeInfo.get(m);
    int a = address.length();
    //If the address of  a node is 10001,
    //an ancestor has as address a left substring
    //of 0001, say 0, 00 or 000.
    //All of them must be added to TreeInfo

    for(int i = 0; i < a;  i++)
    {
        String leftSubstring
                = address.substring(0, i);
        leftSubstring = "1" + leftSubstring;
        int k = stringToInt(leftSubstring);
        //Updating of tree information
        if (k < bitTree.length()-1)
            bitTree = bitTree.substring(0, k)
                + "1" + bitTree.substring(k+1);
        else bitTree = bitTree.substring(0, k)
                + "1" ;
```

```
    }

}



//The tree encoded by binary string b is
//decoded into an array with binary addresses and contents.
//The tree corresponds with the lower
//part of a tree with one more generation.
private  void  createTree(String b )
{


    int l = b.length();
    char c ;
    String s ;
    //First two bits do not matter
    for (Integer i = 2; i < l; i++)
    {
        c = b.charAt(i);
        if (c == '1')
        {
            if ( i < 2*numberOfLeaves)
            {
                s = Integer.toBinaryString(i);
                //prefix "1" is deleted
                s = s.substring(1);
                TreeInfo.add(s);
                s = s.replace('0', '+');
                s = s.replace('1', '-');
                TreeInfo.add(s);
            }
        }
    }
}
```

```
//The tree encoded by binary string b is
//decoded and drawn.
//The tree corresponds with the lower
//part of a tree with one more generation.
private  void  createTree( )
{


    //A random binary string:
    //If bit I is 1, the I node is added to the tree
    bitTree = randomBinaryString();
    createTree(bitTree) ;


    //If a node exists, all its ancestors too.
    //bitTree will be modified in consequence.
    for(int m = 0; m < TreeInfo.size()-1;   m = m+2)
        addAncestors(m);

    TreeInfo.clear();
    createTree(bitTree);
}


//=========================================================
//========= Tree is realized as a virtual 3D shape =======
//=========================================================

/*
There are 6 directions given by
//the natural basis.
One distinct than d is chosen
*/
private int distinctDirection(int d)
{
    int i = r.nextInt(6);
    //System.out.println(i + "  " + d);
    if (i == d) distinctDirection(d);
```

```
    return i;
}


//A direction distinct than  backwards is chosen
private int notBackwards(int d)
{
    int backwards  = 0;
    switch (d)
    {
        case 0: backwards = 1; break;
        case 1: backwards = 0 ; break;
        case 2: backwards = 3; break;
        case 3: backwards = 2; break;
        case 4: backwards = 5; break;
        case 5: backwards = 4; break;
    }
    int i = distinctDirection(backwards);
    return i;
}

//Direction d is realized as a move
//There are 6 directions:
//0: +X, 1 = -X,  2 = +Y,  3 = -Y
//4 = +Z, 5 = -Z.
private  Point3D  move3D(int d)
{

    Point3D p = new Point3D(0,0,0);
    switch (d)
    {
        case 0: p = new Point3D( delta,0,0); break;
        case 1: p = new Point3D(-delta,0,0); break;
        case 2: p = new Point3D(0, delta,0); break;
        case 3: p = new Point3D(0,-delta,0); break;
        case 4: p = new Point3D(0,0, delta); break;
        case 5: p = new Point3D(0,0,-delta); break;
    }
```

```
    return p;
}

//Common initial substring to strings s and t
private  String commonSubstring(String s, String t)
{
    int a = s.length();
    int b = t.length();
    int min = a;
    if (b < a ) min = b;
    String c = "";
    for(int i  = 0; i < min; i++)
    {
        if (s.charAt(i) == t.charAt(i))
            c = c + s.charAt(i);
        else break;
    }
    return c;
}


//Geometric barriers are defined here
private  boolean isForbidden(Point3D thisPoint)
{
    boolean b = false;
    double x = thisPoint.getX();
    double y = thisPoint.getY();
    double z = thisPoint.getZ();
    //Codification of  cube of size 2
    //Unitary edge = 50
    if ((x < 0) || (x > 110)) b = true;
    if ((y < 0) || (y > 110)) b = true;
    if ((z < 0) || (z > 110)) b = true;
    return b;
}
```

```
//Is thisPoint already occupied?
private  boolean isOccupied(Point3D thisPoint)
{
    //printOccupancy();
    int l = Occupancy.size();
    boolean occupied = false;
    if (printIsOccupied)
        System.out.println("isOccupied?");
    //Points at the boundary of the shape are occupied
    if (isForbidden(thisPoint) )
    {
        if (printIsOccupied)
            System.out.println("Forbidden" + thisPoint);
        return true;
    }
    //(0,0,0) is occupied
    if (root3D.equals(thisPoint))
        {
            return true;
        }
    else
    {
        for(int i = 0; i < l; i++)
        {
            if (Occupancy.get(i).equals(thisPoint))
            {
                occupied = true;
                break;
            }
        }
        return occupied;
    }
}


//Is thisPoint already occupied?
private  boolean isPathRepeated(String path)
```

```
{
    int l = Paths3D.size();
    boolean repeated = false;
    if (printIsPathRepeated)
        System.out.println("isPathRepeated?");
    for(int i = 0; i < l; i++)
    {
        if (Paths3D.get(i).equals(path))
        {
            repeated = true;
            break;
        }
    }
    return repeated;
}


//A node has a path that must coincide with
//the paths of its ancestors.
//So, we look for the longest one
private String getAncestralPath(int i)
{
    String addressi = TreeInfo.get(i);

    String ancestralPath = "";
    String candidatePath;
    int f = Paths3D.size();
    List<String> Candidates
            = new  ArrayList<>(2*numberOfLeaves-1);
    for (int j = f-1; j >= 0; j--)
    {
        String addressj = TreeInfo.get(2*j);
        if (printGetAncestralPath)
            System.out.print(" addressi =  " + addressi
                + " addressj = " + addressj);
        String commonAddress
                = commonSubstring(addressi, addressj);
```

```
        int k = commonAddress.length();
        if (k == 0)
        {
            if (printGetAncestralPath)
                System.out.println(" No common subaddress ");
        }
        else
        {
            String path = Paths3D.get(j);
            candidatePath = path.substring(0, k);
            Candidates.add(candidatePath);
            if (printGetAncestralPath)
                System.out.println(" Common address = "
                    + commonAddress
                    + " Pathj = " + path
                    + "  candidate path = "
                    + candidatePath);

        }

    }
    int l = Candidates.size();
    int max = 0;
    for(int k = 0; k < l; k++)
    {
        String s = Candidates.get(k);
        int p = s.length();
        if (p > max)
        {

            max = p;
            ancestralPath = s;
        }
    }
    if (printGetAncestralPath)
                System.out.println(" ancestral path = "
                    + ancestralPath);
```

```
    return ancestralPath;
}


//Node i in the list TreeInfo
//needs an exclusive location in 3D.
//If possible, one is found.
private  String tryToGetPath(int i)
{
    //This is the root of the tree
    Point3D   P1 = new Point3D(0, 0,0);
    //Reinitialization
    lastPoint = P1;
    Point3D thisPoint = P1;
    //A child must share subpaths with their ancestors

    String address = TreeInfo.get(i);
    int l = address.length();

    String ancestralPath = getAncestralPath(i);
    String path = ancestralPath;
    int lengthPath = path.length();
    for (int j = 0; j < lengthPath; j++)
    {
        //Direction at j is read
        direction = Integer.parseInt("" + path.charAt(j));

        //thisPoint moves  along the chosen direction
        thisPoint = thisPoint.add(move3D(direction));

    }

    //New directions are added to complete path
    if (counterOfTrialsToFindPath< limit) //Overflow is interdicted.
    {
        //Directions are concatenated
        for (int j = lengthPath; j < l; j++)
        {
```

```
            //A non backwards new direction is chosen
            direction = notBackwards(direction);
            path = path + direction;
            //Chosen direction is executed in memory
            thisPoint = thisPoint.add( move3D(direction));
        }

        if (printTryToGetPath)
            System.out.println("\nAddress to try = "
                + address
                 + "   Ancetralpath = " + ancestralPath
                + " counterOfTrialsToFindPath= "
                + counterOfTrialsToFindPath
                +   " \nThisPoint = " + thisPoint);

        if ( (isPathRepeated(path))
                || (isOccupied(thisPoint)) )
        {
            if (printTryToGetPath)
                System.out.println("    Path is repeated.");
            counterOfTrialsToFindPath++;
            tryToGetPath(i);
        }
        else
        {
            if (printTryToGetPath)
                System.out.println(
                        "    Path is NOT repeated.");
            Occupancy.add(thisPoint);
            Paths3D.add(path);
        }
    }
    if (counterOfTrialsToFindPath== limit)
    {
        if (printTryToGetPath)
            System.out.println(
                    "    Path not found at trial "
                + counterOfTrialsToFindPath);
```

```
        path =  "-1";
    }

    return path;
}


//Information is created to realize
//and draw tree into 3D space.
private  void treeTo3D()
{
    String path ;

    //Directions are proposed at random,
    //and concatenated if no double occupancy appears
    for (int i = 0; i < TreeInfo.size(); i= i+2)
    {
        if (printTreeTo3D)
            System.out.print("Node i = " + i
                + "  Address = " + TreeInfo.get(i));
        counterOfTrialsToFindPath= 0;
        path = tryToGetPath(i);
        if (printTreeTo3D)
            System.out.println("  Path at treeTo3D = "
            + path);
        if (path.contentEquals("-1"))
        {
            if (printTreeTo3D)
                System.out.println("Truncation");
        }
    }
    if (printTreeTo3D)
        System.out.println("Paths3D size = "
                + Paths3D.size());
    if (printTreeTo3D) printOccupancy();
}
```

```
//============================
//========= Master Method=====
//============================


private   void makeTree( )
{
    initialization();
    //TreeInfo is filled in.
    //A binary string is composed:
    //if bit i is 1, node i is added to the tree.
    //A binary address to each node of the tree
    //is following by a content in TreeInfo.
    createTree();
    //Information is created to realize
    //and draw tree into 3D space
    treeTo3D();
}
}


//==============================
//======= Random trials =========
//==============================



//Is the task done?
//Task: populate lateral and inner points
//with integer coordinates
//of a cube of edge 2
private   boolean weAreDone(ABinaryTree t)
{
    boolean b = false;
    int n = t.Occupancy.size();
    System.out.println("Number of occupied points = " + n);
    if (n == RequiredNumberOfVisitedPoints) b = true;
    return b;
```

```
}


//Various trees are tried and the last is returned
private   ABinaryTree  variousTrees( )
{
    int n = 0;
    ABinaryTree t = null;
    for(int i = 0; i < maxNumberOfTrials; i++)
    {
        t = new ABinaryTree();
        t.counterOfTrialsCube = i;
        if ( weAreDone(t))   break;
        if (t.counterOfTrialsCube < maxNumberOfTrials-1)
        {
            t.TreeInfo.clear();
            t.Occupancy.clear();
            t.Paths3D.clear();
            n = t.counterOfTrialsCube;
        }
    }
    System.out.println("Demanded number of visitied points = " +
            RequiredNumberOfVisitedPoints
            + "\nNumber of done trials  = "
            +  n);
    return t;
}


//================================
//======= 3D graphics ===========
//================================



//Instantiation of the  3D graphics class
private Node create3dContent(ABinaryTree t )
{
```

```
    Project3D  p =  new Project3D( t );
    return new Group(p);
}


//The drawing is synthesized
private class Project3D extends Group
{

//Constructor
Project3D(ABinaryTree t)
{
    //Tree is drawn to expand a 3D bush
    Group g = drawTree(t );
    getChildren().addAll(g );
}

//Returns a 3D line that goes from lastPoint
//to thisPoint
private  Line3D  line3DTo(Point3D thisPoint)
{
    Line3D l = new Line3D(lastPoint, thisPoint);
    lastPoint = thisPoint;
    return l;
}


private   Group drawTree(ABinaryTree t )
{
    Group myGroup  = new Group();
    //This is the root of the tree
    Point3D   P1 = new Point3D(0, 0,0);


    //Put a landmark at the root
    Sphere sphere = new Sphere();
    sphere.setRadius(10);
    sphere.setTranslateX((int) P1.getX());
```

```
        sphere.setTranslateY((int) P1.getY() );
        sphere.setTranslateZ((int) P1.getZ() );
        root.getChildren().add(sphere);

        //Information is created to realize
        //and draw tree into 3D space
        t.treeTo3D();
        t.Occupancy.clear();
        //The tree is drawn:
        for (String Paths3D1 : t.Paths3D)
        {
            //Reinitialization
            lastPoint = P1;
            Point3D thisPoint = P1;
            String path = Paths3D1;
            if (printDrawTree)
                System.out.println( "Path = " + path);
            int length = path.length();
            for (int j = 0; j < length; j++)
            {
                //Direction at j is read
                direction = Integer.parseInt("" + path.charAt(j));
                if (printDrawTree)
                    System.out.print(direction + "   ");
                //thisPoint moves  along the chosen direction
                thisPoint = thisPoint.add(t.move3D(direction));
                //Draw a stroke
                //from lastPoint to thisPoint
                Line3D l = line3DTo(thisPoint);
                myGroup.getChildren().add(l.group);
            }
            if (printDrawTree)
                System.out.println();
            t.Occupancy.add(thisPoint);
        }
        return myGroup;
    }
    }//End of inner class Project3D
```

```
public static void main(String[] args)
{
    launch(args);
}

}//End of program N111  TreeToShapes14
```

6.1 Biblio

Bibliography

[1] JONATHAN "LONESOCK" DUMMER (2015) A Simple Time-Corrected Verlet Integration Method
http://lonesock.net/article/verlet.html
Verified 6/XI/2015 66

[2] SCOTT HOMMEL (2013) Using the Image Ops API
https://docs.oracle.com/javafx/2/image_ops/jfxpub-image_ops.htm
Verified 6/XI/2015 17

[3] JAMA (2012) JAMA : A Java Matrix Package
http://math.nist.gov/javanumerics/jama/
Verified 7/XI/2015 73

[4] LYDIA MICHAEL AND JAMIE A DAVIES (2015) Pattern and regulation of cell proliferation during murine ureteric bud development. Journal of Anatomy
http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1571296/#!po=5.00000
Verified 7/XI/2015 71

[5] WIKIPEDIA (2015) Euler angles
http://en.wikipedia.org/wiki/Euler_angles
Verified 7/XI/2015 79

[6] ORACLE (2015) Oracle JDK 8 and JRE 8 Certified System Configurations
http://www.oracle.com/technetwork/java/javase/certconfig-2095354.htm
Verified 5/XI/2015 v

[7] POLLATRINI (2014) F(X)yz An open source JavaFX 3D Visualization and Component Library
http://birdasaur.github.io/FXyz/
Verified 6/XI/2015 37

[8] ALLA REDKO, 2013 (2013) Using JavaFX UI Controls
https://docs.oracle.com/javafx/2/ui_controls/tree-view.htm
Verified 9/XI/2015 221

[9] M SUZANNE AND H STELLER (2013) Shaping organisms with apoptosis. Cell Death and Differentiation (2013) 20, 669–675; doi:10.1038/cdd.2013.11
http://www.nature.com/cdd/journal/v20/n5/full/cdd201311a.html
Verified 7/XI/2015 71

[10] THE MEDICAL BIOCHEMISTRY PAGE (2015) DNA metabolism, Repair of Damaged DNA (2015) 20, 669–675; doi:10.1038/cdd.2013.11
http://themedicalbiochemistrypage.org/tmbp-sitemap.php
Verified 9/XI/2015 216