JAVA FOR THE STUDY OF EVOLUTION

VOLUME XII

FOSSILS

José del Carmen Rodríguez Santamaría
The EvolJava Community

Contents

1 Worms **1**
- 1.1 Bauplan . . . 1
- 1.2 Old Java style . . . 3
- 1.3 Conclusion . . . 20

2 Introduction to JavaFX: Canvas **21**
- 2.1 A canvas demo . . . 21
- 2.2 Trying movies . . . 24
- 2.3 Conclusion . . . 33

3 JavaFX: transitions **35**
- 3.1 Transitions . . . 35
- 3.2 Timelines . . . 44
- 3.3 Rides . . . 63
- 3.4 The tic-jump technology . . . 66
- 3.5 Conclusion . . . 72

4 Worms in JavaFX **73**
- 4.1 Movement by human design . . . 73
- 4.2 Movement at random . . . 83
- 4.3 Java Lists . . . 104
- 4.4 Movement by evolutionary learning . . . 157
- 4.5 Worms leave fossils . . . 186
- 4.6 Conclusion . . . 187

5 User interfaces in JavaFX **189**
- 5.1 Getting started . . . 189
- 5.2 Examples . . . 190
- 5.3 Conclusion . . . 221

6 Worms as a laboratory **223**
6.1 A GUI for our worms 223
6.2 Improving the code 293
6.3 Conclusion 293

7 JavaFX for more improvement **295**
7.1 General timelines 295
7.2 Breaking long-running tasks 302
7.3 Troubles with memory 307
7.4 JavaFX properties and binding 332
7.5 Check Boxes 353
7.6 Conclusion 359

8 Worms in JavaFx **361**
8.1 Let us try it out 361
8.2 Our final version 366
8.3 Graduation 366
8.4 Conclusion 368

9 Fossils and the falsification of the Evolutionary theory **369**

Answers to exercises **371**

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 twelfth volume: You have been taught that fossils are the proof of evolution. Anymore: here you learn how to argue that fossils are the very proof that the Evolutionary Theory of the origin of species is obviously false. All one needs to do is to simulate evolution including fossil analysis. The falsity of the Evolutionary Theory results from the fact that the fossil record is predicted to be filled in mediocre solutions while nature completely lacks them. Programs are written in JavaFX, the modern Java way of doing graphics and animations. Abundant pedagogic material is provided.

Bogotá, Colombia, *José Rodríguez*
October 2014

Introduction

Fossils are considered by present science as a fundamental proof of the Evolutionary Theory of the origin of species. Our purpose for this volume is to show that to accept that belief one must reject the scientific method. In fact:

According to this method one must work out the postulates of the theory under test to make an account of all predictions that must be next compared with facts. If relevant facts are explained by the theory, this is subjected to corrections to achieve refinement. But if relevant facts contradict the massive predictions of the theory, it must be archived and new ideas must be put forward to be tested.

So, to prove that the Evolutionary Theory is false, all one must do is to simulate evolution including fossil analysis. The falsity of the Evolutionary Theory results from the fact that the fossil record is predicted to be filled in mediocre solutions while nature completely lacks them.

Thus, our purpose in this volume is to illustrate how to make a full fledged simulation of evolution that includes fossil reconstruction and analysis. This is done in program L148 WormsEvoJFXa14. We try to do this according to the demands of modernity so, we go over a long path of prerequisites:

In the first chapter we use Swing to present what the whole dealt consists in: we simulate a population of worms that is taught by Evolution to climb the screen. The simulation includes fossil record and analysis. The rest of this volume is dedicated to rewrite this material using JavaFX. This giro represents the new and professional Java way of making graphics and animations. Since JavaFX is very rich, we present abundant pedagogic material. Although we made many trials to implement our worms in JavaFX, we deleted many of them. The reason is that we decided to favor low size of associated files in order to make more expedite their transfer through Internet.

We need Java 8 and JavaFX. The easiest way of getting them is to download the **netbeans + JDK bundle** from (Oracle, [4], 2014). In case of link outdating, type in Google: *Java SE Downloads - Netbeans + JDK Bundle - Oracle* and look

for a link to Oracle.

NetBeans is the platform of Oracle to run Java. Play with this new tool: in the *start page* select *Try a sample project*, choose *Samples*, next *JavaFX*. Select project *Fireworks*. Expand the directory and find the source folder. Expand it and find the source of the Java program, which ends with *.java*. Select it with return or with the secondary button of the mouse to open it. To run it, click over the editor window and with the secondary button of the mouse open a pop up menu and select the command to run the file. You must see a wonderful animation.

If you know Eclipse, NetBeans must be mastered very quickly. It opens interesting possibilities such as the automatic creation of code.

Next, you must download, import and adapt our programs for the chosen platform. Our programs come in the file *javaevol12P.zip*. Download it and memorize its location. Open NetBeans, go to menu *File* → *Import Project* → *From zip*. Search the zipped program and click OK.

To **run a program**, say *WormsHD*, go to *projects* tab , choose project *eJVolu12P*, expand the tree: *source packages* → *eJVolu12P* → *WormsHD.java*. In the window of the projects tab (not in the editor window) right click over the name of the program and choose in the emergent menu *Run file*. Or, use *Shift* + *F6*. You shall see a window with some worms that climb the screen. To close a running program use the menu *Run* → *Stop Build/Run* or look for a red square under the *Output tab* and activate it.

You are guaranteed to have a happy start. Otherwise, write to me (jose@evoljava.com) to negotiate a truly functional set of instructions to start with.

If you want to work with Eclipse, you are challenged to research how to install Java 8 with the SE Development Kit 8 or higher from Oracle to go next to (Efxclipse, [1], 2014) to download and install the bundle consisting of Eclipse SDK + JavaFX.

Chapter 1

Worms

`Coherence behavior`

1 ***Purpose.*** *We simulate a worm that has various links that can be moved independently. Evolution is given the task of enabling the worm to move its parts coherently that it could rise up a wall. Death and fossil sampling are included. Worms were introduced in Vol II to illustrate how OOP (Object Oriented Programming) works.*

We first compose the code in our usual Java style and next we rewrite the code using JavaFX that today comes packed in Java 8.

1.1 Bauplan

We simulate a worm that climbs up the screen. The task is to develop the software to enable the worm to climb up the screen. We present three solutions: a human design, random assignments and design by evolution.

2 ***The structure of the worm.***

We introduced worms in Vol II. Our worms have links. They can be moved independently but the structure of an organism must be kept as a whole: movements cannot separate the organism which must remain in one piece. Every link has 8 possibilities of movement plus another one of making nothing. We are allowed to move over a reticulate.

One shall check whether or not a given movement of a link disrupts the organism. For instance, if the worm is totally stretched, a forward movement of the

head would disrupt the organism and so it is not allowed. If the worm is stretched, allowed movements are sidewards.

We posit the worm at the bottom of our screen and its task is to arrive at the upper edge. Since movements must be correlated to conserve the structure, movements are packed into the vector mover[]. The structure of the worm is shown in the next figure:

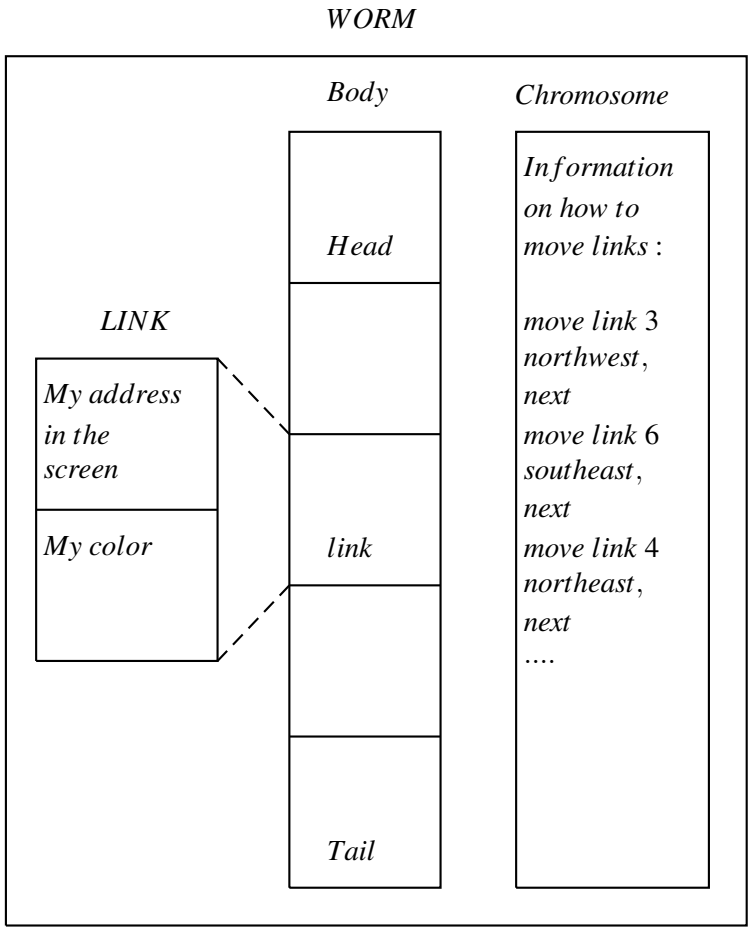


Figure 1.0. The structure of a worm: a body with various links and a chromosome that keeps the information on how to move links.

1.2 Old Java style

Using old Java (the usual Java until Vol XI), we by ourselves instruct a population of worms on how to move their links.

3 ***Worms move: human design***

Let us program with our intelligence our worms to climb the screen.

```
/*
 * Program L3 WormsHD
 A population of worms
 is programmed
 to climb the screen.
 (Human design).

 A worm consists of:
 links plus position plus movers.
 A mover is an ordered list of
 instructions to move:
 which link and in which direction.
 */

import java.awt.Color;
import java.awt.Graphics;
import java.util.Random;

import javax.swing.JFrame;




public class WormsHD extends JFrame
{
 private static final long serialVersionUID = 1L;

 // Constructor
 private WormsHD()
 {
```

```
}

// Instantiation of main class
static WormsHD t = new WormsHD();



///Definition of global variables.
//Number of worms
private static int nWorms = 8;
private static Worm[] worms = new Worm[nWorms];
//Number of links of each worm
private static int nLinks = 7;
//Initial position over the screen of the head
private static int initialRow = 85;
//  Turn on  the random generator
static Random r  = new Random();


/* Definition of inner classes.
 *
   The Move class:
   A Move says in which direction
   a link  must be moved.
*/

static private  class Move
{
 int x;
 int y;

  // The next is a method
  // that  declares
  // how information is acquired.
  // It is a constructor.
  // It is overload by two possibilities.
  //a) Default method: move to the right
  Move()
   {
```

```
     x= 1;
     y= 1;
    }

//b) copy a couple of numbers
Move( int a, int b)
    {
     x= a;
     y= b;
    }
  }
```

```
 // Definition of the class Mover:
 // A Mover keeps two types of information:
 // a) the link of the worm to be moved and
 // b) the direction or move to be chosen.
 private   class Mover
  {
int link ;
    Move   move ;

    //  Movers can acquired information
    // in two ways, they are  overloaded.
    // Form 1: by default
    private Mover()
  {
       link= 0;
       move = new Move();
      }

  // Form 2, from structured data
  private Mover(int a, Move m)
  {
link= a;
move = m;
move.x = m.x;
```

```
move.y = m.y;
  }
 }

 // The class LinkAddress keeps the actual
 // position of a link.
 private   class LinkAddress
 {
  int x;
  int y;

  //Constructor:
  //This is how information is acquired
  LinkAddress(int a, int b)
  {
  x= a;
  y = b;
  }
 }


 /* Here we define the Bauplan of the worm.
    A worm is composite: it has links.
    Mover[ ]  keeps the information
    on how to move its links:
    which  link and in which order
    (in the order that they appear in
    the vector Mover[]), and
    in which direction.
    Each link has its own color.
    Each worm keeps record of
    the actual position of each
    one of its links,
    in linkAddress[]. */
 private class Worm
  {
   Mover mover[ ] ;
   Color color[];
   LinkAddress  linkAddress[ ];
   //move right (side = 1) else left (side = -1)
```

```
   int side = 2*r.nextInt(2) -1;

   //Default initialization
   private Worm( )
   {
mover = new Mover[nLinks];
linkAddress = new LinkAddress[nLinks];
color = new Color[nLinks];
int c1,c2,c3; //BGR components of color
for(int m = 0; m < nLinks; m++)
 {
  c1 = r.nextInt(226);
  c2 = r.nextInt(226);
  c3 = r.nextInt(226);
  color[m] = new Color(c1,c2,c3);
     }
//Which direction to move?
int link;
for(int m = 0; m  < nLinks; m++)
  {
   //Move the tail first and then the others links.
   link = nLinks-m-1;

   Mover mover1;
   if (side == 1)
    mover1  =   t.new Mover(link,new Move(1,1));
    else
     mover1  =   t.new Mover(link,new Move (-1,1));
   mover[m]= mover1;
  }
   }
 }//End of Worm class declaration




//The application begins here
public static void main(String args[])
{
```

```
 int xWin = 200;
 int yWin = 200;
 //t is the instance of wormHD
 t.setSize(xWin,yWin);
 t.setBackground(Color.lightGray);
 t.setVisible(true);
}



//Initial position of worm W at the screen
private static Worm start(int W, Graphics g)
{
 int k, a,b;
 //Auxiliary worm
 Worm  worm  = t.new Worm();
 //Every link is positioned
 for(int  m = 0; m < nLinks; m++)
  {
// All worms are posited at the same initial row
// in vertical position
k = initialRow - nLinks;
a = 20 + 20 * W;
b = k + 12*m;
LinkAddress   ad = t.new LinkAddress (a,b);
worm.linkAddress [m] = ad;

// The color is prepared
g.setColor( worm.color[m]);
// A Rectangle is defined by
// the coordinates of the
// superior right corner,
// width and height.
//The link is painted
g.fillRect(a,b,4,12);
  }
 return worm;
}
```

```
/*
If the worm moved to the left, let it move to the right
and vice verse.
*/
private static void updateMoves(Worm worm )
{
 for(int m = 0; m  < nLinks; m++)
  worm.mover[m].move.x = - worm.mover[m].move.x;
}


//To climb the screen the worm
//translates each one of its links
//according to the information
//consigned in mover[]
private static Worm translate(Graphics g, Worm worm)
{
 // Reads and execute the
 // information to move.
 int n, i, j;

 for (int m = 0; m < nLinks ; m++)
 {
  //Reads the link to be moved
  n = worm.mover[m].link;

  //To move a link:
  //the cell occupied by  the link is cleared,
  //the cell to be occupied is painted.

  //The occupied cell is cleared:
  //it is painted in the background color
  i = worm.linkAddress[n].x;
  j = worm.linkAddress[n].y;
  g.setColor(Color.lightGray);
  g.fillRect(i,j,4,12);

  // A new place is defined:
  //The move to be done is read
  Move  move1;
```

```
  move1 = worm.mover[m].move;
  i = i +4* move1.x;
  j = j - 6*move1.y;
  worm .linkAddress[n].x = i;
  worm .linkAddress[n].y = j;

  //color for painting
  g.setColor( worm.color[nLinks-m-1]);
  g.fillRect(i,j,4,12);
  //If movement is too fast, nothing can be seen.
  //To attune the velocity of the worm,
  //augment or diminish the delay as   pleased
  try
   {
 Thread.sleep(1000L);    // one second = 1000L
   }
 catch (InterruptedException e) {}
 }//End of the m-loop
 return worm;
}



@Override
//Master of painting.
//It is called automatically after main
public    void  paint(Graphics g)
{
 //initialization:
 /*  The index W distinguishes each worm.
 Each worm has its color
 Each worm has nLinks links,
 each part has its own coordinates on the screen,
 which are kept in the linkAddress   array.
 The worms are posited at the bottom of the screen.
 Head are at row InitialRow */
 for (int W = 0; W < nWorms; W++)
  {
worms[W]=  start(W,g);
  }
```

```
 // Climbing nSteps
 int nSteps = 68;
 for (int l = 0; l < nSteps; l++)
  {
   for (int W = 0; W < nWorms; W++)
{
 Worm  worm;
 worm = worms[W];
 updateMoves(worm);
 worm =  translate(g,worm);
 worms[W]=  worm ;
 }
  }
 }
}//End of main class L3 WormsHD
```

4 ***Exercise.*** *Run the program and play with the code.*

As we saw, human design is very effective. And, what about the capabilities of design of randomness?

5 ***Movement at random.*** *Worms are programmed by randomness in the next code to move links.*

```
/*
 * Program L5 WormsRand
 A population of worms
 is programmed by randomness
 to climb the screen.


 A worm consists of:
 links plus position plus movers.
 A mover is an ordered list of
 instructions to move:
 which link and in which direction.
 */
```

```
package ejvolu12p;
import java.awt.Color;
import java.awt.Graphics;
import java.util.Random;

import javax.swing.JFrame;



public class WormsRand extends JFrame
{
 private static final long serialVersionUID = 1L;

 // Constructor
 private WormsRand()
 {  }

 // Instantiation of main class
 static WormsRand t = new WormsRand();


 ///Definition of global variables.
 //Number of worms
 private static int nWorms = 8;
 private static Worm[] worms = new Worm[nWorms];
 //Number of links of each worm
 private static int nLinks = 7;
 //Initial position over the screen of the head
 private static int initialRow = 85;
 //Turn on  the random generator
 static Random r  = new Random();


 /* Definition of inner classes.
  *
    The Move class:
    A Move says in which direction
    a link  must be moved.
 */
```

```
 static private  class Move
 {
  int x;
  int y;

   // The next is a method
   // that  declares
   // how information is acquired.
   // It is a constructor.
   // It is overload by two possibilities.
   //a) Default method
   Move()
    {
     x= 1;
     y= 1;
    }

   // b) The input is a list
   Move(int a, int b)
    {
   x= a;
   y= b;
    }
  }

 // Definition of the class Mover:
 // A Mover keeps two types of information:
 // a) the link of the worm to be moved and
 // b) the direction or move to be chosen.
 private   class Mover
{
int link ;
    Move   move ;

    //  Movers can acquired information
    // in two ways, they are  overloaded.
    // Form 1: by default
    private Mover()
    {
       link = 0;
```

```
       move = new Move();
    }

    // Form 2, from structured data
    private Mover(int a, Move m)
    {
        link = a;
        move = m;
        move.x = m.x;
        move.y = m.y;
    }
}

 // The class LinkAddress keeps the actual
 // position of a link.
 private   class LinkAddress
 {
  int x;
  int y;

    //Constructor:
    //This is how information is acquired
    LinkAddress(int a, int b)
    {
        x= a;
        y = b;
    }
 }


 /* Here we define the archetype of worm.
    A worm is composite: it has links.
    Mover[ ]  keeps the information
    on how to move its links:
    which  link and in which order
    (in the order that they appear in
    the vector Mover[]), and
    in which direction.
    Each link has its own color.
    Each worm keeps record of
```

```
    the actual position of each
    one of its links,
    in linkAddress[]. */
private class Worm
{
    Mover mover[ ] ;
    Color color[];
    LinkAddress  linkAddress[ ];
    int side = r.nextInt(2);

    //Default initialization
    private Worm( )
    {
        mover = new Mover[nLinks];
        linkAddress = new LinkAddress[nLinks];
        color = new Color[nLinks];
        int c1,c2,c3; //BGR components of color
        for(int m = 0; m < nLinks; m++)
        {
            c1 = r.nextInt(226);
            c2 = r.nextInt(226);
            c3 = r.nextInt(226);
            color[m] = new Color(c1,c2,c3);
        }
    }
}//End of Worm class declaration



//Vector with the library of allowed moves
private static Move[] vectMoves = new Move[10];



//The application begins here
public static void main(String args[])
{
    int xWin = 200;
    int yWin = 200;
    //t is the instance of wormHD
```

```
    t.setSize(xWin,yWin);
    t.setBackground(Color.lightGray);
    t.setVisible(true);
    t.setDefaultCloseOperation(EXIT_ON_CLOSE);
}
```

```
//Initial position of worm W at the screen
private static Worm start(int W, Graphics g)
{
 int k, a,b;
 //Auxiliary worm
 Worm  worm  = t.new Worm();
 //Every link is positioned
 for(int  m = 0; m < nLinks; m++)
  {
    // All worms are posited at the same initial row
    // in vertical position
    k = initialRow - nLinks;
    a = 20 + 20 * W;
    b = k + 12*m;
    LinkAddress   ad = t.new LinkAddress (a,b);
    worm.linkAddress [m] = ad;

    // The color is prepared
    g.setColor( worm.color[m]);
    // A Rectangle is defined by
    // the coordinates of the
    // superior right corner,
    // width and height.
    //The link is painted
    g.fillRect(a,b,4,12);
  }
 return worm;
}
```

```
static public void moveDef()
{
 // Here we define 9 cardinal points
 final Move move0 =  new Move(0,1);
  vectMoves[0]   = move0 ;
 final Move move1 =  new Move(1,1);
  vectMoves[1]   = move1 ;
 final Move move2 =  new Move(1,0);
  vectMoves[2]   = move2 ;
 final Move move3 =  new Move(1,-1);
  vectMoves[3]   = move3 ;
 final Move move4 =  new Move(0,-1);
  vectMoves[4]   = move4 ;
 final Move move5 =  new Move(-1,-1);
  vectMoves[5]   = move5 ;
 final Move move6 =  new Move(-1,0);
  vectMoves[6]   = move6 ;
 final Move move7 =  new Move(-1,1);
  vectMoves[7]   = move7 ;
 final Move move8 =  new Move(0,0);
  vectMoves[8]   = move8 ;
}


//There are 8 possible moves
//and one is chosen at random.
private static void chooseMoves(Worm worm )
{
    int side = worm.side;
    int link;

    for(int m = 0; m  < nLinks; m++)
    {
        //Move the tail first and then the others links.
        link = nLinks-m-1;
        //Choose a move at random
        Move move = vectMoves[r.nextInt(9)];
        Mover mover =  t.new Mover(link,move);
        worm.mover[m]= mover;
    }
```

```
    //flip direction
    if ( side == 0)  worm.side   = 1;
        else worm.side= 0;

}



//To climb the screen the worm
//moves each one of its links
//according to the information
//consigned in mover[]
private static Worm translate(Graphics g, Worm worm)
{
    // Reads and execute the
    // information to move.
    int n, i, j;

    for (int m = 0; m < nLinks ; m++)
    {
        //Reads the link to be moved
        n = worm.mover[m].link;

        //To move a link:
        //the cell occupied by  the link is cleared,
        //the cell to be occupied is painted.

        //The occupied cell is cleared:
        //it is painted in the background color
        i = worm.linkAddress[n].x;
        j = worm.linkAddress[n].y;
        g.setColor(Color.lightGray);
        g.fillRect(i,j,4,12);

        // A new place is defined:
        //The move to be done is read
        Move  move1;
        move1 = worm.mover[m].move;
        i = i +4* move1.x;
```

```
        j = j - 6*move1.y;
        worm .linkAddress[n].x = i;
        worm .linkAddress[n].y = j;

        //color for painting
        g.setColor( worm.color[nLinks-m-1]);
        g.fillRect(i,j,4,12);
        //If movement is too fast, nothing can be seen.
        //To attune the velocity of the worm,
        //augment or diminish the delay as   pleased
        try
        {
            Thread.sleep(1000L);    // one second = 1000L
        }
        catch (InterruptedException e) {}
        }//End of the m-loop
        return worm;
}



@Override
//Master of painting.
//It is called automatically after main
public    void  paint(Graphics g)
{
 //initialization:
 /*  The index W distinguishes each worm.
        Each worm has its color
        Each worm has nLinks links,
        each part has its own coordinates on the screen,
        which are kept in the linkAddress   array.
        The worms are posited at the bottom of the screen.
        Heads are at row InitialRow */
    for (int W = 0; W < nWorms; W++)
      worms[W]=  start(W,g);
     moveDef();

    //Climbing nSteps
    int nSteps = 68;
```

```
    for (int l = 0; l < nSteps; l++)
    {
        for (int W = 0; W < nWorms; W++)
        {
            Worm  worm;
            worm = worms[W];
            //Moves are chosen at random
            chooseMoves(worm);
            worm =  translate(g,worm);
            worms[W]=  worm ;
        }
    }
 }
}//End of main class L5 WormsRand
```

6 ***Exercise.*** *Run the program and play with the code.*

7 ***An evolutionary solution is presented in program L7 WormsEvoJOld. Old Java style. This program is not copied here but appears in the zipped file.***

8 ***Exercise.*** *Run the program and play with the code.*

1.3 Conclusion

Using Java Swing, we have shown how is the movement of worms that are programmed by a human being, by randomness and by evolution. Fossilization and ensuing fossil show also has been simulated.

Chapter 2

Introduction to JavaFX: Canvas

`Java graphics for modern times`

9 ***Introduction, purpose and warning.*** *Java was blamed during many years because of the very poor implementation of its graphics facility. During that time much progress about that respect was made by many teams all around the world. All this experience has been now refurnished into a Java extension for graphics along the most high standards. The result is* ***JavaFX*** *that means Java Effects (Oracle, Java platform, [5], 2014). A virtue of JavaFX is that it has a simple machinery for simple tasks and a heavy duty machinery for complex ones. In this introduction we get acquainted with* ***Canvas****, the representative of the simple machinery that will allow us to draw figures as easily as we did it with the ancient style used in the previous chapter. So, we will rewrite the program of moving worms L7 WormsEvoJOld into JavaFX Canvas.*

The Reader is warned that below we will learn to manage the heavy duty machinery of JavaFX. Our one and only purpose for this volume is to implement our program of Worms in that language. It will be inevitable to think that JavaFX is too complicated and that Swing is better because is simpler and nevertheless is good. Besides, JavaFX is an ongoing process that is still being under maturation. Nevertheless, JavaFX is thought to be a tool for professionals that want to get ready for the future that has been already knocking at the door since long ago. It is with that spirit that the Reader must endure all difficulties through the study of present material.

2.1 A canvas demo

Let us learn to use Canvas of JavaFX to make drawings: it is very easy.

10 ***The gallery of CanvasFX is shown by the next code in which we also get acquainted with the general form of a program in JavaFX.***

```
//Program L10 CanvasFX
//How to use Canvas of JavaFX
//to make drawings.


package canvas;

import javafx.application.Application;

import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;

import javafx.scene.paint.Color;
import javafx.scene.shape.ArcType;
import javafx.stage.Stage;


public class CanvasFX extends Application
{

    //This method is called automatically to begin with
       @Override
    public void start(Stage primaryStage)
    {
        //Title
        primaryStage.setTitle("Canvas gallery");
        //Graphics are associated to a tree,
        //root is declared
        Group root = new Group();
        //CanvasFx is declared
        Canvas canvas =   new Canvas(300, 250);
        //Canvas' graphical mode
        GraphicsContext g = canvas.getGraphicsContext2D();
        //Make drawings
```

```
        drawings(g);
        //Centering
        canvas.setTranslateX(20);
        //canvas is added to tree
        root.getChildren().add(canvas);
        //The tree is addressed to stage
        primaryStage.setScene(new Scene(root));
        //Show begins
        primaryStage.show();

    }

    //Drawings using CanvasFX
    private void drawings(GraphicsContext g)
    {
        //Shapes with strokes
        g.setStroke(Color.BLUE);
        g.setLineWidth(5);
        //First point = (10,40)
        g.strokeLine(10,40, 10, 80);
        //Oval = a point + width and height
        g.strokeOval(40, 40, 30, 70);
        //Arc = oval + start angle + extent angle
        g.strokeArc(120, 40, 30, 30, 45, 240, ArcType.OPEN);
        g.strokeArc(180, 40, 30, 30, 45, 240, ArcType.ROUND);
        g.strokeArc(250, 40, 30, 30, 45, 240, ArcType.CHORD);

        //Broken line
        //Vector with x-coordinates
        double[] X1 = new double[]{110, 140, 110, 120};
        //Vector with y-coordinates
        double[] Y1 = new double[]{ 70, 120, 100, 140};
        //4 = number of drawn points
        g.strokePolyline(X1,Y1, 4);

        //Polygon
        double[] X2 = new double[]{160, 190, 160};
        double[] Y2 = new double[]{ 90, 120, 150};
        g.strokePolygon(X2, Y2, 3);
```

```
        //Filled shapes
        g.setFill(Color.RED);
        g.fillArc(10, 150, 30, 30, 45, 240, ArcType.OPEN);
        g.fillArc(60, 150, 30, 30, 45, 240, ArcType.CHORD);
        g.fillArc(110, 150, 30, 30, 45, 240, ArcType.ROUND);
        g.fillPolygon(new double[]{160, 190, 160},
                        new double[]{90, 120, 150}, 3);

        //Drawing a white figure to make a void
        g.setFill(Color.WHITE);
        g.fillOval(160,120,20,50);

        //Drawing text
        g.setFill(Color.CADETBLUE);
        g.fillText("Text in JavaFX Canvas" , 20, 210);
        //Color resetting
        g.setFill(Color.GREEN);
        //Rectangle = a point + width and height.
        g.fillRect(180, 195, 30, 20);
        //Clearing out a rectangular area
        g.clearRect(25, 150, 3, 20);
    }


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

}//End of main class L10 CanvasFX
```

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

2.2 Trying movies

We know how to use CanvasFX to draw some static figures. Let us see whether or not we can use it to generate some dynamics. Because CanvasFx is so similar

as the Graphics of AWT, we will try a straightforward rewriting of a program that encodes for a movie in the AWT system of graphics.

12 ***The code for a movie in java.awt.graphics follows.***

```
//Program L12 MovieAWT
//This class  draws a moving
//face with blinking eyes.
//Face moves vertically.

package movieawt;

import java.awt.Color;
import java.awt.Graphics;

import javax.swing.JFrame;

  public class MovieAWT extends JFrame
{


    private static final long serialVersionUID = 1L;
    private static Long delayLong = 500L;
    private static Long delayShort = 100L;

    public void drawFace(Graphics g, int k)
    {
        //face
        g.setColor(Color.pink);
        g.fillOval(15,15+k,100,150);
        //hair
        g.setColor(Color.black);
        g.fillRect(3,3+k,130,35);
        //mouth
        g.setColor(Color.red);
        g.fillOval(50,120+k,30,20);
        //eyes
        g.setColor(Color.blue);
        g.fillOval(30,50+k,30,20);
```

```
        g.fillOval(70,50+k,30,20);
        //nose
        g.setColor(Color.red);
        g.drawRect(55,70+k,20,40);
    }

    public void  eraseface(Graphics g,int k)
    {
        // face
        g.setColor(Color.white);
        g.fillOval(15,15+k,100,150);
        //hair
        //g.setColor(Color.black);
        g.fillRect(3,3+k,130,35);
        //mouth
        //g.setColor(Color.red);
        g.fillOval(50,120+k,30,20);
        //eyes
        //g.setColor(Color.blue);
        g.fillOval(30,50+k,30,20);
        g.fillOval(70,50+k,30,20);
        //nose
        //g.setColor(Color.red);
        g.drawRect(55,70+k,20,40);
    }

    public void closeEyes (Graphics g, int k)
    {
        // We erase eyes:
        // eyes are painted in pink
        g.setColor(Color.pink);
        g.fillOval(30,50+k,30,20);
        g.fillOval(70,50+k,30,20);
        // We draw smaller eyes
        g.setColor(Color.blue);
        g.fillOval(30,65+k,30,5);
        g.fillOval(70,65+k,30,5);
    }

    public void openEyes (Graphics g, int k)
```

```
    {
        // small eyes are erased
        // We draw smaller eyes
        g.setColor(Color.pink);
        g.fillOval(30,65+k,30,5);
        g.fillOval(70,65+k,30,5);
        //We redraw eyes:
        // eyes are drawing in blue
        // and normal size
        g.setColor(Color.blue);
        g.fillOval(30,50+k,30,20);
        g.fillOval(70,50+k,30,20);

    }

    public void delay(long l1,int l)
    {
while  (System.nanoTime() < l1 + l) {delay(l1,l);}
    }


    public void paint(Graphics g)
    {
        drawFace(g,0);
        // This loop controls the numbers
        // of cycles of the graphic
        for(int kl = 1; kl<=5; kl++)
        {
            // k is the vertical jump
            int k = 10*kl;
            //We erase everything
            g.clearRect(0,0,200,200);
            // We draw face in the new place
            drawFace(g,k);
            // This loop controls the   blinking number
            for(int i=1; i<=2; i++)
            {
                // A delay must be introduced to make
                //  movement visible for a human being.
                System.out.print("Cycle = " + kl);
```

```
                System.out.println(". Blinking = " + i );

                try
                {
                   Thread.sleep(delayLong);
                }
                    catch (InterruptedException e) {}

                closeEyes(g, k);

                try
                {
                Thread.sleep(delayShort);    // one second = 1000L
                }
                    catch (InterruptedException e) {}

                openEyes(g,k);
            }
        }
        System.out.println("Job done.");
    }

    public static void main( String args[] )
    {
        MovieAWT m = new MovieAWT();
        int xWin = 200;
        int yWin = 400;
        m.setSize(xWin,yWin);
        m.setBackground(Color.lightGray);
        m.setVisible(true);
        m.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

}//End of main class L12 MovieAWT
```

13 ***Exercise.*** *Run the program and play with the code.*

Let us rewrite the previous program into CanvasFX.

14 ***The next code tries to produce a movie in CanvasFX.***

```
//Program L14 MovieCanvasFX
//Attempt of using FXCanvas to make a movie

package ejvolu12p;

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.paint.Color;
import javafx.stage.Stage;


public class MovieCanvasFX extends Application
{

    private static final Long delayLong = 500L;
    private static final Long delayShort = 100L;

    //This method is called automatically to begin with
       @Override
    public void start(Stage primaryStage)
    {
        //Title
        primaryStage.setTitle("Canvas gallery");
        //Graphics are associated to a tree,
        //root is declared
        Group root = new Group();
        //CanvasFX is declared
        Canvas canvas =   new Canvas(155, 250);
        //Canvas' graphical mode
        GraphicsContext g = canvas.getGraphicsContext2D();
        //Make drawings
        movie(g);
        //Centering
        canvas.setTranslateX(20);
        //canvas is added to tree
```

```
        root.getChildren().add(canvas);
        //The tree is addressed to stage
        primaryStage.setScene(new Scene(root));
        //Show begins
        primaryStage.show();

    }



    public void drawFace(GraphicsContext g, int k)
    {
        g.setFill(Color.PINK);
        g.fillOval(15,15+k,100,150);
        //hair
        g.setFill(Color.BLACK);
        g.fillRect(3,3+k,130,35);
        //mouth
        g.setFill(Color.RED);
        g.fillOval(50,120+k,30,20);
        //eyes
        g.setFill(Color.BLUE);
        g.fillOval(30,50+k,30,20);
        g.fillOval(70,50+k,30,20);
        //nose
        g.setFill(Color.RED);
        g.fillRect(55,70+k,20,40);
    }

    public void  eraseface(GraphicsContext g,int k)
    {
        g.setFill(Color.WHITE);
g.fillOval(15,15+k,100,150);
//hair
//g.setFill(Color.black);
g.fillRect(3,3+k,130,35);
//mouth
//g.setFill(Color.red);
g.fillOval(50,120+k,30,20);
//eyes
```

```
//g.setFill(Color.blue);
g.fillOval(30,50+k,30,20);
g.fillOval(70,50+k,30,20);
//nose
//g.setFill(Color.red);
g.fillRect(55,70+k,20,40);
}

    public void closeEyes (GraphicsContext g, int k)
    {
        // We erase eyes:
        // eyes are painted in pink
        g.setFill(Color.PINK);
        g.fillOval(30,50+k,30,20);
        g.fillOval(70,50+k,30,20);
        // We draw smaller eyes
        g.setFill(Color.BLUE);
        g.fillOval(30,65+k,30,5);
        g.fillOval(70,65+k,30,5);
    }

    public void openEyes (GraphicsContext g, int k)
    {
        // small eyes are erased
        // We draw smaller eyes
        g.setFill(Color.PINK);
        g.fillOval(30,65+k,30,5);
        g.fillOval(70,65+k,30,5);
        //We redraw eyes:
        // eyes are drawn in blue
        // and normal size
        g.setFill(Color.BLUE);
        g.fillOval(30,50+k,30,20);
        g.fillOval(70,50+k,30,20);

    }



    public void  movie(GraphicsContext g)
```

```
{
    drawFace(g,0);
    // This loop controls the numbers
    // of cycles of the graphic
    for(int kl = 1; kl<=2; kl++)
    {
        // k is the vertical jump
        int k = 10*kl;
        //We erase everything
        g.clearRect(0,0,200,200);
        // We draw face in the new place
        drawFace(g,k);
        // This loop controls the   blinking number
        for(int i=1; i<=2; i++)
        {
            // A delay must be introduced to make
            //  movement visible for a human being.
            System.out.print("Cycle = " + kl);
            System.out.println(". Blinking = " + i );
            //A delay is inserted
            try
                {   // one second = 1000L
                    Thread.sleep(delayLong);
                }
                    catch (InterruptedException e) {}

            closeEyes(g, k);

            //A delay is inserted
            try
                {   // one second = 1000L
                    Thread.sleep(delayShort);
                }
                    catch (InterruptedException e) {}

            openEyes(g,k);
        }
    }
    System.out.println("Job done.");
}
```

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

}//End of main class L14 MovieCanvasFX
```

15 ***Exercise.*** *Run the program and play with the code. Verify that CanvasFX is not appropriate for movies.*

2.3 Conclusion

Canvas, a lightware graphics facility, has allowed us to get acquainted with JavaFX: it is very easy to make static drawings. But it is not good for movies if we use the same approach we took with the graphics facility of AWT: to erase a graphic, we redraw it in white color and to simulate movement we erase the old drawing and redraw it in another place but with the original color. This failure forces us to study the proper solutions for movement provided by JavaFX. This will be done in the next chapter.

Chapter 3

JavaFX: transitions

How to move objects in FavaFX

16 ***Introduction and purpose.*** *We already know how to draw static shapes in JavaFX by resorting to its Canvas facility. We undertake in this chapter the problem of moving graphics. We start by showing how we can draw and move a rectangle that it defined as an object. Because JavaFX is a heavy duty machinery, some simple tasks may look excessively complicated. So, some patience is asked.*

3.1 Transitions

Transformations of drawings in JavaFX are packed in macros called **transitions**. We will pay attention to translations and to the form as transitions can be nested to form great dynamics. The Reader is asked to notice that nesting of both drawing objects and their transitions give rise to a tree. That is why the first node is optionally called root. If one nests transitions but breaks the tree structure, the application will malfunction. Let us begin with drawing anew static drawings that are Java Objects.

17 ***Drawing a static drawings as Objects in JavaFX.***

```
/*
 * Program L17 StaticRectangle
   Draws a rectangle and a circle using JavaFX
 */

package ejvolu12p;
```

```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;



public class StaticRectangle extends Application
{

    @Override
    public void start(Stage 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.
        Rectangle rect1 =   createRectangle();
        //rect1 is attached to the root
        root.getChildren().addAll(rect1);
        //Create a circle
        Circle circle1 = createCircle();
        //The circle is attached to the root
        root.getChildren().addAll(circle1);
        //The scene is shown
        primaryStage.show();
    }

    //A rectangle ready to be used is built
    private Rectangle  createRectangle()
    {
        //Upper left corner at (20,20)
        //width = 100, hight = 100
        Rectangle rect = new Rectangle(20, 20, 100, 100);
```

```
        //Make round corners with span 20
        rect.setArcHeight(20);
        rect.setArcWidth(20);

//A color is prepared
        Color c = Color.DARKCYAN;
        //The rect is given a color
        rect.setFill(c);

        return rect;
    }


    //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;
    }

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

}//End of application L17 StaticRectangle
```

18 ***Exercise.*** *Run the program and play with the code.*

Let us pass now to add movement to our figures. The ready to use JavaFX machinery is very powerful.

19 ***JavaFX moves a rectangle.***

```
//Program L19 MovingRectangle
//We learn how to move a rectangle using JavaFX.
//To achieve that we manipulate inbuilt macros
//called transitions.

package ejvolu12p;

import java.util.Random;

import javafx.animation.TranslateTransition;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.util.Duration;


public class MovingRectangle extends Application
{
    //Turn on  the random generator
    private static final Random r  = new Random();
    //Duration of events
    private static final Duration d = Duration.millis(2000);
    //Inbuilt macro: translation
    TranslateTransition translateTransition ;

    //Starting the film
    private void init(Stage primaryStage)
    {
        //The stage is organized not as a linear array
        //but as a tree.
        Group root = new Group();
        primaryStage.setResizable(false);
        //Dimensions of the screen
        primaryStage.setScene(new Scene(root, 300, 600));
```

```
        //We will work with rectangles
        Rectangle rect =  createRectangle( 40);
            root.getChildren().addAll(rect);
        //Translation is instantiated
        translateTransition =
                new TranslateTransition(d, rect);
        //Vertical shift from initial position
        translateTransition.setByY(100);
        //Number of repetitions.
        translateTransition.setCycleCount(2);
    }

    //Create a rectangle
    private static  Rectangle   createRectangle(int x)
    {
        //Upper left corner at (20,20)
        //width = 100, hight = 100
        Rectangle rect = new Rectangle(x, 100, 10, 10);
        //Make round corners
        rect.setArcHeight(5);
        rect.setArcWidth(5);
        //Color is defined as a random
        //mix of red, green and blue
        int rr,gg,bb;
        rr = r.nextInt(226);
        gg = r.nextInt(226);
        bb = r.nextInt(226);
//The color is prepared
        javafx.scene.paint.Color c
                = javafx.scene.paint.Color.rgb(rr,gg,bb);
        //The color is applied
        rect.setFill(c);
        return rect;
    }


    public void play()
    {
        translateTransition.play();
    }
```

```
    @Override public void stop()
    {
        translateTransition .stop();
    }

    //This method is automatically called at the beginnig.
    @Override public void start(Stage primaryStage)
            throws Exception
    {
        init(primaryStage);
        primaryStage.show();
        play();
    }


    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {
        launch(args);
    }

}//End of main class L19 MovingRectangle
```

20 ***Exercise.*** *Make JavaFX to move a circle horizontally.* *Answer*

And, what about moving various figures? The answer of JavaFX is recursive, simple an smart:

21 ***Scenes with various rectangles that move.***

```
//Program L21 ManyTransitions
//A transition is a change in a graphic.
//Transitions can be  nested.

package ejvolu12p;

import java.util.Random;
import javafx.animation.ParallelTransition;
```

```
import javafx.animation.SequentialTransition;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.util.Duration;


public class ManyTransitions extends Application
{
    //Many transition, possibly on different objects,
    //happen synchronically
    private static ParallelTransition goDown;
    //Many transition, possibly on different objects,
    //happen sequentially.
    private static SequentialTransition goRight;
    //The overall movie is organized sequentially.
    private static SequentialTransition all;

    //Turn on  the random generator
    private static Random r  = new Random();
    //Duration of events
    private static Duration d = Duration.millis(2000);

    //Starting the film
    private void init(Stage primaryStage)
    {
        //Root of FavaFX objects that conform to a tree
        Group root = new Group();
        primaryStage.setResizable(false);
        //Dimensions of the screen
        primaryStage.setScene(new Scene(root, 300, 600));
        //We will work with rectangles
        Rectangle Rect[] = new Rectangle[20];
        int nRects = 5;
        //Rectangles are created, positioned and booked
        for(int i = 0; i < nRects; i++)
        {
```

```
            Rect[i] = createRectangle((i+1)* 40);
            root.getChildren().addAll(Rect[i]);
        }

        //Synchronous movement
         goDown  = new ParallelTransition();
        //All rectangles are moved
         for (int i = 0; i < nRects; i++)
        {
            //Translation
            TranslateTransition translateTransition =
                     new TranslateTransition(d, Rect[i]);
            //Vertical movement
            translateTransition.setByY(100);
            translateTransition.setCycleCount(1);
            //translateTransition is used here and dies.
            goDown.getChildren().addAll(translateTransition);
        }

         //A sequence of movements
         goRight  = new SequentialTransition();
        //All rectangles are moved
         for (int i = 0; i < nRects; i++)
        {
            TranslateTransition tT =
                new TranslateTransition(d, Rect[i]);
            //Horizontal movement
            tT.setByX(20);
            //Vertical movement: direction flips
            tT.setByY(Math.pow(-1, i)*20);
            tT.setCycleCount(1);
            goRight.getChildren().addAll(tT);
        }

         //Choreography: general organization
        all = new SequentialTransition();
        all.getChildren().addAll(goDown,goRight);

    }
```

```
    //Create a rectangle
    private static  Rectangle   createRectangle(int x)
    {
        //Upper left corner at (20,20)
        //width = 100, hight = 100
        Rectangle rect = new Rectangle(x, 100, 10, 10);
        //Make round corners
        rect.setArcHeight(5);
        rect.setArcWidth(5);
        //Color is defined as a random
        //mix of red, green and blue
        int rr,gg,bb;
        rr = r.nextInt(226);
        gg = r.nextInt(226);
        bb = r.nextInt(226);
//The color is prepared
        javafx.scene.paint.Color c = javafx.scene.paint.Color.rgb(rr,gg,bb);
        //The color is applied
        rect.setFill(c);
        return rect;
    }


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

    @Override public void stop()
    {
        all.stop();
    }

    //This method is automatically called at the beginnig.
    @Override public void start(Stage primaryStage)
            throws Exception
    {
        init(primaryStage);
        primaryStage.show();
        play();
```

```
    }


    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {
        launch(args);
    }

}//End of main class L21 ManyTransitions
```

22 ***Exercise.*** *Make an ensemble of 5 circles move synchronized in zig-zag. Caution: you cannot add the same transition twice to a higher level transition but you always can add fresh transitions.* *Answer*

3.2 Timelines

JavaFX has provided the appropriate facility to implement personalized schedules of action: to that aim **timelines** have been created. We begin with simple problems to pass next to more challenging ones.

23 ***Moving a rectangle in a timeline.***

```
//Program L23 MoveRect
//A timeline is a personalized schedule of action.
//This program presents a  movies with
//a moving rectangle along a timeline.

package ejvolu12p;


import java.util.Random;
import javafx.animation.Interpolator;
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.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.util.Duration;



public  class MoveRect extends Application
{
    //A movie needs a clock and a schedule
    private final Timeline timeline = new Timeline();

    //Turn on  the random generator
    static Random r  = new Random();
    //Starting the movie
    private void init(Stage primaryStage)
    {
        //Tree of JavaFX objects
        Group root = new Group();
        primaryStage.setResizable(false);
        //Dimensions of the screen
        primaryStage.setScene(new Scene(root, 300, 120));

        //Create a rectangle that can move.
        //The type of movement is dictated by
        //the interpolator: it is linear.
        Rectangle rect =
                createMovingRectangle(Interpolator.LINEAR);
        //Rect1 is attached to the film
        root.getChildren().addAll(rect);
    }

    //How a moving rectangle is built.
    //Interpolator: the law for movement.
    private Rectangle
        createMovingRectangle(Interpolator interpolator)
```

```
    {
        //Upper left corner at (20,20)
        //width = 100, hight = 100
        Rectangle rect = new Rectangle(20, 20, 100, 100);
        //Make round corners with span 20
        rect.setArcHeight(20);
        rect.setArcWidth(20);
        //Color is defined as a random
        //mix of red, green and blue
        int rr,gg,bb;
        rr = r.nextInt(226);
        gg = r.nextInt(226);
        bb = r.nextInt(226);
//The color is prepared
        Color c = Color.rgb(rr,gg,bb);
        //The rect is painted
        rect.setFill(c);

        //Create a timeline for moving the rectangle:
        //cycle over and over
        timeline.setCycleCount(Timeline.INDEFINITE);
        //Go forth and back
        timeline.setAutoReverse(true);

        //Define in Keyvalue what to do with rect:
        //move it  along the X axis   155 pixels
        //according to the law given by the interpolator
        KeyValue keyValue =
                new KeyValue(rect.translateXProperty(),
                                        155, interpolator);

        //Execute the keyValue using prescribed duration
        KeyFrame keyFrame =
                new KeyFrame(Duration.seconds(4), keyValue);

        //Add the keyframe to the timeline
        timeline.getKeyFrames().add(keyFrame);

        return rect;
    }
```

```
   public void play()
    {
        timeline.play();
    }

    @Override public void stop()
    {
        timeline.stop();
    }

    //This method is automatically called at the beginnig.
    @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  L23 MoveRect
```

24 ***Exercise.*** *Run the program and play with the code.*

25 ***Exercise.*** *Modify the previous program to make a movie with two moving rectangles.* Answer

Let us pass now to insert timelines into a sequential movement.

26 ***A timeline choreography with 3 rectangles.***

```
//Program L26 Move3Rect
//A timeline choreography with 3 rectangles.
//Processes that happen at the same time are attached
//to the same timeline.
//Processes that happen sequencially are attached to
```

```
//a sequentialTransition.

package ejvolu12p;

import java.util.Random;
import javafx.animation.Interpolator;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.SequentialTransition;
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.util.Duration;


public class Move3Rect extends Application
{

    //Our film has two processes or scenes and two schedules
    private final Timeline timeline1 = new Timeline();
    private final Timeline timeline2 = new Timeline();
    //This is the time marshall:
    //it dictates how scenes appear along the movie
    private SequentialTransition animation;
    //Turn on  the random generator
    static Random r  = new Random();
    //Starting the movie
    private void init(Stage primaryStage)
    {
        //JavaFX tree of space-like objects
        Group root = new Group();
        primaryStage.setResizable(false);
        //Dimensions of the screen
        primaryStage.setScene(new Scene(root, 300, 420));
```

```
//Create rectangles.
Rectangle rect1 =
        createRectangle(20,20);
Rectangle rect2 =
        createRectangle(20,140);
Rectangle rect3 =
        createRectangle(20,260);
//Set timeline1 with two cycles
timeline1.setCycleCount(2);
//Go forth and back
timeline1.setAutoReverse(true);

//Define in Keyvalue what to do with rect1:
//move it  along the X axis   155 pixels
//according to the law given by the interpolator
KeyValue keyValue1 =
        new KeyValue(rect1.translateXProperty(),
                     155, Interpolator.LINEAR);
KeyValue keyValue2 =
        new KeyValue(rect2.translateXProperty(),
                     155, Interpolator.LINEAR);

//Execute  synchronically keyValues
//using prescribed duration
KeyFrame keyFrame1 =
        new KeyFrame(Duration.seconds(4),
                     keyValue1,keyValue2);

//Add keyframe1 to  timeline1
timeline1.getKeyFrames().add(keyFrame1);

//Set timeline2
timeline2.setCycleCount(2);
timeline2.setAutoReverse(true);

//Define in Keyvalue what to do with rect3:
//move it  along the X axis   155 pixels
//according to the law given by the interpolator
KeyValue keyValue3 =
        new KeyValue(rect3.translateXProperty(),
```

```
                                155, Interpolator.LINEAR);

        //Execute  keyValue2 using prescribed duration
        KeyFrame keyFrame2 =
              new KeyFrame(Duration.seconds(4), keyValue3);

        //Add the keyframe to the timeline
        timeline2.getKeyFrames().add(keyFrame2);
        //Rects are attached to the film
        root.getChildren().addAll(rect1, rect2, rect3);

        animation = new SequentialTransition();
        animation.getChildren().addAll(timeline1,timeline2);
        animation.setCycleCount(2);
    }

    //A rectangle is returned.
    private Rectangle  createRectangle( int x, int y)
    {
        //Upper left corner at (x,y)
        //width = 100, hight = 100
        Rectangle rect = new Rectangle(x,y, 100, 100);
        //Make round corners with span 20
        rect.setArcHeight(20);
        rect.setArcWidth(20);
        //Color is defined as a random
        //mix of red, green and blue
        int rr,gg,bb;
        rr = r.nextInt(226);
        gg = r.nextInt(226);
        bb = r.nextInt(226);
//The color is prepared
        Color c = Color.rgb(rr,gg,bb);
        //The rect is painted
        rect.setFill(c);

        return rect;
    }

   public void play()
```

```
    {
        animation.play();
    }

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

    //This method is automatically called at the beginnig.
    @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 L26 Move3Rect
```

27 ***Exercise.*** *Add a circle to the previous choreography.* *Answer*

Actors ordinary enter and leave scenes in every movie. Let us make an attempt to do that in JavaFX.

28 ***An attempt to enter an actor into the scene.***

```
//Program L28 SixCircles
//This codes is intended to implement the
//following plan:
//six circles must move in zigzag and
//on finished a new circle must appear
//and move downwards.
//Everyhing is OK with just one problem:
//the sixth circle appears from start
//and not after the stopping of the five circles.
```

```
//In general,  when a new actor is created,
//it is created and shown at the pure beginning
//and not when one expects it to appear.


package sixcircles;

import java.util.Random;
import javafx.animation.ParallelTransition;
import javafx.animation.SequentialTransition;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;
import javafx.util.Duration;


public class SixCircles extends Application
{

    //Everything is organized sequentially.
    private static SequentialTransition all;
    private static SequentialTransition zigzag;

    //Turn on  the random generator
    private static final Random r  = new Random();
    //Duration of events
    private static final Duration d = Duration.millis(2000);

    //Starting the movie
    private void init(Stage primaryStage)
    {
        //Root of tree of space-like objects
        Group root = new Group();
        primaryStage.setResizable(false);
        //Dimensions of the screen
        primaryStage.setScene(new Scene(root, 400, 600));
```

```
//We will work with circles
Circle Circle[] = new Circle[20];
int nCircles = 5;
//Circles are created, positioned and booked
for(int i = 0; i < nCircles; i++)
{
    Circle[i] = createCircle((i+1)* 40);
    root.getChildren().addAll(Circle[i]);
}

 //Choreography: general organization
zigzag = new SequentialTransition();
//Two cycles of zigzag
for(int j = 0; j < 2; j++)
{
    //Synchronous movement to the right
    ParallelTransition goRight  =
            new ParallelTransition();
    //All circles are moved
     for (int i = 0; i < nCircles; i++)
    {
        //Translation
        TranslateTransition translateTransition =
             new TranslateTransition(d, Circle[i]);
        //Diagonal movement
        translateTransition.setByX(100);
        translateTransition.setByY(100);
        translateTransition.setCycleCount(1);
        //translateTransition is used here and dies.
        goRight.getChildren().
            addAll(translateTransition);
    }

    //Synchronous movement to the left
    ParallelTransition goLeft  =
                   new ParallelTransition();
    //All circles are moved
     for (int i = 0; i < nCircles; i++)
    {
        //Translation
```

```
                TranslateTransition translateTransition =
                     new TranslateTransition(d, Circle[i]);
                //Diagonal movement
                translateTransition.setByX(-100);
                translateTransition.setByY(100);
                translateTransition.setCycleCount(1);
                //translateTransition is used here and dies.
                goLeft.getChildren().
                    addAll(translateTransition);
            }

            zigzag.setAutoReverse(true);
            zigzag.getChildren().addAll(goRight,goLeft);
        }
        all = new SequentialTransition();
        //A new actor is created and booked
        Circle circle6 = createCircle(300);
        root.getChildren().addAll(circle6);
        //circle6 is translated
        TranslateTransition translateTransition =
                     new TranslateTransition(d, circle6);
        translateTransition.setByY(400);
                translateTransition.setCycleCount(1);
        //translateTransition is used here and dies.
        all.getChildren().addAll(zigzag,translateTransition);
    }


    //A circle ready to be used is built
    private Circle  createCircle(int x)
    {
        //Center at (x,100)
        //Radius = 10
        Circle circle = new Circle(x, 100, 10);
        //Color is defined as a random
        //mix of red, green and blue
        int rr,gg,bb;
        rr = r.nextInt(226);
        gg = r.nextInt(226);
        bb = r.nextInt(226);
```

```
//The color is prepared
        javafx.scene.paint.Color c =
                javafx.scene.paint.Color.rgb(rr,gg,bb);
        //The circle is given a color
        circle.setFill(c);

        return circle;
    }


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

    @Override public void stop()
    {
        all.stop();
    }

    //This method is automatically called at the beginnig.
    @Override public void start(Stage primaryStage)
            throws Exception
    {
        init(primaryStage);
        primaryStage.show();
        play();
    }


    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {
        launch(args);
    }

}//End of main class L28 SixCircles
```

29 ***Exercise.*** *Run the program and play with the code.*

The solution of the problem of the new actor is presented in the next program.

30 ***The next code enables the new actor to appear when appointed and not before.***

```
//Program L30 SixbCircles
//Five circles must move in zigzag and
//on finished a new circle must appear
//and move downwards.
//Since the sixth circle is created and booked
//at the beginning,
//it is given radius 0 until the moment
//when it is scheduled to appear:
//at that time it is given a positive radius
//and its performance begins.


package ejvolu12p;

import java.util.Random;
import javafx.animation.Interpolator;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.ParallelTransition;
import javafx.animation.PauseTransitionBuilder;
import javafx.animation.SequentialTransition;
import javafx.animation.Timeline;
import javafx.animation.TranslateTransition;
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.shape.Circle;
import javafx.stage.Stage;
import javafx.util.Duration;
```

```
public class SixbCircles extends Application
{

    //Everything is organized sequentially.
    private static SequentialTransition all;
    private static SequentialTransition zigzag;

    //Turn on  the random generator
    private static final Random r  = new Random();
    //Duration of events
    private static final Duration d = Duration.millis(2000);

    //Starting the movie
    private void init(Stage primaryStage)
    {
        Group root = new Group();
        primaryStage.setResizable(false);
        //Dimensions of the screen
        primaryStage.setScene(new Scene(root, 400, 600));
        //We will work with circles
        Circle Circle[] = new Circle[20];
        int nCircles = 5;
        //Circles are created, positioned and booked
        for(int i = 0; i < nCircles; i++)
        {
            Circle[i] = createCircle((i+1)* 40);
            root.getChildren().addAll(Circle[i]);
        }
        //A new actor is created
        Circle circle6 = createCircle(300);
        //Actor is hidden
        circle6.setRadius(0);
        //Actor is booked
        root.getChildren().addAll(circle6);


        //Choreography: general organization of scenes
        zigzag = new SequentialTransition();
        //Two cycles of zigzag
```

```
for(int j = 0; j < 2; j++)
{
    //Synchronous movement to the right
    ParallelTransition goRight  =
            new ParallelTransition();
    //All circles are moved
     for (int i = 0; i < nCircles; i++)
    {
        //Translation
        TranslateTransition translateTransition =
             new TranslateTransition(d, Circle[i]);
        //Diagonal movement
        translateTransition.setByX(100);
        translateTransition.setByY(100);
        translateTransition.setCycleCount(1);
        //translateTransition is used here and dies.
        goRight.getChildren().
            addAll(translateTransition);
    }

    //Synchronous movement to the left
    ParallelTransition goLeft  =
                  new ParallelTransition();
    //All circles are moved
     for (int i = 0; i < nCircles; i++)
    {
        //Translation
        TranslateTransition translateTransition =
             new TranslateTransition(d, Circle[i]);
        //Diagonal movement
        translateTransition.setByX(-100);
        translateTransition.setByY(100);
        translateTransition.setCycleCount(1);
        //translateTransition is used here and dies.
        goLeft.getChildren().
            addAll(translateTransition);
    }

    zigzag.setAutoReverse(true);
    zigzag.getChildren().addAll(goRight,goLeft);
```

```
        }

        //Hidden circle6 is shown
        SequentialTransition s = showST(circle6);
        //Visible circle6 is translated
        TranslateTransition translateTransition =
                     new TranslateTransition(d, circle6);
        translateTransition.setByY(400);
                translateTransition.setCycleCount(1);
        all = new SequentialTransition();
        //translateTransition is used here and dies.
        all.getChildren().
                    addAll(zigzag,s,translateTransition);
    }


    //A circle ready to be used is built
    private Circle  createCircle(int x)
    {
        //Center at (x,100)
        //Radius = 10
        Circle circle = new Circle(x, 100, 10);
        //Color is defined as a random
        //mix of red, green and blue
        int rr,gg,bb;
        rr = r.nextInt(226);
        gg = r.nextInt(226);
        bb = r.nextInt(226);
//The color is prepared
        javafx.scene.paint.Color c =
                javafx.scene.paint.Color.rgb(rr,gg,bb);
        //The circle is given a color
        circle.setFill(c);

        return circle;
    }

    //The new actor circle6 is shown:
    //Its radius is scaled from 0 to 100
```

```
private SequentialTransition showST(Circle circle6)
{
    Duration d2 =   Duration.millis(1);

    //Circle must be shown at true scale
    KeyValue keyValue1 =
            new KeyValue(circle6.scaleXProperty(),
                         1, Interpolator.LINEAR);
    KeyValue keyValue2 =
            new KeyValue(circle6.scaleYProperty(),
                         1, Interpolator.LINEAR);
    Timeline b = new Timeline(
            new KeyFrame(d2,keyValue1,keyValue2 ));

    SequentialTransition s = new SequentialTransition();
    //Radius of circle6 is enlarged and after this
    //it is shown according to the
    //instructions given by timeline b.
    //To do this, a pause of one millisecond is added.
    s.getChildren().add(
            new SequentialTransition(

                PauseTransitionBuilder.create().
                    duration(Duration.ONE).onFinished
                    ((ActionEvent t) ->
                    {circle6.setRadius(100);}).build(),b));
    return s;
}



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

@Override public void stop()
{
    all.stop();
}
```

```
    //This method is automatically called at the beginnig.
    @Override public void start(Stage primaryStage)
            throws Exception
    {
        init(primaryStage);
        primaryStage.show();
        play();
    }


    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {
        launch(args);
    }

}//End of main class L30 SixbCircles
```

31 ***Exercise.*** *Run the program and play with the code.*

32 ***Exercise.*** *Modify the previous program to make the sixth actor to be left alone in the stage.* Answer

33 ***Exercise.*** *Change circles by rectangles in the previous program.* Answer

Let us learn how to add some informative text to our films.

34 ***The next code creates a scene with some titles.***

```
//Program L34  Textual
//Texts are included in a scene.

package ejvolu12p;

import javafx.application.Application;
import javafx.stage.Stage;
```

```
import javafx.animation.AnimationTimer;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.text.Text;


public class Textual extends Application
{

    //Counter (of instants of time)
    private Integer i = 0;

    @Override public void start(Stage stage)
    {
        Group p = new Group();
        Scene scene = new Scene(p);
        stage.setScene(scene);
        stage.setWidth(500);
        stage.setHeight(500);
        stage.show();

        //A circle at the upper left corner
        final Circle circle = new Circle(20,  Color.BISQUE);

        //Constant text
        final Text text = new Text ("Welcome!" );
        text.setStroke(Color.BLACK);
        text.setLayoutX(20);
        text.setLayoutY(20);

        //A changing text that simulates
        //a chronometer
        final Text watch = new Text ("");
        watch.setLayoutX(40);
        watch.setLayoutY(40);
        //Watch is bound to a chronometer in arbitrary units
        AnimationTimer timer = new AnimationTimer()
        {
```

```
            @Override
            public void handle(long l)
            {
                watch.setText("Time  = " + i.toString());
                i++;
            }
        };
        timer.start();


        p.getChildren().addAll(circle, text, watch);
    }


    public static void main(String[] args) {
        Application.launch(args);
    }
  } //End of main class L34 Textual
```

35 ***Exercise.*** *Change circles by rectangles in the previous program.* *Answer*

3.3 Rides

Rides along a complex road is treated by a special tool in JavaFX. Let us see it.

36 ***The next code simulates a ride around a road that consists of two cubic curves.***

```
//Program L36  Ride
//A ride  is simulated

package ejvolu12p;

import javafx.animation.PathTransition;
import javafx.animation.PathTransitionBuilder;
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.Group;
```

```
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.CubicCurveTo;
import javafx.scene.shape.MoveTo;
import javafx.scene.shape.Path;
import javafx.scene.shape.PathBuilder;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.util.Duration;


public class Ride extends Application
{
private PathTransition pathTransition;

private void init(Stage primaryStage)
{
    Group root = new Group();
    primaryStage.setResizable(false);
    primaryStage.setScene(new Scene(root, 250,400));

    //The machine
    Rectangle rect = new Rectangle (0, 0, 15, 10);
    rect.setArcHeight(4);
    rect.setArcWidth(4);
    rect.setFill(Color.RED);
    root.getChildren().add(rect);
    //The road has two cubic segments
    Path path = PathBuilder.create()
        .elements
                (
                new MoveTo(50, 200),
                new CubicCurveTo(10, 30, 100, 150, 200, 120),
                new CubicCurveTo(220, 140, 28, 350, 50, 200)
                )
                .build();
    path.setStroke(Color.DARKBLUE);
    path.getStrokeDashArray().setAll(7d,7d);
    root.getChildren().add(path);
```

```
    //Special JavaFX tool for rides
    pathTransition = PathTransitionBuilder.create()
        .duration(Duration.seconds(5))
        .path(path)
        .node(rect)
        .orientation(PathTransition
                .OrientationType.NONE.ORTHOGONAL_TO_TANGENT)
        .cycleCount(Timeline.INDEFINITE)
        //.autoReverse(true)
        .build();
    }

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

    @Override public void stop()
    {
        pathTransition.stop();
    }

    @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 L36 Ride
```

37 ***Exercise.*** *Run the program and play with the code: design your own machine and a more complex road.*

3.4 The tic-jump technology

Let us use now the **tic-jump technology** that implements the differential approach to complex movements: there is a clock that marks intervals of time with its tics. For every tic, a jump happens. The direction and magnitude of the jump is determined by the present position of the moving object and by some external factors. Movement results from a succession of jumps, which are automatically iterated. This technology admits various objects with particularized management but global management through a loop is not allowed.

38 ***Tics and random jumps are presented in the next code.***

```
//Program L38 Jumps1
//For every tic of a clock,
//a random jump happens.

package ejvolu12p;

import java.util.Random;

import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.util.Duration;

public class Jumps1 extends Application
{

    //Clock
    private Timeline timeline;
    //Generator of pseudo random numbers
    private final Random r = new Random();
```

```
@Override public void start(Stage stage)
{
    Group p = new Group();
    Scene scene = new Scene(p);
    stage.setScene(scene);
    stage.setWidth(150);
    stage.setHeight(150);

    //create a rectangle
    Rectangle rect = new Rectangle(0,0,20,20);
    rect.setFill(Color.BURLYWOOD);
    p.getChildren().add(rect);
    stage.show();

    //Instanciate the clock
    timeline = new Timeline();
    //Automatic repetition of tics
    timeline.setCycleCount(Timeline.INDEFINITE);

    //Duration in within tics
    Duration duration = Duration.millis(2000);
    //A jump is programmed at random
    EventHandler jumpTo;
    jumpTo = (EventHandler<ActionEvent>)
        (ActionEvent t) ->
        {
            int toX =  r.nextInt(100);
            int toY =  r.nextInt(100);
            System.out.println("toX = "
                    + toX + " toY = " + toY);
            rect.setTranslateX(toX);
            rect.setTranslateY(toY);
        };
    //Tic and jump  are synchronized
    KeyFrame keyFrame = new KeyFrame(duration, jumpTo);
    //The new ensemble is booked
    timeline.getKeyFrames().add(keyFrame);
```

```
        timeline.play();

    }


    public static void main(String[] args)
    {
        Application.launch(args);
    }
} //End of main class L38 Jumps1
```

39 ***Exercise.*** *Run the program and play with the code.*

40 ***Exercise.*** *Add a second permanent actor to the previous code. Add also a transient third actor, i.e., one that enters at some time, makes a performance and leaves.*Answer

Our differential approach to complex movements, implemented by the tic-jump technology, must be very appropriate to model a planetary system. So, the next code shows the simplest case, in which the orbit is a perfect circle.

41 ***The simplest planetary system is simulated in the next code.*** *The sun is at the center of the circular orbit of the planet.*

To use the differential approach we need an initial position and the general rule for velocity. The position $P = (x, y)$ of the planet along a circular orbit is parameterized as follows:

$x = rcos(\omega t)$

$y = rsin(\omega t)$

So, the coordinates of the velocity vector V are given by the differentials of these values

$\dot{x} = -\omega rsin(\omega t)$

$\dot{y} = +\omega rcos(\omega t)$

where r is the radius, ω the angular velocity and t the time.

Now, if the position of the planet at time t is $P(t)$ and its velocity is $V(t)$, then the position of the planet at time $t + dt$ is given by

$$P(t+dt) = P(t) + V(t)dt = (x, y) + (\dot{x}, \dot{y})dt$$

or, in coordinates:

$x(t+dt) = x(t) + \dot{x}dt = x(t) - \omega r sin(\omega t)dt$
$y(t+dt) = y(t) + \dot{y}dt = y(t) + \omega r cos(\omega t)dt$

These equations are implemented in the next code.

```
//Program L41 Jumps3
//A planet spins along a circular orbit
//around a sun.

package ejvolu12p;

import java.util.Random;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;
import javafx.util.Duration;

public class Jumps3 extends Application
{

    //Clock
    private Timeline timeline;
    //Generator of pseudo random numbers
    private final Random r = new Random();
    //Counter of tics
    private static Integer nTics = 0;
    //The radius of the sun
    private static Integer rSun = 100;
    //The radius of the orbit
```

```
private static Integer radiusOrbit = 100;
//Angular velocity
private static double omega = 0.005;
//Initial position of the planet
private static double x = 0;
private static double y = radiusOrbit;
//Time is discrete
private static Integer t = 0;

@Override public void start(Stage stage)
{
    Group p = new Group();
    Scene scene = new Scene(p, 420, 420);
    stage.setScene(scene);


    //The sun is centered
    Circle sun = new Circle(0, 0,rSun);
    sun.setFill(Color.GOLD);

    //The planet begins down under the sun
    Circle planet = new Circle(0,radiusOrbit,3);
    planet.setFill(Color.LIGHTSTEELBLUE);

    //Pane to place the ensemble sun + planet
    final StackPane stack = new StackPane();
    stack.getChildren().addAll(sun, planet);
    //Centering
    int shiftX = 200;
    int shiftY = 200;
    stack.setLayoutX(shiftX);
    stack.setLayoutY(shiftY);
    //This was found by trial and error
    y = y - shiftY/2;
    x = x + shiftX/2;


    p.getChildren().add(stack);
```

```
        stage.show();

        //Instanciate the clock
        timeline = new Timeline();
        //Automatic repetition of tics
        timeline.setCycleCount(Timeline.INDEFINITE);
        //Duration in within tics = dt = 1 = one unit of time
        Duration duration = Duration.millis(1);
        //A jump is programmed along a circle
        EventHandler jumpTo;
        jumpTo = (EventHandler<ActionEvent>)
            (ActionEvent e) ->
            {
                //tic
                //Jump of the planet
                x = x - omega*radiusOrbit
                        * Math.sin(omega* t);
                y = y + omega*radiusOrbit
                        * Math.cos(omega* t);
                planet.setTranslateX(x);
                planet.setTranslateY(y);
                //The tic is counted
                t++;
            };
        //Tic and jump  are synchronized
        KeyFrame keyFrame = new KeyFrame(duration, jumpTo);
        //The new ensemble is booked
        timeline.getKeyFrames().add(keyFrame);
        timeline.play();
    }


    public static void main(String[] args)
    {
        Application.launch(args);
    }
} //End of main class L41 Jumps3
```

42 ***Exercise.*** *Run the program and play with the code.*

43 ***Exercise.*** *Change the circle by an ellipse in the previous code. Answer*

44 ***Challenge.*** *Our implementation of the tic-jump technology was made by means of jumps. Try out soft transitions in the previous code to see what happens. Answer*

45 ***Challenge.*** *Begins with Newtons' laws and end with an elliptical movement of a planet. Verify Kepler's laws.*

46 ***Challenge.*** *Model the system composed of Sun + Earth + Moon.*

3.5 Conclusion

We have got acquainted with the fundamentals of JavaFX **animations**. **Timelines** and **transitions** conform the fundamental blocks to build dynamics that manage Java threads in a very friendly form. The **tic-jump technology** is a special option to produce recursive dynamics that results from programming a jump, a small change, for every tic of a watch. JavaFX can be understood as follows: there are actors, space-like objects (rectangles, circles), that get involved in scenes. These actors are all created at the beginning. Their performances are scheduled by time-like objects, timelines and transitions. In particular, timelines regulate the moment of appearance and of disappearance of actors into and out of the scene.

Chapter 4

Worms in JavaFX

```
New tuxedo for great ideas
```

47 ***Purpose.*** *We will use the acquired know-how about animations to rewrite in JavaFX our programs in which an ensemble of worms climb the screen.*

4.1 Movement by human design

We rewrite our programs about worms in JavaFX using transitions and enabling actors to appear and disappear at appointed times in the scene. Our first try shows one human solution to the problem of climbing.

48 ***JavaFX version of the program that enables an ensemble of worms to climb the screen. The know how to climb was composed by human design.***

```
//Program L48 WormsHDJFX
//A population of worms is programmed
//by human design to
//climb the screen.
//Program is written in JavaFX.

package ejvolu12p;

import java.util.Random;
import javafx.animation.ParallelTransition;
import javafx.animation.SequentialTransition;
import javafx.animation.TranslateTransition;
```

```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.util.Duration;



public class WormsHDJFX extends Application
{
    private static WormsHDJFX w = new WormsHDJFX();


    private static   Group p = new Group();

    //Number of steps
    int nSteps = 10;
    //Number of worms
    private static final int nWorms = 8;
    //Worms are kept in an array
    private static Worm[] Worms = new Worm[nWorms];
    //Number of links of each worm
    private static final int nLinks = 7;
    //Initial position over the screen of the head
    private static int initialRow = 150;
    //Generator of pseudo random numbers
    private static final Random r = new Random();
    private static final boolean print1 = false;
    //Each worm has nMovers movers,
    //instructions to move.
    private static int  nMovers = nLinks;
    //Width of the worm = width of each link
    private static int width = 4;
    //Height of each link
    private static int height = 12;

    /* Definition of inner classes.
    *
```

```
   The Move class:
   A Move says in which direction
   a link of the worm must be moved.
   */

private  class Move
{
   int x;
   int y;


   //Default initialization: go to the northeast
   Move()
   {
       x= 1;
       y= 1;
   }

   // Initialize wth a list of two coordinates
   Move(int a, int b)
   {
       x= a;
       y= b;
   }

   // Copy a move
   Move(Move move)
   {
       x= move.x;
       y= move.y;
   }
}//End of class Move


// Definition of the class Mover:
// A Mover keeps two types of information:
// a) the link of the worm to be moved and
// b) the direction or move to be chosen.
//
private   class Mover
```

```
 {
    int link ;
    Move   move;

    //  Movers can acquired information
    // in three ways, they are  overloaded.
    // Form 1: from numbers
private Mover(int l, int mx, int my)
     {
        link= l;
        move.x = mx;
        move.y = my;
     }

    // Form 2, from structured data
    private Mover(int l, Move m)
    {
        link= l;
        move = m;
        move.x = m.x;
        move.y = m.y;
    }

    // Form 3, from structured data
    private Mover(Mover mover)
    {
        link= mover.link;
        move =  new Move(mover.move);
    }
  }//End of class Mover



      private static  Rectangle
           createRectangle()
    {
        //Upper left corner at (20,20)
        Rectangle rect = new Rectangle(20,20,width,height);
        //Make round corners with span 20
```

```
        //rect.setArcHeight(20);
        //rect.setArcWidth(20);
        //Color is defined as a random
        //mix of red, green and blue
        int rr,gg,bb;
        rr = r.nextInt(226);
        gg = r.nextInt(226);
        bb = r.nextInt(226);
//The color is prepared
        Color c = Color.rgb(rr,gg,bb);
        //The rect is painted
        rect.setFill(c);


    return rect;
    }

/* Here we define the class worm.
    A worm is composite: it has links.
    Each link has a rectangular shape
    and  its own color,
    which is defined by three numbers.
    The vector Mover[ ]
    keeps the information
    on how to move its links:
    which  link and in which order
    (in the order that they appear in
    the vector Mover[]), and
    in which direction.
    Each worm keeps record of
    the actual position of each
    one of its links,
    in the associated ractangle. */



private class Worm
{
    //Number of links
    int numberOfLinks = nLinks;
```

```
    //Number of movers = instructions to move
    int numberOfMovers = nLinks;
    //mover[] keeps movers
    Mover mover[ ] = new Mover[nLinks];
    //The shape of each link
    Rectangle rect[] = new Rectangle[20];
    //move right (side = 1) else left (side = -1)
    int side = 2*r.nextInt(2) -1;



  //Default initialization
   private Worm( )
   {
//Rects are created.
        //A color is assigned to each link
        //but different than white
        for(int i = 0; i < numberOfLinks; i++)
            rect[i] = createRectangle();
        //The know how to climb the screen
int link;
for(int m = 0; m  < nLinks; m++)
  {
   //Move the tail first and then the others links.
   link = nLinks-m-1;
   Mover mover1;
           //Flip horizontal direction of movement,
           //Move each link in the y-direction
   if (side == 1)
    mover1 = new Mover(link,new Move(1,1));
    else
     mover1 =  new Mover(link,new Move (-1,1));
   mover[m]= mover1;
  }
 }//End of Worm()

}
```

```
@Override public void start(Stage stage)
    {
        //The scene
        Scene scene = new Scene(p, 200,200);
        stage.setScene(scene);
        stage.setTitle("Human design");
        stage.show();

        //Worms are posited at the start line
        startLine();
        //Worms climb the screen
        climb();
    }
```

```
// All worms are posited at the same initial row
// in vertical position
static private void startLine()
{
    for (int W = 0; W < nWorms; W++)
    {
        int k, x,y;
        //Auxiliary worm
        Worm   worm  = w.new Worm();
        //Each link is positioned
        for(int  link = 0; link < nLinks; link++)
        {
            //Horizontal position
            x = 30 + 20 * W;
            k = initialRow - nLinks;
            //Vertical position
            y = k + height*link;
            //Position is updated
            worm.rect[link].setX(x);
```

```
            worm.rect[link].setY(y);
            //Link is booked
            p.getChildren().add(worm.rect[link]);
        }
        Worms[W]= worm;
    }
}


//To climb the screen the worm W
//translates each one of its links
//according to the information
//consigned in mover[].
//Move m is executed: it moves link m.
private static TranslateTransition translate( int W, int m)
{
    if (print1)
        System.out.println("\n Worm = " + W
                           + " move = " + m);
    //Reads the link to be moved
    int link = Worms[W].mover[m].link;

    //The move to be done is read
    Move  move1;
    move1 = Worms[W].mover[m].move;
    //Move is interpreted
    int byX = 4* move1.x;
    int byY = -6* move1.y;

    //Reads position
    int x = (int) Worms[W].rect[link].getX();
    int y = (int) Worms[W].rect[link].getY();
    if (print1)
        System.out.println(" x = " + x
                           + " y = " + y);
    // A new place is defined:
    x = x + byX;
    y = y + byY;
    if (print1)
        System.out.println(" byX = " + byX
```

```
                            + " byY = " + byY);

    //Address is updated
    Worms[W].rect[link].setX(x);
    Worms[W].rect[link].setY(y);

    //Duration in within tics
    Duration d = Duration.millis(500);
    //A soft movement is programmed
    TranslateTransition translateTransition =
         new TranslateTransition(d, Worms[W].rect[link]);
    //Movement is executed
    translateTransition.setByX(byX);
    translateTransition.setByY(byY);
    //Number of repetitions
    translateTransition.setCycleCount(1);

    return translateTransition;
 }


//Worms climb the screen.
//Various transitions are nested to achieve that.
private    void  climb()
{
    //Overall process
    SequentialTransition allST
                    = new SequentialTransition();
     allST.setCycleCount(1);
     //worms advance nSteps
    for (int i = 0; i < nSteps; i++)
    {
        //Worms advance one step:
        //To move all links, each worm must execute
        //instructions to move in mover[]
        SequentialTransition oneStepST
                        = new SequentialTransition();
        oneStepST.setCycleCount(1);
        //Information is read and decoded
        for (int m = 0; m < nMovers; m++)
```

```
        {
            //All worms execute move m
            ParallelTransition moveMPT
                        = new ParallelTransition();
            moveMPT.setCycleCount(1);
            for (int W = 0; W < nWorms; W++)
            {
                //Direction is flipped
                Worms[W].mover[m].move.x
                            = - Worms[W].mover[m].move.x;
                //A translation of some link
                //is programmed for worm W
                //according to move m
                TranslateTransition tT = translate(W,m);
                tT.setCycleCount(1);
                //Translation is booked
                moveMPT.getChildren().add(tT);
            }
            //The ensamble of translations is booked
            oneStepST.getChildren().add(moveMPT);
        }
    //The movement of worms in one step is booked
    allST.getChildren().add(oneStepST);
    }
    //Dynamics is played
    allST.play();
}


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


} //End of main class L48  WormsHDJFX
```

49 ***Exercise.*** *Run the program and play with the code.*

4.2 Movement at random

Let us use JavaFX to see how worms behave when their links move at random.

50 ***Links move at random. JavaFX version.***

```
//Program L50 WormsRandJFX
//A population of worms is programmed
//by randomness to move links at random.
//Program is written in JavaFX.

package ejvolu12p;

import java.util.Random;
import javafx.animation.ParallelTransition;
import javafx.animation.SequentialTransition;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.util.Duration;



public class WormsRandJFX extends Application
{
    private static WormsRandJFX w = new WormsRandJFX();

    private static   Group p = new Group();

    //Number of steps
    int nSteps = 3;
    //Number of worms
    private static final int nWorms = 5;
    //Worms are kept in an array
    private static Worm[] Worms = new Worm[nWorms];
```

```
    //Number of links of each worm
    private static final int nLinks = 9;
    //Initial position over the screen of the head
    private static int initialRow = 180;
    //Generator of pseudo random numbers
    private static final Random r = new Random();
    //Each worm has nMovers movers,
    //instructions to move.
    private static int  nMovers = nLinks;
    //Width of the worm = width of each link
    private static int width = 4;
    //Height of each link
    private static int height = 12;
    //Vector with 9 possible moves
    private static Move  vectMoves[ ] = new Move[9];
    private static final boolean print1 = false;

    /* Definition of inner classes.
    *
    The Move class:
    A Move says in which direction
    a link of the worm must move.
    */

private  class Move
{
    int x;
    int y;


    //Default initialization
    Move()
    {
        x= 0;
        y= 0;
    }

    // Initialize with numbers
    Move(int a, int b)
    {
```

```
        x= a;
        y= b;
    }

    // Copy a move
    Move(Move move)
    {
        x= move.x;
        y= move.y;
    }
}//End of class Move


// Definition of the class Mover:
// A Mover keeps two types of information:
// a) the link of the worm to be moved and
// b) the direction or move to be chosen.
//
private   class Mover
{
    int link ;
    Move   move;

    //  Movers can acquired information
    // in three ways, they are  overloaded.
    // Form 1: by a list of numbers
    private Mover(int l, int mx, int my)
     {
        link= l;
        move.x = mx;
        move.y = my;
     }

    // Form 2, by structured data
    private Mover(int l, Move m)
    {
        link= l;
        move = m;
        move.x = m.x;
        move.y = m.y;
```

```
    }

    // Form 3, by structured data
    private Mover(Mover mover)
    {
        link= mover.link;
        move =  new Move(mover.move);
    }
  }//End of class Mover




    private static  Rectangle
           createRectangle()
    {
        //Upper left corner at (0,0)
        Rectangle rect = new Rectangle(0,0,width,height);
        //Color is defined as a random
        //mix of red, green and blue
        int rr,gg,bb;
        rr = r.nextInt(226);
        gg = r.nextInt(226);
        bb = r.nextInt(226);
//The color is prepared
        Color c = Color.rgb(rr,gg,bb);
        //The rect is painted
        rect.setFill(c);


    return rect;
    }

/* Here we define the class worm.
    A worm is composite: it has links.
    Each link has a rectangular shape
    and  its own color,
    which is defined by three numbers.
    The vector Mover[ ]
    keeps the information
```

```
    on how to move its links:
    which  link and in which order
    (in the order that they appear in
    the vector Mover[]), and
    in which direction.
    Each worm keeps record of
    the actual position of each
    one of its links,
    in the associated ractangle. */



private class Worm
{
    //Number of links
    int numberOfLinks = nLinks;
    //Number of movers = instructions to move
    int numberOfMovers = nLinks;
    //mover[] keeps movers
    Mover mover[ ] = new Mover[nLinks];
    //The shape of each link
    Rectangle rect[] = new Rectangle[20];
    //move right (side = 1) else left (side = -1)
    int side = 2*r.nextInt(2) -1;



  //Default initialization
   private Worm( )
   {
//Rects are created.
        // A color is assigned to each link
        // but different than white
        for(int i = 0; i < numberOfLinks; i++)
            rect[i] = createRectangle();
        //Instructions to move at random
for(int m = 0; m < nMovers; m++)
        {
            //Select at random  which link to move:
            int l = r.nextInt(nLinks) ;
```

```
            //Select at random  which direction to move on
            Move  move = w.new Move (vectMoves[r.nextInt(9)]);
            //Memorization
            mover[m]= w.new Mover(l,move);
        }
 }//End of Worm()

}
```

```
@Override public void start(Stage stage)
    {
        Scene scene = new Scene(p, 500,500);
        stage.setScene(scene);
        stage.setTitle("Links move at random");
        stage.show();
        //Definition of possible moves
        moveDef();

        //Worms are posited at the start line
        startLine();

        //Worms climb the screen
        climb();
    }


 //Nine basic moves = coordinates where to move in
static private void moveDef()
{
 final Move move0 =  w.new Move(0,1);
   vectMoves[0]   = move0 ;
 final Move move1 =  w.new Move(1,1);
   vectMoves[1]   = move1 ;
 final Move move2 =  w.new Move(1,0);
   vectMoves[2]   = move2 ;
```

```
 final Move move3 =  w.new Move(1,-1);
   vectMoves[3]   = move3 ;
 final Move move4 =  w.new Move(0,-1);
   vectMoves[4]   = move4 ;
 final Move move5 =  w.new Move(-1,-1);
   vectMoves[5]   = move5 ;
 final Move move6 =  w.new Move(-1,0);
   vectMoves[6]   = move6 ;
 final Move move7 =  w.new Move(-1,1);
   vectMoves[7]   = move7 ;
 final Move move8 =  w.new Move(0,0);
   vectMoves[8]   = move8 ;
}


// All worms are posited at the same initial row
// in vertical position
static private void startLine()
{
    for (int W = 0; W < nWorms; W++)
    {
        int  x,y;
        //Auxiliary worm
        Worm   worm = w.new Worm();
        //Each link is positioned
        for(int  link = 0; link < nLinks; link++)
        {
            //Horizontal position
            x = 30 + 100 * W;
            //Vertical position
            y =  initialRow + height*link;
            //Position is updated
            worm.rect[link].setX(x);
            worm.rect[link].setY(y);
            //Link is booked
            p.getChildren().add(worm.rect[link]);
        }
        //Memorization
        Worms[W]= worm;
    }
```

```
}


//Worm W
//translates each one of its links
//according to the information
//consigned in mover[].
//Move m is executed.
private static TranslateTransition translate( int W, int m)
{

    //Read the link to be moved
    int link = Worms[W].mover[m].link;
    if (print1)
        System.out.println("\n Worm = " + W
                           + " move = " + m
                            + " link = " + link);

    //Read move
    Move  move;
    move = Worms[W].mover[m].move;
    //Move is interpreted
    int byX = width* move.x;
    int byY = -(height/2)* move.y;
    if (print1)
        System.out.println(" byX = " + byX
                           + " byY = " + byY);
    //Read position
    int x = (int) Worms[W].rect[link].getX();
    int y = (int) Worms[W].rect[link].getY();
    if (print1)
        System.out.println(" x = " + x
                           + " y = " + y);
    //Position is updated
    x = x + byX;
    y = y + byY;
    if (print1)
        System.out.println(" new x = " + x
                           + " new  y = " + y);
    Worms[W].rect[link].setX(x);
```

```
    Worms[W].rect[link].setY(y);

    //Duration in within tics
    Duration d = Duration.millis(500);
    //A soft jump is programmed
    TranslateTransition translateTransition =
         new TranslateTransition(d, Worms[W].rect[link]);
    //Jump is executed
    translateTransition.setByX(byX);
    translateTransition.setByY(byY);
    //Number of repetitions
    translateTransition.setCycleCount(1);

    return translateTransition;
 }


//Worms climb the screen.
//Various transitions are nested to achieve that.
private    void  climb()
{
    //Overall process
    SequentialTransition allST
                   = new SequentialTransition();

     allST.setCycleCount(1);
     //worms advance nSteps
    for (int i = 0; i < nSteps; i++)
    {
        //Worms advance one step:
        //To move all links, each worm must execute
        //instructions to move in mover[]
        SequentialTransition oneStepST
                       = new SequentialTransition();
        oneStepST.setCycleCount(1);
        //Information is read and decoded
        for (int m = 0; m < nMovers; m++)
        {
            //All worms execute move m
            ParallelTransition moveMPT
```

```
                    = new ParallelTransition();
            moveMPT.setCycleCount(1);
            for (int W = 0; W < nWorms; W++)
            {
                //Direction is flipped
                Worms[W].mover[m].move.x
                            = - Worms[W].mover[m].move.x;
                //A translation of some link
                //is programmed for worm W
                //according to move m
                TranslateTransition tT = translate(W,m);
                tT.setCycleCount(1);
                //Translation is booked
                moveMPT.getChildren().add(tT);
            }
            //The ensamble of translations is booked
            oneStepST.getChildren().add(moveMPT);
        }
    //The movement of worms in one step is booked
    allST.getChildren().add(oneStepST);
    }
    //Dynamics is played
    allST.play();
}


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


} //End of main class L50  WormsRandJFX
```

51 ***Exercise.*** *Run the program and play with the code.*

Randomness is best appreciated when it can be repeatedly run. We have two strategies to include repetitions. First: we can make the same worms to start at the same startline the needed number of times. Second: we can kill all worms at the end of each session and rebuilt a new world for each new try. We implement the first strategy and the second is left as an exercise.

52 ***Repetitions through repositioning.*** *The next code in JavaFX instructs links of worms to move at random. The possibility of automatic repetition is included by repositioning worms at the start line.*

```
package ejvolu12p;

//Program L52 WormsRandJFX2
//A population of worms is programmed
//by randomness to
//climb the screen.
//Repetitions are allowed. To that aim,
//worms are repositioned at the
//start line to begin a new try.
//Program is written in JavaFX.



import java.util.Random;
import javafx.animation.ParallelTransition;
import javafx.animation.SequentialTransition;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.util.Duration;



public class WormsRandJFX2 extends Application
{
    private static WormsRandJFX2 w = new WormsRandJFX2();
    //Overall process
    private static  SequentialTransition nRepetitionsST
                    = new SequentialTransition();
    //Number of repetitions
    private static int nRepetitions = 5;
```

```
    private static int repCounter = 0;

    private static   Group p = new Group();
    //Number of steps
    int nSteps = 3;
    //Number of worms
    private static final int nWorms = 5;
    //Worms are kept in an array
    private static Worm[] Worms = new Worm[nWorms];
    //Number of links of each worm
    private static final int nLinks = 9;
    //Initial position of the head
    private static int initialRow = 180;
    //Generator of pseudo random numbers
    private static final Random r = new Random();
    //Each worm has nMovers movers,
    //instructions to move.
    private static int  nMovers = 3;
    //Width of the worm = width of each link
    private static int width = 4;
    //Height of each link
    private static int height = 12;
    //Vector with 9 possible moves
    private static Move  vectMoves[ ] = new Move[9];

    private static final boolean print1 = false;
    //Initial position of worms
    private static int[][][]  xy = new int[nWorms][nLinks][2];


    /* Definition of inner classes.
    *
    The Move class:
    A Move says in which direction
    a link of the worm must be moved.
    */

 private  class Move
 {
    int x;
```

```
    int y;


    //Default initialization
    Move()
    {
        x= 0;
        y= 0;
    }

    // Initialize wth  coordinates
    Move(int a, int b)
    {
        x= a;
        y= b;
    }

    // Copy a move
    Move(Move move)
    {
        x= move.x;
        y= move.y;
    }
}//End of class Move


// Definition of the class Mover:
// A Mover keeps two types of information:
// a) the link of the worm to be moved and
// b) the direction or move to be chosen.
//
private   class Mover
{
    int link ;
    Move   move;

    // Movers can acquired information
    // in three ways, they are  overloaded.
    // Form 1: by a list
    private Mover(int l, int mx, int my)
```

```
     {
        link= l;
        move.x = mx;
        move.y = my;
     }

    // Form 2, by structured data
    private Mover(int l, Move m)
    {
        link= l;
        move = m;
        move.x = m.x;
        move.y = m.y;
    }

    // Form 3, by structured data
    private Mover(Mover mover)
    {
        link= mover.link;
        move =  new Move(mover.move);
    }
  }//End of class Mover




    private static  Rectangle
           createRectangle()
    {
        //Upper left corner at (0,0)
        Rectangle rect = new Rectangle(0,0,width,height);
        //Color is defined as a random
        //mix of red, green and blue
        int rr,gg,bb;
        rr = r.nextInt(226);
        gg = r.nextInt(226);
        bb = r.nextInt(226);
//The color is prepared
        Color c = Color.rgb(rr,gg,bb);
        //The rect is painted
```

```
        rect.setFill(c);
    return rect;
    }

/* Here we define the class worm.
    A worm is composite: it has links.
    Each link has a rectangular shape
    and  its own color,
    which is defined by three numbers.
    The vector Mover[ ]
     keeps the information
    on how to move its links:
    which  link and in which order
    (in the order that they appear in
    the vector Mover[]), and
    in which direction.
    Each worm keeps record of
    the actual position of each
    one of its links,
    in the associated ractangle. */



private class Worm
{
    //Number of links
    int numberOfLinks = nLinks;
    //Number of movers = instructions to move
    int numberOfMovers = nLinks;
    //mover[] keeps movers
    Mover mover[ ] = new Mover[nLinks];
    //The shape of each link
    Rectangle rect[] = new Rectangle[20];
    //move right (side = 1) else left (side = -1)
    int side = 2*r.nextInt(2) -1;



  //Default initialization
   private Worm( )
```

```
   {
//Rects are created.
        // A color is assigned to each link
        // but different than white
        for(int i = 0; i < numberOfLinks; i++)
            rect[i] = createRectangle();
        //Instructions to move at random
for(int m = 0; m < nMovers; m++)
        {
            //Select at random  which link to move:
            int l = r.nextInt(numberOfLinks) ;
            //Select at random  which direction to move on
            Move  move = w.new Move (vectMoves[r.nextInt(9)]);
            //Memorization
            mover[m]= w.new Mover(l,move);
        }
 }//End of constructor Worm()

}//End of class Worm



@Override public void start(Stage stage)
    {
        //The scene
        Scene scene = new Scene(p, 500,500);
        stage.setScene(scene);
        stage.setTitle("Links move at random");
        stage.show();
        //Definition of possible moves
        moveDef();

        nRepetitionsST.setCycleCount(1);
        for(int i = 0; i < nRepetitions; i++)
        {
            repCounter = i;
            //Worms are posited at the start line
            startLine();
            //Worms climb the screen
            SequentialTransition oneTrialST  =  climb();
```

```
            nRepetitionsST.getChildren().add(oneTrialST);
        }
    //Movie is played
    nRepetitionsST.play();
    }


 //Nine basic moves = coordinates where to move in
static private void moveDef()
{
 final Move move0 =  w.new Move(0,1);
   vectMoves[0]   = move0 ;
 final Move move1 =  w.new Move(1,1);
   vectMoves[1]   = move1 ;
 final Move move2 =  w.new Move(1,0);
   vectMoves[2]   = move2 ;
 final Move move3 =  w.new Move(1,-1);
   vectMoves[3]   = move3 ;
 final Move move4 =  w.new Move(0,-1);
   vectMoves[4]   = move4 ;
 final Move move5 =  w.new Move(-1,-1);
   vectMoves[5]   = move5 ;
 final Move move6 =  w.new Move(-1,0);
   vectMoves[6]   = move6 ;
 final Move move7 =  w.new Move(-1,1);
   vectMoves[7]   = move7 ;
 final Move move8 =  w.new Move(0,0);
   vectMoves[8]   = move8 ;
}


//All worms are posited at the same initial row
//in vertical position.
//There is one procedure for the first time
//(all links must be booked),
//and another procedure for later times:
//no booking but worms must be repositioned
//at the start line.
static public void startLine()
{
```

```
for (int W = 0; W < nWorms; W++)
{
    int  x,y;
    //Auxiliary worm
    Worm   worm;
    if (repCounter == 0)
        worm  = w.new Worm();
    else
        worm = Worms[W];
    //Each link is positioned
    for(int  link = 0; link < nLinks; link++)
    {
        //Horizontal position
        x = 30 + 50 * W;
        //Vertical position
        y = initialRow + height*link;

       //Position is updated
        worm.rect[link].setX(x);
        worm.rect[link].setY(y);
        //Link is booked for the   0-th trial
        if (repCounter == 0)
        {
            p.getChildren().add(worm.rect[link]);
            xy[W][link][0] = x;
            xy[W][link][1] = y;
        }
        else //link is repositioned
        {
            //Duration
            Duration d = Duration.millis(1);
            //A soft jump is programmed
            TranslateTransition translateTransition =
                new TranslateTransition(d,
                                        worm.rect[link]);

            //Repositioning at initial position
            translateTransition.setToX(0);
            translateTransition.setToY(0);
```

```
                //Number of repetitions
                translateTransition.setCycleCount(1);
                //Repositioning is booked
                nRepetitionsST.getChildren()
                        .add(translateTransition);
            }
        }
        Worms[W]= worm;
    }
}



//To climb the screen the worm W
//translates each one of its links
//according to the information
//consigned in mover[].
//Move m is executed: it moves link m.
private static TranslateTransition
                    translate( int W, int m)
{
    if (print1)
        System.out.println("\n Worm = " + W
                           + " move = " + m);
    //Read the link to be moved
    int link = Worms[W].mover[m].link;
    //Read position
    int x = (int) Worms[W].rect[link].getX();
    int y = (int) Worms[W].rect[link].getY();
    if (print1)
        System.out.println(" x = " + x
                           + " y = " + y);
    //The move to be done is read
    Move  move1;
    move1 = Worms[W].mover[m].move;
    //Move is interpreted
    int byX = 4* move1.x;
    int byY = -6* move1.y;
```

```
    // A new place is defined:
    x = x + byX;
    y = y + byY;
    if (print1)
        System.out.println(" byX = " + byX
                           + " byY = " + byY);

    //Address is updated
    Worms[W].rect[link].setX(x);
    Worms[W].rect[link].setY(y);

    //Duration in within tics
    Duration d = Duration.millis(500);
    //A soft jump is programmed
    TranslateTransition translateTransition =
         new TranslateTransition(d, Worms[W].rect[link]);
    //Jump is executed
    translateTransition.setByX(byX);
    translateTransition.setByY(byY);
    //Number of repetitions
    translateTransition.setCycleCount(1);

    return translateTransition;
 }


//Worms climb the screen.
//Various transitions are nested to achieve that.
public    SequentialTransition  climb()
{
    //Overall process
    SequentialTransition oneTrialST
                    = new SequentialTransition();
     oneTrialST.setCycleCount(1);
     //worms give nSteps steps
    for (int i = 0; i < nSteps; i++)
    {
        //Worms advance one step:
        //To move all links, each worm must execute
        //instructions to move in mover[]
```

```
        SequentialTransition oneStepST
                        = new SequentialTransition();
        oneStepST.setCycleCount(1);
        //Information is read and decoded
        for (int m = 0; m < nMovers; m++)
        {
            //All worms execute move m
            ParallelTransition moveMPT
                        = new ParallelTransition();
            moveMPT.setCycleCount(1);
            for (int W = 0; W < nWorms; W++)
            {
                Worm  worm;
                worm = Worms[W];
                //Direction is flipped
                worm.mover[m].move.x
                            = - worm.mover[m].move.x;
                //A translation of some link
                //is programmed for worm W
                //according to move m
                TranslateTransition tT = translate(W,m);
                tT.setCycleCount(1);
                //Translation is booked
                moveMPT.getChildren().add(tT);
            }
            //The ensamble of translations is booked
            oneStepST.getChildren().add(moveMPT);
        }
    //The movement of worms in one step is booked
    oneTrialST.getChildren().add(oneStepST);
    }
    return oneTrialST;
}



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

```
} //End of main class L52 WormsRandJFX2
```

53 ***Exercise. The previous program implements an exact repetition. This is not what we mean when we speak of giving randomness another try: randomness must invent something new for each new try. Implement this new directive.*** *Answer*

54 ***Exercise. Implement the second strategy to include repetitions: begin a completely new world for each repetition.*** *Answer*

4.3 Java Lists

Our worms are highly variable: their number might be modified as well as the number of their links and of instructions to move. To deal with ensembles of varying length, Java has provided the class List, which functions as a linear array but with the possibility to change its length. In consequence, the class List has procedures to add and delete objects. This possibility will be used below to model a trend towards complexity, which is represented by the addition of a new link to our worms when some worm of the present population has been shown to be able to climb the screen.

We start with the previous program: it is reformulated in the language of list and a link is added for each new generation.

55 ***Previous program is rewritten with the use of Java List instead of the usual arrays.*** *One link is added for each generation.*

```
//Program L55 WormsRandJFX5
//A population of worms is programmed
//by randomness to
//climb the screen.
//Repetitions are allowed for a certain
//number of generations.
//Each generations has its own
//population of worms.
//One link is added for each new generation.
//Program is written in JavaFX.

package ejvolu12p;
```

```
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javafx.animation.Interpolator;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.ParallelTransition;
import javafx.animation.PauseTransitionBuilder;
import javafx.animation.SequentialTransition;
import javafx.animation.Timeline;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.util.Duration;



public class WormsRandJFX5 extends Application
{
    private static WormsRandJFX5 w = new WormsRandJFX5();
    //Overall process
    private static  SequentialTransition nGenerationsST
                    = new SequentialTransition();
    //Number of repetitions or generations
    private static int nGenerations = 5;
    private static int gen = 0;

    //The scene
    private static   Group p = new Group();


    //Number of steps
    int nSteps = 1;
```

```
    //Number of worms
    private static  int nWorms = 5;
    //Worms are kept in a list
    private static List<Worm> Worms = new ArrayList<>();
    //Number of links of each worm
    private static  int nLinks = 4;
    //Initial position  of the head
    private static int initialRow = 180;
    //Generator of pseudo random numbers
    private static final Random r = new Random();
    //Each worm has nMovers movers,
    //instructions to move.
    private static int  nMovers = 3;
    //Width of the worm = width of each link
    private static int width = 4;
    //Height of each link
    private static int height = 12;
    //Array with 9 possible moves
    private static Move  vectMoves[ ] = new Move[9];

    private static final boolean print1 = true;



    /* Definition of inner classes.
    *
    The Move class:
    A Move says in which direction
    a link of the worm must move.
    */

 private  class Move
 {
    int x;
    int y;


    //Default initialization
    Move()
    {
```

```
        x= 0;
        y= 0;
    }

    // Initialize with  two coordinates
    Move(int a, int b)
    {
        x= a;
        y= b;
    }

    // Copy a move
    Move(Move move)
    {
        x= move.x;
        y= move.y;
    }
}//End of class Move


// Definition of the class Mover:
// A Mover keeps two types of information:
// a) the link of the worm to be moved and
// b) the direction or move to be chosen.
//
private   class Mover
{
    int link ;
    Move   move;

    //  Movers can acquired information
    // in three ways, they are  overloaded.
    // Form 1: by a list
    private Mover(int l, int mx, int my)
     {
        link= l;
        move.x = mx;
        move.y = my;
     }
```

```
    // Form 2, by structured data
    private Mover(int l, Move m)
    {
        link= l;
        move = new Move(m);
    }

    // Form 3, by structured data
    private Mover(Mover mover)
    {
        link= mover.link;
        move =  new Move(mover.move);
    }
  }//End of class Mover



//Rects that belongs in the worms of the first generation
//are ready to move. Others are hidden.
private static  Rectangle
          createRectangle(int gen)
{
    //Upper left corner at (0,0)
    Rectangle rect = new Rectangle(0,0,width,height);
    //Color is defined as a random
    //mix of red, green and blue
    int rr,gg,bb;
    rr = r.nextInt(226);
    gg = r.nextInt(226);
    bb = r.nextInt(226);
    //The color is prepared
    Color c = Color.rgb(rr,gg,bb);
    //The rect is painted
    rect.setFill(c);
    //Rects of the first generation (gen = 0)
    //are ready to move,
    //otherwise they got hidden
    if (gen > 0)
    {
        rect.setHeight(0);
```

```
        rect.setWidth(0);
    }
    return rect;
}


/*  Here we define the class worm.
    A worm is composite: it has links.
    Each link has a rectangular shape
    and  its own color,
    which is defined by three numbers.
    The vector Mover[ ]
    keeps the information
    on how to move its links:
    which  link and in which order
    (in the order that they appear in
    the vector Mover[]), and
    in which direction.
    Each worm keeps record of
    the actual position of each
    one of its links,
    in the associated ractangle. */



private class Worm
{
    //Number of links
    int numberOfLinks = nLinks;
    //Number of movers = instructions to move
    int numberOfMovers = nLinks;
    //List mover  keeps movers
    List<Mover> Mover = new ArrayList<>();
    //The shape of each link
    List<Rectangle> Rect = new ArrayList<>();
    //move right (side = 1) else left (side = -1)
    int side = 2*r.nextInt(2) -1;
```

```
    //Default initialization:
    //worms of the first generation are
    //ready to move. Others get hidden.
    private Worm(int gen )
    {
//Rects are created.
        // A color is assigned to each link
        // but different than white
        for(int i = 0; i < numberOfLinks; i++)
            Rect.add(createRectangle(gen));
        //Instructions to move at random
for(int m = 0; m < nMovers; m++)
        {
            //Select at random  which link to move:
            int l = r.nextInt(nLinks) ;
            //Select at random  which direction to move on
            Move  move = w.new Move (vectMoves[r.nextInt(9)]);
            //Memorization
            Mover.add(w.new Mover(l,move));
        }
    }//End of constructor Worm(int  gen)

}//End of class Worm



@Override public void start(Stage stage)
    {
        Scene scene = new Scene(p, 500,500);
        stage.setScene(scene);
        stage.setTitle("Links move at random");
        stage.show();
        //Definition of possible moves
        moveDef();

        nGenerationsST.setCycleCount(1);
        for(int i = 0; i < nGenerations; i++)
        {
            gen = i;
            if (print1)
```

```
                System.out.println("\nGen = " + gen);
            //Worms are posited at the start line
            startLine(gen);
            //Worms express their information to move
            SequentialTransition oneGenST  =  express();
            nGenerationsST.getChildren().add(oneGenST);

            //Worms get hidden
            ParallelTransition hideST  =  hideST();
            if (gen < nGenerations-1)
            {
                //Clear Worms as list and hide drawn worms
                Worms.clear();
                nGenerationsST.getChildren().add(hideST);
            }
            //Add a new link
            nLinks = nLinks+1;
        }
    //Movie is played
    nGenerationsST.play();
    }


 //Nine basic moves = coordinates where to move in
static private void moveDef()
{
 final Move move0 =  w.new Move(0,1);
   vectMoves[0]   = move0 ;
 final Move move1 =  w.new Move(1,1);
   vectMoves[1]   = move1 ;
 final Move move2 =  w.new Move(1,0);
   vectMoves[2]   = move2 ;
 final Move move3 =  w.new Move(1,-1);
   vectMoves[3]   = move3 ;
 final Move move4 =  w.new Move(0,-1);
   vectMoves[4]   = move4 ;
 final Move move5 =  w.new Move(-1,-1);
   vectMoves[5]   = move5 ;
 final Move move6 =  w.new Move(-1,0);
   vectMoves[6]   = move6 ;
```

```
 final Move move7 =  w.new Move(-1,1);
   vectMoves[7]   = move7 ;
 final Move move8 =  w.new Move(0,0);
   vectMoves[8]   = move8 ;
}


// All worms are posited at the same initial row
// in vertical position.
static public void startLine(int gen)
{
    for (int W = 0; W < nWorms; W++)
    {
        int  x,y;
        //Auxiliary worm
        Worm   worm = w.new Worm(gen);


        //Each link is positioned
        for(int  link = 0; link < nLinks; link++)
        {
            //Horizontal position
            x = 30 + 100 * W;
            //Vertical position
            y = initialRow + height*link;

            //Position is updated
            worm.Rect.get(link).setX(x);
            worm.Rect.get(link).setY(y);

            //Link is shown
            SequentialTransition show
                        = showST(worm.Rect.get(link));
            //Link is booked
            p.getChildren().add(worm.Rect.get(link));
            //Show transition is booked
            nGenerationsST.getChildren().add(show);
        }
        //worm is located at position W in list Worms
        Worms.add(worm);
```

```
    }

}



//To climb the screen the worm W
//translates each one of its links
//according to the information
//consigned in the list mover.
//Move m is executed: it moves link m.
private static TranslateTransition
                    translate( int W, int m)
{
    if (print1)
        System.out.println();
    //Reads the link to be moved
    int link = Worms.get(W).Mover.get(m).link;
    //Reads position
    int x = (int) Worms.get(W).Rect.get(link).getX();
    int y = (int) Worms.get(W).Rect.get(link).getY();

    //The move to be done is read
    Move  move1;
    move1 = Worms.get(W).Mover.get(m).move;
    //Move is interpreted
    int byX = 4* move1.x;
    int byY = -6* move1.y;
    // A new place is defined:
    x = x + byX;
    y = y + byY;
    if (print1)
        System.out.println("\n Worm = " + W
                        + " move = " + m
                        + " link = " + link
                        + " \n x = " + x
                        + " y = " + y
                        + " \nbyX = " + byX
                        + " byY = " + byY);
```

```
    //Address is updated
    Worms.get(W).Rect.get(link).setX(x);
    Worms.get(W).Rect.get(link).setY(y);

    //Duration in within tics
    Duration d = Duration.millis(500);
    //A soft jump is programmed
    TranslateTransition translateTransition =
         new TranslateTransition(d, Worms.get(W).Rect.get(link));
    //Jump is executed
    translateTransition.setByX(byX);
    translateTransition.setByY(byY);
    //Number of repetitions
    translateTransition.setCycleCount(1);

    return translateTransition;
 }


//Worms express their information to move.
//Various transitions are nested to achieve that.
public    SequentialTransition  express()
{
    //Overall process
    SequentialTransition oneGenST
                    = new SequentialTransition();
     oneGenST.setCycleCount(1);
     //worms advance nSteps
    for (int i = 0; i < nSteps; i++)
    {
        //Worms advance one step:
        //To move all links, each worm must execute
        //instructions to move in mover[]
        SequentialTransition oneStepST
                        = new SequentialTransition();
        oneStepST.setCycleCount(1);
        //Information is read and decoded
        for (int m = 0; m < nMovers; m++)
        {
            //All worms execute move m
```

```
            ParallelTransition moveMPT
                        = new ParallelTransition();
            moveMPT.setCycleCount(1);
            for (int W = 0; W < nWorms; W++)
            {
                //A translation of some link
                //is programmed for worm W
                //according to move m
                TranslateTransition tT = translate(W,m);
                tT.setCycleCount(1);
                //Translation is booked
                moveMPT.getChildren().add(tT);
            }
            //The ensamble of translations is booked
            oneStepST.getChildren().add(moveMPT);
        }
    //The movement of worms in one step is booked
    oneGenST.getChildren().add(oneStepST);
    }
    return oneGenST;
}


    //A hidden rect gets shown
    private static SequentialTransition showST(Rectangle rect)
    {
        Duration d1 =   Duration.millis(1);

        //Rect must be shown at true scale
        KeyValue keyValue1 =
                new KeyValue(rect.scaleXProperty(),
                             1, Interpolator.LINEAR);
        KeyValue keyValue2 =
                new KeyValue(rect.scaleYProperty(),
                             1, Interpolator.LINEAR);
        Timeline b = new Timeline(
                new KeyFrame(d1,keyValue1,keyValue2 ));

        SequentialTransition s = new SequentialTransition();
        Duration d2 =   Duration.millis(1);
```

```
        //Width of  rect is enlarged
        //and after this
        //it is shown according to the
        //instructions given by timeline b.
        //To do this, a pause  is added.
        s.getChildren().add(
                new SequentialTransition(

                    PauseTransitionBuilder.create().
                        duration(d2)
                            .onFinished
                            ((ActionEvent t) ->
                            {
                                if (gen > 0)//Hidden -> show
                                {
                                    rect.setWidth(width);
                                    rect.setHeight(height);
                                }
                            })
                            .build(),b));
        return s;
    }


    //Worms are hidden after their performance.
    private static ParallelTransition  hideST()
    {

        ParallelTransition hidePT
                    = new ParallelTransition();
        hidePT.setCycleCount(1);
        //Hide all worms
        for (int W = 0; W < nWorms; W++)
        {
            //Hide each link
            for(int  link = 0; link < nLinks; link++)
            {
                Duration d2 =   Duration.millis(1);
                SequentialTransition ss
                    = new SequentialTransition();
```

```
                //Rectangles are scaled down to zero
                KeyValue keyValue1 =
                    new KeyValue(Worms.get(W).Rect.get(link)
                        .scaleXProperty(),
                            0, Interpolator.LINEAR);
                KeyValue keyValue2 =
                    new KeyValue(Worms.get(W).Rect.get(link)
                        .scaleYProperty(),
                            0, Interpolator.LINEAR);
                //KeyValues are  inserted into a timeline
                Timeline b = new Timeline(
                    new KeyFrame(d2,keyValue1,keyValue2 ));
                //Timeline is inserted into a transition
                ss.getChildren().addAll(b);
                hidePT.getChildren().add(ss);
            }
        }
        return hidePT;
    }



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


} //End of main class L55 WormsRandJFX5
```

56 ***Exercise.*** *Rewrite program L53 WormsRandJFX3 using Java List and enable the addition of one link per generation.* Answer

57 ***Climbing by twisting randomness.***

The task we have to solve is to instruct a set of rectangles to coordinately move in such a way that the movement of the whole ensemble mimics the climbing of a worm up a wall. This task can eventually be solved by randomness because it can achieve anything if it is empowered with sufficient resources, particularly with

time. The problem is that the needed time for solving a rather complex problem may diverge beyond the age of the universe no matter how powerful your computing machines might be. Nevertheless, one can twist randomness very simply: just pick up those events that happen at random and that are useful for the purpose you have. To show how this functions, we challenge randomness with a game: beginning with two rectangles, randomness proposes moves but the only moves that are executed are those that make the worm climb and that are not disruptive. If some worm succeeds because it can climb the screen, a new link is added. And so on. The code follows:

```
//Program L57 WormsRandJFX7
//A population of worms is programmed
//by twisted randomness to
//climb the screen.
//Each generations has its own
//population of worms.
//All worms have two links at the start.
//One link is added when some worm is shown
//to climb the screen.
//Program is written in JavaFX.



package ejvolu12p;

import static java.lang.Math.pow;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javafx.animation.Interpolator;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.ParallelTransition;
import javafx.animation.PauseTransitionBuilder;
import javafx.animation.SequentialTransition;
import javafx.animation.Timeline;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
```

```
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Text;
import javafx.stage.Stage;
import javafx.util.Duration;



public class WormsRandJFX7 extends Application
{
    private static WormsRandJFX7 w = new WormsRandJFX7();
    //Overall process
    private static  SequentialTransition nGenerationsST
                    = new SequentialTransition();
    //Number of repetitions or generations
    private static int nGenerations = 10;
    //Actual generation
    private static int gen = 0;

    //Duration of events
    private static final Duration d = Duration.millis(100);

    private static   Group p = new Group();


    //Number of steps given for each try
    int nSteps = 8;
    //Number of worms
    private static  int nWorms = 8;
    //Worms are kept in a list
    private static List<Worm> Worms = new ArrayList<>();
    //Number of links of each worm
    private static  int nLinks = 2;
    //Initial position  of the head
    private static int initialRow = 280;
    //Generator of pseudo random numbers
    private static final Random r = new Random();
```

```
   //Each worm has nMovers movers,
   //instructions to move.
   private static int  nMovers = 3;
   //Width of the worm = width of each link
   private static int width = 6;
   //Height of each link
   private static int height = 6;
   //Array with 9 possible moves
   private static Move  vectMoves[ ] = new Move[9];

   private static final boolean print1 = false;
   private static final boolean print2 = false;
   //Text to be drawn
   private static  Text genCount = new Text ("");
   //Two links cannot occupy the same place
   static boolean isOccupied[][]  = new boolean[600][300];
   //Is a worm capable of climbing the screen?
   private static  int Fitness[] =  new int[21];
   //Was climbing good enough?
   private static boolean bullseye  = false;
   //List of generations at which success was made
   private static String record = "";
   //Measures of the screen
   private static int xWin = 1200;
   private static int yWin = 850;




   /* Definition of inner classes.
   *
   The Move class:
   A Move says in which direction
   a link of the worm must move.
   */

private  class Move
{
   int x;
```

```
    int y;


    //Default initialization
    Move()
    {
        x= 0;
        y= 0;
    }

    // Initialize with   two coordinates
    Move(int a, int b)
    {
        x= a;
        y= b;
    }

    // Copy a move
    Move(Move move)
    {
        x= move.x;
        y= move.y;
    }
}//End of class Move


// Definition of the class Mover:
// A Mover keeps two types of information:
// a) the link of the worm to be moved and
// b) the direction or move to be chosen.
//
private   class Mover
{
    int link ;
    Move   move;

    //  Movers can acquired information
    // in three ways, they are  overloaded.
    // Form 1: by a list of numbers
    private Mover(int l, int mx, int my)
```

```
     {
        link= l;
        move.x = mx;
        move.y = my;
     }

    // Form 2, by structured data: link + move
    private Mover(int l, Move m)
    {
        link= l;
        move = new Move(m);
    }

    // Form 3, by structured data: mover
    private Mover(Mover mover)
    {
        link= mover.link;
        move =  new Move(mover.move);
    }
  }//End of class Mover



//Rects that belong in the worms of the first generation
 //are visible. Others are hidden.
private static  Rectangle
           createRectangle(int gen)
{
    //Upper left corner at (0,0)
    Rectangle rect = new Rectangle(0,0,width,height);
    //Color is defined as a random
    //mix of red, green and blue
    int rr,gg,bb;
    rr = r.nextInt(226);
    gg = r.nextInt(226);
    bb = r.nextInt(226);
    //The color is prepared
    Color c = Color.rgb(rr,gg,bb);
    //The rect is painted
    rect.setFill(c);
```

```
    //Rects of the first generation (gen = 0)
    //are ready to move,
    //otherwise they got hidden
    if (gen > 0)
    {
        rect.setHeight(0);
        rect.setWidth(0);
    }
    return rect;
}


/*  Here we define the class worm.
    A worm is composite: it has links.
    Each link has a rectangular shape
    and  its own color,
    which is defined by three numbers.
    The vector Mover[ ]
    keeps the information
    on how to move its links:
    which  link and in which order
    (in the order that they appear in
    the vector Mover[]), and
    in which direction.
    Each worm keeps record of
    the actual position of each
    one of its links,
    in the associated rectangle. */



private class Worm
{
    //Number of links
    int numberOfLinks = nLinks;
    //Number of movers = instructions to move
    int numberOfMovers = nLinks;
    //List mover  keeps movers
    List<Mover> Mover = new ArrayList<>();
    //The shape of each link
```

```
    List<Rectangle> Rect = new ArrayList<>();



    //Default initialization:
    //worms of the first generation are
    //ready to move. Others get hidden.
    private Worm(int gen )
    {
// Rects are created.
        // A color is assigned to each link
        // but different than white
        for(int i = 0; i < numberOfLinks; i++)
            Rect.add(createRectangle(gen));
        //Instructions to move at random
for(int m = 0; m < nMovers; m++)
        {
            //Select at random  which link to move:
            int l = r.nextInt(nLinks) ;
            //Select at random  which direction to move on
            Move  move
                    = w.new Move (vectMoves[r.nextInt(9)]);
            //Memorization
            Mover.add(w.new Mover(l,move));
        }
    }//End of constructor Worm(int  gen)

}//End of class Worm


//General setting
@Override public void start(Stage stage)
    {
        Scene scene = new Scene(p, xWin,yWin);
        stage.setScene(scene);
        stage.setTitle("Links move at random");
        stage.show();
        //Definition of possible moves
        moveDef();
```

```
//Text to draw
genCount = new Text ("");
genCount.setLayoutX(40);genCount.setLayoutY(40);
p.getChildren().addAll(genCount);
nGenerationsST.setCycleCount(1);
//Many attempts are allowed
for(int i = 0; i < nGenerations; i++)
{
    voidOccupationMatrix();//Voidness
    gen = i;
    if (print2)
        System.out.println("\nGen = " + gen);
    //Generation number is drawn
    Timeline timeline = drawGen(gen);
    nGenerationsST.getChildren().addAll( timeline );
    //Worms are posited at the start line
    startLine(gen);
    //Worms express their information to move
    SequentialTransition oneGenST  =  express();
    nGenerationsST.getChildren().add(oneGenST);
    //Is some worm able to climb?
    assessFitness();
    //Worms get hidden
    ParallelTransition hideST  =  hideST();
    if (gen < nGenerations - 1 )
    {
        //Clear Worms as list and hide drawn worms
        Worms.clear();
        nGenerationsST.getChildren().add(hideST);
    }
    //Success?
    Integer genI = gen;
    if (bullseye)
    {
        //If climbing was successful, add a new link
         nLinks = nLinks+1;
         //List of generations with sucess
        record = record + "  " + genI.toString();
    }
}
```

```
    //Movie is played
    nGenerationsST.play();

    System.out.println("Successful generations = "
            + record);
    }


// To avoid overwriting of worms,
// an occupation record is kept:
// The screen is divided in cells 6x6.
// At the beginning every cell is empty=false.
// Occupancy is recorded in isOccupied[][],
// which is a boolean matrix.
// A boolean  variable can be
// true= occupied,  else false= not occupied.
private static void  voidOccupationMatrix()
{
   //System.out.println("occupationMatrix");
    for(int i = 0; i < 600; i++)
    for(int j = 0; j < 300; j++)
        isOccupied[i][j]= false;
}

//Nine basic moves = coordinates where to move in
static private void moveDef()
{
    final Move move0 =  w.new Move(0,1);
       vectMoves[0]   = move0 ;
    final Move move1 =  w.new Move(1,1);
       vectMoves[1]   = move1 ;
    final Move move2 =  w.new Move(1,0);
       vectMoves[2]   = move2 ;
    final Move move3 =  w.new Move(1,-1);
       vectMoves[3]   = move3 ;
    final Move move4 =  w.new Move(0,-1);
       vectMoves[4]   = move4 ;
    final Move move5 =  w.new Move(-1,-1);
       vectMoves[5]   = move5 ;
    final Move move6 =  w.new Move(-1,0);
```

```
       vectMoves[6]   = move6 ;
    final Move move7 =  w.new Move(-1,1);
       vectMoves[7]   = move7 ;
    final Move move8 =  w.new Move(0,0);
       vectMoves[8]   = move8 ;
}


// All worms are posited at the same initial row
// in vertical position.
static public void startLine(int gen)
{
    for (int W = 0; W < nWorms; W++)
    {
        int  x,y;
        //Auxiliary worm
        Worm   worm = w.new Worm(gen);


        //Each link is positioned
        for(int  link = 0; link < nLinks; link++)
        {
            //Horizontal position
            x = 30 + 150 * W;
            //Vertical position
            y = initialRow + height*link;

            //Position is updated
            worm.Rect.get(link).setX(x);
            worm.Rect.get(link).setY(y);

            //Link is shown
            SequentialTransition show
                        = showST(worm.Rect.get(link));
            //Link is booked
            p.getChildren().add(worm.Rect.get(link));
            //Show transition is booked
            nGenerationsST.getChildren().add(show);
        }
        //worm is located at position W in list Worms
```

```
        Worms.add(worm);
    }

}


//Draws the generation number
private static Timeline drawGen(int gen)
{
    Integer genI = gen;
    String message =  "Generation  = " +
                    genI.toString();
    //A JavaFX event committed to print a text is generated
    EventHandler printText =
        (EventHandler<ActionEvent>)
            (ActionEvent t ) ->
                {
                    genCount.setText(message);
                };
    //Presentation of printText must be
    //synchronized with the overall film so,
    //duration d is specified
    KeyFrame keyFrame = new KeyFrame(d, printText);

    Timeline timeline = new Timeline();
    timeline.setCycleCount(1);
    //add the keyframe to the timeline
    timeline.getKeyFrames().add(keyFrame);

    return timeline;
}


//Is move m disruptive?
//(A worm is and must remain as a whole,
//nevertheless, some elasticity is allowed.)
private static boolean isDisruptive(int m, Worm worm )
{

    // A move is disruptive if the target address
```

```
// is farther than certain distance
// from the neighboring links.
boolean isDisruptive = false;
int  link;
int  x,y,  x1,y1, x2,y2, x3,y3;
double d1;

double  d2;

//Read the link to be moved
link = worm.Mover.get(m).link;
//Read the   address of the link;
x  = (int) worm.Rect.get(link).getX();
y  = (int) worm.Rect.get(link).getY();
//The   move to be done is read
Move move = w.new Move(worm.Mover.get(m).move);

//The target address is calculated
x1 = x + width*move.x;
y1 = y -  width*move.y;

//The addresses of the two neighboring links are read
//The disruption is calculated
//Case: intermediate link
if ((0 < link) & (link < nLinks-1))
 {
  x2 = (int) worm.Rect.get(link-1).getX();

  y2 = (int) worm.Rect.get(link-1).getY();

  d1 =    pow(  ( pow( (x2 - x1),2)
                        +  pow((y2 - y1),2)),0.5);

  x3 = (int) worm.Rect.get(link+1).getX();
  y3 = (int) worm.Rect.get(link+1).getY();
  d2 =  pow(  (pow( (x3 - x1),2)
                        + pow((y3 - y1),2)),0.5);


  // Here we decide whether or not
```

```
      // a move is disruptive.
      //To augment elasticity, augment the threshold

      if ((d1 > 1.415*width) || (d2 > 1.415*width))
                  isDisruptive = true;

     }

    //First link
    if (link == 0)
    {
        x2 = (int) worm.Rect.get(1).getX();
        y2 = (int) worm.Rect.get(1).getY();
        d1 = pow( (pow( (x2 - x1),2) + pow((y2 - y1),2)),0.5);
        if( d1 > 1.1*width) isDisruptive = true;
     }
     //Last link
     if( link == nLinks-1 )
      {
        x2 = (int) worm.Rect.get(nLinks - 2).getX();
        y2 = (int) worm.Rect.get(nLinks - 2).getY();
        d2 = pow( (pow( (x2 - x1),2) + pow((y2 - y1),2)),0.5);
        if ( d2 > 1.1*width ) isDisruptive =   true;
       }
      return isDisruptive;
 }




  //Is the move m of worm W not disruptive?
//If it is not, execute it.
private static SequentialTransition
                  exeIf( Worm worm, int m)
{
    if (print1)
        System.out.println(" exeIf");
    //Reads the link to be moved
    int link = worm.Mover.get(m).link;
    //Reads position
```

```
int x = (int) worm.Rect.get(link).getX();
int y = (int) worm.Rect.get(link).getY();

//The move to be done is read
Move  move;
move = worm.Mover.get(m).move;
//Move is interpreted
int byX = width* move.x;
int byY = -height* move.y;
//Target address is calculated:
int x1 = x + byX;
int y1 = y + byY;
if (x1 < 0 ) x1 = 0;
if (y1 < 0 ) y1 = 0;
if (print1)
    System.out.println(
        " move = " + m   + " link = " + link
        + " \n x = " + x + " y = " + y
        + " \nbyX = " + byX  + " byY = " + byY);


//Occupied cell
int i = (int)  x / width;
int j = (int) y / height;


 //Target cell
int k =(int) x1/width;
int l = (int) y1/height;


//Move is always executed
SequentialTransition a =      new   SequentialTransition();

TranslateTransition translateTransition =
     new TranslateTransition(d, worm.Rect.get(link));
//Jump is executed
translateTransition.setByX(byX);
translateTransition.setByY(byY);
//Number of repetitions
```

```
translateTransition.setCycleCount(1);
a.getChildren().add(translateTransition);



//Is the move disruptive?
boolean   isMoveDisruptive =  isDisruptive(m,worm);
if (print2)
{
    if (isMoveDisruptive)
        System.out.println("move is Disruptive");
    else
        System.out.println("move is not  Disruptive");
}
//Disruptive moves are undone
if  (((isOccupied[k][l])) || ((isMoveDisruptive)))
{
    byX = -byX;
    byY = -byY;
}
else //move is not disruptive -> it is not undone
{
    byX = 0;
    byY = 0;
    worm.Rect.get(link).setX(x1);
    worm.Rect.get(link).setY(y1);
    isOccupied[i][j]= false;
    isOccupied[k][l]= true;
}



//Move is possibly undone
TranslateTransition translateTransition2 =
     new TranslateTransition(d, worm.Rect.get(link));
//Jump is programmed
translateTransition2.setByX(byX);
translateTransition2.setByY(byY);
//Number of repetitions
```

```
    translateTransition2.setCycleCount(1);
    a.getChildren().add(translateTransition2);

return a;
 }



//Worms express their information to move.
public    SequentialTransition  express()
{
    //Overall process
    SequentialTransition expressST
                    = new SequentialTransition();
     expressST.setCycleCount(1);
     //Worms advance nSteps steps
    for (int i = 0; i < nSteps; i++)
    {
        //Worms advance one step:
        //To move all links, each worm must execute
        //instructions to move
        SequentialTransition oneStepST
                        = new SequentialTransition();
        oneStepST.setCycleCount(1);
        //Information is read and decoded
        for (int m = 0; m < nMovers; m++)
        {
            if (print1)
                System.out.println("nMovers = " + nMovers);
            //All worms execute move m
            ParallelTransition moveMPT
                        = new ParallelTransition();
            moveMPT.setCycleCount(1);
            for (Worm worm: Worms)
            {
                //A translation of some link
                //is programmed for worm W
                //according to move m.
                //If it is not disruptive,
                //it is executed
```

```
                SequentialTransition tT = exeIf(worm,m);
                tT.setCycleCount(1);
                //Translation is booked
                moveMPT.getChildren().add(tT);
            }
            //The ensamble of translations is booked
            oneStepST.getChildren().add(moveMPT);
        }
    //The movement of worms in one step is booked
    expressST.getChildren().add(oneStepST);
    }
    return expressST;
}



    //A hidden rect gets shown
    private static
        SequentialTransition showST(Rectangle rect)
    {
        Duration d1 =   Duration.millis(1);

        //Rect must be shown at true scale
        KeyValue keyValue1 =
                new KeyValue(rect.scaleXProperty(),
                             1, Interpolator.LINEAR);
        KeyValue keyValue2 =
                new KeyValue(rect.scaleYProperty(),
                             1, Interpolator.LINEAR);
        Timeline b = new Timeline(
                new KeyFrame(d1,keyValue1,keyValue2 ));

        SequentialTransition s = new SequentialTransition();
        Duration d2 =   Duration.millis(1);
        //Width of  rect is enlarged
        //and after this
        //it is shown according to the
        //instructions given by timeline b.
        //To do this, a pause  is added.
```

```
        s.getChildren().add(
                new SequentialTransition(

                    PauseTransitionBuilder.create().
                        duration(d2)
                            .onFinished
                            ((ActionEvent t) ->
                            {
                                if (gen > 0)//Hidden -> show
                                {
                                    rect.setWidth(width);
                                    rect.setHeight(height);
                                }
                            })
                            .build(),b));
        return s;
    }


    //Worms are hidden after their performance.
    private static ParallelTransition  hideST()
    {

        ParallelTransition hidePT
                    = new ParallelTransition();
        hidePT.setCycleCount(1);
        //Hide all worms
        for (int W = 0; W < nWorms; W++)
        {
            //Hide each link
            for(int  link = 0; link < nLinks; link++)
            {
                Duration d2 =   Duration.millis(1);
                SequentialTransition ss
                    = new SequentialTransition();
                //Rectangles are scaled down to zero
                KeyValue keyValue1 =
                    new KeyValue(Worms.get(W).Rect.get(link)
                        .scaleXProperty(),
                             0, Interpolator.LINEAR);
```

```
                KeyValue keyValue2 =
                    new KeyValue(Worms.get(W).Rect.get(link)
                        .scaleYProperty(),
                            0, Interpolator.LINEAR);
                //KeyValues are  inserted into a timeline
                Timeline b = new Timeline(
                    new KeyFrame(d2,keyValue1,keyValue2 ));
                //Timeline is inserted into a transition
                ss.getChildren().addAll(b);
                hidePT.getChildren().add(ss);
            }
        }
        return hidePT;
    }


// Every worm is assigned a fitness
//The nearer to the top, the fitter.
//If climbing has been significative,
//the complexity of the game rises up one level:
//a true bullseye is reported (to add
//next a new link to all worms).
private static void assessFitness()
{
    int yOfTail;
    double yOfHead;

    bullseye  = false;
    for(int W = 0; W <  nWorms; W++)
    {
        // We get the y-coordinate of the tail
        yOfTail = (int) Worms.get(W).Rect.get(nLinks-1).getY();
        yOfHead =  Worms.get(W).Rect.get(0).getY();
        //Climbing up is worth  a point
        Fitness[W] = 100*(yWin - yOfTail);
        if(print1)
            System.out.println("Fitness of worm "
                    + W + " = " + Fitness[W]
                    + " head = " + yOfHead);
```

```
        //if (headPosition < initialRow - width*nLinks)
        if (yOfHead < initialRow - 2 * height)
        {
            bullseye = true;
            if (print1)
            System.out.println( "There was advance W = " + W );
        }
    }
}



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


} //End of main class L57 WormsRandJFX7
```

58 ***Exercise.*** *Run the program and play with the code. Try out other setting for the parameters that regulate elasticity. Invent and run new definitions of disruption.*

59 ***Challenge.*** *The previous program reinitializes everything with a completely new set of worms. Develop the alternative in which worms are repositioned at the startline for each new generation.*

60 ***Challenge.*** *Invent and run new definitions of disruption.*

61 ***Java evolves: builders became deprecated***

JavaFX represents a great leap in the development of Java. This demonstrates that Java evolves. According to what we claim, this must imply that the history of Java must be filled in bugs whose correction have given rise to more bugs. Is that true? Of course it is. As an example, the version of JavaFX that comes with Java8 has as a terrible modification the absence of builders that made javaFX 2.2 so popular but that were declared deprecated by Java8 because of the poisoning with intractable mathematical problems (this is referred to a bug in Java when it was property of Sun Microsystems).

Deprecated classes are tolerated but not encouraged in new versions of Java because each new version defines its own semantics and slight changes might cause severe damage to correct understanding. Deprecated classes point to bugs of design. So, let us get rid of the PauseTransitionBuilder, a builder that we have continuously used. The next code shows how can it be replaced by acceptable instructions. The same idea can be applied to all builders.

```
//Program L61 WormsRandJFX8
//A population of worms is programmed
//by twisted randomness to
//climb the screen.
//All moves are executed
//but  nondisruptive moves are
//not undone.
//Each generations has its own
//population of worms.
//All worms have two links at the start.
//One link is added when some worm is shown
//to climb the screen.
//Program is written in JavaFX.



package ejvolu12p;

import static java.lang.Math.pow;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javafx.animation.Interpolator;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.ParallelTransition;
import javafx.animation.SequentialTransition;
import javafx.animation.Timeline;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
```

```
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Text;
import javafx.stage.Stage;
import javafx.util.Duration;



public class WormsRandJFX8 extends Application
{
    private static WormsRandJFX8 w = new WormsRandJFX8();
    //Overall process
    private static  SequentialTransition nGenerationsST
                    = new SequentialTransition();
    //Number of repetitions or generations
    private static int nGenerations = 10;
    //Actual generation
    private static int gen = 0;

    //Duration of events
    private static final Duration d = Duration.millis(100);


    private static   Group p = new Group();


    //Number of steps given for each try
    int nSteps = 8;
    //Number of worms
    private static  int nWorms = 8;
    //Worms are kept in a list
    private static List<Worm> Worms = new ArrayList<>();
    //Number of links of each worm
    private static  int nLinks = 2;
    //Initial position over the screen of the head
    private static int initialRow = 280;
    //Generator of pseudo random numbers
```

```
private static final Random r = new Random();
//Each worm has nMovers movers,
//instructions to move.
private static int  nMovers = 3;
//Width of the worm = width of each link
private static int width = 6;
//Height of each link
private static int height = 6;
//Array with 9 possible moves
private static Move  vectMoves[ ] = new Move[9];

private static final boolean print1 = false;
private static final boolean print2 = false;
//Text to be drawn
private static  Text genCount = new Text ("");
//Two links cannot occupy the same place
static boolean isOccupied[][]  = new boolean[600][300];
//Is a worm capable of climbing the screen?
private static  int Fitness[] =  new int[21];
//Was climbing good enough?
private static boolean bullseye  = false;
//List of generations at which success was made
private static String record = "";
//Size of the screen
private static int xWin = 1200;
private static int yWin = 850;
//Grouping in order to ease erasing
private static List<Rectangle> allRects = new ArrayList<>();




/* Definition of inner classes.
*
The Move class:
A Move says in which direction
a link of the worm must move.
*/
```

```
private  class Move
{
   int x;
   int y;


   //Default initialization
   Move()
   {
       x= 0;
       y= 0;
   }

   // Initialize with   two coordinates
   Move(int a, int b)
   {
       x= a;
       y= b;
   }

   // Copy a move
   Move(Move move)
   {
       x= move.x;
       y= move.y;
   }
}//End of class Move


// Definition of the class Mover:
// A Mover keeps two types of information:
// a) the link of the worm to be moved and
// b) the direction or move to be chosen.
//
private   class Mover
{
   int link ;
   Move   move;

   //  Movers can acquired information
```

```
    // in three ways, they are  overloaded.
    // Form 1: by a list of numbers
    private Mover(int l, int mx, int my)
     {
        link= l;
        move.x = mx;
        move.y = my;
     }

    // Form 2, by structured data: link + move
    private Mover(int l, Move m)
    {
        link= l;
        move = new Move(m);
    }

    // Form 3, by structured data: mover
    private Mover(Mover mover)
    {
        link= mover.link;
        move =  new Move(mover.move);
    }
  }//End of class Mover



//Rects that belong in the worms of the first generation
 //are visible. Others are hidden.
private static  Rectangle
          createRectangle(int gen)
{
    //Upper left corner at (0,0)
    Rectangle rect = new Rectangle(0,0,width,height);
    //Color is defined as a random
    //mix of red, green and blue
    int rr,gg,bb;
    rr = r.nextInt(226);
    gg = r.nextInt(226);
    bb = r.nextInt(226);
    //The color is prepared
```

```
    Color c = Color.rgb(rr,gg,bb);
    //The rect is painted
    rect.setFill(c);
    //Rects of the first generation (gen = 0)
    //are ready to move,
    //otherwise they got hidden
    if (gen > 0)
    {
        rect.setHeight(0);
        rect.setWidth(0);
    }
    return rect;
}


/*  Here we define the class worm.
    A worm is composite: it has links.
    Each link has a rectangular shape
    and  its own color,
    which is defined by three numbers.
    The vector Mover[ ]
    keeps the information
    on how to move its links:
    which  link and in which order
    (in the order that they appear in
    the vector Mover[]), and
    in which direction.
    Each worm keeps record of
    the actual position of each
    one of its links,
    in the associated rectangle. */



private class Worm
{
    //Number of links
    int numberOfLinks = nLinks;
    //Number of movers = instructions to move
    int numberOfMovers = nLinks;
```

```
    //List mover  keeps movers
    List<Mover> Mover = new ArrayList<>();
    //The shape of each link
    List<Rectangle> Rect = new ArrayList<>();




    //Default initialization:
    //worms of the first generation are
    //ready to move. Others get hidden.
    private Worm(int gen )
    {
//Rects are created.
        // A color is assigned to each link
        // but different than white
        for(int i = 0; i < numberOfLinks; i++)
            Rect.add(createRectangle(gen));
        //Instructions to move at random
for(int m = 0; m < nMovers; m++)
        {
            //Select at random  which link to move:
            int l = r.nextInt(nLinks) ;
            //Select at random  which direction to move on
            Move  move
                    = w.new Move (vectMoves[r.nextInt(9)]);
            //Memorization
            Mover.add(w.new Mover(l,move));
        }
    }//End of constructor Worm(int  gen)

}//End of class Worm


//General setting
@Override public void start(Stage stage)
    {
        Scene scene = new Scene(p, xWin,yWin);
        stage.setScene(scene);
        stage.setTitle("Links move at random");
```

```
stage.show();
//Definition of possible moves
moveDef();
//Text to draw
genCount = new Text ("");
genCount.setLayoutX(40);genCount.setLayoutY(40);
p.getChildren().addAll(genCount);
nGenerationsST.setCycleCount(1);

//Many attempts are allowed
for(int i = 0; i < nGenerations; i++)
{
    voidOccupationMatrix();//Voidness
    gen = i;
    if (print2)
        System.out.println("\nGen = " + gen);
    //Generation number is drawn
    Timeline timeline = drawGen(gen);
    nGenerationsST.getChildren().addAll( timeline );
    //Worms are posited at the start line
    startLine(gen);
    //Worms express their information to move
    SequentialTransition oneGenST  =  express();
    nGenerationsST.getChildren().add(oneGenST);
    //Is some worm able to climb?
    assessFitness();
    //Worms get hidden
    ParallelTransition hideST  =  hideST();
    if (gen < nGenerations - 1 )
    {
        //Clear Worms as list and hide drawn worms
        Worms.clear();
        nGenerationsST.getChildren().add(hideST);
    }
    //Success?
    Integer genI = gen;
    if (bullseye)
    {
        //If climbing was successful, add a new link
         nLinks = nLinks+1;
```

```
                //List of generations with sucess
                record = record + "  " + genI.toString();
            }
        }
    //Film is played
    nGenerationsST.play();

    System.out.println("Successful generations = "
            + record);
    }


// To avoid overwriting of worms,
// an occupation record is kept:
// The screen is divided in cells 6x6.
// At the beginning every cell is empty=false.
// Occupancy is recorded in isOccupied[][],
// which is a boolean matrix.
// A boolean  variable can be
// true = occupied,  else false= not occupied.
private static void  voidOccupationMatrix()
{
   //System.out.println("occupationMatrix");
    for(int i = 0; i < 600; i++)
    for(int j = 0; j < 300; j++)
        isOccupied[i][j]= false;
}

//Nine basic moves = coordinates where to move in
static private void moveDef()
{
    final Move move0 =  w.new Move(0,1);
       vectMoves[0]   = move0 ;
    final Move move1 =  w.new Move(1,1);
       vectMoves[1]   = move1 ;
    final Move move2 =  w.new Move(1,0);
       vectMoves[2]   = move2 ;
    final Move move3 =  w.new Move(1,-1);
       vectMoves[3]   = move3 ;
    final Move move4 =  w.new Move(0,-1);
```

```
       vectMoves[4]   = move4 ;
    final Move move5 =  w.new Move(-1,-1);
       vectMoves[5]   = move5 ;
    final Move move6 =  w.new Move(-1,0);
       vectMoves[6]   = move6 ;
    final Move move7 =  w.new Move(-1,1);
       vectMoves[7]   = move7 ;
    final Move move8 =  w.new Move(0,0);
       vectMoves[8]   = move8 ;
}


// All worms are posited at the same initial row
// in vertical position.
static public void startLine(int gen)
{
    for (int W = 0; W < nWorms; W++)
    {
        int  x,y;
        //Auxiliary worm
        Worm   worm = w.new Worm(gen);


        //Each link is positioned
        for(int  link = 0; link < nLinks; link++)
        {
            //Horizontal position
            x = 30 + 150 * W;
            //Vertical position
            y = initialRow + height*link;

            //Position is updated
            worm.Rect.get(link).setX(x);
            worm.Rect.get(link).setY(y);

            //Link is shown
            SequentialTransition show
                        = showST(worm.Rect.get(link));
            //Link is booked
            p.getChildren().add(worm.Rect.get(link));
```

```
            allRects.add(worm.Rect.get(link));
            //Show transition is booked
            nGenerationsST.getChildren().add(show);
        }
        //worm is located at position W in list Worms
        Worms.add(worm);
    }

}


//Draws the generation number
private static Timeline drawGen(int gen)
{
    Integer genI = gen;
    String message =  "Generation  = " +
                    genI.toString();
    //A JavaFX event committed to print a text is generated
    EventHandler printText =
        (EventHandler<ActionEvent>)
            (ActionEvent t ) ->
                {
                    genCount.setText(message);
                };
    //Presentation of printText must be
    //synchronized with the overall film so,
    //duration d is specified
    KeyFrame keyFrame = new KeyFrame(d, printText);

    Timeline timeline = new Timeline();
    timeline.setCycleCount(1);
    //add the keyframe to the timeline
    timeline.getKeyFrames().add(keyFrame);

    return timeline;
}


//Is move m disruptive?
//(A worm is and must remain as a whole,
```

```
//nevertheless, some elasticity is allowed.)
private static boolean isDisruptive(int m, Worm worm )
{

    // A move is disruptive if the target address
    // is farther than certain distance
    // from the neighboring links.
    boolean isDisruptive = false;
    int  link;
    int  x,y,  x1,y1, x2,y2, x3,y3;
    double d1;

    double  d2;

    //Read the link to be moved
    link = worm.Mover.get(m).link;
    //Read the   address of the link;
    x  = (int) worm.Rect.get(link).getX();
    y  = (int) worm.Rect.get(link).getY();
    //The   move to be done is read
    Move move = w.new Move(worm.Mover.get(m).move);

    //The target address is calculated
    x1 = x + width*move.x;
    y1 = y -  width*move.y;

    //The addresses of the two neighboring links are read
    //The disruption is calculated
    //Case: intermediate link
    if ((0 < link) & (link < nLinks-1))
     {
      x2 = (int) worm.Rect.get(link-1).getX();

      y2 = (int) worm.Rect.get(link-1).getY();

      d1 =    pow(  ( pow( (x2 - x1),2)
                           +  pow((y2 - y1),2)),0.5);

      x3 = (int) worm.Rect.get(link+1).getX();
      y3 = (int) worm.Rect.get(link+1).getY();
```

```
      d2 =  pow(  (pow( (x3 - x1),2)
                           + pow((y3 - y1),2)),0.5);


      // Here we decide whether or not
      // a move is disruptive.
      //To augment elasticity, augment the threshold

      if ((d1 > 1.415*width) || (d2 > 1.415*width))
                  isDisruptive = true;

     }

    //First link
    if (link == 0)
    {
        x2 = (int) worm.Rect.get(1).getX();
        y2 = (int) worm.Rect.get(1).getY();
        d1 = pow( (pow( (x2 - x1),2) + pow((y2 - y1),2)),0.5);
        if( d1 > 1.1*width) isDisruptive = true;
     }
     //Last link
     if( link == nLinks-1 )
      {
        x2 = (int) worm.Rect.get(nLinks - 2).getX();
        y2 = (int) worm.Rect.get(nLinks - 2).getY();
        d2 = pow( (pow( (x2 - x1),2) + pow((y2 - y1),2)),0.5);
        if ( d2 > 1.1*width ) isDisruptive =   true;
       }
      return isDisruptive;
 }



//Is the move m of worm W not disruptive?
//If it is not, execute it.
private static SequentialTransition
                  exeIf( Worm worm, int m)
{
```

```
if (print1)
    System.out.println(" exeIf");
//Read the link to be moved
int link = worm.Mover.get(m).link;
//Read position
int x = (int) worm.Rect.get(link).getX();
int y = (int) worm.Rect.get(link).getY();

//The move to be done is read
Move  move;
move = worm.Mover.get(m).move;
//Move is interpreted
int byX = width* move.x;
int byY = -height* move.y;
//Target address is calculated:
int x1 = x + byX;
int y1 = y + byY;
if (x1 < 0 ) x1 = 0;
if (y1 < 0 ) y1 = 0;
if (print1)
    System.out.println(
        " move = " + m   + " link = " + link
        + " \n x = " + x + " y = " + y
        + " \nbyX = " + byX  + " byY = " + byY);


//Occupied cell
int i = (int)  x / width;
int j = (int) y / height;


 //Target cell
int k =(int) x1/width;
int l = (int) y1/height;


//Move is always executed
SequentialTransition a =      new   SequentialTransition();

TranslateTransition translateTransition =
```

```
     new TranslateTransition(d, worm.Rect.get(link));
//Jump is executed
translateTransition.setByX(byX);
translateTransition.setByY(byY);
//Number of repetitions
translateTransition.setCycleCount(1);
a.getChildren().add(translateTransition);



//Is the move disruptive?
boolean   isMoveDisruptive =  isDisruptive(m,worm);
if (print2)
{
    if (isMoveDisruptive)
        System.out.println("move is Disruptive");
    else
        System.out.println("move is not  Disruptive");
}
//Disruptive moves are undone
if  (((isOccupied[k][l])) || ((isMoveDisruptive)))
{
    byX = -byX;
    byY = -byY;
}
else //move is not disruptive -> it is not undone
{
    byX = 0;
    byY = 0;
    worm.Rect.get(link).setX(x1);
    worm.Rect.get(link).setY(y1);
    isOccupied[i][j]= false;
    isOccupied[k][l]= true;
}



//Move is possibly undone
TranslateTransition translateTransition2 =
```

```
          new TranslateTransition(d, worm.Rect.get(link));
    //Jump is programmed
    translateTransition2.setByX(byX);
    translateTransition2.setByY(byY);
    //Number of repetitions
    translateTransition2.setCycleCount(1);
    a.getChildren().add(translateTransition2);

return a;
 }



//Worms express their information to move.
public    SequentialTransition  express()
{
    //Overall process
    SequentialTransition expressST
                    = new SequentialTransition();
     expressST.setCycleCount(1);
     //Worms advance nSteps steps
    for (int i = 0; i < nSteps; i++)
    {
        //Worms advance one step:
        //To move all links, each worm must execute
        //instructions to move
        SequentialTransition oneStepST
                        = new SequentialTransition();
        oneStepST.setCycleCount(1);
        //Information is read and decoded
        for (int m = 0; m < nMovers; m++)
        {
            if (print1)
                System.out.println("nMovers = " + nMovers);
            //All worms execute move m
            ParallelTransition moveMPT
                        = new ParallelTransition();
            moveMPT.setCycleCount(1);
            for (Worm worm: Worms)
            {
```

```
                //A translation of some link
                //is programmed for worm W
                //according to move m.
                //If it is not disruptive,
                //it is executed
                SequentialTransition tT = exeIf(worm,m);
                tT.setCycleCount(1);
                //Translation is booked
                moveMPT.getChildren().add(tT);
            }
            //The ensamble of translations is booked
            oneStepST.getChildren().add(moveMPT);
        }
    //The movement of worms in one step is booked
    expressST.getChildren().add(oneStepST);
    }
    return expressST;
}



    //A hidden rect gets shown
    private static
        SequentialTransition showST(Rectangle rect)
    {
        SequentialTransition s = new SequentialTransition();
        Duration d2 =   Duration.millis(1);
        //Rect is enlarged
        KeyValue keyValue3 =
            new KeyValue(rect.widthProperty(),width,
                Interpolator.LINEAR);
        KeyValue keyValue4 =
            new KeyValue(rect.heightProperty(),height,
                Interpolator.LINEAR);
        //KeyValues are  inserted into a timeline
        Timeline b = new Timeline(
            new KeyFrame(d2,keyValue3,keyValue4 ));
        //Timeline is inserted into a transition
        s.getChildren().addAll(b);
```

```
        return s;
    }




    //Worms are hidden after their performance.
    private static ParallelTransition  hideST()
    {

        ParallelTransition hidePT
                    = new ParallelTransition();
        hidePT.setCycleCount(1);
        Duration d2 =   Duration.millis(1);
        //Hide all rects
        for(Rectangle rect: allRects)
        {
           //Rects are made unvisible
            KeyValue keyValue =
                    new KeyValue(rect
                            .visibleProperty(), false
                            ,Interpolator.LINEAR);
            Timeline b = new Timeline(
                    new KeyFrame(d2,keyValue));
            hidePT.getChildren().add(b);
        }

        return hidePT;
    }



// Every worm is assigned a fitness
//The nearer to the top, the fitter.
//If climbing has been significative,
//the complexity of the game rises up one level:
//a true bullseye is reported (to add
//next a new link to all worms).
```

```
private static void assessFitness()
{
    int yOfTail;
    double yOfHead;

    bullseye  = false;
    for(int W = 0; W <  nWorms; W++)
    {
        // We get the y-coordinate of the tail
        yOfTail = (int) Worms.get(W).Rect.get(nLinks-1).getY();
        yOfHead =  Worms.get(W).Rect.get(0).getY();
        //Climbing up is worth  a point
        Fitness[W] = 100*(yWin - yOfTail);
        if(print1)
            System.out.println("Fitness of worm "
                    + W + " = " + Fitness[W]
                    + " head = " + yOfHead);

        //if (headPosition < initialRow - width*nLinks)
        if (yOfHead < initialRow - 2 * height)
        {
            bullseye = true;
            if (print1)
            System.out.println( "There was advance W = " + W );
        }
    }
}



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


} //End of main class L61 WormsRandJFX8
```

62 ***Exercise.*** *Run the program and play with the code.*

63 ***Challenge.*** *The task of showing and hiding objects can be dealt with color, width and height, scale or visibility. Play with them to choose the one you most like.*

4.4 Movement by evolutionary learning

We are ready now to write a JavaFX program in which evolution functions as a teacher. Evolution is based on selection, the differential reproduction of individuals. There are infinitely many diverse types of selection and at this time we choose **positive proportional selection**, according to which the probability of reproduction of any individual is proportional to fitness.

64 ***Positive proportional selection: an element is chosen for reproduction in proportion to its fitness. Test this program carefully: if it is wrong, everything from now on is also wrong.***

```
/*
Program L64 PositiveSelection.
An element is chosen for reproduction
according to its fitness.
Based on Program E153 whateverD
 */

package positiveselection;




import java.util.Random;

public class PositiveSelection
{

    //Vector of fitnesses in arbitrary units
    private static final int Fitness[] =  {3, 1, 6, 1, 4};
    //Number of individuals
    private static final  int nWorms = Fitness.length;
    //Cumulative distribution
```

```
    private static final int cumulate[] = new int[nWorms];
    // Counters are in count
    private static final int count[] = new int[nWorms];
    //Number of trials
    private static final int nTrials = 10000;
    //printing flag
    private static final boolean print = false;
    // Turn on of the random generator
    static Random r  = new Random();

public static void main(String[] args)
{
    //Initialization
    for (int i=0; i< nWorms; i++)
        count[i]=0;

    /*
    for (int i=0; i< nWorms; i++)
      Fitness[i]=r.nextInt(10);
          */

    cumulate[0] = Fitness[0];
    for (int i = 1; i < nWorms; i++)
        cumulate[i]= cumulate[i-1] + Fitness[i];



    System.out.println( "i \tfitness cumulate ");
    for (int i = 0; i < nWorms; i++)
        System.out.println( i + "\t" + Fitness[i] + " \t "
                + cumulate[i]);


    //Maximal value
    int max = cumulate[nWorms-1];

    //System.out.println("Generated numbers: ");
    for (int i = 0; i < nTrials;i++)
    {
        //A random integer less than max is generated
```

```
        //with a uniform distribution
        int n = r.nextInt(max);
        if (print)
            System.out.println("n =  " + n);
        //The uniform distribution is transformed according
        //to the cumulate distribution
        if (n < cumulate[0])
        {
            if (print)
                System.out.println(
                        "chosen individual  = 0 ");
            count[0]++;
        }
        for (int j=0; j < nWorms-1;j++)
        {

            int l = j+1;
            if ( ( cumulate[j] <= n) & (n < cumulate[l]) )
            {
                if (print)
                    System.out.println(
                        "Chosen individual = " + l);
                count[l]++;
            }
        }

    }

   //Output
    System.out.println("\nEvents and their observed "
            + "and expected frequencies: ");
    int sum1 = 0;
    int sum2 = 0;
    for (int i=0; i<nWorms;i++)
    {
        System.out.println(i + "\t" + count[i] + "\t"
                +  Fitness[i]*nTrials/cumulate[nWorms-1]);
        sum1 = sum1 + count[i];
        sum2 = sum2 + Fitness[i]*nTrials/cumulate[nWorms-1];
```

```
    }
    System.out.println("Totals  " + sum1 + "\t"
                +  sum2);
  }
}//End of main class L64 PositiveSelection
```

65 ***Exercise.*** *Implement* ***negative proportional selection*** *according to which the probability of death before reproduction is the higher the less is the fitness of the individual. A given numbers of elements are purged but survivors are subjected to sampling with a uniform distribution to form the next generation. Compare your answer with that given by Program G51 syntax8 of Vol VI.*

Let us now propose a JavaFX version of our worms.

66 ***Evolution teaches worms to climb the screen. JavaFX code.***

```
//Program L66 WormsEvoJFX
//A population of worms is programmed
//by evolution to
//climb the screen.
//All moves are executed
//but  nondisruptive moves are
//not undone.
//Each generations has its own
//population of worms.
//All worms have two links at the start.
//One link is added when some worm is shown
//to climb the screen.
//Program is written in JavaFX.


package ejvolu12p;

import static java.lang.Math.pow;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
```

```
import javafx.animation.Interpolator;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.ParallelTransition;
import javafx.animation.SequentialTransition;
import javafx.animation.Timeline;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Text;
import javafx.stage.Stage;
import javafx.util.Duration;



public class WormsEvoJFX extends Application
{
    private static WormsEvoJFX w = new WormsEvoJFX();
    //Overall process
    private static  SequentialTransition evolutionST
                    = new SequentialTransition();
    //Number of repetitions or generations
    private static int nGenerations = 50;
    //Actual generation
    private static int gen = 0;
    //Duration of graphical events
    private static final Duration d = Duration.millis(100);
    //Group of spacelike JavaFX objects
    private static   Group p = new Group();
    //Number of steps given for each try
    int nSteps = 8;
    //Number of worms
    private static  int nWorms = 8;
    //Worms are kept in a list
```

```
private static List<Worm> Worms = new ArrayList<>();
//Helping files
private static List<Worm> WormsTwo = new ArrayList<>();
private static Worm[] OldWorms ;
//Number of links of each worm
private static  int nLinks = 2;
//Initial position  of the head
private static int initialRow = 280;
//Generator of pseudo random numbers
private static final Random rd = new Random();
//Each worm has nMovers movers,
//instructions to move.
private static int  nMovers = 5;
//Width of the worm = width of each link
private static int width = 6;
//Height of each link
private static int height = 6;
//Array with 9 possible moves,
//9 possible directions of movement
private static Move  vectMoves[ ] = new Move[9];
private static final boolean print1 = false;
private static final boolean print2 = false;
private static final boolean printReproduction = true;

//Text to be drawn
private static  Text genCount = new Text ("");
//Two links cannot occupy the same place
static boolean isOccupied[][]  = new boolean[600][300];
//Is a worm capable of climbing the screen?
private static  int Fitness[] =  new int[21];
//Was climbing good enough?
private static boolean bullseye  = false;
//List of generations at which success was made
private static String record = "";
//Measures of the screen
private static int xWin = 1260;
private static int yWin = 850;
//Grouping of links in order to ease erasing
private static List<Rectangle> allRects
                        = new ArrayList<>();
```

```
    //The number of point mutations of the offspring
    private static int numbPointMutations = 5;




    /* Definition of inner classes.
    *
    The Move class:
    A Move says in which direction
    a link of the worm must be moved.
    */

 private  class Move
 {
    int x;
    int y;


    //Default initialization
    Move()
    {
        x= 0;
        y= 0;
    }

    // Initialize with two coordinates
    Move(int a, int b)
    {
        x= a;
        y= b;
    }

    // Copy a move
    Move(Move move)
    {
        x= move.x;
        y= move.y;
    }
 }//End of class Move
```

```
// Definition of the class Mover:
// A Mover keeps two types of information:
// a) the link of the worm to be moved and
// b) the direction or move to be chosen.
//
private   class Mover
{
   int link ;
   Move   move;

   //  Movers can acquired information
   //  in three ways, they are  overloaded.
   //  Form 1: by  numbers
   private Mover(int l, int mx, int my)
    {
       link= l;
       move.x = mx;
       move.y = my;
    }

   // Form 2, by structured data: link + move
   private Mover(int l, Move m)
   {
       link= l;
       move = new Move(m);
   }

   // Form 3, by structured data: mover
   private Mover(Mover mover)
   {
       link= mover.link;
       move =  new Move(mover.move);
   }
 }//End of class Mover


//A random color is synthesized
private static Color randomColor()
```

```
 {
    //Color is defined as a random
    //mix of red, green and blue
    int r,g,b;
    r = rd.nextInt(226);
    g = rd.nextInt(226);
    b = rd.nextInt(226);
    //The color is prepared
    Color c = Color.rgb(r,g,b);
    return c;
 }


//Rects that belong in the worms of the first generation
 //are visible. Others are hidden.
private static  Rectangle
           createRectangle(int gen)
{
    //Upper left corner at (0,0)
    Rectangle rect = new Rectangle(0,0,width,height);

    //The color is prepared
    Color c = randomColor();
    //The rect is painted
    rect.setFill(c);
    //Rects of the first generation (gen = 0)
    //are visible, otherwise they are not
    if (gen > 0)
    {
        rect.setHeight(0);
        rect.setWidth(0);
    }
    return rect;
}


/*  Here we define the class worm.
    A worm is composite: it has links.
    Each link has a rectangular shape
    and  its own color.
```

```
    The vector Mover[ ]
    keeps the information
    on how to move its links:
    which  link and in which order
    (in the order that they appear in
    the vector Mover[]), and
    in which direction.
    Each worm keeps record of
    the actual position of each
    one of its links,
    in the associated rectangle. */



private class Worm
{
    //Number of links
    int numberOfLinks = nLinks;
    //Number of movers = instructions to move
    int numberOfMovers = nLinks;
    //List mover  keeps movers
    List<Mover> Mover = new ArrayList<>();
    //The shape of each link
    List<Rectangle> Rect = new ArrayList<>();

    //Default initialization:
    //worms of the first generation are
    //visible. Others not.
    private Worm()
    {
//Visible rects are created.
        // A color is assigned to each link
        // but different than white
        for(int i = 0; i < numberOfLinks; i++)
            Rect.add(createRectangle(0));
        //Instructions to move at random
for(int m = 0; m < nMovers; m++)
        {
            //Select at random  which link to move:
            int l = rd.nextInt(nLinks) ;
```

```
            //Select at random  which direction to move on
            Move  move
                    = w.new Move (vectMoves[rd.nextInt(9)]);
            //Memorization
            Mover.add(w.new Mover(l,move));

        }
    }//End of constructor Worm()

    //Worm W is read from the old population
    public Worm(int W)
    {
        numberOfLinks = nLinks;
        numberOfMovers = nMovers;
        //Fake initialization
        for(int i = 0; i < numberOfLinks; i++)
               Rect.add(createRectangle(1));
        for(int m = 0; m < nMovers; m++)
        {
            //Select at random  which link to move:
            int l = rd.nextInt(nLinks) ;
            //Select at random  which direction to move on
            Move  move
                    = w.new Move (vectMoves[rd.nextInt(9)]);
            //Memorization
            Mover.add(w.new Mover(l,move));
        }
        //Read true color
        for(int i = 0; i <  OldWorms[W].numberOfLinks; i++)
        {
           Rect.get(i).setFill( OldWorms[W]
                                .Rect.get(i).getFill());
        }
        //Read true movers
        for(int i = 0; i < OldWorms[W].numberOfMovers; i++)
           Mover.set(i, OldWorms[W].Mover.get(i));
    }

}//End of class Worm
```

```
//General setting
@Override public void start(Stage stage)
    {
        Scene scene = new Scene(p, xWin,yWin);
        stage.setScene(scene);
        stage.setTitle("Evolution teaches worms "
                       + "to climb the screen");
        stage.show();
        //Definition of possible moves
        moveDef();
        //Text to draw
        genCount = new Text ("");
        genCount.setLayoutX(40);genCount.setLayoutY(40);
        p.getChildren().addAll(genCount);
        evolutionST.setCycleCount(1);

        //A sequential loop
        for(int i = 0; i < nGenerations; i++)
        {
            //Record of occupancy = void
            voidOccupationMatrix();
            //Generation
            gen = i;
            if (print2)
                System.out.println("\nGen = " + gen);
            //Generation number is drawn
            Timeline timeline = drawGen(gen);
            evolutionST.getChildren().addAll( timeline );
            //Worms are created and posited at the start line
            startLine();
            //Worms express their information to move
            SequentialTransition oneGenST = oneGeneration();
            evolutionST.getChildren().add(oneGenST);
            //Drawn worms get hidden
            ParallelTransition hideST  =  hideST();
```

```
            if (gen < nGenerations - 1 )
            {
                //Helping array
                Worm[] file1 = new Worm[50];
                //Information is recorded in an array
                OldWorms = Worms.toArray(file1);
                //Destroy everything
                //for the sake of a new beginning
                Worms.clear();
                WormsTwo.clear();
                evolutionST.getChildren().add(hideST);
            }
            //Success?
            Integer genI = gen;
            if (bullseye)
            {
                //If climbing was successful, add a new link
                 nLinks = nLinks+1;
                 //List of generations with sucess
                record = record + "  " + genI.toString();
            }
        }
    //Movie is played
    evolutionST.play();

    System.out.println("Successful gens = "
            + record);
    }


// To avoid overwriting of worms,
// an occupation record is kept:
// The screen is divided in cells width x height.
// At the beginning every cell is empty=false.
// Occupancy is recorded in isOccupied[][],
// which is a boolean matrix.
// A boolean  variable can be
// true= occupied,  else false= not occupied.
private static void  voidOccupationMatrix()
{
```

```
   //System.out.println("occupationMatrix");
    for(int i = 0; i < 600; i++)
    for(int j = 0; j < 300; j++)
        isOccupied[i][j]= false;
}

//Nine basic moves = irections where to move in
static private void moveDef()
{
    final Move move0 =  w.new Move(0,1);
       vectMoves[0]   = move0 ;
    final Move move1 =  w.new Move(1,1);
       vectMoves[1]   = move1 ;
    final Move move2 =  w.new Move(1,0);
       vectMoves[2]   = move2 ;
    final Move move3 =  w.new Move(1,-1);
       vectMoves[3]   = move3 ;
    final Move move4 =  w.new Move(0,-1);
       vectMoves[4]   = move4 ;
    final Move move5 =  w.new Move(-1,-1);
       vectMoves[5]   = move5 ;
    final Move move6 =  w.new Move(-1,0);
       vectMoves[6]   = move6 ;
    final Move move7 =  w.new Move(-1,1);
       vectMoves[7]   = move7 ;
    final Move move8 =  w.new Move(0,0);
       vectMoves[8]   = move8 ;
}


//All worms are posited at the same initial row
//in vertical position.
static public void startLine()
{
    for (int W = 0; W < nWorms; W++)
    {
        int  x,y;
        //Auxiliary worm
         Worm   worm;
```

```
         if (gen == 0)//Create worm
             worm = w.new Worm();
         else //Copy worm W from old generation
             worm = w.new Worm(W);

        //Each link is positioned
        for(int  link = 0; link < nLinks; link++)
        {
            //Horizontal position
            x = 140 + 140 * W;

            //Vertical position
            y = initialRow + height*link;

            //Position is updated
            worm.Rect.get(link).setX(x);
            worm.Rect.get(link).setY(y);

            //Link is shown
            ParallelTransition show
                        = showPT(worm.Rect.get(link));
            //Link is booked
            p.getChildren().add(worm.Rect.get(link));
            //Link is grouped
            allRects.add(worm.Rect.get(link));
            //Show-transition is booked
            evolutionST.getChildren().add(show);
        }
        //worm is located at position W in lists Worms
        Worms.add(worm);

        if (print1)
            System.out.println("W = " + W + " nLinks = "
                + Worms.get(W).numberOfLinks  );
    }

}
```

```
//Draw the generation number
private static Timeline drawGen(int gen)
{
    Integer genI = gen;
    String message =  "Generation  = " +
                   genI.toString();
    //A JavaFX event committed to print a text is generated
    EventHandler printText =
        (EventHandler<ActionEvent>)
            (ActionEvent t ) ->
                {
                    genCount.setText(message);
                };
    //Presentation of printText must be
    //synchronized with the overall movie so,
    //duration d is specified
    KeyFrame keyFrame = new KeyFrame(d, printText);

    Timeline timeline = new Timeline();
    timeline.setCycleCount(1);
    //add the keyframe to the timeline
    timeline.getKeyFrames().add(keyFrame);

    return timeline;
}



//Is move m disruptive for the worm?
//(A worm is and must remain as a whole,
//nevertheless, some elasticity is allowed.)
private static boolean isDisruptive(int m, Worm worm )
{
    //A move is disruptive if the target address
    //is farther than certain distance
    //from the neighboring links.
    boolean isDisruptive = false;
    int  link;
```

```
int  x,y,  x1,y1, x2,y2, x3,y3;
double d1;

double  d2;

//Read the link to be moved
link = worm.Mover.get(m).link;
//Read the   address of the link;
x  = (int) worm.Rect.get(link).getX();
y  = (int) worm.Rect.get(link).getY();
//The   move to be done is read
Move move = w.new Move(worm.Mover.get(m).move);

//The target address is calculated
x1 = x + width*move.x;
y1 = y -  width*move.y;

//The addresses of the two neighboring links are read
//The disruption is calculated
//Case: intermediate link
if ((0 < link) & (link < nLinks-1))
{
    x2 = (int) worm.Rect.get(link-1).getX();

    y2 = (int) worm.Rect.get(link-1).getY();

     d1 =    pow(  ( pow( (x2 - x1),2)
                        +  pow((y2 - y1),2)),0.5);

    x3 = (int) worm.Rect.get(link+1).getX();
    y3 = (int) worm.Rect.get(link+1).getY();
    d2 =  pow(  (pow( (x3 - x1),2)
                        + pow((y3 - y1),2)),0.5);


    // Here we decide whether or not
    // a move is disruptive.
    //To augment elasticity, augment the threshold

    if ((d1 > 1.415*width) || (d2 > 1.415*width))
```

```
                    isDisruptive = true;

     }

    //First link
    if (link == 0)
    {
        x2 = (int) worm.Rect.get(1).getX();
        y2 = (int) worm.Rect.get(1).getY();
        d1 = pow( (pow( (x2 - x1),2) + pow((y2 - y1),2)),0.5);
        if( d1 > 1.1*width) isDisruptive = true;
     }
     //Last link
    if( link == nLinks-1 )
    {
        x2 = (int) worm.Rect.get(nLinks - 2).getX();
        y2 = (int) worm.Rect.get(nLinks - 2).getY();
        d2 = pow( (pow( (x2 - x1),2) + pow((y2 - y1),2)),0.5);
        if ( d2 > 1.1*width ) isDisruptive =   true;
    }
    return isDisruptive;
 }



//Execute move m for worm W.
//If it is  disruptive, undo it.
private static SequentialTransition
                    exeIf( Worm worm, int m)
{
    if (print1)
        System.out.println(" exeIf");
    //Reads the link to be moved
    int link = worm.Mover.get(m).link;
    //Reads position
    int x = (int) worm.Rect.get(link).getX();
    int y = (int) worm.Rect.get(link).getY();

    //The move to be done is read
```

```
Move  move;
move = worm.Mover.get(m).move;
//Move is interpreted
int byX = width* move.x;
int byY = -height* move.y;
//Target address is calculated:
int x1 = x + byX;
int y1 = y + byY;
if (x1 < 0 ) x1 = 0;
if (y1 < 0 ) y1 = 0;
if (print1)
    System.out.println(
        " move = " + m   + " link = " + link
        + " \n x = " + x + " y = " + y
        + " \nbyX = " + byX  + " byY = " + byY);


//Occupied cell
int i = (int)  x / width;
int j = (int) y / height;


 //Target cell
int k =(int) x1/width;
int l = (int) y1/height;


//Move is always executed
SequentialTransition a = new SequentialTransition();

TranslateTransition translateTransition =
     new TranslateTransition(d, worm.Rect.get(link));
//Jump is executed
translateTransition.setByX(byX);
translateTransition.setByY(byY);
//Number of repetitions
translateTransition.setCycleCount(1);
a.getChildren().add(translateTransition);
```

```
    //Is the move disruptive?
    boolean   isMoveDisruptive =  isDisruptive(m,worm);
    if (print1)
    {
        if (isMoveDisruptive)
            System.out.println("move is Disruptive");
        else
            System.out.println("move is not  Disruptive");
    }
    //Disruptive moves are undone
    if  (((isOccupied[k][l])) || ((isMoveDisruptive)))
    {
        byX = -byX;
        byY = -byY;
    }
    else //move is not disruptive -> it is not undone
    {
        byX = 0;
        byY = 0;
        worm.Rect.get(link).setX(x1);
        worm.Rect.get(link).setY(y1);
        isOccupied[i][j]= false;
        isOccupied[k][l]= true;
    }




    //Move is possibly undone
    TranslateTransition translateTransition2 =
         new TranslateTransition(d, worm.Rect.get(link));
    //Jump is programmed
    translateTransition2.setByX(byX);
    translateTransition2.setByY(byY);
    //Number of repetitions
    translateTransition2.setCycleCount(1);
    a.getChildren().add(translateTransition2);

return a;
```

```
 }


// Every worm is assigned a fitness
//The nearer to the top, the fitter.
//If climbing has been significative,
//the complexity of the game rises up one level:
//a true bullseye is reported (to add
//next a new link to all worms).
private static void assessFitness()
{
    int yOfTail;
    double yOfHead;

    bullseye  = false;
    for(int W = 0; W <  nWorms; W++)
    {
        // We get the y-coordinate of the tail
        yOfTail = (int) Worms.get(W)
                            .Rect.get(nLinks-1).getY();
        yOfHead =  Worms.get(W).Rect.get(0).getY();
        //Fitness measures how much the head has  been lifted
        Fitness[W] = (yWin - yOfTail);
        if(print1)
            System.out.println("Fitness of worm "
                    + W + " = " + Fitness[W]
                    + " head = " + yOfHead);
        //If worm W is good for climbing
        //its head must be high enough
        if (yOfHead < initialRow - 2 * height)
        {
            //Worms that climb are rewarded
            Fitness[W] = 100 * Fitness[W];
            bullseye = true;
            if (printReproduction)
            System.out.println(
                    "There was advance W = " + W );
        }
    }
}
```

```
//Worms express their information to move.
public SequentialTransition  express()
{
    //Overall process
    SequentialTransition expressST
                   = new SequentialTransition();
     expressST.setCycleCount(1);
     //Worms advance nSteps steps
    for (int i = 0; i < nSteps; i++)
    {
        //Worms advance one step:
        //To move all links, each worm must execute
        //instructions to move
        SequentialTransition oneStepST
                        = new SequentialTransition();
        oneStepST.setCycleCount(1);
        //Information to move is read and decoded
        for (int m = 0; m < nMovers; m++)
        {
            if (print1)
                System.out.println("nMovers = " + nMovers);
            //All worms execute move m
            ParallelTransition moveMPT
                        = new ParallelTransition();
            moveMPT.setCycleCount(1);
            //This is a parallel loop for a list
            for (Worm worm: Worms)
            {
                //A translation of some link
                //is programmed for worm W
                //according to move m.
                //If it is not disruptive,
                //it is executed
                SequentialTransition tT = exeIf(worm,m);
                tT.setCycleCount(1);
                //Translation is booked
                moveMPT.getChildren().add(tT);
            }
```

```
            //The ensamble of translations is booked
            oneStepST.getChildren().add(moveMPT);
        }
    //The movement of worms in one step is booked
    expressST.getChildren().add(oneStepST);
    }
    return expressST;
}



    //A hidden rect gets shown
    private static
        ParallelTransition showPT(Rectangle rect)
    {
        ParallelTransition s = new ParallelTransition();
        Duration d2 =   Duration.millis(1);
        //Rect is enlarged
        KeyValue keyValue3 =
            new KeyValue(rect.widthProperty(),width,
                Interpolator.LINEAR);
        KeyValue keyValue4 =
            new KeyValue(rect.heightProperty(),height,
                Interpolator.LINEAR);
        //KeyValues are  inserted into a timeline
        Timeline b = new Timeline(
            new KeyFrame(d2,keyValue3,keyValue4 ));
        //Timeline is inserted into a transition
        s.getChildren().addAll(b);

        return s;
    }



    //Worms are hidden after their performance.
    private static ParallelTransition  hideST()
```

```
{

    ParallelTransition hidePT
                = new ParallelTransition();
    hidePT.setCycleCount(1);
    Duration d2 =   Duration.millis(1);
    //Hide all rects
    for(Rectangle rect: allRects)
    {
       //Rects are made unvisible
        KeyValue keyValue =
                new KeyValue(rect
                        .visibleProperty(), false
                        ,Interpolator.LINEAR);
        Timeline b = new Timeline(
                new KeyFrame(d2,keyValue));
        hidePT.getChildren().add(b);
    }

    return hidePT;
}




//====================================
//=========Evolutionary part==========
//====================================


//One generation = expression + assessement
//+ reproduction, recombination and mutation.
public    SequentialTransition  oneGeneration()
{
    SequentialTransition oneGenST
            = new SequentialTransition();
    //Worms express their information to move
    SequentialTransition expressST = express();
    oneGenST.getChildren().addAll(expressST);
```

```
        //Fitness is measured
        assessFitness();
        //Reproduction is proportional to fitness
        reproduction();
        //Population is replaced by a recombinat one
        recombination();
        //Information to move suffers mutation
        mutation();

        return oneGenST;
    }




/*  To generate a worm of the next generation,
    an individual is sampled at random
    from the old population
    and cloned
    with a probability that is
    proportional to its fitness.
    */
private static void reproduction()
{
    if (printReproduction)
        System.out.println( "\nGeneration = " + gen);
    //Cumulative distribution
    Double cumulate[] = new Double[nWorms];
    // Counters
    int count[] = new int[nWorms];
    //Number of trials
    int nTrials = nWorms;

    //Initialization
    WormsTwo.clear();
    WormsTwo.addAll(Worms);
    for (int i=0; i< nWorms; i++)
        count[i]=0;
```

```
//Cumulative distribution
cumulate[0] = (double) Fitness[0];
for (int i=1; i< nWorms; i++)
    cumulate[i]= cumulate[i-1] + Fitness[i];

//Report
if (printReproduction)
{
    System.out.println( "\ni \tfitness cumulate ");

for (int i = 0; i < nWorms; i++)
    System.out.println( i + "\t" + Fitness[i] + " \t "
            + cumulate[i]);
}

//Maximal value
double max = cumulate[nWorms-1];
//Counter of chosen worms
int W = 0;
//System.out.println("Generated numbers: ");
for (int i = 0; i < nTrials; i++)
{
    //A random integer less than max is generated
    //with a uniform distribution
    int n = rd.nextInt((int) max);

    //System.out.println("n =  " + n);
    //Probability of cloning is made
    //in proportion to fitness.
    //Is worm 0 to be chosen?
    if (n < cumulate[0])
    {
        /*
        if (printReproduction)
            System.out.println( "chosen individual  = 0 ");
        */
        WormsTwo.set(W, Worms.get(0));
        count[0]++;
        W++;
    }
```

```
        //Is worm l to be chosen?
        for (int l = 1; l < nWorms; l++)
        {

            if ( ( cumulate[l-1] <= n) & (n < cumulate[l]) )
            {
                /*
                if (printReproduction)
                    System.out.println(
                        "Chosen individual = " + l);
                */
                count[l]++;
                WormsTwo.set(W, Worms.get(l));
                W++;
            }
        }

    }
    if (printReproduction)
    {
        System.out.println("\nEvents and their observed "
            + "and expected frequencies: ");
        for (int i=0; i<nWorms;i++)
            System.out.println(i + "\t" + count[i] + "\t"
                +  Fitness[i]*nTrials/cumulate[nWorms-1]);
    }

    //WormsTwo is copied into Worms
    Worms.clear();
    Worms.addAll(WormsTwo);

}


//Recombination is implemented
private static void recombination()
{

    //Every worm is replaced by a recombinant
    for(int W = 0; W < nWorms; W++ )
```

```
    {
        //A  binary recombinant operator
        //is assembled at random.

        int rec[] = new int[210];
        for (int j = 0; j< nMovers; j++)
  rec[j]= rd.nextInt(2);
//Another worm is chosen at random for recombination
        int m = rd.nextInt(nWorms);
   //Basal initialization of the recombinant
        Worm  worm = Worms.get(W);
for(int j = 0; j< nMovers; j++)
        {
            // Recombination is executed.
            //For 0, take from the partner worm m
            //For 1, take from worm W.
            if (rec[j] == 0)
            worm.Mover.set(j, Worms.get(m).Mover.get(j));
        }

        //Recombinant is memorized
        WormsTwo.set(W, worm);
        if(print2)
        {
            System.out.println("Old move.x = " + Worms.get(W).Mover.ge

            System.out.println("New Rec move.x = " + worm.Mover.get(0)
        }

    }
    //Old population is replaced by the recombinated one
    Worms.clear();
    Worms.addAll(WormsTwo);
}




//Blind mutations of infomation to move
```

```
private static void moverMutation()
{
    int   m, n;
    //Let us try out blind  mutations
    //in specific Movers,
    //in the  link  to be moved
    //and in the direction of movement.
    for(int W = 0; W < nWorms; W++)
    {
        //Each offspring receives
        //numbPointMutation mutations
        for(int k = 0; k< numbPointMutations; k++)
        {
            //Select at random  which mover to mutate:
            m =  rd.nextInt(nMovers );
            //Select at random which link  to move:
            n = rd.nextInt( nLinks) ;
            if(print2)
                System.out.println("Old move.x = "
                     + Worms.get(W).Mover.get(m).move.x);
            //Select at random which direction to move on.
            Move move = vectMoves[rd.nextInt(9) ];
            //Mover m is assembled from link   + move
            Worms.get(W).Mover.set(m, w.new Mover(n,move));
            if(print2)
                System.out.println("New Mutation move.x = "
                    + Worms.get(W).Mover.get(m).move.x);
            //Color of link n is changed
            //(possibly various times)
            Worms.get(W).Rect.get(n).setFill(randomColor());
        }
    }
}


    //Information to move is changed
    private static void mutation(   )
    {
        moverMutation();
    }
```

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


} //End of main class L66 WormsEvoJFX
```

67 ***Exercise.*** *Run the program and play with the code.*

68 ***Challenge.*** *In the previous program the old population of worms is recorded in an array and then its is completely destroyed to begin a new generation with a recently created population. Use the alternate strategy of repositioning old worms at the startline for each new generation to develop the corresponding algorithm.*

69 ***Exercise.*** *Furnish evolution with one or more hints and test next how good they are.**Answer*

70 ***Challenge.*** *We proved in program L48 WormsHDJFX that the problem of making worms to move upwards is a very easy one for a human being and that it can be solved using as many moves as links there are in the worm. All one needs to do is to move links northwest beginning with the tail and moving subsequent links in ascendant order. So, by looking at previous code, we can say in hindsight: evolution can be implemented as bad as desired. The problem is then to invent an implementation that makes honor to evolution. Would the Reader create one?*

4.5 Worms leave fossils

We know enough about JavaFX to include fossils in our simulation.

71 ***Exercise. Include a fossil study in our simulation in which evolution teaches worms to climb the screen. Use your program to prove else reject the proposition that fossils exhibit a variability in complexity and perfection. Explicitly, there must appear fossils with different number of links, a fact that denotes diversity of complexity. Nevertheless, one must find that a significant part of reconstructed fossils can not climb the screen so, there is a clear track of diversity of perfection. If one adds the idea that more links corresponds with later times, one concludes that the fossil study evidences an evolution of both complexity and perfection.*** *Answer*

4.6 Conclusion

Evolution is obviously false: on one hand, nature around us is very complex and extremely perfect and these characteristics can also be ascribed to life in the past as evidenced by the fossil record. On the other, this is contrary to what one expects were evolution the explanation of our existence: evolution must be plagued at every stage with imperfection and this must be evidenced as in the fossil record as in present populations. More detailedly, our simulation of worms that are being taught by evolution to climb the screen supports next claims:

1. Evolution can create a trend towards higher and higher complexity: i.e., more and more links.

2. The evolution of complexity is inescapably reflected in the fossil record through a variability in the number of links, some fossils have 2 links, others 3 or more.

3. There are evolutionary environments in which mediocrity is the trade mark of the evolution of complexity from the stand of the fossil record: it is entirely composed of reconstructed worms that in general are unable to climb the screen although they can somehow kick and revolve or move but in the wrong direction.

4. Nothing like this is found in nature, whose perfection and complexity are from every every point of view overwhelming. Therefore, species can not be explained in general by natural evolution.

Chapter 5

User interfaces in JavaFX

`The User commands what to do`

72 ***Objective.*** *Any simulation has a rich set of parameters. To change them, we usually modify the code directly. This is not the best option for a program that is intended to be used by a large audience. The reason is that it is easy to damage the code causing the need of beginning anew with a fresh version of the program. Our purpose for this chapter is to show how to use the facility provided by JavaFX to develop a Graphic User Interface, a tool that allows to set parameters without the risk of causing damage to the code.*

5.1 Getting started

A pdf with the basic material about management of UI (User interfaces) in JavaFX can be downloaded from (Oracle [6], 2014) (upper right corner)

A list with all types of controls can be found on Part II, page 17. Using hyperlinks one can go directly to desired control. The easiest form of learning how to manage these tools is to run the desired example, whose code can be found at the end of the pdf (part IV, pag 341). One opens a project with the name of the chosen application and replaces the content of the opening page with the code. Some housekeeping would be necessary, say, to erase navigating signals of the pdf or to accommodate some texts. Once the program has been run as it is, one proceeds to personalization.

5.2 Examples

Here we provide some examples of personalization such as it is needed for our project about worms.

73 *Welcome label.* *The next code presents some information to the user.*

```
/*Program L73 Welcome
 * A welcome message is presented
 * inside a button.
 */

package ejvolu12p;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;


public class Welcome extends Application
{

    @Override
    public void start(Stage primaryStage)
    {
        //Button is declared
        Button btn = new Button();
        //Button is initialized
        btn.setText("A population of worms is programmed\n"
            + "by evolution to\n"
            + "climb the screen.\n"
            + "All moves are executed\n"
            + "but disruptive moves are\n"
            + "undone.\n"
            + "Each generations has its own\n"
            + "population of worms.\n"
            + "All worms have two links at the start.\n"
            + "One link is added when some worm is \n"
```

```
            + "shown to climb the screen.\n"
            + "A fossil study is included.\n"
            + "Program is written in JavaFX.\n" );

        //A button must be glued to a pane
        StackPane root = new StackPane();
        root.getChildren().add(btn);

        Scene scene = new Scene(root, 300, 250);

        primaryStage.setTitle("Welcome!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }


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

}//End of main class L73 Welcome
```

74 ***Exercise.*** *Run the program and play with the code.*

75 ***Capturing numbers.*** *A text field is used to capture a string that is next transformed into a number.*

```
//Program L75 CapturingNumbers
//Numbers are captured as strings
//by means of fieldTexts.
//They are  next transformed to numbers,
//if that is possible, and
//their sum is calculated and reported.


package ejvolu12p;

import javafx.application.Application;
```

```
import static javafx.application.Application.launch;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;


public class CapturingNumbers extends Application
{

@Override
public void start(Stage stage)
{
    Group root = new Group();
    Scene scene = new Scene(root, 300, 250);
    stage.setScene(scene);
    stage.setTitle("Capturing numbers");
    GridPane grid = new GridPane();
    grid.setPadding(new Insets(10, 10, 10, 10));
    grid.setVgap(5);
    grid.setHgap(5);
    scene.setRoot(grid);

    //A message
    final Label label1 = new Label();
    //Position
    GridPane.setConstraints(label1, 3, 0);
    //Width
    GridPane.setColumnSpan(label1, 2);
    //Label1 is booked
    grid.getChildren().add(label1);
    //Initialization
    label1.setText("Enter first number");

    //First number is captured
```

```
final TextField number1 = new TextField();
//Position
GridPane.setConstraints(number1, 3, 3);
//Booking
grid.getChildren().add(number1);

//Second number is captured
final TextField number2 = new TextField();
number2.setPromptText("Enter second number");
GridPane.setConstraints(number2, 3, 9);
grid.getChildren().add(number2);

//A button to send data for processing
Button OKBt = new Button("OK");
GridPane.setConstraints(OKBt, 3, 15);
grid.getChildren().add(OKBt);

//Output is presented
final Label sum = new Label();
GridPane.setConstraints(sum, 3, 25);
GridPane.setColumnSpan(sum, 2);
grid.getChildren().add(sum);

//Listener is attached to OKBt
OKBt.setOnAction((ActionEvent e) ->
{
    if
    ((number1.getText().length() != 0)
            && (number2.getText().length() != 0))
        {
            //String -> number
            final double f1
                = Double.parseDouble(number1.getText());
            final double f2
                = Double.parseDouble(number2.getText());
            //Sum
            final double f3 = f1+f2;
            sum.setText("Sum = " + f3);
        }
```

```
    else
        {
            sum.setText("Strings cannot be empty");
        }
    });


    stage.show();
}


public static void main(String[] args)
{
    launch(args);
}
}//End of main class L75 CapturingNumbers
```

76 ***Exercise.*** *Run the program and play with the code.*

77 ***Restricted capturing.*** *When we give to the User the possibility to pick up one option from among a list, we use a ComboBox. The next program shows how to use this to build a calculator.*

```
//Program L77 RestrictedCapturing
//A comboBox is used to display several options
//from which the User must pick up one.
//A console is also included to show
//a large amount of results.
//This program reads two integers,
//the type of arithmetic integer operation,
//executes it
//and prints a report to a console.


package restrictedcapturing;

import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.event.ActionEvent;
```

```
import javafx.event.Event;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;


public class RestrictedCapturing extends Application
{

//Declarations
final Button OKBt = new Button ("OK");
//Console to write output
final TextArea console = new TextArea ("");
//Captured strings
//First number
private static String s1 = "";
//Second number
private static String s2 = "";
//
//private static String s3 = "";
private static String operation = "";
//Associated numbers
private static Integer n1 = 0;
private static Integer n2 = 1;
private static Integer output = 0;



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



@Override public void start(Stage stage)
```

```
{
    stage.setTitle("RestrictedCapturing");
    Scene scene = new Scene(new Group(), 600, 270);


    //First number is captured
    final ComboBox firstNumber = new ComboBox();
    //Options
    firstNumber.getItems()
            .addAll("1", "2", "3", "4", "5");

    //Prompt
    firstNumber.setPromptText("Choose a number");
    //Prompt can be redefined
    firstNumber.setEditable(false);
    //Option is picked up
    firstNumber.setOnAction((Event ev) ->
    {
        s1 =
        firstNumber.getSelectionModel()
                .getSelectedItem().toString();
        n1 = Integer.parseInt(s1);
        System.out.println(n1);
    });


    //Second number is captured
    final ComboBox secondNumber = new ComboBox();
    secondNumber.getItems()
            .addAll("6", "7", "8", "9", "0");

    secondNumber.setPromptText("Choose a number");
    //Prompt cannot be redefined
    secondNumber.setEditable(false);
    secondNumber.setOnAction((Event ev) ->
    {
        s2 =
        secondNumber.getSelectionModel()
                .getSelectedItem().toString();
        n2 = Integer.parseInt(s2);
```

```
});


//A mathematical operation is chosen
final ComboBox operationComboBox = new ComboBox();
operationComboBox.getItems().addAll(
"Sum", "Subtraction",
        "Multiplication", "Integer Division");
operationComboBox.setValue("Sum");
operationComboBox.setEditable(true);

operationComboBox.setOnAction((Event ev) ->
{
    operation =
    operationComboBox.getSelectionModel()
            .getSelectedItem().toString();


});


//Listener of OK button
OKBt.setOnAction((ActionEvent e) ->
{
    if (firstNumber.getValue() != null &&
        !firstNumber.getValue().toString().isEmpty())
    {
        if (operation != null)
            {
                //setText(item);
                if (operation.contains("Sum"))
                {
                    output = n1 + n2;
                    operation = "Sum";
                }
                else if
                    (operation.contains("Subtraction"))
                {
                    output = n1 - n2;
                    operation = "Subtraction";
```

```
                    }
                    else if
                        (operation.contains("Multiplication"))
                    {
                        output = n1 * n2;
                        operation = "Multiplication";
                    }
                    else
                    {
                        output = n1 / n2;
                        operation = "Integer Division";
                    }
                }

            console.setText( "Inputs = " + s1 + " and " +  s2
                    + "\nOperation  = " + operation
                    + "\nOuput = " + output);
        }

    });

    //Grid is initialized
    GridPane grid = new GridPane();
    grid.setVgap(4);
    grid.setHgap(10);
    grid.setPadding(new Insets(5, 5, 5, 5));
    //Row   0
    grid.add(new Label("First number: "), 0, 0);
    grid.add(firstNumber, 1, 0);
    grid.add(new Label("Operation: "), 2, 0);
    grid.add(operationComboBox, 3, 0);
    //Row 1
    grid.add(new Label("Second number: "), 0, 1);
    grid.add(secondNumber, 1, 1);
    //Row 2
    grid.add(console, 0, 2, 4, 1);
    //Row 3
    grid.add(OKBt, 0, 3);
```

```
    Group root = (Group)scene.getRoot();
    root.getChildren().add(grid);
    stage.setScene(scene);
    stage.show();
}
}//End of main class L77 RestrictedCapturing
```

78 ***Exercise.*** *Run the program and play with the code.*

79 ***Exercise.*** *Change in the previous code the integer division by ordinary division.* Answer

80 ***Radio buttons.*** *When one chooses one option among a set of mutually exclusive possibilities, one uses a radio button group.*

```
//Program L80 RadioButtonShow
//One option is selected among a set of
//mutually exclusive choices.

package ejvolu12p;

import javafx.application.Application;
import javafx.beans.value.ObservableValue;
import javafx.geometry.Insets;
import javafx.scene.*;
import javafx.scene.control.*;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

public class RadioButtonShow extends Application
{

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

```
@Override
public void start(Stage stage)
{
    Scene scene = new Scene(new Group());
    stage.setTitle("Evolution");
    stage.setWidth(1000);
    stage.setHeight(100);
    final ToggleGroup group = new ToggleGroup();
    //First radio button
    RadioButton rb1 = new RadioButton("Randomness");
    rb1.setToggleGroup(group);
    rb1.setUserData("Randomness");
    //Second radio button
    RadioButton rb2 = new RadioButton("Evolution without hints");
    rb2.setToggleGroup(group);
    rb2.setUserData("Evolution without hints");
    //Third radio button
    RadioButton rb3 = new RadioButton("Evolution with hint1 "
            + "= gene amplification");
    rb3.setToggleGroup(group);
    rb3.setUserData("Evolution with hint1"
                                + " = gene amplification");
    //Fourth radio button
    RadioButton rb4 = new RadioButton("Evolution with hint2 "
                                            + "= move(1,1)");
    rb4.setToggleGroup(group);
    rb4.setUserData("Evolution with hint2 = move(1,1)");

    //Listener
    group.selectedToggleProperty()
            .addListener((ObservableValue<?
                extends Toggle> ov, Toggle t1, Toggle t2) ->
            {
                RadioButton chk = (RadioButton)t2
                        .getToggleGroup().getSelectedToggle();
                System.out.println("Selected Radio Button - "
                        + chk.getText());
            });
```

```
    //Booking
    HBox hbox = new HBox();
    hbox.getChildren().add(rb1);
    hbox.getChildren().add(rb2);
    hbox.getChildren().add(rb3);
    hbox.getChildren().add(rb4);
    hbox.setSpacing(10);
    //Presentation
    hbox.setSpacing(50);
    hbox.setPadding(new Insets(20, 10, 10, 20));
    //Booking
    ((Group) scene.getRoot()).getChildren().add(hbox);
    stage.setScene(scene);
    stage.show();
}
}//Program L80 RadioButtonShow
```

81 ***Exercise.*** *Run the program and play with the code.*

82 ***Reading a datum from a slider.*** *Instead of typing a number, a slider is used to input a number analogically. The next program uses one to set the separation of two objects.*

```
//Program L82 SliderShow
//A slider is used to get data
//that are printed to the console.

package slidershow;



import javafx.application.Application;
import javafx.beans.value.ObservableValue;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.layout.GridPane;
import javafx.scene.paint.Color;
```

```
import javafx.stage.Stage;



public class SliderShow extends Application
{

//Slider declaration and symbolic initialization
final Slider mySlider = new Slider(0, 200, -1);
//Message for the User
final Label promptMessage
        = new Label("Drag the thumb of the slider!");
final static Color messageColor = Color.BLUE;



@Override
public void start(Stage stage)
{
    Group root = new Group();
    Scene scene = new Scene(root, 700, 200);
    stage.setScene(scene);
    stage.setTitle("Slider show");

    //Setting of parameters of the slider
    mySlider.setMin(0);
    mySlider.setMax(200);
    mySlider.setValue(0);
    mySlider.setShowTickLabels(true);
    mySlider.setShowTickMarks(true);
    mySlider.setMajorTickUnit(100);
    mySlider.setMinorTickCount(5);
    mySlider.setBlockIncrement(10);

    //Message is given a color
    promptMessage.setTextFill(messageColor);


   //new value of slider is detected and reported
    mySlider.valueProperty()
```

```
        .addListener((ObservableValue<? extends Number> ov,
                Number old_val, Number new_val) ->
        {
            System.out.println(new_val.doubleValue());
        });


    //A  grid pane (as a table)
    GridPane grid = new GridPane();

    //Vertical  unit
    grid.setVgap(100);

    //Position of slider: column 1, row 0
    GridPane.setConstraints(mySlider, 1,0);
    //Slider is booked
    grid.getChildren().add(mySlider);

    //Message is positioned: column 0, row 1
    GridPane.setConstraints(promptMessage, 0,1);
    //Message is booked
    grid.getChildren().add(promptMessage);
    //Pane is booked
    root.getChildren().add(grid);
    stage.show();

}

public static void main(String[] args)
{
    launch(args);
}
}//End of main class L82 SliderShow
```

83 ***Exercise.*** *Run the program and play with the code.*

84 ***Charts to report results.*** *JavaFx provides ready to use macros for the making of professional charts. In the next code we see how to pack data for a bar chart and how to display it.*

```
//Program L84 ChartShow
//A barchart is shown.
//XYchars are defined by maps of the form:
//x -> f(x) = y.

package chartshow;

import javafx.application.Application;

import static javafx.application.Application.launch;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.Scene;
import javafx.scene.chart.BarChart;
import javafx.scene.chart.CategoryAxis;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.XYChart;
import javafx.scene.chart.XYChart.Series;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;


public class ChartShow extends Application
{

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

    @Override
    public void start(Stage primaryStage)
    {
        //Declaration of x and y axes
        CategoryAxis xAxis = new CategoryAxis();
        NumberAxis yAxis = new NumberAxis();
        //Declaration of chart
        BarChart barChart = new BarChart(xAxis, yAxis);
```

```
        //Data is provided
        barChart.setData(getChartData());
        barChart.setTitle("Series of random numbers");
        primaryStage.setTitle("Uniform distribution over (0,1)");

        StackPane root = new StackPane();
        root.getChildren().add(barChart);
        primaryStage.setScene(new Scene(root, 400, 250));
        primaryStage.show();
    }

    //Data for a chart is fabricated.
    //It must be a map that associates a number
    //to a string.  Our numbers are generated
    //at random and the string is an Integer.toString.
    private ObservableList<XYChart.Series<String, Double>>
        getChartData()
        {
            //Data is packed into an ObservableList
            ObservableList<XYChart.Series<String, Double>>
                map = FXCollections.observableArrayList();
            Series<String, Double> series = new Series<>();

            series.setName("Random numbers");

            //Fabrication of data:
            //It is a map that associates a random number
            //to an index as string
            for (Integer index = 0; index < 10; index++)
            {
                double rand =  Math.random() ;
                series.getData().add(new XYChart.Data
                        (Integer.toString(index), rand));
            }
            map.addAll(series);
            return map;
        }
}//End of main class L84 ChartShow
```

85 ***Exercise.*** *Run the program and play with the code.*

86 ***Menus.*** *The next code illustrates how menus operate in JavaFX.*

```
//Program L86 MenuShow
//A menu is used to control
//the presentation of 3 labels.
//JavaFX version.

package ejvolu12p;


import java.util.AbstractMap.SimpleEntry;
import java.util.Map.Entry;
import javafx.application.Application;
import javafx.beans.value.ObservableValue;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.input.*;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.TextAlignment;
import javafx.stage.Stage;

public class MenuShow extends Application
{

//A string is mapped to a color
final Entry<String, Color>[] effects = new Entry[]
{
new SimpleEntry<>("RED",Color.RED),
new SimpleEntry<>("GREEN", Color.GREEN),
new SimpleEntry<>("BLUE", Color.BLUE)
};
```

```
//Three labels
final static Label title = new Label();
final static Label author = new Label();
final static Label info = new Label();

//Vertical box with just one column
//to accommodate labels
final static VBox vbox = new VBox();

//Declaration of menu items
CheckMenuItem titleView ;
CheckMenuItem authorView ;
CheckMenuItem infoView ;

Menu hideView;


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

@Override
public void start(Stage stage)
{
    stage.setTitle("Menu Show");
    Scene scene = new Scene(new VBox(), 400, 350);
    //Used fonts
    title.setFont(new Font("Verdana Bold", 22));
    author.setFont(new Font("Arial Italic", 10));
    info.setWrapText(true);
    info.setTextAlignment(TextAlignment.JUSTIFY);
    //Vertical box specifications
    vbox.setAlignment(Pos.CENTER);
    vbox.setSpacing(10);
    vbox.setPadding(new Insets(0, 10, 0, 10));
    vbox.getChildren().addAll(title, author,  info);

    //Menubar declaration and initialization
```

```
MenuBar menuBar = new MenuBar();

// --- Menu General
Menu menuGeneral = new Menu("General");
//Menu item declaration and initialization
MenuItem setTexts = new MenuItem("SetText");
//Listener
setTexts.setOnAction((ActionEvent t) ->
{
    setTexts();
    vbox.setVisible(true);
});
//MenuItems added to menu
menuGeneral.getItems().addAll(setTexts);

// --- MenuItem clear
MenuItem clear = new MenuItem("Clear");
//Can be sparked from a key
clear.setAccelerator(KeyCombination
        .keyCombination("Ctrl+X"));

//MenuItem exit
MenuItem exit = new MenuItem("Exit");
exit.setOnAction((ActionEvent t) ->
{
    //Finish everything and exit
    System.exit(0);
});
//Insertions
menuGeneral.getItems()
        .addAll(clear, new SeparatorMenuItem(), exit);


// --- Menu Color
Menu menuColor = new Menu("Color");

Menu menuEffect = new Menu("Color Effect");
final ToggleGroup colorEffect = new ToggleGroup();
for (Entry<String, Color> colorFilling : effects)
{
```

```
    RadioMenuItem itemEffect
            = new RadioMenuItem(colorFilling.getKey());
    itemEffect.setUserData(colorFilling.getValue());
    itemEffect.setToggleGroup(colorEffect);
    menuEffect.getItems().add(itemEffect);
}
//NoEffects menu
final MenuItem noEffects = new MenuItem("No Effects");
noEffects.setDisable(true);
//Listener
Boolean activated = false;
 noEffects.setOnAction((ActionEvent t) ->
{
   info.setTextFill(Color.BLACK);
   noEffects.setDisable(!activated);
});


//Listener
colorEffect.selectedToggleProperty().addListener(
(ObservableValue<? extends Toggle> ov, Toggle old_toggle,
Toggle new_toggle) ->
{
    if (colorEffect.getSelectedToggle() != null)
    {

        Color color = (Color) colorEffect
                .getSelectedToggle().getUserData();
        info.setTextFill(color);

        noEffects.setDisable(false);
    }
    else
    {
        noEffects.setDisable(true);

    }
});
menuColor.getItems().addAll(menuEffect, noEffects);
```

```
// --- Menu Hide
hideView = new Menu("Hide");
hideView.setDisable(true);
//Initialziation
titleView = createMenuItem ("Title", title);
authorView = createMenuItem ("Author", author);
infoView = createMenuItem ("Info", info);
//Booking
hideView.getItems().addAll(titleView, authorView,
    infoView);
menuBar.getMenus()
        .addAll(menuGeneral, menuColor, hideView);
titleView.setVisible(false);
titleView.setVisible(false);
titleView.setVisible(false);

//Listener
clear.setOnAction((ActionEvent t) ->
{
    vbox.setVisible(false);
    titleView.setSelected(false);
    authorView.setSelected(false);
    infoView.setSelected(false);

});


//Context Menu that reveals who the author is
final ContextMenu cm = new ContextMenu();
MenuItem cmItem1 = new MenuItem("Who is the author?");
author.setContextMenu(cm);

cm.getItems().add(cmItem1);
author.addEventHandler(MouseEvent
        .MOUSE_CLICKED, (MouseEvent e) ->
{
    if (e.getButton() == MouseButton.SECONDARY)
    {
        author.setText("The author is me");
```

```
        }
    });

    //Booking
    ((VBox) scene.getRoot())
            .getChildren().addAll(menuBar, vbox);
    stage.setScene(scene);
    stage.show();

}


//Text of labels are managed
private void setTexts()
{
    title.setText("This is the title");
    author.setText("Author");
    info.setText("Right click over the Author");

    author.setVisible(true);
    title.setVisible(true);
    info.setVisible(true);

    titleView.setSelected(true);
    authorView.setSelected(true);
    infoView.setSelected(true);

    hideView.setDisable(false);
}


//Items of the hide menu are managed
private static CheckMenuItem
            createMenuItem (String s, final Node node)
{
    CheckMenuItem cmi = new CheckMenuItem(s);
    //Listener
    cmi.selectedProperty().addListener
        ((ObservableValue<? extends Boolean>
            ov, Boolean oldVal, Boolean newVal) ->
```

```
            {
                node.setVisible(newVal);
                cmi.setVisible(newVal);
                cmi.setSelected(newVal);
                vbox.setVisible(true);
            });
    return cmi;
}

}//End of main class L86 MenuShow
```

87 ***Exercise.*** *Run the program and play with the code.*

88 ***Tabs.*** *Controls are grouped in pages which can be separated by tabs, such as it is shown here.*

```
//Program L88 Tabs
//Tabs allows to separate GUI by chapters

package ejvolu12p;

import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.Tab;
import javafx.scene.control.TabPane;
import javafx.scene.layout.BorderPane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;


public class Tabs extends Application
{


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

```
    {
        primaryStage.setTitle("Tabs");
        Group root = new Group();
        Scene scene = new Scene(root, 400, 300, Color.WHITE);

        //This is the outer pane
        BorderPane outerPane = new BorderPane();
        //Pane to host tabs
        TabPane tabPane = new TabPane();

        //FirstTab
        Tab firstTab = new Tab();
        firstTab.setText("First page");
        //Add a label
        Label firstLabel = new Label("Information 1");
        firstTab.setContent(firstLabel);
        tabPane.getTabs().add(firstTab);

        //Second tab
        Tab secondTab = new Tab();
        secondTab.setText("Second page");
        //Add a label
        Label secondLabel = new Label("Part two");
        secondTab.setContent(secondLabel);
        tabPane.getTabs().addAll(secondTab);

        //TabPane is booked and  positioned
        outerPane.setCenter(tabPane);
        //OuterPane is booked
        root.getChildren().add(outerPane);
        primaryStage.setScene(scene);
        primaryStage.show();
    }


    public static void main(String[] args)
    {
        launch(args);
    }
}//End of main class L88 Tabs
```

89 ***Exercise.*** *Run the program and play with the code.*

90 ***Displaying information.*** *JavaFX offers various options to display huge amounts of information. The next is both simple and effective.*

```
 //Program L90 PaginationShow
//A nice option to display information.

package ejvolu12p;

import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.Scene;
import javafx.scene.control.Pagination;
import javafx.scene.control.TextArea;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;



public class PaginationShow extends Application
{
//Tool for information display
private Pagination pagination;
//Information to be displayed
final String[] textPages = new String[]
{
    "This is the most important expression",
    "This is also very important",
    "This is important",
    "This is interesting",
    "This is secondary",
    "This is not very important",
    "This is irrelevant",
    "This is what will kill you."
};
```

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

//Number of items per page
public int itemsPerPage()
{
    return 1;
}

//Pieces of information are associated to pages
public VBox createPage(int pageIndex)
{
    VBox box = new VBox(1);
    int page = pageIndex * itemsPerPage();
    for (int i = page; i < page + itemsPerPage(); i++)
    {
        TextArea text = new TextArea(textPages[i]);
        text.setFont(Font.font(18-i));
        text.setWrapText(true);
        box.getChildren().add(text);
    }
return box;
}

@Override
public void start(final Stage stage) throws Exception
{
    //Instantiation and initialization = 0th page
    pagination = new Pagination(textPages.length, 0);
    //Colored border
    pagination.setStyle("-fx-border-color:red;");
    //Pagination master
    pagination.setPageFactory((Integer pageIndex) ->
    {
        if (pageIndex >= textPages.length)
        {
```

```
            return null;
        }
        else
        {
            return createPage(pageIndex);
        }
    });
    //Pane to place pages
    AnchorPane anchor = new AnchorPane();
    //Margins
    AnchorPane.setTopAnchor(pagination, 20.0);
    AnchorPane.setRightAnchor(pagination, 20.0);
    AnchorPane.setBottomAnchor(pagination, 10.0);
    AnchorPane.setLeftAnchor(pagination, 10.0);
    //Booking
    anchor.getChildren().addAll(pagination);
    Scene scene = new Scene(anchor, 400, 250);
    stage.setScene(scene);
    stage.setTitle("PaginationSample");
    stage.show();
}
}//L90 PaginationShow
```

91 ***Exercise.*** *Run the program and play with the code.*

92 ***Stopping an animation.*** *The next code shows how to stop an animation. To that aim, we will devise a clock.*

```
//Program L92 StopFX
//A tutorial that shows how to
//control start/stop/pause/resume   an animation.
//JavaFX version.


package ejvolu12p;
```

```
import javafx.animation.Animation;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
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.*;
import javafx.scene.shape.Line;
import javafx.scene.transform.Rotate;
import javafx.stage.Stage;
import javafx.util.Duration;



public class StopFX extends Application
{

 //Timeline for watch
 private static final Timeline watch = new Timeline();
 //A rotation is initialized with angle 0 degrees
 //and position (200,200)
 final Rotate rotationTransform = new Rotate(0, 200,200);




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



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

```
Group p = new Group();
Scene scene = new Scene(p, 400, 350);
stage.setTitle("Watch");
//Hand of the watch
Line hand = new Line(200, 200, 200, 100);
p.getChildren().addAll(hand);

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

// --- Menu General
Menu menuGeneral = new Menu("Control");
//Menu item declaration and initialization
MenuItem start = new MenuItem("Start");

//Listener
start.setOnAction((ActionEvent t) ->
{
    //Initial angle
    rotationTransform.setAngle(0);
    watch.playFromStart();

});
//MenuItems added to menu
menuGeneral.getItems().addAll(start);


//Menu item declaration and initialization
MenuItem pause = new MenuItem("Pause");
//Listener
pause.setOnAction((ActionEvent t) ->
{
    watch.pause();
});
//MenuItems added to menu
menuGeneral.getItems().addAll(pause);


//Menu item declaration and initialization
```

```
MenuItem resume = new MenuItem("Resume");
//Listener
resume.setOnAction((ActionEvent t) ->
{
    watch.play();
});
//MenuItems added to menu
menuGeneral.getItems().addAll(resume);

//Menu item declaration and initialization
MenuItem stop = new MenuItem("Stop");
//Listener
stop.setOnAction((ActionEvent t) ->
{
    watch.stop();
});
//MenuItems added to menu
menuGeneral.getItems().addAll(stop);


//MenuItem exit
MenuItem exit = new MenuItem("Exit");
exit.setOnAction((ActionEvent t) ->
{
    //Finish everything and exit
    System.exit(0);
});

//Insertions
menuGeneral.getItems()
        .addAll( new SeparatorMenuItem(), exit);

//Booking
menuBar.getMenus()
        .addAll(menuGeneral);
p.getChildren().addAll(menuBar);
stage.setScene(scene);
stage.show();

//Rotation is added to hand
```

```
    hand.getTransforms().add(rotationTransform);

    //Timeline is specified:
    //Give a round tour (360 degrees) for each  5  seconds
    watch.getKeyFrames().addAll(
        new KeyFrame(Duration.seconds(5),
            new KeyValue(
                rotationTransform.angleProperty(),360) ));
    //Repeat tour over and over
    watch.setCycleCount(Animation.INDEFINITE);

    //Enable start  only when animation is stopped
    start.disableProperty().bind(watch.statusProperty()
            .isNotEqualTo(Animation.Status.STOPPED));
    //Enable pause only when animation is running
    pause.disableProperty().bind(watch.statusProperty()
            .isNotEqualTo(Animation.Status.RUNNING));
    //Enable resume only when animation is paused
    resume.disableProperty().bind(watch.statusProperty()
            .isNotEqualTo(Animation.Status.PAUSED));
    //Enable stop only when animation is not stopped
    stop.disableProperty().bind(watch.statusProperty()
            .isEqualTo(Animation.Status.STOPPED));
}



}//End of main class L92 StopFX
```

93 ***Exercise.*** *Run the program and play with the code.*

94 ***Exercise.*** *Devise a watch with two handles, one for, say, minutes, and other for seconds.* Answer

95 ***Research.*** *Developers tend to consider that functional units are beautiful per se. Nevertheless, experts learn that there is a type of beauty that adds functionality. For instance, we already mentioned the evolvable style of programming that eases the task of modifying software. By the same token, JavaFX provides additional options to embellish GUIs. They appear in the chapter of Cascading Style Sheets*

(CSS) of the cited pdf. In some cases the change is nice and so, the Reader is invited to study the theme to use that power.

5.3 Conclusion

We have got acquainted with some built-in JavaFX User Interface controls. In spite of its complex and tight syntax, we consider that JavaFX is very nice and very powerful.

Chapter 6

Worms as a laboratory

Measuring and reporting

96 ***Objective.*** *We provide a GUI for our program about worms. The result shall be an application that can be used to easily design experiments and that can be safely run by everybody without touching the code and with no knowledge of Java.*

6.1 A GUI for our worms

Let us apply the know-how that was acquired in the last chapter to improve our application about worms.

97 ***The next code adds a JavaFX GUI to our program of worms.***

```
//Program L97 WormsEvoJFX4
//A population of worms is programmed
//by evolution to
//climb the screen.
//All moves are executed
//but  nondisruptive moves are
//not undone.
//All worms have two links at the start.
//One link is added when some worm is shown
//to climb the screen.
//A fossil study is included.
//The  simulation includes the accepted by us fact that
//evolution can create complexity and is
//addressed to tackle two questions:
```

```
// 1. Which is the price that evolution
//      must pay for complexity?
// 2. Does imperfect fossils appear
//      in fossil sampling
//      and in which proportion?
//A suitable GUI is provided.
//Program is written in JavaFX.


package ejvolu12p;


import java.awt.AWTException;
import java.io.IOException;
import static java.lang.Math.pow;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.animation.Interpolator;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.ParallelTransition;
import javafx.animation.SequentialTransition;
import javafx.animation.Timeline;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.event.Event;
import javafx.event.EventHandler;
import javafx.geometry.Bounds;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
```

```
import javafx.scene.chart.BarChart;
import javafx.scene.chart.CategoryAxis;
import javafx.scene.chart.LineChart;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.PieChart;
import javafx.scene.chart.XYChart;
import javafx.scene.control.Button;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.control.Pagination;
import javafx.scene.control.RadioButton;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.ScrollPane.ScrollBarPolicy;
import javafx.scene.control.SeparatorMenuItem;
import javafx.scene.control.Slider;
import javafx.scene.control.Tab;
import javafx.scene.control.TabPane;
import javafx.scene.control.TextArea;
import javafx.scene.control.Toggle;
import javafx.scene.control.ToggleGroup;
import javafx.scene.image.Image;
import javafx.scene.input.Clipboard;
import javafx.scene.input.ClipboardContent;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.FlowPane;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Font;
import javafx.scene.text.Text;
import javafx.stage.Stage;
import javafx.util.Duration;
import javax.imageio.ImageIO;
```

```
public class WormsEvoJFX4 extends Application
{
    //Instantiation
    private static WormsEvoJFX4 w = new WormsEvoJFX4();
    //Overall drawing process
    private static  SequentialTransition allST
                    = new SequentialTransition();
    //Drawing process for evolution
    private static  SequentialTransition evolutionST
                    = new SequentialTransition();
    //Number of repetitions or generations
    private static int nGenerations = 50;
    //Actual generation
    private static int gen = 0;
    //Duration of graphical events
    private static final Duration d = Duration.millis(100);
    //The scene
    private static Group p = new Group();
    //Number of steps given for each try
    private static int nSteps = 7;
    //Number of worms
    private static  int nWorms = 8;
    //Worms are kept in a list
    private static List<Worm> Worms = new ArrayList<>();
    //Helping files
    private static List<Worm> WormsTwo = new ArrayList<>();
    private static Worm[] OldWorms ;
    //Every worm leaves a fossil
    private  static List<Fossil> Fossils = new ArrayList<>();
    //Counter  of fossils
    private static int nFossils = 0;
    //Number of   fossils to be sampled
    private static int nSampledFossils = 5;
    //Number of links of each worm
    private static  int nLinks = 2;
    //Elasticicty of inner links:
    //inner links can get separated without geting disrupted
    private static double innerElasticity = 1.415;
```

```
//Elasticity of head and tail
private static double borderElasticity = 1.1;
//Width of the worm = width of each link
private static int widthOfLinks = 6;
//Height of each link
private static int heightOfLinks = 6;
//Each worm has nMovers movers,
//instructions to move.
private static int  nMovers = 5;
//Initial vertical position of the head over the screen
private static int initialRow = 280;

//Generator of pseudo random numbers
private static final Random rd = new Random();
//Array with 9 possible moves
//that are possible directions of movement
private static Move  vectMoves[ ] = new Move[9];
private static final boolean print1 = false;
private static final boolean print2 = false;
private static final boolean print3 = false;
private static final boolean printReproduction = true;
//Text to be drawn
private static  Text genCount = new Text ("");
//Two links cannot occupy the same place
static boolean isOccupied[][]  = new boolean[600][300];
//Is a worm capable of climbing the screen?
private static  int Fitness[] =  new int[21];
//Was climbing good enough?
private static boolean bullseye  = false;
//Measures of the screen
private static int xWin = 1260;
private static int yWin = 1000;
//Grouping of links in order to ease erasing
private static List<Rectangle> allRects
                        = new ArrayList<>();
private static List<Rectangle> lastRects
                        = new ArrayList<>();
//The number of point mutations of the offspring
private static int nPointMutations = 5;
//Gene amplification follows:
```

```
//nSourceMovers is the number of different Movers
//to be copied.
private static int nSourceMovers = 2;
//ncopies is the number of new copies
//of each copied Mover in
 //the gene amplification method.
private static int   nCopies = 2;

//List of generations at which a worm were found
//to climb the screen
private static List<Integer> bullseyes = new ArrayList<>();

//Slider for shifting the position of heads of worms:
//declaration and symbolic initialization
final Slider verticalShiftSlider = new Slider(-100, 100, 0);

//Slider for the elasticity of inner links
final Slider innerElasticitySlider = new Slider(0.5, 2, 0);

//Slider for the elasticity of
//bordering links    (head and tail)
final Slider borderElasticitySlider
                    = new Slider(0.5, 2, 0);
//Dynamics can be fueled by randomness,
//by evolution without hints,
//or by evolution with hints
private static String typeOfProcess
                    = "Evolution without hints";
//Blackboard to write a report
private static TextArea console
                    = new TextArea ("Use copy-paste");

//Click over this button to begin simulation
final Button goBt = new Button ("Go");

//This is the outer pane to place tabs
BorderPane outerPane = new BorderPane();

//Pane to place a console
private static FlowPane flowPane;
```

```
//Declaration of line chart
private static LineChart newLinkChart;

//Declaration of bar chart of perfection of fossils
private static BarChart qualityChart;

//Declaration of pie chart of perfection of fossils
private static PieChart qualityPieChart;

//Number of perfect fossils
private static int nPerfect;
//Number of imperfect fossils
private static int nImperfect;
//Quality of fossils
private static String quality = "";

//Tool for information display
private Pagination pagination;
//Information to be displayed.
//Pages are separated by a comma.
final String[] textPages = new String[]
{
    "A population of worms is programmed "
    + "by evolution to "
    + "climb the screen."
    , "All moves are executed: "
    + "disruptive moves are undone "
    + "but  nondisruptive moves remain."
    ,"One link is added when some worm is shown "
    + "to climb the screen."
    ,"A fossil study is included."
    ,"The  simulation includes the accepted by us"
    + " fact that "
    + "evolution can create complexity and is"
    + " addressed to tackle two questions:"
    + " \n   1. Which is the price that evolution "
    + "must pay for complexity?"
    + "\n   2. Does imperfect fossils appear "
    + "in fossil sampling "
    + "and in which proportion?"
```

```
    };

    /* Definition of inner classes.
    *
    The Move class:
    A Move says in which direction
    a link of the worm must be moved.
    */

 private  class Move
 {
    int x;
    int y;


    //Default initialization
    Move()
    {
        x= 0;
        y= 0;
    }

    // Initialize with a list of two coordinates
    Move(int a, int b)
    {
        x= a;
        y= b;
    }

    // Copy a move
    Move(Move move)
    {
        x= move.x;
        y= move.y;
    }
 }//End of class Move


 // Definition of the class Mover:
```

```
// A Mover keeps two types of information:
// a) the link of the worm to be moved and
// b) the direction or move to be chosen.
//
private   class Mover
{
   int link ;
   Move   move;

   //  Movers can acquired information
   // in three ways, they are  overloaded.
   // Form 1: by a list of numbers
   private Mover(int l, int mx, int my)
    {
       link= l;
       move.x = mx;
       move.y = my;
    }

   // Form 2, by structured data: link + move
   private Mover(int l, Move m)
   {
       link= l;
       move = new Move(m);
   }

   // Form 3, by structured data: mover
   private Mover(Mover mover)
   {
       link= mover.link;
       move =  new Move(mover.move);
   }
 }//End of class Mover


//A random color is synthesized
private static Color randomColor()
{
   //Color is defined as a random
   //mix of red, green and blue
```

```
    int r,g,b;
    r = rd.nextInt(226);
    g = rd.nextInt(226);
    b = rd.nextInt(226);
    //The color is prepared
    Color c = Color.rgb(r,g,b);
    return c;
 }


//Rects are created
private static  Rectangle
           createRectangle(int gen)
{
    //Upper left corner at (0,0)
    Rectangle rect =
        new Rectangle(0,0,widthOfLinks,heightOfLinks);

    //The color is prepared
    Color c = randomColor();
    //The rect is painted
    rect.setFill(c);
    //Rects of the first generation (gen = 0)
    //are visible, otherwise they are not
    if (gen > 0)
    {
        rect.setHeight(0);
        rect.setWidth(0);
    }
    return rect;
}


/*  Here we define the class worm.
    A worm is composite: it has links.
    Each link has a rectangular shape
    and  its own color.
    The vector Mover[ ]
     keeps the information
    on how to move its links:
```

```
    which  link and in which order
    (in the order that they appear in
    the vector Mover[]), and
    in which direction.
    Each worm keeps record of
    the actual position of each
    one of its links,
    in the associated rectangle. */



private class Worm
{
    //Number of links
    int numberOfLinks = nLinks;
    //Number of movers = instructions to move
    int numberOfMovers = nLinks;
    //List Mover  keeps movers
    List<Mover> Mover = new ArrayList<>();
    //The shape of each link
    List<Rectangle> Rect = new ArrayList<>();
    //move right (side = 1) else left (side = -1)
    int side = 2*rd.nextInt(2) -1;

    //Default initialization: at random.
    //Worms of the first generation are
    //visible. Others not.
    private Worm()
    {
        //Visibility of links: 0 = visible; 1 = unvisible
        int k= 0;
        if ( (typeOfProcess.equals("Randomness"))
            && (gen == 0))    k = 0;
        else  k = 1;

        //A color is assigned to each link
        //but different than white
        for(int i = 0; i < numberOfLinks; i++)
            Rect.add(createRectangle(k));
```

```
        //Instructions to move at random
for(int m = 0; m < nMovers; m++)
        {
            //Select at random  which link to move:
            int l = rd.nextInt(nLinks) ;
            //Select at random  which direction to move on
            Move  move
                    = new Move (vectMoves[rd.nextInt(9)]);
            //Memorization
            Mover.add(new Mover(l,move));

        }
    }//End of constructor Worm()


    //Constructor from worm W of the old generation.
    private Worm(int W)
    {
        numberOfLinks = nLinks;
        numberOfMovers = nMovers;
        //Symbolic initialization
        for(int i = 0; i < numberOfLinks; i++)
               Rect.add(createRectangle(1));
        for(int m = 0; m < nMovers; m++)
        {
            //Select at random  which link to move:
            int l = rd.nextInt(nLinks) ;
            //Select at random  which direction to move on
            Move  move
                    = new Move (vectMoves[rd.nextInt(9)]);
            //Memorization
            Mover.add(new Mover(l,move));
        }
        //Read true color
        for(int i = 0; i <  OldWorms[W].numberOfLinks; i++)
        {
            Rect.get(i).setFill( OldWorms[W]
                                 .Rect.get(i).getFill());
        }
        //Read true movers
```

```
        for(int m = 0; m < OldWorms[W].numberOfMovers; m++)
           Mover.set(m, OldWorms[W].Mover.get(m));
    }

    //Human design initialization
    private Worm(String HD )
    {
numberOfLinks = nLinks;
        numberOfMovers = nLinks;
        //Rects
        for(int i = 0; i < numberOfLinks; i++)
               Rect.add(createRectangle(1));
        //The know how to climb the screen
int link;
for(int l = 0; l  < nLinks; l++)
        {
            //Move the tail first and then the others links.
            link = nLinks-l-1;
            Mover mover;

            //Move each link in the y-direction
            if (side == 1)
                mover = new Mover(link,new Move(1,1));
            else
                mover =  new Mover(link,new Move (-1,1));
            Mover.add(mover);
            if (print3)
                System.out.println(l + "WormHD link = "
                + mover.link + " move.x = " + mover.move.x
                + " move.y= " + mover.move.y);
        }
    }//End of Worm()


    //Constructor from a fossil
    private Worm(Fossil fossil)
    {
        numberOfLinks = fossil.numberOfLinks;
        numberOfMovers = fossil.numberOfMovers;
        //Rects are initialized, 1 = unvisible
```

```
        for(int i = 0; i < numberOfLinks; i++)
               Rect.add(createRectangle(1));

        //Movers are copied
        for(int m = 0; m < numberOfMovers; m++)
        {
            //Link is read:
            int link = fossil.Mover.get(m).link;

            //Move is read
            Move  move
                    = w.new Move (fossil.Mover.get(m).move);
            //Mover is built
            Mover mover = w.new Mover(link,move);
            //Memorization
            Mover.add(mover);
            if (print3)
                System.out.println(m
                        + "Fossil -> worm link = "
                + mover.link + " move.x = " + mover.move.x
                + " move.y= " + mover.move.y);
        }
    }

}//End of class Worm



//Inner class Fossil.
//Fossils keep relevant information about worms
private class Fossil
{
    int numberOfLinks;
    int numberOfMovers;
    //The shape of each link
    List<Rectangle> Rect = new ArrayList<>();
    //List mover  keeps movers
    List<Mover> Mover = new ArrayList<>();
```

```
    //Constructor of a fossil from a worm
    //Worm -> fossil
    private Fossil(Worm worm)
    {
        numberOfLinks = worm.numberOfLinks;
        numberOfMovers = worm.numberOfMovers;
        //Rects are created
         for(int i = 0; i < numberOfLinks; i++)
               Rect.add(createRectangle(1));

        //Movers are cloned
        for(int m = 0; m < numberOfMovers; m++)
            Mover.add(new Mover(worm.Mover.get(m)));
    }
}//End of Fossil




//General setting
@Override public void start(Stage stage)
{
    //The scene
    Scene scene = new Scene(p, xWin,yWin);
    stage.setScene(scene);
    stage.setTitle("Evolution teaches worms "
                   + "to climb the screen");
    stage.show();
    //Definition of possible moves
    moveDef();



    //========BEGINNING OF GUI============

    //========MENUBAR===========

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

```
// --- Menu Program
Menu menuProgram = new Menu("Program");

//Instantiation and initialization = 0th page
pagination = new Pagination(textPages.length, 0);
//Colored border
pagination.setStyle("-fx-border-color:red;");
//Pagination master
pagination.setPageFactory((Integer pageIndex) ->
{
    if (pageIndex >= textPages.length)
    {
        return null;
    }
    else
    {
        return createPage(pageIndex);
    }
});

//Pane to place pages
AnchorPane anchor = new AnchorPane();

//Margins
AnchorPane.setTopAnchor(pagination, 20.0);
AnchorPane.setRightAnchor(pagination, 20.0);
AnchorPane.setBottomAnchor(pagination, 10.0);
AnchorPane.setLeftAnchor(pagination, 10.0);
//Booking
anchor.getChildren().addAll(pagination);

//Pane
FlowPane s = new FlowPane();
s.getChildren().add(anchor);
//Pane is  hidden
s.setOpacity(0);
s.setLayoutX(100);
s.setLayoutX(100);
```

```
//Click over this button to begin simulation
final Button closeBt = new Button ("Close");
s.getChildren().add(closeBt);

p.getChildren().add(s);
//Listener of OK button
closeBt.setOnAction((ActionEvent e) ->
{
    s.setOpacity(0);

});


//Menu item declaration and initialization
MenuItem displayInfo = new MenuItem("Information");
//Listener
displayInfo.setOnAction((ActionEvent t) ->
{
 s.setOpacity(1);
});

//MenuItems added to menu
menuProgram.getItems().addAll(displayInfo);


//MenuItem exit
MenuItem exit = new MenuItem("Exit");
exit.setOnAction((ActionEvent t) ->
{
    //Finish everything and exit
    System.exit(0);
});
//Insertions
menuProgram.getItems()
        .addAll(
                new SeparatorMenuItem(), exit);

    menuBar.getMenus()
        .addAll(menuProgram);
 p.getChildren().add(menuBar);
```

```
//======== Tabs=========


//Position of outerPane

outerPane.setLayoutX(100);
outerPane.setLayoutY(500);
outerPane.setOpacity(0);

//Pane to host tabs
TabPane tabPane = new TabPane();


//=====Worms Tab
Tab wormsTab = new Tab();
wormsTab.setText("Worms");

//Number of worms is captured
final ComboBox nWormsCB = new ComboBox();
//Options
nWormsCB.getItems()
        .addAll("1",  "2", "3", "4", "5","6",
                "7", "8", "9", "10");
nWormsCB.setEditable(false);
nWormsCB.setValue(nWorms);
//Listener
nWormsCB.setOnAction((Event ev) ->
{
    String s1 =
    nWormsCB.getSelectionModel()
            .getSelectedItem().toString();
    nWorms = Integer.parseInt(s1);
    System.out.println(nWorms);
});


//Number of links is captured
final ComboBox nLinksCB = new ComboBox();
```

```
//Options
nLinksCB.getItems()
        .addAll(  "2", "3", "4", "5","6",
                "7", "8", "9", "10");
nLinksCB.setEditable(false);
nLinksCB.setValue(nLinks);
//Listener
nLinksCB.setOnAction((Event ev) ->
{
    String s1 =
    nLinksCB.getSelectionModel()
            .getSelectedItem().toString();
    nLinks = Integer.parseInt(s1);
    System.out.println(nLinks);
});


//Width of links
final ComboBox widthOfLinksCB = new ComboBox();
//Options
widthOfLinksCB.getItems()
        .addAll(  "2", "4", "6", "8", "10", "12");
widthOfLinksCB.setEditable(false);
widthOfLinksCB.setValue(widthOfLinks);
widthOfLinksCB.setOnAction((Event ev) ->
{
    String s1 =
    widthOfLinksCB.getSelectionModel()
            .getSelectedItem().toString();
    widthOfLinks = Integer.parseInt(s1);
    System.out.println(widthOfLinks);
});

//Height of links
final ComboBox heightOfLinksCB = new ComboBox();
//Options
heightOfLinksCB.getItems()
        .addAll(  "2", "4", "6", "8", "10", "12");
heightOfLinksCB.setEditable(false);
heightOfLinksCB.setValue(heightOfLinks);
```

```
heightOfLinksCB.setOnAction((Event ev) ->
{
    String s1 =
    heightOfLinksCB.getSelectionModel()
            .getSelectedItem().toString();
    heightOfLinks = Integer.parseInt(s1);
    System.out.println(heightOfLinks);
});



//Slider for the elasticity of inner links
innerElasticitySlider.setMin(0.5);
innerElasticitySlider.setMax(2);
innerElasticitySlider.setValue(innerElasticity);
innerElasticitySlider.setShowTickLabels(true);
innerElasticitySlider.setShowTickMarks(true);
innerElasticitySlider.setMajorTickUnit(0.5);
innerElasticitySlider.setMinorTickCount(1);
innerElasticitySlider.setBlockIncrement(0.1);

innerElasticitySlider.valueProperty().addListener((
ObservableValue<? extends Number> ov, Number old_val,
Number new_val) ->
{
    innerElasticity =    new_val.doubleValue();
    System.out.println(innerElasticity);
});


//Slider for the elasticity of bordering links,
//head and tail
borderElasticitySlider.setMin(0.5);
borderElasticitySlider.setMax(2);
borderElasticitySlider.setValue(borderElasticity);
borderElasticitySlider.setShowTickLabels(true);
borderElasticitySlider.setShowTickMarks(true);
borderElasticitySlider.setMajorTickUnit(0.5);
borderElasticitySlider.setMinorTickCount(1);
borderElasticitySlider.setBlockIncrement(0.1);
```

```
borderElasticitySlider.valueProperty().addListener((
ObservableValue<? extends Number> ov, Number old_val,
Number new_val) ->
{
    borderElasticity =    new_val.doubleValue();
    System.out.println(borderElasticity);
});



//Number of steps is captured
final ComboBox nStepsCB = new ComboBox();
//Options
nStepsCB.getItems()
        .addAll( "1", "2", "3", "4", "5","6",
                "7", "8", "9", "10");
nStepsCB.setEditable(false);
nStepsCB.setValue(nSteps);
nStepsCB.setOnAction((Event ev) ->
{
    String s1 =
    nStepsCB.getSelectionModel()
            .getSelectedItem().toString();
    nSteps = Integer.parseInt(s1);
    System.out.println(nSteps);
});


//Number of steps is captured
final ComboBox nMoversCB = new ComboBox();
//Options
nMoversCB.getItems()
        .addAll( "1", "2", "3", "4", "5","6",
                "7", "8", "9", "10");
nMoversCB.setEditable(false);
nMoversCB.setValue(nMovers);
nMoversCB.setOnAction((Event ev) ->
{
    String s1 =
```

```
    nMoversCB.getSelectionModel()
            .getSelectedItem().toString();
    nMovers = Integer.parseInt(s1);
    System.out.println(nMovers);
});


//Slider for Position of heads
verticalShiftSlider.setMin(-100);
verticalShiftSlider.setMax(100);
verticalShiftSlider.setValue(0);
verticalShiftSlider.setShowTickLabels(true);
verticalShiftSlider.setShowTickMarks(true);
verticalShiftSlider.setMajorTickUnit(100);
verticalShiftSlider.setMinorTickCount(50);
verticalShiftSlider.setBlockIncrement(50);

verticalShiftSlider.valueProperty().addListener((
ObservableValue<? extends Number> ov, Number old_val,
Number new_val) ->
{
    initialRow = (int) (280 +  new_val.doubleValue());
    System.out.println(initialRow);
});




//GridPane  of wormsTab
GridPane gridWT = new GridPane();
gridWT.setVgap(4);
gridWT.setHgap(10);
gridWT.setPadding(new Insets(5, 5, 5, 5));
//Row   0
gridWT.add(new Label("Choose the number of "
        + "worms per generation: "), 0, 0);

gridWT.add(nWormsCB, 1, 0);
```

```
//Row   1
gridWT.add(new Label("Choose the "
        + "number of links: "), 0, 1);
gridWT.add(nLinksCB, 1,1);

gridWT.add(new Label("Choose the width "
        + "of links: "), 2, 1);
gridWT.add(widthOfLinksCB, 3,1);

gridWT.add(new Label("Choose the height "
        + "of links: "), 4, 1);
gridWT.add(heightOfLinksCB, 5,1);

//Row   2
gridWT.add(new Label("Choose the inner "
        + "elasticity: "), 0, 2);
gridWT.add(innerElasticitySlider, 1,2);

gridWT.add(new Label("Choose the border "
        + "elasticity: "), 2, 2);
gridWT.add(borderElasticitySlider, 3,2);

//Row   3
gridWT.add(new Label("Choose the number "
        + "of steps: "), 0, 3);
gridWT.add(nStepsCB, 1,3);

//Row   4
gridWT.add(new Label("Choose the number "
        + "of movers: "), 0, 4);
gridWT.add(nMoversCB, 1,4);

//Row   5
gridWT.add(new Label("Vertical shift of "
        + "the head: "), 0, 5);
gridWT.add(verticalShiftSlider, 1,5);

wormsTab.setContent(gridWT);
tabPane.getTabs().add(wormsTab);
```

```
//============Evolution tab
Tab evolutionTab = new Tab();
evolutionTab.setText("Evolution");




//Number of generations is captured
final ComboBox nGenerationsCB = new ComboBox();
//Options
nGenerationsCB.getItems()
        .addAll(  "5", "10", "20", "50","100",
                "200", "300", "400", "500");
nGenerationsCB.setEditable(false);
nGenerationsCB.setValue(nGenerations);
nGenerationsCB.setOnAction((Event ev) ->
{
    String s1 =
    nGenerationsCB.getSelectionModel()
            .getSelectedItem().toString();
    nGenerations = Integer.parseInt(s1);
    System.out.println(nGenerations);
});


//Number of point mutation per child
final ComboBox nPointMutationsCB = new ComboBox();
//Options
nPointMutationsCB.getItems()
        .addAll( "1", "2", "3", "4", "5","6",
                "7", "8", "9", "10");
nPointMutationsCB.setEditable(false);
nPointMutationsCB.setValue(nMovers);
nPointMutationsCB.setOnAction((Event ev) ->
{
    String s1 =
    nPointMutationsCB.getSelectionModel()
```

```
            .getSelectedItem().toString();
    nPointMutations = Integer.parseInt(s1);
    System.out.println(nPointMutations);
});

//Number of nSourceMovers =
//number of movers that serve as pattern to be cloned
final ComboBox nSourceMoversCB = new ComboBox();
//Options
nSourceMoversCB.getItems()
        .addAll( "1", "2", "3", "4", "5","6",
                "7", "8", "9", "10");
nSourceMoversCB.setEditable(false);
nSourceMoversCB.setValue(nSourceMovers);
nSourceMoversCB.setOnAction((Event ev) ->
{
    String s1 =
    nSourceMoversCB.getSelectionModel()
            .getSelectedItem().toString();
    nSourceMovers = Integer.parseInt(s1);
    System.out.println(nSourceMovers);
});

 //Number of copies of each clone
final ComboBox nCopiesCB = new ComboBox();
//Options
nCopiesCB.getItems()
        .addAll( "1", "2", "3", "4", "5","6",
                "7", "8", "9", "10");
nCopiesCB.setEditable(false);
nCopiesCB.setValue(nCopies);
nCopiesCB.setOnAction((Event ev) ->
{
    String s1 =
    nCopiesCB.getSelectionModel()
            .getSelectedItem().toString();
    nCopies = Integer.parseInt(s1);
    System.out.println(nCopies);
});
```

```
//Type of process
final ToggleGroup group = new ToggleGroup();

//First radio button
RadioButton rb1 = new RadioButton("Randomness");
rb1.setToggleGroup(group);
rb1.setUserData("Randomness");
//Second radio button
RadioButton rb2 =
        new RadioButton("Evolution without hints");
rb2.setToggleGroup(group);
rb2.setUserData("Evolution without hints");
typeOfProcess = "Evolution without hints";
rb2.setSelected(true);
//Third radio button
RadioButton rb3 =
        new RadioButton("Evolution with hint1 "
        + "= gene amplification");
rb3.setToggleGroup(group);
rb3.setUserData("Evolution with hint1"
                + " = gene amplification");
//Fourth radio button
RadioButton rb4 =
        new RadioButton("Evolution with hint2"
                                + " = move(1,1)");
rb4.setToggleGroup(group);
rb4.setUserData("Evolution with hint2"
                                + " = move(1,1)");
//Fifth radio button
RadioButton rb5 =
        new RadioButton("Evolution "
                + "+ hint 1 +  hint2 ");
rb5.setToggleGroup(group);
rb5.setUserData("Evolution "
                + "+ hint 1 +  hint2 ");
//Sixth radio button
RadioButton rb6 =
        new RadioButton("Comparison");
rb6.setToggleGroup(group);
```

```
rb6.setUserData("Comparison");
rb6.setDisable(true);
//Sixth radio button
RadioButton rb7 =
        new RadioButton("Human design");
rb7.setToggleGroup(group);
rb7.setUserData("Human design");



//Listener
group.selectedToggleProperty()
    .addListener((ObservableValue<?
        extends Toggle> ov, Toggle t1, Toggle t2) ->
    {
        RadioButton chk = (RadioButton)t2
            .getToggleGroup().getSelectedToggle();
        if (print3)
            System.out.println("Selected Radio Button = "
            + chk.getText());

        if (chk.getText().contains("Randomness"))
        {
            System.out.println("Randomness");
            typeOfProcess = "Randomness";
        }
        else if
           (chk.getText().contains(
                   "Evolution without hints"))
        {
            System.out.println(
                    "Evolution without hints");
            typeOfProcess = "Evolution without hints";

        }
        else if
            (chk.getText().contains(
                    "Evolution with hint1 "
                    + "= gene amplification"))
        {
```

```
    System.out.println(
            "Evolution with hint1 "
            + "= gene amplification");
    typeOfProcess =  "Evolution with hint1 "
            + "= gene amplification";

}
else if
    (chk.getText().contains(
            "Evolution with hint2"
                    + " = move(1,1)"))
{
    System.out.println(
            "Evolution with hint2 "
            + "=  move(1,1)");
    typeOfProcess =  "Evolution with hint2"
                    + " = move(1,1)";
}
else if
    (chk.getText().contains(
           "Evolution "
        + "+ hint 1 +  hint2 "))
{
    System.out.println(
            "Evolution "
        + "+ hint 1 +  hint2 ");
    typeOfProcess =  "Evolution "
        + "+ hint 1 +  hint2 ";
}
else if
    (chk.getText().contains(
           "Comparison"))
{
    System.out.println(
            "Comparison");
    typeOfProcess =  "Comparison";
}
else if
    (chk.getText().contains(
           "Human design"))
```

```
            {
                System.out.println(
                        "Human design");
                typeOfProcess =  "Human design";
            }


        });

    //Booking
    VBox vbox = new VBox();
    vbox.getChildren().add(rb1);
    vbox.getChildren().add(rb2);
    vbox.getChildren().add(rb3);
    vbox.getChildren().add(rb4);
    vbox.getChildren().add(rb5);
    vbox.getChildren().add(rb6);
    vbox.getChildren().add(rb7);
    vbox.setSpacing(5);
    //Presentation
    vbox.setPadding(new Insets(20, 10, 10, 20));


    /*
    //Alternate disposition
            HBox hbox = new HBox();
    hbox.getChildren().add(rb1);
    hbox.getChildren().add(rb2);
    hbox.getChildren().add(rb3);
    hbox.getChildren().add(rb4);
    hbox.getChildren().add(rb5);
    hbox.getChildren().add(rb6);
    //Presentation
    hbox.setSpacing(5);
    hbox.setPadding(new Insets(20, 10, 10, 20));
    */
```

```
//GridPane  of wormsTab
GridPane gridET = new GridPane();
gridET.setVgap(4);
gridET.setHgap(10);
gridET.setPadding(new Insets(5, 5, 5, 5));
//Row   0
gridET.add(new Label("Choose the number "
        + "of  generations: "), 0, 0);
gridET.add(nGenerationsCB, 1, 0);

//Row   1
gridET.add(new Label("Choose the number "
        + "of  point mutations: "), 0, 1);
gridET.add(nPointMutationsCB, 1, 1);

//Row   2
gridET.add(new Label("Choose the number of  source "
        + "movers  in gene amplification: "), 0, 2);
gridET.add(nSourceMoversCB, 1, 2);

//Row   3
gridET.add(new Label("Choose the number of  copies "
        + "in gene amplification: "), 0, 3);
gridET.add(nCopiesCB, 1, 3);

//Row   4
gridET.add(new Label("Choose the type "
        + "of process: "), 0, 4);
gridET.add(vbox, 0, 5);


evolutionTab.setContent(gridET);
tabPane.getTabs().add(evolutionTab);



//============Fossils tab
Tab fossilsTab = new Tab();
fossilsTab.setText("Fossils");
```

```
//Number of sampled fossils is captured
final ComboBox nSampledFossilsCB = new ComboBox();
//Options
nSampledFossilsCB.getItems()
        .addAll(  "5", "10", "20", "50","100",
                "200", "300");
nSampledFossilsCB.setEditable(false);
nSampledFossilsCB.setValue(nSampledFossils);
nSampledFossilsCB.setOnAction((Event ev) ->
{
    String s1 =
    nSampledFossilsCB.getSelectionModel()
            .getSelectedItem().toString();
    nSampledFossils = Integer.parseInt(s1);
    System.out.println(nSampledFossils);
});


//GridPane  of fossilsTab
GridPane gridFT = new GridPane();
gridFT.setVgap(4);
gridFT.setHgap(10);
gridFT.setPadding(new Insets(5, 5, 5, 5));
//Row   0
gridFT.add(new Label("Choose the number of  "
        + "fossils: "), 0, 0);
gridFT.add(nSampledFossilsCB, 1, 0);



fossilsTab.setContent(gridFT);
tabPane.getTabs().add(fossilsTab);


//Report tab
//Contains the blackboard of the application
Tab reportTab = new Tab();
reportTab.setText("Go + Report ");
```

```
ScrollPane scrollPane = new ScrollPane();

//Show scrollBars?
scrollPane.setHbarPolicy(ScrollBarPolicy.NEVER);
scrollPane.setVbarPolicy(ScrollBarPolicy.ALWAYS);
//Height of scrollPane
scrollPane.setPrefHeight(400);

//Pane to place a console
flowPane = new FlowPane();
flowPane.setPadding(new Insets(5, 0, 5, 0));
flowPane.setVgap(4);
flowPane.setHgap(4);
//One column
flowPane.setPrefWrapLength(100);
//Color for background = Pale blue
flowPane.setStyle("-fx-background-color: DAE6F3;");

 Button copyPasteBtn = new Button();
copyPasteBtn.setText("You can copy  output to"
        + " the clipboard and paste it into a document."
        + "\nText: click + control + A (or E) + C."
        + "\nGraphics: Accommodate each one "
        + "and click corresponding button.");
//position
copyPasteBtn.setLayoutX(400);
copyPasteBtn.setLayoutY(40);
copyPasteBtn.setOpacity(0);
flowPane.getChildren().add(copyPasteBtn);


//Listener of OK button
goBt.setOnAction((ActionEvent e) ->
{
    //Date of experiment is reported
    Calendar cal = Calendar.getInstance();
    String format = "yyyy-MM-dd HH:mm:ss";
    SimpleDateFormat sdf = new SimpleDateFormat(format);
    String startTime =  sdf.format(cal.getTime());
```

```
        console.appendText("Program L97 WormsEvoJFX4 "
                    + "\nDate of experiment = " + startTime);
        //CopyPaste button is made visible
        copyPasteBtn.setOpacity(1);
        //OK button gets unvisible
        goBt.setOpacity(0);
        //Parameters are reported
        appendParameters();
        //Simulation begins
        go();
        Calendar cal2 = Calendar.getInstance();
        String endTime =  sdf.format(cal2.getTime());
        console.appendText("\n\nExperiment ended by = "
                + endTime);
    });
    //OK button is booked
    flowPane.getChildren().add(goBt);



    //Console to write output
    console = new TextArea ("");
    //Unmodifiable
    console.setEditable(false);
    //Booking
    flowPane.getChildren().add(console);


    //Line chart for the generation at which
    //a new link was added.
    //Declaration of x and y axes
    CategoryAxis xAxis1 = new CategoryAxis();
    NumberAxis yAxis1 = new NumberAxis();
    xAxis1.setLabel("New link");
    yAxis1.setLabel("Generation");
    //Instantiation
    newLinkChart = new LineChart(xAxis1, yAxis1);
    //Size
    newLinkChart.setPrefSize(500, 330);
```

```
//Trigger of screeshot
Button saveLinkBtn = new Button("Save newLinkChart "
        + "to clipboard");
//Listener
saveLinkBtn.setOnAction((ActionEvent t) ->
{
    try
    {
        //Rectangle for screenshot is gotten
        Bounds b = newLinkChart.getBoundsInParent();
        int x = (int)Math.round(stage.getX()
                + outerPane.getLayoutX()   );
        int y = (int)Math.round(stage.getY()
                + outerPane.getLayoutY() + 61);
        int w = (int)Math.round(b.getWidth());
        int h = (int)Math.round(b.getHeight() + 28);
        //Multipurpose tool
        java.awt.Robot robot = new java.awt.Robot();
        //Rectangle for screenshot
        java.awt.image.BufferedImage bi
            = robot.createScreenCapture
                (new java.awt.Rectangle(x, y, w, h));
        //Declaration of type of data
        java.io.ByteArrayOutputStream stream
                = new java.io.ByteArrayOutputStream();
        //Formatting
        ImageIO.write(bi, "png", stream);
        //Image is created as image of screenshot
        Image image = new Image(new java
            .io
            .ByteArrayInputStream(
                stream.toByteArray()), w, h, true, true);
        //Declaration of clipboard
        ClipboardContent cc = new ClipboardContent();
        //Clipboard is loaded with image
        cc.putImage(image);
        //Loading is finished
```

```
            Clipboard.getSystemClipboard().setContent(cc);
        }   //In case of problems
        catch (AWTException | IOException ex)
            {
                Logger.getLogger(WormsEvoJFX4
                .class.getName()).log(Level.SEVERE, null, ex);
            }
});
//Booking
flowPane.getChildren().add(newLinkChart);
//vertical space in withing items
flowPane.setVgap(25);
//Booking
flowPane.getChildren().addAll( saveLinkBtn);



//BarChart for the quality of fossils:
//Perfect fossils can climb, imperfect not.
CategoryAxis xAxis2 = new CategoryAxis();
NumberAxis yAxis2 = new NumberAxis();
xAxis2.setLabel("Quality of reconstructed fossils ("
        + nSampledFossils + ")");
yAxis2.setLabel("Absolute frequency");
//Instantiation
qualityChart = new BarChart(xAxis2, yAxis2);

//Size
qualityChart.setPrefSize(500, 330);

flowPane.getChildren().add(qualityChart);



//Screenshot for qualityChart
Button saveQualityBtn = new Button("Save "
        + "qualityChart to clipboard");
saveQualityBtn.setOnAction((ActionEvent t) ->
```

```
    {
        try
        {
            //Rectangle for screenshot is gotten
            Bounds b = newLinkChart.getBoundsInParent();
            int x = (int)Math.round(stage.getX()
                    + outerPane.getLayoutX()   );
            int y = (int)Math.round(stage.getY()
                    + outerPane.getLayoutY() + 61);
            int w = (int)Math.round(b.getWidth());
            int h = (int)Math.round(b.getHeight() + 28);
            //Multipurpose tool
            java.awt.Robot robot = new java.awt.Robot();
            //Rectangle for screenshot
            java.awt.image.BufferedImage bi
                    = robot.createScreenCapture
                        (new java.awt.Rectangle(x, y, w, h));
            //Declaration of type of data
            java.io.ByteArrayOutputStream stream
                    = new java.io.ByteArrayOutputStream();
            //Formatting
            ImageIO.write(bi, "png", stream);
            //Image is created as image of screenshot
            Image image = new Image(new java
                    .io
                    .ByteArrayInputStream(
                    stream.toByteArray()), w, h, true, true);
            //Declaration of clipboard
            ClipboardContent cc = new ClipboardContent();
            //Clipboard is loaded with image
            cc.putImage(image);
            //Loading is finished
            Clipboard.getSystemClipboard().setContent(cc);
        }   //In case of problems
        catch (AWTException | IOException ex)
            {
                Logger.getLogger(WormsEvoJFX4
                .class.getName()).log(Level.SEVERE, null, ex);
            }
    });
```

```
flowPane.getChildren().addAll( saveQualityBtn);




ObservableList<PieChart.Data> qualityPieChartData =
            FXCollections.observableArrayList(
            new PieChart.Data("Perfect", 20),
            new PieChart.Data("Imperfect", 20));

qualityPieChart = new PieChart(qualityPieChartData);
qualityPieChart.setPrefSize(500, 330);
flowPane.getChildren().add(qualityPieChart);
final Label caption = new Label("");
caption.setTextFill(Color.DARKORANGE);
caption.setStyle("-fx-font: 24 arial;");




//Screenshot for qualityChart
Button saveQualityPieBtn
    = new Button("Save qualityPieChart to clipboard");
saveQualityPieBtn.setOnAction((ActionEvent t) ->
{
    try
    {
        //Rectangle for screenshot is gotten
        Bounds b = newLinkChart.getBoundsInParent();
        int x = (int)Math.round(stage.getX()
                + outerPane.getLayoutX()   );
        int y = (int)Math.round(stage.getY()
                + outerPane.getLayoutY() + 61);
        int w = (int)Math.round(b.getWidth());
        int h = (int)Math.round(b.getHeight() + 28);
        //Multipurpose tool
        java.awt.Robot robot = new java.awt.Robot();
        //Rectangle for screenshot
        java.awt.image.BufferedImage bi
```

```
                = robot.createScreenCapture
                    (new java.awt.Rectangle(x, y, w, h));
            //Declaration of type of data
            java.io.ByteArrayOutputStream stream
                    = new java.io.ByteArrayOutputStream();
            //Formatting
            ImageIO.write(bi, "png", stream);
            //Image is created as image of screenshot
            Image image = new Image(new java
                    .io
                    .ByteArrayInputStream(
                    stream.toByteArray()), w, h, true, true);
            //Declaration of clipboard
            ClipboardContent cc = new ClipboardContent();
            //Clipboard is loaded with image
            cc.putImage(image);
            //Loading is finished
            Clipboard.getSystemClipboard().setContent(cc);
        }   //In case of problems
        catch (AWTException | IOException ex)
            {
                Logger.getLogger(WormsEvoJFX4
                .class.getName()).log(Level.SEVERE, null, ex);
            }
    });
    flowPane.getChildren().addAll( saveQualityPieBtn);




    //Filling and booking
    scrollPane.setContent(flowPane);
    reportTab.setContent(scrollPane);
    tabPane.getTabs().add(reportTab);


    //======
```

```
    //Welcome button
    Button welcomeBtn = new Button();
    welcomeBtn.setText("Set parameters and "
            + "clik  over the Go button, "
            + "under the Go + Report tab, "
            + "to begin simulation."
            + "\nTo confirm reading, click over here.");
    //Position
    welcomeBtn.setLayoutX(400);
    welcomeBtn.setLayoutY(40);
    //Listener
    welcomeBtn.setOnAction((ActionEvent event) ->
    {
        welcomeBtn.setOpacity(0);
        //GUI is made visible
        outerPane.setOpacity(1);
    });
    //Booking
    p.getChildren().add(welcomeBtn);

    //=========TABPANE
    //TabPane is booked and  positioned
    outerPane.setCenter(tabPane);
    //OuterPane is booked
    p.getChildren().add(outerPane);



    //================================
    //======== END OF GUI=============
    //================================


}


//Start simulation
private static void go()
{
    //Text to be drawn
```

```
genCount = new Text ("");
genCount.setLayoutX(40);genCount.setLayoutY(60);
p.getChildren().addAll(genCount);
evolutionST.setCycleCount(1);

//Evolution for nGenerations
for(int i = 0; i < nGenerations; i++)
{
    //The scene is void
    voidOccupationMatrix();
    //Generation
    gen = i;
    //if (print2)
        System.out.println("\nGen = " + gen);
    //Generation number is drawn
    Timeline timeline = drawGen(gen);
    evolutionST.getChildren().addAll( timeline );
    //Worms are created and posited at the start line
    startLine(typeOfProcess);
    //Every worm leaves a fossil
    fossilization();
    //Worms express their information to move
    SequentialTransition oneGenST = oneGeneration();
    evolutionST.getChildren().add(oneGenST);
    //Drawn worms get hidden
    ParallelTransition hideST  =  hideST();
    evolutionST.getChildren().add(hideST);
    //Helping array
    Worm[] file1 = new Worm[50];
    //Information is recorded in an array
    OldWorms = Worms.toArray(file1);

    //Destroy everything
    //for the sake of a new beginning
    Worms.clear();
    WormsTwo.clear();

    //Success?
    Integer genI = gen;
    if (bullseye)
```

```
        {
            //If climbing was successful, add a new link
             nLinks = nLinks+1;

            //List of successful generations
            bullseyes.add(genI);
        }

    }
    //EvolutionST is booked
    allST.getChildren().add(evolutionST);

    console.appendText("\n\n====OUTPUT===== ");
    console.appendText("\n\n=SUCCESSFUL GENERATIONS= ");
    Integer size = bullseyes.size();
            for(Integer index = 0; index < size; index++)
            {
    console.appendText("\n" + bullseyes.get(index));

            }
    //Data for newLinkChart is provided
    newLinkChart.setData(getNewLinkChartData());
newLinkChart.setTitle("Cost of complexity under \n"
            + typeOfProcess);



  //Fossil study
    SequentialTransition fsST = fossilSampling();
    //Data for qualityChart of fossils is provided
    qualityChart.setData(getQualityChartData() );
    qualityChart.setTitle("Quality of fossils under \n"
            + typeOfProcess);


    ObservableList<PieChart.Data> qualityPieChartData =
        FXCollections.observableArrayList(
        //1 is added in case nPerfect = 0
        new PieChart.Data("Perfect", 10000*nPerfect+1),
        new PieChart.Data("Imperfect", 10000*nImperfect));
```

```
    qualityPieChart.setData(qualityPieChartData);

    qualityPieChart.setTitle("Quality of fossils under \n"
            + typeOfProcess
    + "\nPerfect = " + nPerfect
    + "    Imperfect = " + nImperfect);

    allST.getChildren().add(fsST);
    //Film is played
    allST.play();
}

//Number of items per page
private int itemsPerPage()
{
    return 1;
}

//Pieces of information are associated to pages
private VBox createPage(int pageIndex)
{
    VBox box = new VBox(1);
    int page = pageIndex * itemsPerPage();
    for (int i = page; i < page + itemsPerPage(); i++)
    {
        TextArea text = new TextArea(textPages[i]);
        text.setPrefSize(20, 270);
        text.setFont(Font.font(18-i));
        text.setWrapText(true);
        box.getChildren().add(text);
    }
return box;
}



//Data for a newLinkChart is fabricated.
//It must be a map that associates a number
//to a string.
private static ObservableList<XYChart.Series<String, Double>>
```

```
    getNewLinkChartData()
    {
        //Data is packed into an ObservableList
        ObservableList<XYChart.Series<String, Double>>
            map = FXCollections.observableArrayList();
        XYChart.Series<String, Double> series
                        = new XYChart.Series<>();

        series.setName("Generation at which a new link "
                + "was added  during " + nGenerations
                + " generations");

        //Fabrication of data:
        //It is a series of couplets.
        //First item = index of new link.
        //Second item = generation at which link was added
        Integer size = bullseyes.size();
        for(Integer index = 0; index < size; index++)
        {

            series.getData().add(new XYChart.Data
                    (Integer.toString(index),
                            bullseyes.get(index)));
        }
        map.addAll(series);
        return map;
    }

//Data for qualityChart is fabricated.
//It must be a map that associates a number
//to a string.
private static ObservableList<BarChart.Series<String, Double>>
    getQualityChartData()
    {
        //Data is packed into an ObservableList
        ObservableList<BarChart.Series<String, Double>>
            map = FXCollections.observableArrayList();
        BarChart.Series<String, Double> series
                        = new BarChart.Series<>();
```

```
        series.setName("Absolute frequency");

        //Fabrication of data:
        //It is a series of couplets.
        //First item = quality descriptor.
        //Second item = absolute frequency

        series.getData().add(new BarChart.Data
                ("Perfect",
                        nPerfect));
        series.getData().add(new BarChart.Data
                ("Imperfect",
                        nImperfect));
        System.out.println("\nnPerfect = " + nPerfect
        + "\n nImperfect = " + nImperfect);
        map.addAll(series);
        return map;
    }



//Parameters are published on the console
private static void appendParameters()
{
    console.appendText("\n\n====PARAMETERS====");

    console.appendText("\n\n=WORMS= ");
    console.appendText("\nNumber of worms per "
            + "generation = " + nWorms);
    console.appendText("\nNumber of links = " + nLinks);
    console.appendText("\nWidth of links = "
            + widthOfLinks);
    console.appendText("\nHeight of links = "
            + heightOfLinks);
    console.appendText("\nInner elasticity = "
            + innerElasticity);
    console.appendText("\nBorder elasticity = "
            + borderElasticity);
    console.appendText("\nNumber of steps = " + nSteps);
    console.appendText("\nNumber of movers = " + nMovers);
```

```
    console.appendText("\nInitial row of the head = "
            + initialRow);

    console.appendText("\n\n=EVOLUTION=");
    console.appendText("\nNumber of  generations = "
            + nGenerations);
    console.appendText("\nNumber of point mutations = "
            + nPointMutations);
    console.appendText("\nNumber of source movers in "
            + "gene amplification = " + nSourceMovers);
    console.appendText("\nNumber of  copies in "
            + "gene amplification = " + nCopies);
    console.appendText("\nType of process = "
            + typeOfProcess);

    console.appendText("\n\n =FOSSILS=");
    console.appendText("\nNumber of fossils = "
            + nSampledFossils);



}



// To avoid overwriting of worms,
// an occupation record is kept:
// The screen is divided in cells width x hight.
// At the beginning every cell is empty=false.
// Occupancy is recorded in isOccupied[][],
// which is a boolean matrix.
// A boolean  variable can be
// true= occupied,  else false= not occupied.
private static void  voidOccupationMatrix()
{
   //System.out.println("occupationMatrix");
    for(int i = 0; i < 600; i++)
    for(int j = 0; j < 300; j++)
        isOccupied[i][j]= false;
}
```

```
//Nine basic moves = directions where to move in
static private void moveDef()
{
    final Move move0 =  w.new Move(0,1);
       vectMoves[0]   = move0 ;
    final Move move1 =  w.new Move(1,1);
       vectMoves[1]   = move1 ;
    final Move move2 =  w.new Move(1,0);
       vectMoves[2]   = move2 ;
    final Move move3 =  w.new Move(1,-1);
       vectMoves[3]   = move3 ;
    final Move move4 =  w.new Move(0,-1);
       vectMoves[4]   = move4 ;
    final Move move5 =  w.new Move(-1,-1);
       vectMoves[5]   = move5 ;
    final Move move6 =  w.new Move(-1,0);
       vectMoves[6]   = move6 ;
    final Move move7 =  w.new Move(-1,1);
       vectMoves[7]   = move7 ;
    final Move move8 =  w.new Move(0,0);
       vectMoves[8]   = move8 ;
}


static private Worm chooseGenerator(int W, String process)
{
    //Auxiliary worm
    Worm   worm = null;

    //Generate a worm at random
    if (process.equals("Randomness"))
        worm = w.new Worm();
    //Copy worm W from old generation
    if (process.equals("Evolution without hints"))
        worm = w.new Worm(W);
    //Copy worm W from old generation
    if (process.equals("Evolution with hint1 "
                       + "= gene amplification"))
        worm = w.new Worm(W);
```

```
    //Copy worm W from old generation
    if (process.equals("Evolution with hint2"
                                + " = move(1,1)"))
        worm = w.new Worm(W);
    //Copy worm W from old generation
    if (process.equals("Evolution "
                  + "+ hint 1 +  hint2 "))
        worm = w.new Worm(W);
    //Moves are designed by human intelligence
    if (process.equals("Human design"))
    {
        String HD = "HD";
        nGenerations = 5;
        nMovers = nLinks;
        worm = w.new Worm(HD);
    }
    return worm;
}


//All worms are posited at the same initial row
//in vertical position.
static private void startLine(String process)
{
    for (int W = 0; W < nWorms; W++)
    {
        int  x,y;
        //Auxiliary worm
         Worm   worm;
         //Worms are created at generation 0
         if (gen == 0)
         {
             if (process.equals("Human design"))
                worm = chooseGenerator(W, process);
             else worm = w.new Worm();
         }
         else //worms are inherited
             worm = chooseGenerator(W, process);

        //Each link is positioned
```

```
        for(int  link = 0; link < nLinks; link++)
        {
            //Horizontal position
            x = 140 + 140 * W;

            //Vertical position
            y = initialRow + heightOfLinks*link;

            //Position is updated
            worm.Rect.get(link).setX(x);
            worm.Rect.get(link).setY(y);

            //Link is shown
            ParallelTransition show
                        = showPT(worm.Rect.get(link));
            //Link is booked
            p.getChildren().add(worm.Rect.get(link));
            //Link is grouped
            allRects.add(worm.Rect.get(link));
            //Show-transition is booked
            evolutionST.getChildren().add(show);
        }
        //worm is located at position W in lists Worms
        Worms.add(worm);

        if (print1)
            System.out.println("W = " + W + " nLinks = "
                + WormsTwo.get(W).numberOfLinks  );
    }

}



//Draw the generation number
private static Timeline drawGen(int gen)
{
    Integer genI = gen;
    String message =  "Generation  = " +
```

```
                genI.toString();
    //A JavaFX event committed to print a text is generated
    EventHandler printText =
        (EventHandler<ActionEvent>)
            (ActionEvent t ) ->
                {
                    genCount.setText(message);
                };
    //Presentation of printText must be
    //synchronized with the overall film so,
    //duration d is specified
    KeyFrame keyFrame = new KeyFrame(d, printText);

    Timeline timeline = new Timeline();
    timeline.setCycleCount(1);
    //add the keyframe to the timeline
    timeline.getKeyFrames().add(keyFrame);

    return timeline;
}



//Is move m disruptive for the worm?
//(A worm is and must remain as a whole,
//nevertheless, some elasticity is allowed.)
private static boolean isDisruptive(int m, Worm worm )
{
    //A move is disruptive if the target address
    //is farther than certain distance
    //from the neighboring links.
    boolean isDisruptive = false;
    int  link;
    int  x,y,  x1,y1, x2,y2, x3,y3;
    double d1;

    double  d2;

    //Read the link to be moved
```

```
link = worm.Mover.get(m).link;
//Read the   address of the link;
x  = (int) worm.Rect.get(link).getX();
y  = (int) worm.Rect.get(link).getY();
//The   move to be done is read
Move move = w.new Move(worm.Mover.get(m).move);

//The target address is calculated
x1 = x + widthOfLinks*move.x;
y1 = y -  widthOfLinks*move.y;

//The addresses of the two neighboring links are read
//The disruption is calculated
//Case: intermediate link
if ((0 < link) & (link < nLinks-1))
{
    x2 = (int) worm.Rect.get(link-1).getX();

    y2 = (int) worm.Rect.get(link-1).getY();

     d1 =    pow(  ( pow( (x2 - x1),2)
                        +  pow((y2 - y1),2)),0.5);

    x3 = (int) worm.Rect.get(link+1).getX();
    y3 = (int) worm.Rect.get(link+1).getY();
    d2 =  pow(  (pow( (x3 - x1),2)
                        + pow((y3 - y1),2)),0.5);


    // Here we decide whether or not
    // a move is disruptive.
    if ((d1 > innerElasticity * widthOfLinks)
            || (d2 > innerElasticity * widthOfLinks))
               isDisruptive = true;

 }

//First link
if (link == 0)
{
```

```
        x2 = (int) worm.Rect.get(1).getX();
        y2 = (int) worm.Rect.get(1).getY();
        d1 = pow( (pow( (x2 - x1),2) + pow((y2 - y1),2)),0.5);
        if( d1 > borderElasticity * widthOfLinks)
            isDisruptive = true;
     }
     //Last link
    if( link == nLinks-1 )
    {
        x2 = (int) worm.Rect.get(nLinks - 2).getX();
        y2 = (int) worm.Rect.get(nLinks - 2).getY();
        d2 = pow( (pow( (x2 - x1),2)
                + pow((y2 - y1),2)),0.5);
        if ( d2 > borderElasticity *widthOfLinks )
            isDisruptive =   true;
    }
    return isDisruptive;
 }



//Execute move m for worm W.
//If it is  disruptive, undo it.
private static SequentialTransition
                    exeIf( Worm worm, int m)
{
    if (print1)
        System.out.println(" exeIf");
    //Read the link to be moved
    int link = worm.Mover.get(m).link;
    //Read position
    int x = (int) worm.Rect.get(link).getX();
    int y = (int) worm.Rect.get(link).getY();

    //The move to be done is read
    Move  move;
    move = worm.Mover.get(m).move;
    //Move is interpreted
    int byX = widthOfLinks* move.x;
```

```
int byY = -heightOfLinks* move.y;
//Target address is calculated:
int x1 = x + byX;
int y1 = y + byY;
if (x1 < 0 ) x1 = 0;
if (y1 < 0 ) y1 = 0;
if (print1)
    System.out.println(
        " move = " + m   + " link = " + link
        + " \n x = " + x + " y = " + y
        + " \nbyX = " + byX  + " byY = " + byY);


//Occupied cell
int i = (int)  x / widthOfLinks;
int j = (int) y / heightOfLinks;


 //Target cell
int k =(int) x1/widthOfLinks;
int l = (int) y1/heightOfLinks;


//Move is always executed
SequentialTransition moveRect
        = new SequentialTransition();

TranslateTransition translateTransition =
     new TranslateTransition(d, worm.Rect.get(link));
//Jump is executed
translateTransition.setByX(byX);
translateTransition.setByY(byY);
//Number of repetitions
translateTransition.setCycleCount(1);
moveRect.getChildren().add(translateTransition);



//Is the move disruptive?
boolean   isMoveDisruptive =  isDisruptive(m,worm);
```

```
    if (print1)
    {
        if (isMoveDisruptive)
            System.out.println("move is Disruptive");
        else
            System.out.println("move is not  Disruptive");
    }
    //Disruptive moves are undone
    if  (((isOccupied[k][l])) || ((isMoveDisruptive)))
    {
        byX = -byX;
        byY = -byY;
    }
    else //move is not disruptive -> it is not undone
    {
        byX = 0;
        byY = 0;
        worm.Rect.get(link).setX(x1);
        worm.Rect.get(link).setY(y1);
        isOccupied[i][j]= false;
        isOccupied[k][l]= true;
    }




    //Move is possibly undone
    TranslateTransition translateTransition2 =
         new TranslateTransition(d, worm.Rect.get(link));
    //Jump is programmed
    translateTransition2.setByX(byX);
    translateTransition2.setByY(byY);
    //Number of repetitions
    translateTransition2.setCycleCount(1);
    moveRect.getChildren().add(translateTransition2);

    return moveRect;
}
```

```
// Every worm is assigned a fitness
//The nearer to the top, the fitter.
//If climbing has been significative,
//the complexity of the game rises up one level:
//a true bullseye is reported (to add
//next a new link to all worms).
private static void assessFitness()
{
    int yOfTail;
    double yOfHead;

    bullseye  = false;
    for(int W = 0; W <  nWorms; W++)
    {
        // We get the y-coordinate of the tail
        yOfTail = (int) Worms.get(W)
                            .Rect.get(nLinks-1).getY();
        yOfHead =  Worms.get(W).Rect.get(0).getY();
        //Fitness measures how much the head has  been lifted
        Fitness[W] = (yWin - yOfTail);
        if(print1)
            System.out.println("Fitness of worm "
                    + W + " = " + Fitness[W]
                    + " head = " + yOfHead);
        //If worm W is good for climbing
        //its head must be high enough
        if (yOfHead < initialRow - 2 * heightOfLinks)
        {
            //Worms that climb are rewarded
            Fitness[W] = 100 * Fitness[W];
            bullseye = true;
            if (printReproduction)
            System.out.println(
                    "There was advance W = " + W );
        }
    }
}
```

```
//Worms express their information to move.
private  static  SequentialTransition  express()
{
    //Overall process
    SequentialTransition expressST
                  = new SequentialTransition();
     expressST.setCycleCount(1);
     //Worms advance nSteps
    for (int i = 0; i < nSteps; i++)
    {
        //Worms advance one step:
        //To move all links, each worm must execute
        //instructions to move
        SequentialTransition oneStepST
                       = new SequentialTransition();
        oneStepST.setCycleCount(1);
        //Information to move is read and decoded
        for (int m = 0; m < nMovers; m++)
        {
            if (print1)
                System.out.println("nMovers = " + nMovers);
            //All worms execute move m
            ParallelTransition moveMPT
                        = new ParallelTransition();
            moveMPT.setCycleCount(1);
            //This is a parallel loop for a list
            for (Worm worm: Worms)
            {
                //Direction is flipped in human design
                if (typeOfProcess.contains("Human design"))
                    worm.Mover.get(m).move.x
                            = - worm.Mover.get(m).move.x;
                //A translation of some link
                //is programmed for worm W
                //according to move m.
                //If it is not disruptive,
                //it is executed
                SequentialTransition tT = exeIf(worm,m);
                tT.setCycleCount(1);
                //Translation is booked
```

```
                moveMPT.getChildren().add(tT);
            }
            //The ensamble of translations is booked
            oneStepST.getChildren().add(moveMPT);
        }
    //The movement of worms in one step is booked
    expressST.getChildren().add(oneStepST);
    }
    return expressST;
}



    //A hidden rect gets shown
    private static
        ParallelTransition showPT(Rectangle rect)
    {
        ParallelTransition s = new ParallelTransition();
        Duration d2 =   Duration.millis(1);
        //Rect is enlarged
        KeyValue keyValue3 =
            new KeyValue(rect.widthProperty(),widthOfLinks,
                Interpolator.LINEAR);
        KeyValue keyValue4 =
            new KeyValue(rect.heightProperty(),heightOfLinks,
                Interpolator.LINEAR);
        //KeyValues are  inserted into a timeline
        Timeline b = new Timeline(
            new KeyFrame(d2,keyValue3,keyValue4 ));
        //Timeline is inserted into a transition
        s.getChildren().addAll(b);

        return s;
    }
```

```
//Worms are hidden after their performance.
private static ParallelTransition  hideST()
{

    ParallelTransition hidePT
                = new ParallelTransition();
    hidePT.setCycleCount(1);
    Duration d2 =   Duration.millis(1);
    //Hide all rects
    for(Rectangle rect: allRects)
    {
       //Rects are made unvisible
        KeyValue keyValue =
                new KeyValue(rect
                        .visibleProperty(), false
                        ,Interpolator.LINEAR);
        Timeline b = new Timeline(
                new KeyFrame(d2,keyValue));
        hidePT.getChildren().add(b);
    }

    return hidePT;
}




//====================================
//=========Evolutionary part==========
//====================================


//One generation = expression + assessement
//+ reproduction, recombination and mutation.
private static   SequentialTransition  oneGeneration()
{
    SequentialTransition oneGenST
            = new SequentialTransition();
    //Worms express their information to move
```

```
        SequentialTransition expressST = express();
        oneGenST.getChildren().addAll(expressST);
        //Fitness is measured
        assessFitness();
        //Reproduction is proportional to fitness
        reproduction();
        //Population is replaced by a recombinat one
        recombination();
        //Information to move suffers mutation
        mutation();

        return oneGenST;
    }

/*  To generate a worm of the next generation,
an individual is sampled at random
from the old population
and cloned
with a probability that is
proportional to its fitness.
*/
private static void reproduction()
{
    if (printReproduction)
        System.out.println( "\nGeneration = " + gen);
    //Cumulative distribution
    Double cumulate[] = new Double[nWorms];
    // Counters
    int count[] = new int[nWorms];
    //Number of trials
    int nTrials = nWorms;

    //Initialization
    WormsTwo.clear();
    WormsTwo.addAll(Worms);
    for (int i=0; i< nWorms; i++)
        count[i]=0;

    //Cumulative distribution
    cumulate[0] = (double) Fitness[0];
```

```
for (int i=1; i< nWorms; i++)
    cumulate[i]= cumulate[i-1] + Fitness[i];

//Report
if (printReproduction)
{
    System.out.println( "\ni \tfitness cumulate ");

for (int i = 0; i < nWorms; i++)
    System.out.println( i + "\t" + Fitness[i] + " \t "
            + cumulate[i]);
}

//Maximal value
double max = cumulate[nWorms-1];
//Counter of chosen worms
int W = 0;
//System.out.println("Generated numbers: ");
for (int i = 0; i < nTrials; i++)
{
    //A random integer less than max is generated
    //with a uniform distribution
    int n = rd.nextInt((int) max);

    //System.out.println("n =  " + n);
    //Probability of cloning is made
    //in proportion to fitness.
    //Is worm 0 to be chosen?
    if (n < cumulate[0])
    {
        /*
        if (printReproduction)
            System.out.println( "chosen individual  = 0 ");
        */
        WormsTwo.set(W, Worms.get(0));
        count[0]++;
        W++;
    }
    //Is worm l to be chosen?
    for (int l = 1; l < nWorms; l++)
```

```
        {

            if ( ( cumulate[l-1] <= n) & (n < cumulate[l]) )
            {
                /*
                if (printReproduction)
                    System.out.println(
                        "Chosen individual = " + l);
                */
                count[l]++;
                WormsTwo.set(W, Worms.get(l));
                W++;
            }
        }

    }
    if (printReproduction)
    {
        System.out.println("\nEvents and their observed "
            + "and expected frequencies: ");
        for (int i=0; i<nWorms;i++)
            System.out.println(i + "\t" + count[i] + "\t"
                +  Fitness[i]*nTrials/cumulate[nWorms-1]);
    }

    //WormsTwo is copied into Worms
    Worms.clear();
    Worms.addAll(WormsTwo);

}



//Recombination is implemented
private static void recombination()
{

    //Every worm is replaced by a recombinant
    for(int W = 0; W < nWorms; W++ )
    {
```

```
        //A  binary recombinant operator
        //is assembled at random.

        int rec[] = new int[210];
        for (int j = 0; j< nMovers; j++)
  rec[j]= rd.nextInt(2);
//Another worm is chosen at random for recombination
        int m = rd.nextInt(nWorms);
   //Basal initialization of the recombinant
        Worm  worm = Worms.get(W);
for(int j = 0; j< nMovers; j++)
        {
            // Recombination is executed.
            //For 0, take from the partner worm m
            //For 1, take from worm W.
            if (rec[j] == 0)
            worm.Mover.set(j, Worms.get(m).Mover.get(j));
        }

        //Recombinant is memorized
        WormsTwo.set(W, worm);
        if(print2)
        {
            System.out.println("Old move.x = "
                    + Worms.get(W).Mover.get(0).move.x);

            System.out.println("New Rec move.x = "
                    + worm.Mover.get(0).move.x);
        }

    }
    //Old population is replaced by the recombinated one
    Worms.clear();
    Worms.addAll(WormsTwo);
}
```

```
//Blind mutations of infomation to move
private static void moverMutation()
{
    int   m, n;
    //Let us try out blind  mutations
    //in specific Movers,
    //in the  link  to be moved
    //and in the direction of movement.
    for(int W = 0; W < nWorms; W++)
    {
        //Each offspring receives
        //numbPointMutation mutations
        for(int k = 0; k< nPointMutations; k++)
        {
            //Select at random  which mover to mutate:
            m =  rd.nextInt(nMovers );
            //Select at random which link  to move:
            n = rd.nextInt( nLinks) ;
            if(print2)
                System.out.println("Old move.x = "
                     + Worms.get(W).Mover.get(m).move.x);
            //Select at random which direction to move on.
            Move move = vectMoves[rd.nextInt(9) ];
            //Mover m is assembled from link + move
            Worms.get(W).Mover.set(m, w.new Mover(n,move));
            if(print2)
                System.out.println("New Mutation move.x = "
                    + Worms.get(W).Mover.get(m).move.x);
            //Color of link n is changed
            //(possibly various times)
            Worms.get(W).Rect.get(n).setFill(randomColor());
        }
    }
}


//Hint1 = some chosen moves replace others
//in within the same worm
private static void geneAmplification()
{
```

```
    //  System.out.println("geneAmplification");
    int    m,    mm;
    int    link;
    //All worms are scanned
    for(int W = 0; W < nWorms; W++)
    {
        //nSourceMovers are amplified
        for (int i = 0; i < nSourceMovers; i++)
        {
            //Select at random which Mover to be read
            m = rd.nextInt(nMovers);
            //Move to be amplified is read
            Move move =  Worms.get(W).Mover.get(m).move;
            //The Move is amplified
            for(int k = 0; k< nCopies; k++)
            {
                //Select at random the place for replacement:
                //throw a number from 1 to nMovers.
                mm = rd.nextInt(nMovers) ;
                //Link is read
                link = Worms.get(W).Mover.get(mm).link;
                //Mover mm is assembled from link + move
                //and amplified
                Worms.get(W).Mover
                    .set(mm, w.new Mover(link,move));
            }
        }
    }
}


//Hint2: move = Move(1,1) = move northeast
private static void move11()
{
    //  System.out.println("move11");
    int    m,    mm;
    int    link;
    //All worms are scanned
    for(int W = 0; W < nWorms; W++)
    {
```

```
        //nSourceMovers are amplified
        for (int i = 0; i < nSourceMovers; i++)
        {
            //Select at random which Mover to be read
            m = rd.nextInt(nMovers);
            //Move11
            Move move = w.new Move(1,1);
            //The Move is amplified
            for(int k = 0; k< nCopies; k++)
            {
                //Select at random the place for replacement:
                //throw a number from 1 to nMovers.
                mm = rd.nextInt(nMovers) ;
                //Link is read
                link = Worms.get(W).Mover.get(mm).link;
                //Mover mm is assembled from link + move
                //and cloned
                Worms.get(W).Mover
                    .set(mm, w.new Mover(link,move));
            }
        }
    }
}


//Information to move is changed
private static void mutation(   )
{

    //Mutation of movers
    moverMutation();
    //Mutations on request
    if (typeOfProcess.contains("Evolution with hint1 "
                        + "= gene amplification"))
        geneAmplification( );
    if (typeOfProcess.contains("Evolution with hint2"
                               + " = move(1,1)"))
        move11();
    if (typeOfProcess.contains("Evolution "
                    + "+ hint 1 +  hint2 "))
```

```
    {
      geneAmplification( );
      move11();
    }
}


//Worm ->fossil
//All worms leave their fossils
private static void fossilization()
{
    for(int W = 0; W < nWorms; W++)
    {
//Worm W leaves its fossil
        Fossil fossil = w.new Fossil(Worms.get(W));
        Fossils.add(fossil);
if (print1)
            System.out.println("Fossil number = "
    + nFossils
    + " nLinks = "
            + Fossils.get(nFossils).numberOfLinks);
        //Counter
nFossils++;
    }
}

//Fitness of worm is measured.
private static double fitness(Worm worm)
{
    double yOfTail;
    double fitness;
    // We get the y-coordinate of the tail
    yOfTail =  worm.Rect.get(nLinks-1).getY();
    //The nearer to the top of the screen (yOfTail = 0),
    //the fitter:
    fitness  = 100*(yWin - yOfTail);
    if(print1)
        System.out.println("Fitness of fossil = "
                        + fitness);
    //Classification according to perfection
```

```
    //Perfect = can climb
    //Imperfect = cannot climb
    double yOfHead =  worm.Rect.get(0).getY();
    if (yOfHead < initialRow - 2 * heightOfLinks)
    {
       nPerfect++;
       quality = "Perfect";
        if (print3)
        System.out.println( "perfect " );
    }
    else
    {
        nImperfect++;
        quality = "Imperfect";
    }

    return fitness;
}

//Draw the fossil number
private static Timeline drawFossilNumber(int f)
{
    Integer F = f;
    String message =  "Fossil  = " +
                    F.toString();
    //A JavaFX event committed to print a text is generated
    EventHandler printText =
        (EventHandler<ActionEvent>)
            (ActionEvent t ) ->
                {
                    genCount.setText(message);
                };
    //Presentation of printText must be
    //synchronized with the overall film so,
    //duration d is specified
    KeyFrame keyFrame = new KeyFrame(d, printText);

    Timeline timeline = new Timeline();
    timeline.setCycleCount(1);
    //add the keyframe to the timeline
```

```
    timeline.getKeyFrames().add(keyFrame);

    return timeline;
}

//Reconstructed worm is given an initial position
private static ParallelTransition startLine(Worm worm)
{
    voidOccupationMatrix();
    nLinks = worm.numberOfLinks ;
    int x,y;
    ParallelTransition showWorm = new ParallelTransition();
    showWorm.setCycleCount(1);
    for(int  link = 0; link < nLinks; link++)
    {
        //Horizontal position
        x = 140;
        //Vertical position
        y = initialRow + heightOfLinks*link;
        //Position is updated
        worm.Rect.get(link).setX(x);
        worm.Rect.get(link).setY(y);
        //Link is shown
        ParallelTransition showPT
                    = showPT(worm.Rect.get(link));
        //Show-transition is booked
        showWorm.getChildren().add(showPT);
        //Link is booked
        p.getChildren().add(worm.Rect.get(link));
        //Link is grouped to ease hidding
        lastRects.add(worm.Rect.get(link));
    }
    return showWorm;
}


//Worm expresses its movers
private static SequentialTransition
        expressMovers(Worm worm)
{
```

```
    SequentialTransition expressMoversST
                    = new SequentialTransition();
    expressMoversST.setCycleCount(1);
    // d is the number of steps.
    // m is the index of the mover;
    for(int d = 0; d < nSteps ; d++)
    {
        for(int m  = 0 ; m <  worm.numberOfMovers; m ++)
        {
            //Direction is flipped in human design
                if (typeOfProcess.contains("Human design"))
                    worm.Mover.get(m).move.x
                            = - worm.Mover.get(m).move.x;
            //Movers a re expressed
            SequentialTransition exeMovers = exeIf(worm,m);
            exeMovers.setCycleCount(1);
            //Execution of movers is booked
            expressMoversST.getChildren().add(exeMovers);
        }
    }
    return expressMoversST;
}


//Reconstructed fossil  is  hidden after its performance.
    private static ParallelTransition  hideLastRectsST()
    {

        ParallelTransition hidePT
                    = new ParallelTransition();
        hidePT.setCycleCount(1);
        Duration d2 =   Duration.millis(1);
        //Hide all rects
        for(Rectangle rect: lastRects)
        {
            //Rects are made unvisible
            KeyValue keyValue =
                    new KeyValue(rect
                            .visibleProperty(), false
                            ,Interpolator.LINEAR);
```

```
            Timeline b = new Timeline(
                    new KeyFrame(d2,keyValue));
            hidePT.getChildren().add(b);
        }
        return hidePT;
    }


//==============FOSSILS================

//Fossils are sampled and reconstructed
//to measure their functionality
private static  SequentialTransition fossilSampling()
{
    if (print3)
        System.out.println("\nFOSSIL SAMPLING\n");
    console.appendText("\n\n=FOSSIL SAMPLING=\n");
    if (print3)
        System.out.println("Perfection "
            + "= number of links *fitness");
    console.appendText("\nPerfection "
            + "= number of links *fitness");
    if (print3)
        System.out.println("\nFossil number, number of links,"
  + "fitness, perfection\n");
    console.appendText("\nFossil number, number of links, "
  + "fitness, perfection");
    console.appendText("\nTotal number of fossils = "
            + nFossils);
    double fitness;

    SequentialTransition fossilSamplingST
          = new SequentialTransition();
    for(int i = 0; i < nSampledFossils; i++)
    {
        //A fossil is chosen at random
        int f = rd.nextInt(nFossils);
        //Fossil is reconstructed to a worm
        Worm worm =   w.new  Worm(Fossils.get(f));
        //Worm is positioned at the startline
```

```
        ParallelTransition showWorm = startLine(worm);
        fossilSamplingST.getChildren().addAll( showWorm );
        //Generation number is drawn
        Timeline timeline = drawFossilNumber(f);
        fossilSamplingST.getChildren().addAll( timeline );
        //Movers of worm are unfolded
        SequentialTransition expressMoversST
                = expressMovers(worm);
        fossilSamplingST.getChildren().add(expressMoversST);
        fitness = fitness(worm);
        //Drawn rects of worm get hidden
        ParallelTransition hideLastRectsST
                =  hideLastRectsST();
        fossilSamplingST.getChildren().add(hideLastRectsST);
        if (print3)
            System.out.println(f + "\t" + nLinks
            + "\t" + fitness + "\t\t" + fitness*nLinks);

        console.appendText("\n" + f + "\t" + nLinks
            + "\t" + fitness + "\t\t" + fitness*nLinks
            + "\t" + quality);
    }

    double nPerf = nPerfect;
    double nImperf = nImperfect;
    double propPerfect = 100* nPerf/(nPerf + nImperf);
    double propImperfect = 100* nImperf/(nPerf + nImperf);
    console.appendText("\nPerfect fossils = " + nPerfect
            + " (" + propPerfect + ")%" );
    console.appendText("\nImperfect fossils = " + nImperfect
            + " (" + propImperfect + "%)" );

    return fossilSamplingST;
}



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

```
        Application.launch(args);
}


} //End of main class L97 WormsEvoJFX4
```

98 ***Exercise.*** *Run the program and play with the code.*

99 ***Important exercise.*** *The simulation includes the accepted by us fact that evolution can create complexity and is addressed to tackle two questions:*

1. Which is the price that evolution must pay for complexity?

2. Does imperfect fossils appear in fossil sampling and in which proportion?

The answers to these questions, whatever they could be, are of maximal importance because they have the possibility to represent a fierce rejection to the Evolutionary Theory that predicts a huge amount a imperfect fossils. So, it is important to make sure that the code is correct, i.e., that it fairly and sharply does what it promises. Study the code and its output enough to confidently take a decision about this and make your best to support it. Answer

6.2 Improving the code

Our previous program offers various possibilities for improvement. Some are realizable just now but others need more know-how and shall postponed.

100 ***Exercise.*** *We have various realizable tasks related to the GUI. Add to the previous program the possibility to 1) Graduate the horizontal distance in within worms. 2) Graduate the duration of translation of links. 3) Take a screenshot of the worms 4) Regulate purposefully an coherently the enabling-disabling of the diverse items of the GUI. In relation with the code, simplify the part corresponding to screenshots: it is repeated various times.* Answer

We have still some ideas to improve our code but we need to know more JavaFX. This will be the theme of the next chapter.

6.3 Conclusion

We have seen how to make a program into a laboratory. Nevertheless, we has as yet some things to learn.

Chapter 7

JavaFX for more improvement

`Clever ideas for delicate problems`

101 ***Introduction and purpose.*** *A heavy bug of design of previous program is that it gets dumb when it is commanded to run during a large number of generations. Besides, the amount of consumed resources is an exaggerated growing function of the number of generations in such a way that one almost needs supercomputing for 500 generations and 8 worms. It seems that we need to know more JavaFX to get rid of these problems. So, let us revise some material about timelines and the management of concurrent tasks and threads. We will also bring forth more examples of binding, thanks to which one ties the value of parameter to another's value.*

7.1 General timelines

We have used timelines to accommodate movement of objects. But the formalism we used was rather specific. Let us learn the general formalism to appreciate how powerful it is.

Warning: program ColorfulCircles2 and higher variants run well for Netbeans 8 but not for Netbeans 8.0.1. Hopefully, they will run again for future versions of netbeans. This shows how hard is to design software: any improvement generates bugs, whose correction generates more bugs. So, we must learn to be infinitely tolerable with both ourselves and with others.

102 ***General timelines and parallel movement.*** *The next code illustrates how a Timeline can host parallel, independent and recurrent movement of many objects. It also illustrates how JavaFX allows things to be made wonderful besides being*

functional. The general formalism of a timeline is shown: it is defined by an initial state at start time and the end state at final time.

```
//Program L102 ColorfulCircles2

//The program shows how a timeline can host parallel,
//independent  movement
//of many objects.
//It also illustrates how JavaFX allows things to be
//made wonderful besides being functional.
//General formalism of a timeline.

//Original: ColorfulCircles by Oracle,
//packed with Netbeans 8.
//Modified  by JR ,
//the EvolJava Community (2014)


/*Note of the Owner:
 *
 * Copyright (c) 2008, 2012 Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under
 * the following license:
 *
 * Redistribution and use in source and binary forms,
 * with or without
 * modification, are permitted provided that the
 * following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the
 *    above copyright
 *    notice, this list of conditions and the following
 *    disclaimer.
 *  - Redistributions in binary form must reproduce the
 *    above copyright
```

```
 *    notice, this list of conditions and the following
 *    disclaimer in
 *    the documentation and/or other materials provided
 *    with the distribution.
 *  - Neither the name of Oracle Corporation nor
 *    the names of its
 *    contributors may be used to endorse or promote
 *    products derived
 *    from this software without specific prior written
 *    permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT
 * HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL
 * THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
 * IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 */



package colorfulcircles2;

import static java.lang.Math.random;
```

```
import java.util.ArrayList;
import java.util.List;
import javafx.animation.Animation;
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.Group;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.effect.BlendMode;
import javafx.scene.effect.BoxBlur;
import javafx.scene.paint.Color;
import javafx.scene.paint.CycleMethod;
import javafx.scene.paint.LinearGradient;
import javafx.scene.paint.Stop;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Rectangle;
import javafx.scene.shape.StrokeType;
import javafx.stage.Stage;
import javafx.util.Duration;

/**
 * A sample that demonstrates how to draw
 * and paint shapes, apply visual
 * effects, blend colors in overlapping
 * objects, and animate objects.
 *
 * @see javafx.scene.effect.BlendMode
 * @see javafx.scene.effect.BoxBlur
 * @see javafx.scene.shape.Circle
 * @see javafx.scene.Group
 * @see javafx.scene.paint.LinearGradient
 * @see javafx.animation.Timeline
 */
public class ColorfulCircles2 extends Application
{
    private static final double WIDTH = 495, HEIGHT = 480;
    private Timeline animation;
```

```
private void init(Stage primaryStage)
{
    Group root = new Group();
    primaryStage.setResizable(false);
    primaryStage.setScene(new Scene(root, 495,480));
    primaryStage.setTitle("Heavenly blows");
    Group layer1 = new Group();
    for(int i=0; i<15;i++)
    {
        Circle circle = new Circle(200,Color
                .web("white",0.05f));
        circle.setStrokeType(StrokeType.OUTSIDE);
        circle.setStroke(Color.web("white",0.2f));
        circle.setStrokeWidth(4f);
        layer1.getChildren().add(circle);
    }
    // create second list of circles
    Group layer2 = new Group();
    for(int i=0; i<20;i++)
    {
        Circle circle = new Circle(70,Color
                .web("white",0.05f));
        circle.setStrokeType(StrokeType.OUTSIDE);
        circle.setStroke(Color.web("white",0.1f));
        circle.setStrokeWidth(2f);
        layer2.getChildren().add(circle);
    }
    // create third list of circles
    Group layer3 = new Group();
    for(int i=0; i<10;i++)
    {
        Circle circle = new Circle(150,Color
                .web("white",0.05f));
        circle.setStrokeType(StrokeType.OUTSIDE);
        circle.setStroke(Color.web("white",0.16f));
        circle.setStrokeWidth(4f);
        layer3.getChildren().add(circle);
    }
    // Set a blur effect on each layer
```

```
layer1.setEffect(new BoxBlur(30,30,3));
layer2.setEffect(new BoxBlur(2,2,2));
layer3.setEffect(new BoxBlur(10,10,3));
// create a rectangle size
//of window with colored gradient
Rectangle colors = new Rectangle(WIDTH, HEIGHT,
        new LinearGradient(0f,1f,1f,0f,true
            , CycleMethod.NO_CYCLE
            , new Stop(0,Color.web("#f8bd55")),
            new Stop(0.14f,Color.web("#c0fe56")),
            new Stop(0.28f,Color.web("#5dfbc1")),
            new Stop(0.43f,Color.web("#64c2f8")),
            new Stop(0.57f,Color.web("#be4af7")),
            new Stop(0.71f,Color.web("#ed5fc2")),
            new Stop(0.85f,Color.web("#ef504c")),
            new Stop(1,Color.web("#f2660f")))
);
colors.setBlendMode(BlendMode.OVERLAY);
// create main content
Group group = new Group(
        new Rectangle(WIDTH, HEIGHT, Color.BLACK),
        layer1,
        layer2,
        layer3,
        colors
);
Rectangle clip = new Rectangle(WIDTH, HEIGHT);
clip.setSmooth(false);
group.setClip(clip);
root.getChildren().add(group);

// create list of all circles
List<Node> allCircles = new ArrayList<>();
allCircles.addAll(layer1.getChildren());
allCircles.addAll(layer2.getChildren());
allCircles.addAll(layer3.getChildren());
// Create an animation to parallely and
//independently
//move every circle in allCircles
animation = new Timeline();
```

```
        //This can be done with lists only.
        //Otherwise, use a for loop
        allCircles.stream().forEach((circle) ->
        {
            //Make two transitions, one after the other.
            //First: from time 0 to 10000
            //Second from time 10000 to 20000
            for(int i = 0; i < 2; i++)
            {
                Duration dZero = Duration.millis(10000*i);
                Duration dFinal
                        = Duration.millis(10000*(i+1));
                animation.getKeyFrames().addAll
                    (
                        // set start position at time dZero
                        new KeyFrame(dZero,
                            new KeyValue(circle
                                .translateXProperty()
                                    ,i*WIDTH),
                            new KeyValue(circle
                                .translateYProperty(),0)
                        ),
                        // set end position at time dFinal
                        new KeyFrame(dFinal,
                            new KeyValue(circle
                                .translateXProperty()
                                ,random()*WIDTH),
                            new KeyValue(circle
                                .translateYProperty()
                                ,random()*HEIGHT)
                        )
                    );

            }
        });
        //Eternal repeating
        animation.setCycleCount(Animation.INDEFINITE);
        //animation.setAutoReverse(true);
    }
```

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

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

    @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 L102 ColorfulCircles2
```

103 ***Exercise.*** *The previous program shows two independent blows, one after the other. Modify it to show that one can use the same scheme to represent recurrent movement (chained movements such that one final state is the initial state of the next).* *Answer*

7.2 Breaking long-running tasks

Let us see now how JavaFX deals with long-running tasks that possibly cause the application to get dumb, unresponsive. The basic idea is to provide the possibility to intercept long running tasks to be canceled or to be listened at. This is done by the JavaFX Service Class (Fedortsova [2], 2014) as it shown below.

104 ***Dealing with unresponsiveness of long-running tasks.*** *The next code for JavaFX shows how to control long-running tasks.*

```
//Program L104 StartStopRestart
//This program shows how to deal in JavaFX
//with long-running computing tasks
//that may cause unresponsiveness of the application.
//The program provides the possibility
//to start a task, to cancel it,
//and to give a fresh start to the task.
//The program uses the
//JavaFx Service Class that can start, stop and restart
//a task from the beginning.
//But it cannot resume a stopped, cancelled task.
//Long-running task and UI-management are executed in isolation
//one from the other.

//Next line is for NetBeans. Delete it for Eclipse.
package startstoprestart;


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.control.ProgressIndicator;
import javafx.stage.Stage;


public class StartStopRestart extends Application
{


//=======Inner class that contains the long task
private static class LongTask extends Service<String>
{
```

```
    @Override //createTask() must return a task
    protected Task<String> createTask()
    {

        return new Task<String>()
        {
            @Override //If startButton is clicked
            protected String call()
            {
                 counter++;
                Integer iterations;
                int max = 200000;
                for (iterations = 0; iterations < max;
                        iterations++)
                {
                    System.out.println("Run  " + counter
                            + " Iteration " + iterations);
                    //If cancelButton is clicked
                    if (isCancelled())
                    {
                       break;
                    }
                }
                return iterations.toString();
            }
        };
    }
}//End of inner class LongTask



//========= UI management

private static Group p = new Group();
private static  int counter = 0;

@Override
public void start(Stage stage) throws Exception
{
```

```
LongTask service = new LongTask();


//Running task indicator
final ProgressIndicator progressIndicator
        = new ProgressIndicator();
//Booking
p.getChildren().add(progressIndicator);

Scene scene = new Scene(p, 400, 350);
stage.setTitle("StartStopRestart");

//Declaration of buttons
Button startButton = new Button("Start");
Button cancelButton = new Button("Cancel");
Button newRunButton = new Button("newRun");
Button exitButton = new Button("Exit");

//Properties of buttons
cancelButton.setOpacity(0);
newRunButton.setOpacity(0);

startButton.setLayoutX(100);
startButton.setLayoutY(100);

cancelButton.setLayoutX(100);
cancelButton.setLayoutY(130);

newRunButton.setLayoutX(100);
newRunButton.setLayoutY(160);

exitButton.setLayoutX(100);
exitButton.setLayoutY(190);

//Listeners
startButton.setOnAction((ActionEvent actionEvent) ->
{

    service.start();
    startButton.setOpacity(0);
```

```
    cancelButton.setOpacity(1);

});


cancelButton.setOnAction((ActionEvent actionEvent) ->
{
    service.cancel();
    newRunButton.setOpacity(1);
    cancelButton.setOpacity(0);

});



newRunButton.setOnAction((ActionEvent actionEvent) ->
{
    service.restart();
    newRunButton.setOpacity(0);
    cancelButton.setOpacity(1);
});

exitButton.setOnAction((ActionEvent actionEvent) ->
{
    //Finish everything and exit
    System.exit(0);
});

//What to do if task is accomplished?
service.setOnSucceeded((WorkerStateEvent t) ->
{
    System.out.println("done:"
                    + t.getSource().getValue());
    newRunButton.setOpacity(1);
    cancelButton.setOpacity(0);
});


//Booking
p.getChildren().addAll(startButton, cancelButton,
```

```
                                newRunButton, exitButton);

    //Stage filling and showing
    stage.setScene(scene);
    stage.show();

    }//End of start()


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

}//End of main class L104 StartStopRestart
```

105 ***Exercise.*** *Add to the previous program the possibility to reset the number of iterations upon each task calling. Also, enable the progress indicator to show the amount of task done.* Answer

Let us deal now with a subtle but terrible bug of design of our programs.

7.3 Troubles with memory

It seems that our programs need huge amounts of memory. Is that all that we can say?

106 ***Overflowing of needed resources.***

Our programs about worms run well on a desktop for 100 generations but not for 300 while for 500 it gets too boring and too overloaded with computing requirements: thanks to the System Monitor of Ubuntu it was possible to know that the program needs for 8 worms and 500 generations almost 10 GiB of RAM, 2.6 of which are for the Ubuntu system. We have found two reasons for this behavior. Firstly, the program demands to memorize a fossil for every generated worm to next make a sampling and, secondly, our animations seem to be recorded and played from memory only after all computations have been completed.

Now, JavaFX has been thought to be the standard in both high quality and mobility. In fact, it is planned to be run on every mobile that also has Java 8. It would be our fault if our program cannot be run in a cellular phone because the

lack of memory resources. So, it is a matter of honor to find a convenient solution to these troubles.

To attack the problem we propose three ideas. First: instead of making the sampling of fossils after the last generation, one can program sampling along the simulation and so no memorization of all fossils would be necessary. Second, instead of playing the whole animation after calculation of the complete simulation, we might command to play each generation just after it has been calculated and so, no (automatic) recording would be necessary. Besides, the displaying of animations was absolutely necessary to make sure that the program was correct but once this has been positively decided, animations might be categorized in research as superfluous and therefore disposable. Thus, one must enable the program with the possibility to dispense with animations. But, beware, to silence the playing of animations it is enough to silence the line that contains the command to play it. This causes an effective diminishing of needed resources but not enough to make the application accessible for mobile crafts. So, we need to learn how to measure used memory in Java and how to optimize its use.

To measure consumed memory we can use the next piece of code:

```
long fm = Runtime.getRuntime().freeMemory();
long tm = Runtime.getRuntime().totalMemory();
long um = tm-fm;
System.out.println("Total  memory = " + tm
          +"\nFree  memory = " + fm
          + "\nUsed memory = " + um);
```

107 ***Exercise.*** *Insert the aforementioned piece of code to measure used memory into program L103 ColorfulCircles3. Insertion can be done at the end of the init method. Find the loop that regulates the number of steps of the dance and change that number to verify that the more steps the dance has, the more memory is needed. This shows that animation is calculated, completely recorded and then played from memory.*

To diminish the consumption of memory, we need to command the application to present short pieces just as they are calculated.

108 ***Animation played by pieces.*** *The next code shows how to play a piece of an animation just after it has been calculated.*

```
//Program L108 ColorfulCircles4
```

```
//The program shows how a timeline can host parallel,
//independent  movement
//of many objects.
//It also illustrates how JavaFX allows things to be
//made wonderful besides being functional.
//General formalism of a timeline.

//Original: ColorfulCircles by Oracle
//Modified  by JR,
//the EvolJava Community (2014)


/*Note of the Owner:
 *
 * Copyright (c) 2008, 2012 Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under
 * the following license:
 *
 * Redistribution and use in source and binary forms,
 * with or without
 * modification, are permitted provided that the
 * following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the
 *    above copyright
 *    notice, this list of conditions and the following
 *    disclaimer.
 *  - Redistributions in binary form must reproduce the
 *    above copyright
 *    notice, this list of conditions and the following
 *    disclaimer in
 *    the documentation and/or other materials provided
 *    with the distribution.
 *  - Neither the name of Oracle Corporation nor
 *    the names of its
 *    contributors may be used to endorse or promote
```

```
 *    products derived
 *    from this software without specific prior written
 *    permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT
 * HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL
 * THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
 * IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 */



package colorfulcircles4;

import static java.lang.Math.random;
import java.util.ArrayList;
import java.util.List;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;
```

```
import javafx.event.ActionEvent;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.effect.BlendMode;
import javafx.scene.effect.BoxBlur;
import javafx.scene.layout.Region;
import javafx.scene.paint.Color;
import javafx.scene.paint.CycleMethod;
import javafx.scene.paint.LinearGradient;
import javafx.scene.paint.Stop;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Rectangle;
import javafx.scene.shape.StrokeType;
import javafx.stage.Stage;
import javafx.util.Duration;

/**
 * A sample that demonstrates how to draw
 * and paint shapes, apply visual
 * effects, blend colors in overlapping
 * objects, and animate objects.
 *
 * @see javafx.scene.effect.BlendMode
 * @see javafx.scene.effect.BoxBlur
 * @see javafx.scene.shape.Circle
 * @see javafx.scene.Group
 * @see javafx.scene.paint.LinearGradient
 * @see javafx.animation.Timeline
 */
public class ColorfulCircles4 extends Application
{
    private static final double WIDTH = 495, HEIGHT = 480;


    private HeavenlyDance heavenlyDance;
```

```
private void init(Stage primaryStage)
{
    Group root = new Group();
    primaryStage.setResizable(false);
    primaryStage.setScene(new Scene(root, WIDTH,HEIGHT));
    primaryStage.setTitle("Heavenly dance");
    //Instantiation
    heavenlyDance = new HeavenlyDance();
    //Booking
    root.getChildren().add(heavenlyDance);
}



public static class HeavenlyDance extends Region
{

    //Counter of played pieces
    int counter = 0;
    //Constructor = initializer
    public HeavenlyDance()
    {
      showPiece(counter);
    }


    //Calculates and shows one piece of the animation
    private void showPiece(int l)
    {

        System.out.println("\n Try = " + l);

        //getChildren().removeAll();

        Group layer1 = new Group();
    for(int i=0; i<15;i++)
    {
        Circle circle = new Circle(200,Color
                .web("white",0.05f));
```

```
        circle.setStrokeType(StrokeType.OUTSIDE);
        circle.setStroke(Color.web("white",0.2f));
        circle.setStrokeWidth(4f);
        layer1.getChildren().add(circle);
    }
    // create second list of circles
    Group layer2 = new Group();
    for(int i=0; i<20;i++)
    {
        Circle circle = new Circle(70,Color
                .web("white",0.05f));
        circle.setStrokeType(StrokeType.OUTSIDE);
        circle.setStroke(Color.web("white",0.1f));
        circle.setStrokeWidth(2f);
        layer2.getChildren().add(circle);
    }
    // create third list of circles
    Group layer3 = new Group();
    for(int i=0; i<10;i++)
    {
        Circle circle = new Circle(150,Color
                .web("white",0.05f));
        circle.setStrokeType(StrokeType.OUTSIDE);
        circle.setStroke(Color.web("white",0.16f));
        circle.setStrokeWidth(4f);
        layer3.getChildren().add(circle);
    }
    // Set a blur effect on each layer
    layer1.setEffect(new BoxBlur(30,30,3));
    layer2.setEffect(new BoxBlur(2,2,2));
    layer3.setEffect(new BoxBlur(10,10,3));
    // create a rectangle size
    //of window with colored gradient
    Rectangle colors = new Rectangle(WIDTH, HEIGHT,
            new LinearGradient(0f,1f,1f,0f,true
                , CycleMethod.NO_CYCLE
                , new Stop(0,Color.web("#f8bd55")),
                new Stop(0.14f,Color.web("#c0fe56")),
                new Stop(0.28f,Color.web("#5dfbc1")),
                new Stop(0.43f,Color.web("#64c2f8")),
```

```
                new Stop(0.57f,Color.web("#be4af7")),
                new Stop(0.71f,Color.web("#ed5fc2")),
                new Stop(0.85f,Color.web("#ef504c")),
                new Stop(1,Color.web("#f2660f")))
);
colors.setBlendMode(BlendMode.OVERLAY);
// create main content
Group group = new Group(
        new Rectangle(WIDTH, HEIGHT, Color.BLACK),
        layer1,
        layer2,
        layer3,
        colors
);
Rectangle clip = new Rectangle(WIDTH, HEIGHT);
clip.setSmooth(false);
group.setClip(clip);
getChildren().add(group);

// create list of all circles
List<Node> allCircles = new ArrayList<>();
allCircles.addAll(layer1.getChildren());
allCircles.addAll(layer2.getChildren());
allCircles.addAll(layer3.getChildren());
// Create an animation to parallely and
//independently
//move every circle in allCircles
final Timeline animation = new Timeline();

//This can be done with lists only.
//Otherwise, use a for loop
allCircles.stream().forEach((circle) ->
{
    int i = 0;
    double xZero  = i*WIDTH;
    double yZero  = 0;
    //Make recurrent transitions, one after the other.
    //From time 10000*i
    //to time  10000*(i+1)
    //1 second = 1000;
```

```
int n = 3; //n > 2
for(i = 0; i < n; i++)
{
    //Initial and final times
    Duration dZero = Duration.millis(10000*i );
    Duration dFinal
            = Duration.millis(10000*(i+1));
    //Initial and final positions
    double xFinal;
    double yFinal;
    if (i < n-1)
    {
        xFinal  = random()*WIDTH;
        yFinal  = random()*HEIGHT;
    }
    else
    {
        xFinal  = WIDTH;
        yFinal  = 0;
    }


    //Translation from intial position
    //at initial time
    //to
    //final position at final time
    animation.getKeyFrames().addAll
        (
            // set start position at time dZero
            new KeyFrame(dZero,
                new KeyValue(circle
                    .translateXProperty()
                        ,xZero),
                new KeyValue(circle
                    .translateYProperty(),yZero)
            ),
            // set end position at time dFinal
            new KeyFrame(dFinal,
                new KeyValue(circle
```

```
                            .translateXProperty()
                            ,xFinal),
                        new KeyValue(circle
                            .translateYProperty()
                            ,yFinal)
                    )
                );
            //Recurrence is here
            xZero  = xFinal;
            yZero  = yFinal;
            //Feedback
            System.out.println("xZero = " + xZero);
        }
    });
    animation.setCycleCount(1);
    animation.play();

    //Call itself for a new iteration.
    //To execute only when animation has been done
    animation.setOnFinished((ActionEvent me) ->
    {
        getChildren().remove(group);

        //Run Garbage Collector to empty trash
        System.gc();
        System.gc();


        //Show piece of animation
        if (counter < 3) showPiece(counter);
    });


    //Memory usage
    long fm = Runtime.getRuntime().freeMemory();
    long tm = Runtime.getRuntime().totalMemory();
    long um = tm-fm;
    System.out.println("Total  memory in = " + tm
        +"\nFree  memory = " + fm
        + "\nUsed memory = " + um
```

```
        +  "\ncounter = " + counter);
        counter++;
        }
    }//End of inner class HeavenlyDance



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


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

    }
}////End of main class L108 ColorfulCircles4
```

109 ***Exercise.*** *Run the program and play with the code. Play enough to confidently accept else reject the conclusion of the Author: the proposed remedy resulted to be worst than the illness. In fact, the program correctly presents a piece of animation just after it has been calculated but the needed memory is surprisingly large for more than one generation. The possible cause is that Java does not obey orders if a preprogrammed solution exists and memory management is one of those cases.*

In view of our negative results, an idea whose exploration turns now to be mandatory is the following: since JavaFX has no acceptable cleaning process, the solution might be to create objects as minimally as possible. In other words, we must reuse JavaFX objects as much as possible. The next code illustrates how this is done over another flavor of a previous program.

110 ***Reuse of worms.*** *The next code shows over a model problem how to reuse old worms for a new generation. No serendipity at all.*

```
//Program L110 WormsRandJFX9
```

```
//A population of worms is programmed
//by randomness to
//move their links
//during certain number of generations.
//For each new generation:
//worms are repositioned at startLine,
//information to move is reset and
//one link is added.
//Program is written in JavaFX.
//This is a refurnished version of
//program L56 WormsRandJFX6

package ejvolu12p;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javafx.animation.Interpolator;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.ParallelTransition;
import javafx.animation.SequentialTransition;
import javafx.animation.Timeline;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.util.Duration;



public class WormsRandJFX9 extends Application
{
    private static WormsRandJFX9 w = new WormsRandJFX9();
    //Overall process
```

```
private static  SequentialTransition nGenerationsST
                = new SequentialTransition();
//Number of repetitions or generations
private static int nGenerations = 20;
private static int gen = 0;

//The scene
private static   Group p = new Group();

//Number of worms
private static  int nWorms = 3;
//Worms are kept in a list
private static List<Worm> Worms = new ArrayList<>();
//Number of links of each worm
private static  int nLinks = 6;
//Initial position over the screen of the head
private static int initialRow = 180;
//Generator of pseudo random numbers
private static final Random r = new Random();
//Each worm has nMovers movers,
//instructions to move.
private static int  nMovers = 3;
//Width of the worm = width of each link
//Number of steps
int nSteps = 3;
private static int width = 4;
//Height of each link
private static int height = 12;
//Array with 9 possible moves
private static Move  vectMoves[ ] = new Move[9];

private static final boolean print1 = true;
//Number of ms of translation events.
private static int durationMs = 500;
//Pane to ease coordinate management:
//It is seem what you (probably) expect
private static StackPane stackPane = new StackPane();
/* Definition of inner classes.
*
The Move class:
```

```
   A Move says in which direction
   a link of the worm must move.
   */

private  class Move
{
   int x;
   int y;


   //Default initialization
   Move()
   {
       x= 0;
       y= 0;
   }

   // Initialize wth a list of two coordinates
   Move(int a, int b)
   {
       x= a;
       y= b;
   }

   // Copy a move
   Move(Move move)
   {
       x= move.x;
       y= move.y;
   }
}//End of class Move


// Definition of the class Mover:
// A Mover keeps two types of information:
// a) the link of the worm to be moved and
// b) the direction or move to be chosen.
//
private   class Mover
{
```

```
    int link ;
    Move   move;

    //  Movers can acquired information
    // in three ways, they are  overloaded.
    // Form 1: by a list of numbers
    private Mover(int l, int mx, int my)
     {
        link= l;
        move.x = mx;
        move.y = my;
     }

    // Form 2, by structured data
    private Mover(int l, Move m)
    {
        link= l;
        move = new Move(m);
    }

    // Form 3, by structured data
    private Mover(Mover mover)
    {
        link= mover.link;
        move =  new Move(mover.move);
    }
  }//End of class Mover



//Rects that belongs in the worms of the first generation
 //are visible and ready to move.
 //Others are hidden.
private static  Rectangle
           createRectangle(boolean visible)
{
    //Upper left corner at (0,0)
    Rectangle rect = new Rectangle(0,0,width,height);
    //Color is defined as a random
    //mix of red, green and blue
```

```
    int rr,gg,bb;
    rr = r.nextInt(226);
    gg = r.nextInt(226);
    bb = r.nextInt(226);
    //The color is prepared
    Color c = Color.rgb(rr,gg,bb);
    //The rect is painted
    rect.setFill(c);
    //Rects of the first generation (gen = 0)
    //are ready to move,
    //otherwise they got hidden
    if ( !visible)   rect.setOpacity(0);
    stackPane.getChildren().addAll(rect);

    return rect;
}


/*  Here we define the class worm.
    A worm is composite: it has links.
    Each link has a rectangular shape
    and  its own color,
    which is defined by three numbers.
    The vector Mover[ ]
     keeps the information
    on how to move its links:
    which  link and in which order
    (in the order that they appear in
    the vector Mover[]), and
    in which direction.
    Each worm keeps record of
    the actual position of each
    one of its links,
    in the associated ractangle. */



private class Worm
{
    //Number of links
```

```
    int numberOfLinks = nLinks;
    //Number of movers = instructions to move
    int numberOfMovers = nLinks;
    //List mover  keeps movers
    List<Mover> Mover = new ArrayList<>();
    //The shape of each link
    List<Rectangle> Rect = new ArrayList<>();

    //Default initialization:
    //worms of the first generation are
    //ready to move. Others get hidden.
    private Worm()
    {
//Rects are created.
        // A color is assigned to each link
        // but different than white
        for(int i = 0; i < numberOfLinks; i++)
            Rect.add(createRectangle(true));
        //Instructions to move at random
for(int m = 0; m < nMovers; m++)
        {
            //Select at random  which link to move:
            int l = r.nextInt(nLinks) ;
            //Select at random  which direction to move on
            Move  move = w.new Move (vectMoves[r.nextInt(9)]);
            //Memorization
            Mover.add(w.new Mover(l,move));
        }
    }//End of constructor Worm(int  gen)

}//End of class Worm



 //Nine basic moves = directions where to move in
static private void moveDef()
{
 final Move move0 =  w.new Move(0,1);
   vectMoves[0]   = move0 ;
 final Move move1 =  w.new Move(1,1);
```

```
   vectMoves[1]   = move1 ;
 final Move move2 =  w.new Move(1,0);
   vectMoves[2]   = move2 ;
 final Move move3 =  w.new Move(1,-1);
   vectMoves[3]   = move3 ;
 final Move move4 =  w.new Move(0,-1);
   vectMoves[4]   = move4 ;
 final Move move5 =  w.new Move(-1,-1);
   vectMoves[5]   = move5 ;
 final Move move6 =  w.new Move(-1,0);
   vectMoves[6]   = move6 ;
 final Move move7 =  w.new Move(-1,1);
   vectMoves[7]   = move7 ;
 final Move move8 =  w.new Move(0,0);
   vectMoves[8]   = move8 ;
}


@Override public void start(Stage stage)
    {
        //The scene
        Scene scene = new Scene(p, 300,400);
        stage.setScene(scene);
        stage.setTitle("Links move at random");
        stage.show();
        //Definition of possible moves
        moveDef();
        //Run the overall process only once
        nGenerationsST.setCycleCount(1);
        //Initial worms are created
        createWorms();
        for(int i = 0; i < nGenerations; i++)
        {
            gen = i;
            System.out.println("Gen = " + gen);
            //Worms are posited at the start line
            startLine();
            //One link is added to all worms
            addLink();
            //Worms climb the screen
```

```
            SequentialTransition oneGenST  =  express();
            nGenerationsST.getChildren().add(oneGenST);
            memoryUsage();
        }
    //Film is played
    nGenerationsST.play();
    p.getChildren().add(stackPane);
    memoryUsage();
    }

//Initial worms are created
static public void createWorms()
{
    for (int W = 0; W < nWorms; W++)
    {

        Worm   worm = w.new Worm();


        //Each link is positioned
        int x,y;
        for(int  link = 0; link < nLinks; link++)
        {
            //Horizontal position
            x = 30 + 100 * W;
            //Vertical position
            y = initialRow + height*link;

            //Position is updated
            worm.Rect.get(link).setX(x);
            worm.Rect.get(link).setY(y);

        }
        //worm is added at position W in list Worms
        Worms.add(worm);
    }

}
```

```
//The head of all worms are posited at the same initial row
//in vertical position.
static public void startLine()
{
    // Create an animation to parallely and
        //independently
        //move every link
        ParallelTransition positionZeroPT = new ParallelTransition();;
    for (int W = 0; W < nWorms; W++)
    {
        int  x,y;
        //Auxiliary worm
        Worm   worm  = Worms.get(W);

        //Each link is positioned
        for(int  link = 0; link < nLinks; link++)
        {
            //Horizontal position
            x = 30 + 100 * W;
            //Vertical position
            y = initialRow + height*link;

            //Address is updated
            Worms.get(W).Rect.get(link).setX(x);
            Worms.get(W).Rect.get(link).setY(y);

            //Worms are positioned at initial position
            Duration d = Duration.millis(1 );
            //A soft jump is programmed
            TranslateTransition translateTransition =
                new TranslateTransition(d,
                                        worm.Rect.get(link));
            //Repositioning at initial position.
            translateTransition.setToX(x);
            translateTransition.setToY(y);
            //Number of repetitions
            translateTransition.setCycleCount(1);
            //Repositioning is booked
            positionZeroPT.getChildren()
                    .add(translateTransition);
```

```
        }
        //Invent new moves
                Worms.set(W,resetMoves(Worms.get(W)));
    }
    //Booking
    nGenerationsST.getChildren()
                        .add(positionZeroPT);

    //Recently added links are  shown
     if (gen > 0)
           {
             for (int W = 0; W < nWorms; W++)
             {
                ParallelTransition show
                        = showST(Worms.get(W)
                                .Rect.get(nLinks-1));
                //Show transition is booked
                nGenerationsST.getChildren().add(show);
             }
           }
}


//Moves at random are created
private static Worm resetMoves(Worm worm)
{
    //Instructions to move at random
    for(int m = 0; m < nMovers; m++)
    {
        //Select at random  which link to move:
        int l = r.nextInt(nLinks) ;
        //Select at random  which direction to move on
        Move  move = w.new Move (vectMoves[r.nextInt(9)]);
        //Memorization
        worm.Mover.set(m, w.new Mover(l,move));
    }
    return worm;
}
```

```
//Move m of worm W is executed.
private static TranslateTransition
                    translate( int W, int m)
{
    if (print1)
        System.out.println();
    //Reads the link to be moved
    int link = Worms.get(W).Mover.get(m).link;
    //Reads position
    int x = (int) Worms.get(W).Rect.get(link).getX();
    int y = (int) Worms.get(W).Rect.get(link).getY();

    //The move to be done is read
    Move  move1;
    move1 = Worms.get(W).Mover.get(m).move;
    //Move is interpreted
    int byX = width* move1.x;
    int byY = -height* move1.y;
    // A new place is defined:
    x = x + byX;
    y = y + byY;
    if (print1)
        System.out.println("\n Worm = " + W
                        + " move = " + m
                        + " link = " + link
                        + " \n x = " + x
                        + " y = " + y
                        + " \nbyX = " + byX
                        + " byY = " + byY);

    //Address is updated
    Worms.get(W).Rect.get(link).setX(x);
    Worms.get(W).Rect.get(link).setY(y);

    //Duration in within tics
    Duration d = Duration.millis(durationMs);
    //A soft jump is programmed
    TranslateTransition translateTransition =
         new TranslateTransition(d, Worms.get(W).Rect.get(link));
```

```
    //Jump is executed
    translateTransition.setByX(byX);
    translateTransition.setByY(byY);
    //Number of repetitions
    translateTransition.setCycleCount(1);

    return translateTransition;
 }



//Worms express their information to move.
public    SequentialTransition  express()
{
    //Overall process
    SequentialTransition oneTrialST
                    = new SequentialTransition();
     oneTrialST.setCycleCount(1);
     //worms advance nSteps
    for (int i = 0; i < nSteps; i++)
    {
        //Worms advance one step:
        //To move all links, each worm must execute
        //instructions to move
        SequentialTransition oneStepST
                        = new SequentialTransition();
        oneStepST.setCycleCount(1);
        //Information is read and decoded
        for (int m = 0; m < nMovers; m++)
        {
            //All worms execute move m
            ParallelTransition moveMPT
                        = new ParallelTransition();
            moveMPT.setCycleCount(1);
            for (int W = 0; W < nWorms; W++)
            {
                //A translation of some link
                //is programmed for worm W
                //according to move m
                TranslateTransition tT = translate(W,m);
```

```
                tT.setCycleCount(1);
                //Translation is booked
                moveMPT.getChildren().add(tT);
            }
            //The ensamble of translations is booked
            oneStepST.getChildren().add(moveMPT);
        }
    //The movement of worms in one step is booked
    oneTrialST.getChildren().add(oneStepST);
    }
    return oneTrialST;
}


    //One link is added to all worms
    public static void addLink()
    {
        //All worms are visited
        for (int W = 0; W < nWorms; W++)
        {
            //New unvisible rect is created and added to the worm
            Worms.get(W).Rect.add(createRectangle(false));


        }
        //Bookkeeping
        nLinks++;
    }



    //A hidden rect gets shown
    private static ParallelTransition showST(Rectangle rect)
    {
        ParallelTransition s = new ParallelTransition();
        Duration d2 =   Duration.millis(1);

        //Rect gets visible
        KeyValue keyValue =
            new KeyValue(rect.opacityProperty(),1,
```

```
                Interpolator.LINEAR);
        //KeyValue is  inserted into a timeline
        Timeline b = new Timeline(
            new KeyFrame(d2,keyValue ));
        //Timeline is inserted into a transition
        s.getChildren().addAll(b);
        return s;
    }


    private static void memoryUsage()
{
    //Memory usage
    long fm = Runtime.getRuntime().freeMemory();
    long tm = Runtime.getRuntime().totalMemory();
    long um = tm-fm;
    System.out.println("Total  memory = " + tm
        +"\nFree  memory = " + fm
        + "\nUsed memory = " + um );
}

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


} //End of main class L110 WormsRandJFX9
```

111 ***Exercise.*** *Run the program and play with the code.*

112 ***Exercise.*** *Modify the previous program to add a link only when some worm is shown to climb the screen.* Answer

113 ***Permanent research.*** *We have explored some ways to deal with the problem of large amounts of memory needed by JavaFX animations. This problem is important since it interferes the expansion of JavaFX to the world of mobiles. So, keep an eye on the solutions proposed by the literature.*

7.4 JavaFX properties and binding

Let us see more examples of a very nice tool of JavaFX that is used to enslave a value to another: properties and their binding.

114 ***Binding.*** *JavaFX proposes a binding when one needs to enslave a parameter to another. A condition must be met: parameters must be declared as* ***properties****. The next code illustrates how this is implemented for numbers whose type is integer and double.*

```
//Program L114 BindingDemo
//One can use the inbuilt bindings
//for numbers of type Integer and double.
//And one also can devise own bindings.



/* Adapted from

 * http://docs.oracle.com/javafx/2/binding/jfxpub-binding.htm

Scott Hommel

Technical Writer, Oracle, June 2013.
 */

package ejvolu12p;

import javafx.beans.binding.Bindings;
import javafx.beans.binding.DoubleBinding;
import javafx.beans.binding.NumberBinding;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.IntegerProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.beans.property.SimpleIntegerProperty;

public class BindingDemo
{
```

```
/*Two integers are bound so that their values are
always added together.
There are three variables involved:
num1 (a dependency), num2 (a dependency),
and sum (the bound result). The dependency types
are both IntegerProperty,
and the binding itself is NumberBinding.
*/
public static void m1()
{
    //Declaration and initialization with integers
    IntegerProperty num1 = new SimpleIntegerProperty(1);
    IntegerProperty num2 = new SimpleIntegerProperty(2);
    //num1 + num2 as a bound value
    NumberBinding sum = num1.add(num2);
    //Computation and report
    System.out.println(
            num1.getValue()
            + " + "
            + num2.getValue() + " = "
            + sum.getValue());
    //Partial resetting
    num1.set(2);
    System.out.println(
            num1.getValue()
            + " + "
            + num2.getValue() + " = "
            + sum.getValue());
}



//You could also use the Binding class to do the same thing
public static void m2()
{
   IntegerProperty num1 = new SimpleIntegerProperty(3);
   IntegerProperty num2 = new SimpleIntegerProperty(4);
   NumberBinding sum = Bindings.add(num1,num2);
   System.out.println(
```

```
            num1.getValue()
            + " + "
            + num2.getValue() + " = "
            + sum.getValue());
   num1.setValue(5);
   //Next line cannot be used without thread management
   //System.err.println(sum.getValue());
   System.out.println(
            num1.getValue()
            + " + "
            + num2.getValue() + " = "
            + sum.getValue());
}


//The two approaches can be combined
public static void m3()
{
    //Number of objects of type A
    IntegerProperty numA = new SimpleIntegerProperty(5);
    //Unitary price of objects of type A
    IntegerProperty priceA = new SimpleIntegerProperty(6);
    //Number of objects of type B
    IntegerProperty numB = new SimpleIntegerProperty(7);
    //Unitary price of objects of type B
    IntegerProperty priceB = new SimpleIntegerProperty(8);
    //Total = numA * priceA + numB * priceB
    NumberBinding total =
      Bindings.add(numA.multiply(priceA),numB.multiply(priceB));
    System.out.println(
            numA.getValue()
                    + "*" + priceA.getValue()
            + " + "
            + numB.getValue()
                    + "*" + priceB.getValue() + " = "
            + total.getValue());
    //Number of objects type A is redefined
    numA.setValue(2);
    //Computation and report
    System.out.println(
```

```
            numA.getValue()
                    + "*" + priceA.getValue()
            + " + "
            + numB.getValue()
                    + "*" + priceB.getValue() + " = "
            + total.getValue());
}


//Properties also can be of type double
//along a custum binding
public static void m4()
{
    //Declaration and initilization with doubles:
    //The same scheme as in m3()
    final DoubleProperty a = new SimpleDoubleProperty(11);
    final DoubleProperty b = new SimpleDoubleProperty(12);
    final DoubleProperty c = new SimpleDoubleProperty(13);
    final DoubleProperty d = new SimpleDoubleProperty(14);
    //Declaring a custum  binding
    DoubleBinding output = new DoubleBinding()
    {

        {
            super.bind(a, b, c, d);
        }

        @Override
        protected double computeValue()
        {
            return (a.get() * b.get()) + (c.get() * d.get());
        }
    };
    //Calculation and report
    System.out.println(
            a.getValue()
                    + "*" + b.getValue()
            + " + "
            + c.getValue()
                    + "*" + d.getValue() + " = "
```

```
            + output.getValue());
    //Partial redefinition
    b.set(3);
       System.out.println(
            a.getValue()
                    + "*" + b.getValue()
            + " + "
            + c.getValue()
                    + "*" + d.getValue() + " = "
            + output.getValue());
}


public static void main(String[] args)
{
    System.out.println("m1");
    m1();
    System.out.println("m2");
    m2();
    System.out.println("m3");
    m3();
    System.out.println("m4");
    m4();
}

}//End of main class L114 BindingDemo
```

115 ***Exercise.*** *Run the program and play with the code.*

116 ***Costume properties.*** *Binding JavaFX machinery is under forging but on completion it will lend a full service to customized properties beyong the next code:*

```
//Program L116 PropertyDemo
//A new property is devised.


/* Adapted from
```

```
 * http://docs.oracle.com/javafx/2/binding/jfxpub-binding.htm

Scott Hommel

Technical Writer, Oracle, June 2013.
*/


package ejvolu12p;


import javafx.beans.Observable;
import javafx.beans.binding.Bindings;
import javafx.beans.binding.NumberBinding;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;


public class PropertyDemo
{

private static boolean addListener = true;

/*
     Class Bill implements a single property
    named amountDue.
*/
static class Bill
{

    // Define a variable to store the property
    private DoubleProperty amountDue
            = new SimpleDoubleProperty();

    // Define a getter for the property's value
    public final double getAmountDue()
    {return amountDue.get();}
```

```
    // Define a setter for the property's value
    public final void setAmountDue(double value)
    {amountDue.set(value);}

     // Define a getter for the property itself
    public DoubleProperty amountDueProperty()
    {return amountDue;}

}



public static void main1()
{
    Bill electricBill = new Bill();
    electricBill.setAmountDue(80);
    //Listener of changes in amountDue
    electricBill.amountDueProperty()
        .addListener(new ChangeListener()
        {
            @Override
            public void changed(ObservableValue o,
                    Object oldVal,  Object newVal)
            {
                System.out.println("New value = " + newVal
                 + " Electric bill has changed!");
            }
        });

    electricBill.setAmountDue(100.00);
    electricBill.setAmountDue(100.00);

}



 /*
    The bill total (a binding) will be marked as invalid
```

```
the first time it detects a change
in one of its dependencies. However,
the binding object will recalculate itself
only if the total is actually requested again.
*/
public static void main(String[] args)
{
    System.out.println("Main1");
    main1();
    System.out.println("\nMain");
    //Initialization in zeroes
    Bill bill1 = new Bill();
    Bill bill2 = new Bill();
    Bill bill3 = new Bill();

    //total = bill1 + bill2 + bill3
    NumberBinding total =
      Bindings.add(bill1.amountDueProperty()
        .add(bill2.amountDueProperty()),
        bill3.amountDueProperty());

    System.out.println("Total1 = " + total.getValue());


    total.addListener((Observable o) ->
    {
        System.out
                .println("Total has been recalculated.");
    });

    //Completely new account
    bill1.setAmountDue(200.00);
    bill2.setAmountDue(100.00);
    bill3.setAmountDue(75.00);

    // Recalculation and report
     System.out.println("Total2 = " + total.getValue());

    //Partial resetting
    bill3.setAmountDue(150.00);
```

```
        // Recalculation and report
        System.out.println("Total3 = " + total.getValue());

    }

}//End of main class L116 PropertyDemo
```

117 ***Exercise.*** *Run the program and play with the code.*

118 ***Exercise.*** *Compose a program that automatically calculates and recalculates a bill and dress it in an appropriate GUI.*

Answer

119 ***Research.*** *One can implement a GUI for a bill with buttons but this is not the best option. The perfect instrument is a JavaFX table. Use one to implement a bill that automatically updates itself.*

120 ***Properties of controls.*** *JavaFX objects have many inbuilt properties ready to be used. In the next code we see how to manage the properties of fonts to bind the text of a welcome message to a font that has been chosen by the User.*

```
//Program L120 FontProperties

//Adapted from
//http://stackoverflow.com/questions/
//19433726/javafx-binding-simple-example

//This program allows to choose a font
//from a Choice Box
//and automatically to rewrite a sample text
//into the chosen font.
//This is done through a JavaFX  binding that
//binds the font of the text
//to the election made by the User.



package ejvolu12p;
```

```
import javafx.application.Application;
import javafx.beans.property.ObjectProperty;
import javafx.beans.property.SimpleObjectProperty;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.Scene;
import javafx.scene.control.ChoiceBox;
import javafx.scene.control.Label;
import javafx.scene.control.SingleSelectionModel;
import javafx.scene.layout.GridPane;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class FontProperties extends Application
{
    //Label with text in  chosen font
    Label fontLabel = new Label("The font of this text "
            + "is bound to  chosen font");

    //Pane for label and choice box
    GridPane gridPane = new GridPane();

    //Available fonts
    public ObservableList availableFonts
            = FXCollections.observableArrayList(
                "DejaVu Sans Mono",
                "Nimbus Sans L",
                "Ubuntu",
                "Ubuntu Light",
                "Ubuntu Condensed",
                "Arial",
                "Garuda",
                "Gentium",
                "Kinnari",
                "Loma",
                "Saab"
            );
```

```
//Declaration and initialization of the official font.
//To achieve a binding,
//theFont is not declared as a font
//but as a property.
public ObjectProperty<Font> theFont
        = new SimpleObjectProperty<>
            (Font.font("DejaVu Sans Mono",18));



@Override
public void start(Stage stage) throws Exception
{
    Scene scene = new Scene(gridPane);


    //Available fonts
    ChoiceBox availableFontsCB
        = new ChoiceBox(availableFonts);


    gridPane.setPrefSize(500,100);

    gridPane.add(availableFontsCB,0,0);
    gridPane.add(fontLabel,0,2);
    //The machinery to select font
    SingleSelectionModel  fontSelectionModel
            = availableFontsCB.getSelectionModel();

    //Listener
    fontSelectionModel
        .selectedIndexProperty()
        .addListener(new ChangeListener()
        {
            @Override
            public void changed(ObservableValue value,
                    Object oldValue, Object newValue)
            {
                //Index of chosen font
```

```
                    int selectedFontIndex
                        = fontSelectionModel
                            .selectedIndexProperty()
                            .getValue();
                    //Chosen font is set as the official font
                    theFont.set(Font.font((String)
                        availableFonts
                        .get(selectedFontIndex),14));
                }
            });



        //The  font of fontLabel gets bound to
        //and dependent from the chosen font
        fontLabel.fontProperty().bind(theFont);
        //stage is filled in
        stage.setScene(scene);
        stage.sizeToScene();
        stage.setTitle("Choose a font");
        stage.show();

    }

    public static void main(String[] args) {
        Application.launch("mybinding.SimpleBindingMain");
    }
}//End of main class L120 FontProperties
```

121 ***Exercise.*** *Run the program and play with the code.*

122 ***Connecting two graphical objects with a third one.*** *The next marvelous example of the literature shows the very spirit behind the binding procedures of JavaFX.*

```
//Program L122 ConnectedBall
/*
    This program illustrates how to use a line
```

```
    to bind two balls.
    JavaFx inbuilt properties are used to clearly
    show the spirit behind JavaFX binding procedures.

    Slight modification  from:
    skrb / ConnectedBall.java; Marz 2014.
    https://gist.github.com/skrb/4335101

 */

package ejvolu12p;

import javafx.application.Application;

import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.input.MouseEvent;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Line;
import javafx.stage.Stage;



public class ConnectedBall extends Application
{

//Inner class Ball based on a circle
class Ball extends Circle
{
    private double dragBaseX;
    private double dragBaseY;

    public Ball(double centerX, double centerY,
                double radius)
    {
        super(centerX, centerY, radius);

        setOnMousePressed((MouseEvent event) ->
```

```
        {
            dragBaseX = event.getSceneX() - getCenterX();
            dragBaseY = event.getSceneY() - getCenterY();
        });

        setOnMouseDragged((MouseEvent event) ->
        {
            setCenterX(event.getSceneX() - dragBaseX);
            setCenterY(event.getSceneY() - dragBaseY);
        });
    }
}


//Inner class ConnectionLine that creates a line
//that is by its very nature the connecting line between
//two balls.
class ConnectionLine extends Line
{
    public ConnectionLine(Ball startBall, Ball endBall)
    {
        //Start point of line is bound to first ball
        startXProperty().bind(startBall.centerXProperty());
        startYProperty().bind(startBall.centerYProperty());
        //End point of line is bound to second ball
        endXProperty().bind(endBall.centerXProperty());
        endYProperty().bind(endBall.centerYProperty());
    }
}



@Override
public void start(Stage stage) throws Exception
{
    Group root = new Group();

    initShapes(root);

    Scene scene = new Scene(root, 400, 400);
```

```
    stage.setScene(scene);
    stage.setTitle("Drag anyone of the two balls");
    stage.show();
}

//Two balls and a connecting line are delcared, specified
//and booked
private void initShapes(Group root)
{
    //Declaration and specification of center and radious
    Ball ball1 = new Ball(100, 200, 20);
    ball1.setFill(Color.RED);
    //Booking
    root.getChildren().add(ball1);
    Ball ball2 = new Ball(300, 200, 20);
    ball2.setFill(Color.RED);
    root.getChildren().add(ball2);
    //Connecting line is declared and specified
    ConnectionLine connectionLine
            = new ConnectionLine(ball1, ball2);
    connectionLine.setStroke(Color.CYAN);
    connectionLine.setStrokeWidth(5);
    //Booking of line.
    //Change 0 by 1, 2, 3
    root.getChildren().add(0, connectionLine);
}


public static void main(String... args)
{
    launch(args);
}
}//End of main class L122 ConnectedBall
```

123 ***Exercise.*** *Run the program and play with the code.*

124 ***Challenge.*** *Add to the previous code the possibility to drag the line but without distorting the unity of the three elements. You might consider to use a bidirectional binding.*

Since we like sliders to get input data, it would be good to get acquainted with the next example of the literature:

125 ***A slider controls a progress bar.*** *The code follows:*

```
//Program L125  SliderbarBinding

/**
 *Adapted from
 *
 * 8 True Stories about JavaFX, skrb /2014
 * http://skrb.hatenablog.com/entry/2014/06/25/195708
 */

package sliderbarbinding;

import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.ProgressBar;
import javafx.scene.control.Slider;
import javafx.stage.Stage;


public class SliderbarBinding extends Application
{



 @Override
public void start(Stage stage) throws Exception
{
    Group root = new Group();


    //Declaration and instantiation of a slider
```

```
    Slider slider = new Slider (0.0, 1.0, 0.0);
    //Position
    slider.setLayoutY(20);

    //Declaration and instantiation of a progress bar
    ProgressBar  bar = new ProgressBar();
    //Position
    bar.setLayoutY(80);
    //Dimensions are homogenized
    bar.setPrefSize(135, 20);
    //Booking
    root.getChildren().addAll(slider,bar);

    //A helping double property is declared
    DoubleProperty value = new SimpleDoubleProperty();
    //Helping property gets bound to the slider reading
    value.bind(slider.valueProperty());
    //Reading of progress bar is bound to value
    bar.progressProperty().bind(value);


    Scene scene = new Scene(root, 400, 400);
    stage.setScene(scene);
    stage.setTitle("Drag the thumb of the slider");
    stage.show();
}

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

}//End of main class L125  SliderbarBinding
```

126 ***Exercise.*** *Run the program and play with the code.*

The next code achieves a binding by brute force:

127 ***A binding without properties.*** *A timeline is used to control the separation*

of two rectangles by means of a slider. The code follows:

```
//Program L127 SliderBinding
//A slider is used to control the distance
//in within two rectangles.
//Code uses no JavaFX properties.

package ejvolu12p;



import java.util.Random;
import javafx.animation.KeyFrame;
import javafx.animation.SequentialTransition;
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.beans.value.ObservableValue;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.layout.GridPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.util.Duration;


public class SliderBinding extends Application
{

//Slider declaration and symbolic initialization
final Slider distanceSlider = new Slider(0, 200, -1);
```

```
//Message for the User
final Label promptMessage
        = new Label("Drag the thumb of the slider!");
final static Color messageColor = Color.BLUE;

//Width of rects
private static final int width = 63;
//Height of each rect
private static final int height = 63;
//Generator of pseudo random numbers
private static final Random rd = new Random();


//Value given by the slider
DoubleProperty newValue = new SimpleDoubleProperty(1);

//Overall drawing process
private static final  SequentialTransition St
            = new SequentialTransition();


//Duration of graphical events
private static final Duration d
            = Duration.millis(100);

//A random color is synthesized
private static Color randomColor()
 {
    //Color is defined as a random
    //mix of red, green and blue
    int r,g,b;
    r = rd.nextInt(226);
    g = rd.nextInt(226);
    b = rd.nextInt(226);
    //The color is prepared
    Color c = Color.rgb(r,g,b);
    return c;
 }
```

```
//A rect is created.
private static  Rectangle
           createRectangle(int x, int y)
{
    //Upper left corner at (x,y)
    Rectangle rect = new Rectangle(x,y,width,height);
    //The color is prepared
    Color c = randomColor();
    //The rect is painted
    rect.setFill(c);
    return rect;
}



@Override
public void start(Stage stage)
{
    Group root = new Group();
    Scene scene = new Scene(root, 700, 200);
    stage.setScene(scene);
    stage.setTitle("Slider show");

    //Setting of parameters of the slider
    distanceSlider.setMin(0);
    distanceSlider.setMax(200);
    distanceSlider.setValue(0);
    distanceSlider.setShowTickLabels(true);
    distanceSlider.setShowTickMarks(true);
    distanceSlider.setMajorTickUnit(100);
    distanceSlider.setMinorTickCount(5);
    distanceSlider.setBlockIncrement(10);
    //Message is given a color
    promptMessage.setTextFill(messageColor);

    //Two rects
    Rectangle rect1 =  createRectangle(217,55);
    Rectangle rect2 =  createRectangle(280, 55);
    root.getChildren().add(rect1);
    root.getChildren().add(rect2);
```

```
//Change of slider is detected
//and used to command separation of rects.
KeyFrame kFSlider
            = new KeyFrame
    (d, (ActionEvent event) ->
    {
        //Value of sliding is read
        distanceSlider.valueProperty()
                .addListener((ObservableValue<?
                    extends Number> ov,
                    Number old_val, Number new_val) ->
                {
                    //Rect2 is moved accordingly
                    rect2.setTranslateX(0.625*new_val
                            .doubleValue());
                });
    });

//Timeline for binding slider change to
//separation of rectangles
Timeline tl = new Timeline();
tl.setCycleCount(Timeline.INDEFINITE);
//Graphcial event is added to the timeline
tl.getKeyFrames().add(kFSlider);
//Timeline is added to sequentialTransition
St.getChildren().addAll( tl);



//A  pane with units
GridPane grid = new GridPane();
grid.setPadding(new Insets(10, 10, 10, 10));
//Vertical and horizontal units
grid.setVgap(10);
grid.setHgap(70);
//Position
GridPane.setConstraints(distanceSlider, 1,0);
grid.getChildren().add(distanceSlider);
```

```
    //Message is positioned
    GridPane.setConstraints(promptMessage, 0,10);
    //Message is booked
    grid.getChildren().add(promptMessage);
    //Pane is booked
    root.getChildren().add(grid);

    stage.show();
    St.play();
}

public static void main(String[] args)
{
    launch(args);
}
}//End of main class L127 SliderBinding
```

128 ***Exercise.*** *Run the program and play with the code.*

129 ***Exercise.*** *Rewrite the previous program in terms of JavaFX properties and their bindings.* Answer

7.5 Check Boxes

The necessity of the next control was asserted while a new version of Worms was being developed.

130 ***CheckBoxes.*** *These are boxes that admit parallel selection. By contrast, a toggle group of radio buttons admits exclusive selection. The next code illustrates how CheckBoxes operate.*

```
//Program L130 CheckBoxShow
//These are boxes that admit parallel
//selection-deselection.
```

```
package ejvolu12p;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.value.ObservableValue;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.FlowPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;

public class CheckBoxShow extends Application
{
    //Three rectangles
    Rectangle rectBlack = new Rectangle(90, 20);
    Rectangle rectYellow = new Rectangle(90, 20);
    Rectangle rectRed = new Rectangle(90, 20);
    //Names of rects
    final String[] names = new String[]
            {"BLACK", "YELLOW", "RED"};
    //Colors of rects
    final Color[] colors = new Color[]
        {Color.BLACK, Color.YELLOW, Color.RED};
    //CheckBoxes
    final CheckBox[] cbs = new CheckBox[names.length];

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


@Override
public void start(Stage stage)
{
    Scene scene = new Scene(new Group());
```

```
stage.setTitle("Checkbox Sample");
stage.setWidth(450);
stage.setHeight(150);

//Three rectangles are instantiated
//Round corners
rectBlack.setArcHeight(10);
rectBlack.setArcWidth(10);
//Color
rectBlack.setFill(Color.BLACK);


rectYellow.setArcHeight(10);
rectYellow.setArcWidth(10);
rectYellow.setFill(Color.YELLOW);


rectRed.setArcHeight(10);
rectRed.setArcWidth(10);
rectRed.setFill(Color.RED);

//Properties of CheckBoxes
for (int i = 0; i < names.length; i++)
{
    final CheckBox cb = cbs[i] = new CheckBox(names[i]);
    cbs[i].setSelected(true);

}

//Listener of   CheckBox 0
cbs[0].selectedProperty().addListener(
    (ObservableValue<? extends Boolean> ov,
    Boolean old_val, Boolean new_val) ->
    {
        rectBlack.setFill( new_val ? colors[0] : null);

    });

//Listener of   CheckBox 1
cbs[1].selectedProperty().addListener(
```

```
        (ObservableValue<? extends Boolean> ov,
        Boolean old_val, Boolean new_val) ->
        {
            rectYellow.setFill( new_val ? colors[1] : null);

        });

    //Listener of  CheckBox 2
    cbs[2].selectedProperty().addListener(
        (ObservableValue<? extends Boolean> ov,
        Boolean old_val, Boolean new_val) ->
        {
            rectRed.setFill( new_val ? colors[2] : null);

        });

    //Container for CheckBoxes
    VBox vbox = new VBox();
    vbox.getChildren().addAll(cbs);
    vbox.setSpacing(5);


    //Container for rects
    FlowPane flowPane = new FlowPane();
    flowPane.getChildren().add(rectBlack);
    flowPane.getChildren().add(rectYellow);
    flowPane.getChildren().add(rectRed);
    //Great container
    HBox root = new HBox();
    root.getChildren().add(vbox);
    root.getChildren().add(flowPane);
    root.setSpacing(40);
    root.setPadding(new Insets(20, 10, 10, 20));
    ((Group) scene.getRoot()).getChildren().add(root);
    stage.setScene(scene);
    stage.show();
}
}//End of main class L130 CheckBoxShow
```

131 ***Exercise.*** *Run the program and play with the code.*

Let us learn how to generate random colors with a procedure that is JavaFX specific. There are 148 colors with a proper name in JavaFX that we put into an array.

132 ***The next code shows how to generate and manage random colors in JavaFX.***

```
//Program L132 CopyJavaFXColor
//This program chooses at random one JavaFX color,
//fills a rect in  it,
//reads the applied color from the rect,
//and paints a second rect in the same color.

package ejvolu12p;


import java.lang.reflect.Field;
import java.util.Random;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;


public class CopyJavaFXColor extends Application
{

    //Generator of pseudo random numbers
    private static final Random rd = new Random();

   @Override
   public void start(final Stage stage) throws Exception
   {
      final Group rootGroup = new Group();
      final Scene scene
```

```
        = new Scene(rootGroup, 400, 300, Color.WHITE);
stage.setScene(scene);
stage.setTitle("JavaFX colors + get + copy");
final VBox colorsPane = new VBox();
//Colors with a name in JavaFX
final Field[] fields = Color.class.getFields();
Color AllColors[] = new Color[1000];

int N = 0;
System.out.println("Color number, color as string,"
          + " name of color in JavaFX");
//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 with colors
         AllColors[N] = color;
         //The name of the color
         final String colorName = field.getName();
         System.out.println("N = " + N + "  "
              + color.toString() + " " + colorName);
         N++;
      }
      catch (IllegalAccessException illegalAccessEx)
      {
         System.out.println("IllegalAccesException");
      }
   }
}

//Two rectangles are created
Rectangle rect1 = new Rectangle(30,40,50,70);
Rectangle rect2 = new Rectangle(20,30,100,200);
//A color is chosen at Random.
//The 0-th color is transparent: it is left out
int i = rd.nextInt(N-1)+1;
```

```
      //First rect is filled in chosen color
      rect1.setFill(AllColors[i]);
      i = rd.nextInt(N-1)+1;
      //Second rect is filled in another random color
      rect2.setFill(AllColors[i]);
      //To test the methods of JavaFX,
      //the color is read from rect1 and applied to rect2.
      //The final result must be two rects
      //painted in the same color.
      rect2.setFill(rect1.getFill());
      //Booking
      colorsPane.getChildren().addAll(rect1,  rect2);
      rootGroup.getChildren().add(colorsPane);
      stage.show();

   }


   public static void main(final String[] arguments)
   {
      Application.launch(arguments);
   }
}//End of main class L132 CopyJavaFXColor
```

133 ***Exercise.*** *Run the program and play with the code.*

7.6 Conclusion

We have seen various tools of JavaFX to deal with problems that can be dispensed if only one can get satisfied with solutions that have been tailored for work at home. These were related mainly to the huge amount of memory demanded by JavaFX animations and unresponsiveness of long-running task. But if one aspires to produce professional solutions, one cannot avoid sophistication. As we have seem, JavaFX was precisely devised to be satisfactory to the professional while trying to be accessible to the beginner. It seems to us that small troubles in the path of sophistication are solved by very complex machineries that must be ex-profeso designed and that need a long time for maturation, i.e., the present distribution is prey of bugs and the same is expected from some future ones.

Chapter 8

Worms in JavaFx

```
Each task is daunting
```

134 ***Introduction and purpose.*** *We explored in the previous chapter some ideas in relation with unresponsiveness of long-running tasks and animations that consume too much memory. So, let us apply them to improve the code about worms.*

8.1 Let us try it out

We begin by sharing a trouble that has strongly sued us as the next exercise shows.

135 ***Comment.*** *We developed program L135 WormsEvoJFX6 (not inluded): the program correctly implements the process of Randomness and of Human design (the exactness of copying colors was not tested because it is irrelevant for these cases). What did happened when dealing with reproduction by random sampling? In that case, each worm of the next generation is a clone of a worm of the present generation that is chosen at random. To achieve the cloning task, a second list of worms, WormsTWo, was used in previous versions. To test the idea that the creation of new worms created interference with the first ones, the second list was deleted and a helping list of Shadows was introduced. This list t has no JavaFX objects but nevertheless keeps all the relevant information of worms. These shadows are then transferred to the List Worms. The Author considers that this new copying methodology correctly copies the moves that instruct worms to climb. This was inferred after looking exhaustively to printing of the whole content of worms and shadows. But the copy process of colors fails. Both conclusions seem to be naturally drawn by letting the program run during 50 generations. It happens that if a worm is not sampled, it disappears forever while sampled worms may have various copies. So, by generation 50 the whole population consists of clones of a*

single worm and so all of them must have the same dynamic behavior and the same colors in all links. The dynamic behavior is perfect but colors not. Therefore, we have here a bug in Java or most probably in JavaFX. Let us notice that we have previously tested whether or not JavaFX methods to copy-paste color works. For that aim, we specifically developed program L132 CopyJavaFXColor. Nevertheless, program L135 WormsEvoJFX6 presents a remedy to another bug: if a new link is added to all worms, it is not visible in the 0-th worm. The remedy was to command twice to draw that link. These bugs operates neither for Randomness nor for Human Design but only for Evolution under reproduction by random sampling (others subprocesses of evolution was not tested as yet).

136 ***Circumventing the color-copy-paste-bug off.***

This bug consist in that the program does not copy exactly the colors of links when a worm is replaced by another when Evolution under reproduction by random sampling is run. We instinctively have tied the bug to the very method reproductionByRandomsampling(). We have made many attempts to understand and circumvent this bug:

- To see what happened at detail, worms and shadows were furnished with methods to print their complete content. So much information was useless.
- The default value of the setCycleCount property is 1. Nevertheless, the value was explicitly set to 1 all around. That was not the problem.
- Transitions are nested and must conform to a tree. This is achieved naturally if one adds a transition to the transition of the next higher rank that is nearest upwards in the code. If one adds a transition to an undue node, interference is caused to the animation that can get spoiled. This was revised but nothing suspicious was found.
- To decide whether or not the cause was a problem with the getting and setting of color, links were commanded to be painted in a single color. This was executed perfectly. So, getting and setting seem to be OK.
- To rule out any possible interference in within timelines and lists of Worms, copy- paste operations were changed in such a way that properties of list were not used: an array of colors was defined in within each worm and used to keep colors and to get them back. This rendered no positive result. Additionally, lists of complex objects were changed by arrays. No positive result was found.

- To know whether or not there was interference from others JavaFX items, say, menus, controls for worms, type of process and fossils that are booked to a tabPane, all that was nullified by surrounding the code with the tags /* */. No positive result was found.

- To test the incidence of the very color-copy-paste JavaFX methods, three different modes were tried: a fillTransition with start and final states specified, by using a keyValue and at last by using an EventHandler. A full timeline with start and final times specified was not tested.

- To understand the phenomenon at detail, a command to publish the new color for link 0 was added at various places. It appeared that the expected cloning of colors after having achieved monomorphism, as evidenced by the moves of worms, was not observed. The copy-paste procedure from worm to shadow was found to operate correctly but the procedure that reads the color from the shadow, applies it to the worm and reads it for publication fails. This seems to imply that the setting of color by the corresponding JavaFX method that works for a simple demo surrenders itself to the attack of an unknown factor.

- An intermediate rect that was declared as local variable was deleted and replaced by a global static field. No change was observed.

- To research thread troubles, a command to print change of color was inserted directly into the EventHandler that changes color. Suspicion was not verified.

- The study of the output of all those commands to print colors evidenced that worms never changed their colors. This was contrary to reality: worms changed colors of its links but the changes were not the due ones. This contradiction could be explained by assuming that worms change colors but that this operation is automatically reversed. In consequence, the setAutoReverse property was set to false (actually, this is its default value). Nothing happened. Anyway, at the end of the simulation a non programmed change of color used to appear. This change disappear. Animated by this positive result, the same operation was added to the transition that shows a recently added link. This was done because when a new link was added to all worms, it was not visible in worm 0. Nothing happened.

- Panes (Gordon [3], 2013) appeared as a possible cause of interference so, they were revised. The only suspicious fact was that the outer Pane was not declared static. After corresponding amendment nothing happened.

- To try to circumvent the problem, colors were decided to be managed outside worms, in arrays that are fields, globl variables (arrays attached to worms or shadows did not do it). This functioned and was adopted as our solution.

- We have employed some 50 hours battling with this color-copy-paste bug. Many tracks of our battle can be found in program L136 WormsEvoJFX7, not included. Hundreds of experiences of this type has programmed our mind to reject by instinct the Evolutionary Theory: where are the tracks of the battle with so many bug that there should had happened along the development of the software that sustains life to achieve such a marvelous simplicity that I see in my eyes? Nowhere: life is and has been perfect. So, the Evolutionary Theory, be it Darwinian or Lamarckian, is obviously false.

137 ***Challenge.*** *We have not won the battle against the color-copy-paste bug but have circumvented it. Make an effort to win: no bug must be remedied with ad hoc solutions.*

138 ***Arrays else lists.*** *We might consider now the option to choose arrays else list to implement our program Worms. Since evolution is open ended, just as list are while arrays are not, it would be magnificent to choose lists in spite of the fact that lists consume more memory and more time to be processed. Nevertheless, when trying to implement this directive to deal with colors, we failed even when we used a trick to resolve conflicts among lists and thread: it is to use specific constructors as in the next lines:*

```
//Worms are kept in a list
private static List<Worm> Worms
    = Collections.synchronizedList(new ArrayList<>());
//Helping list
private static List<Shadow> Shadows
    = Collections.synchronizedList(new ArrayList<>());
```

These efforts are consigned in program L138 WormsEvoJFX8, not included.

139 ***Comment.*** *More improvements are presented in program L139 WormsEvoJFX9, not included. the code.*

Our next task is to enable the possibility to break a long running task derived from a large number of generations. To achieve this, the code must undergo a very deep restructuring because the computational process and the animation must be

separated one from another in order that the Service Class, responsible for canceling a computational task, could operate. This means that we must run a virtual simulation (one without JavaFX). That is why we introduce the new class VirtualWorm that is the same as Worms with exception that JavaFX rectangles of Worms are replaced by Corners, a list of Point2D items that keep the position of the upper left corner of the rectangles.

140 ***Program L140 WormsEvoJFXa10, not included, is a modified code that enables future separation of computing task (a virtual animation) and animation. In this version, VirtualWorms go shoulder to shoulder with Worms.***

141 ***Comment.*** *If many generations of evolution are programmed, the ensuing computation can be broken from the Menu Program $->$ Cancel. This was enabled by the new class* `LongTask extends Service<String>` *in line 2917.*

142 ***Exercise.*** *In program L140 WormsEvoJFXa10 almost every method that process a Worm has a duplicate to process a VirtualWorm. Duplication of genetic code has been put forward as a fundamental mechanism for natural evolution. But we see here an example of a systematic duplication of many methods in a reuse of software made by a human being. How would you argue that the present systematic duplication was not made by natural evolution but by a human being? Answer*

143 ***Exercise.*** *Previous program runs well until the appearing of JavaFX objects in line 2974 in withing the class LongTask. Separate this code from LongTask that it could be run outside the Service Class to get a full virtual simulation with correct charts. Hint: separation is not enough because of troubles with threads. So, you need to revise program L105 StartStopRestart2 to see how to command to do something upon complete execution of LongTask. Answer*

We have a virtual simulation (without JavaFX objects) in version L143 WormsEvoJFXa11 and a truly JavaFX simulation in Program L139 WormsEvoJFX9. Our final task is put them together in such a way that the JavaFX simulation could appear as an option. There are many ways of doing that. Our aim cannot be to minimize the creation of new code because that will be done if two parallel branches are defined from the beginning, one with animation and the other without it. This solution is of no help to test the virtual simulation. So, our strategy is to keep all virtual worms in the list RecordedWorms that is next handed over to the animation that must execute their moves. Our first step is the following:

144 ***The code program L144 WormsEvoJFXa12, not included, separates virtual animation from the real, JavaFX animation. Arrangements were made until***

Human design runs well. *A bug that hid a recently added link to worm zero was cured: it was due to reuse of worm 0 as a recipient for enlivened fossils. We stopped reuse and created a new worm, helpingWorm, and that was all to it.*

145 ***Fake exercise.*** *Run the program and play with the code.*

146 ***Exercise for the Author.*** *Take program L144 WormsEvoJFXa12 and make it running not only for worms under Human Design but also under Randomness and Evolution. Fix the bug that draws a little square near to the top at right hand of the screen. Make sure that to see the JavaFX animation is optional and that runs for whatever number of worms one chooses.* Answer

147 ***Challenge.*** *We have made some changes to a given program whose fundamental move was a massive duplication. So, discuss the following proposition: duplication is the more frequent move along evolution of software that is driven by human intelligence.*

8.2 Our final version

We at last are ready to present what we consider is a satisfactory program.

148 ***This is it.*** *We ordered the methods of our previous program according to their functional role. The possibility to run and repeat a virtual simulation (one that does not touch at the JavaFX GUI) was enabled. Other improvements were tried out. The result is presented in program L148 WormsEvoJFXa14. The code, with nearly 5000 lines, is included in the attached .zip file.*

149 ***Exercise.*** *Run the program and play with the code.*

8.3 Graduation

150 ***To graduate from this volume you have various realizable tasks and some challenges that include opportunities to exercise your personality:***

1. *Selectively enable printing capabilities of results and add some new ones to test the correctness of the code part by part. You might also use the debugger of your platform. Correct found bugs. Report to me (jose@evoljava.com) conceptual bugs, those that cause an unfair implementation of a given process.*

2. *Rewrite the displayed information to fit you personal beliefs and wisdom. Keep in mind that the EvolJava Community has no official belief. Instead, it is conformed by persons that have beliefs and that love to test them using Java, if only that is possible.*

3. *One option afforded by the program is to repeat a simulation. In that case no chart is produced. Add a chart, and the way to save it to clipboard, that draws the series of relative frequencies of imperfect fossils along repeats. Hint: imitate the procedure that builds the chart newLinkChart that draws the generation at which an additional link was added.*

4. *Colors of worms track reproduction as by random sampling as by selection. Extend this to all other subprocesses of evolution.*

5. *Implement the comparative study of all processes.*

6. *JavaFX colors include some very light versions. This may cause problems for the beholder. So, enable the possibility to darken a color. This can be done by passing a color through the operator* `.darker()`*, which operate as a suffix.*

7. *Fossils and shadows were originally created to fit different purposes but after many modifications they ended to be the same. This originates redundant code. Carry out the corresponding simplification.*

8. *The animation can be calculated and played completely or it can be nullified. Include an intermediate mode in which only successful generations are shown. Include other options.*

9. *The saved memory when the animation is nullified is important but not sufficient. At present 50 megas are needed for no animation and more than 100 with animation. Would you craft something really functional?*

10. *Our Worms represent what must be the standard of the study of evolution: at the same time that shows that evolution is capable of solving problems it also shows that this capability has a price: to make many unsuccessful trials. They are so many that they might dominate every fossil study. So, you can try to extract a template of the code that it may be easily adapted to other problems. Of course, a booklet with appropriate instructions is also welcome.*

11. *Once sparked, present version of the program cannot be reset. The Author considers that resetting is not worth the effort. If you disagree, give a try.*

The implementation of the first three tasks gives you the right to add your name to the list of contributors to this software and to distribute your new version.

8.4 Conclusion

Much effort was invested in trying to improve the code. Nevertheless, we did not got rid of the pre-processing time of produced animation and the ensuing unresponsiveness. A question springs: Do we have the correct insight and the appropriate style?

Chapter 9

Fossils and the falsification of the Evolutionary theory

Ideas after hard work

151 ***Challenge.*** *Express your personal comments in regard with the role of fossil reconstruction and the assessment of the idea that the Evolutionary Theory is obviously false.*

Answers to exercises

Problems of Chapter 3

20, page 40. The next code uses JavaFX to move a circle horizontally:

```
//Program L20 MovingCircle
//JavaFX makes a circle move horizontally

package movingcircle;

import java.util.Random;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;
import javafx.util.Duration;


public class MovingCircle extends Application
{
    //Turn on  the random generator
    private static final Random r  = new Random();
    //Duration of events
    private static final Duration d = Duration.millis(2000);
```

```
    //Inbuilt macro: translation
    TranslateTransition translateTransition ;


    private void init(Stage primaryStage)
    {

        Group root = new Group();
        primaryStage.setResizable(false);
        //Dimensions of the screen
        primaryStage.setScene(new Scene(root, 270, 120));
        //We will work with circles
        Circle circle =  createCircle();
            root.getChildren().addAll(circle);
        //Translation is instantiated
        translateTransition =
                new TranslateTransition(d, circle);
        //Vertical shift from initial position
        translateTransition.setByX(100);
        //Turn on Autoreverse
        translateTransition.setAutoReverse(true);
        //Number of repetitions.
        translateTransition.setCycleCount(5);
    }


    //A circle ready to be used is built
    private Circle  createCircle()
    {
        //Center at (100,70)
        //Radius = 50
        Circle circle = new Circle(100,70, 50);

//A color is prepared
        Color c = Color.FUCHSIA;
        //The circle is given a color
        circle.setFill(c);
```

```
        return circle;
    }


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

    @Override public void stop()
    {
        translateTransition .stop();
    }

    //This method is automatically called at the beginnig.
    @Override public void start(Stage primaryStage)
            throws Exception
    {
        init(primaryStage);
        primaryStage.show();
        play();
    }


    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {
        launch(args);
    }

}//End of main class L20 MovingCircle
```

22, page 44. The next JavaFX code makes a set of five circles move in zig-zag at the same time.

```
//Program L22 FiveCircles
//Five circles move in zigzag.

package ejvolu12p;

import java.util.Random;
import javafx.animation.ParallelTransition;
import javafx.animation.SequentialTransition;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;
import javafx.util.Duration;


public class FiveCircles extends Application
{

    //Everything is organized sequentially.
    private static SequentialTransition zigzag;

    //Turn on  the random generator
    private static final Random r  = new Random();
    //Duration of events
    private static final Duration d = Duration.millis(2000);

    //Starting the film
    private void init(Stage primaryStage)
    {
        //JavaFX objects are organized as a tree.
        Group root = new Group();
        primaryStage.setResizable(false);
        //Dimensions of the screen
        primaryStage.setScene(new Scene(root, 350, 600));
        //We will work with circles
```

```
Circle Circle[] = new Circle[20];
int nCircles = 5;
//Circles are created, positioned and booked
for(int i = 0; i < nCircles; i++)
{
    Circle[i] = createCircle((i+1)* 40);
    root.getChildren().addAll(Circle[i]);
}

 //Choreography: general organization
zigzag = new SequentialTransition();
//Two cycles of zigzag
for(int j = 0; j < 2; j++)
{
    //Synchronous movement to the right
    ParallelTransition goRight  =
            new ParallelTransition();
    //All circles are moved
     for (int i = 0; i < nCircles; i++)
    {
        //Translation
        TranslateTransition translateTransition =
             new TranslateTransition(d, Circle[i]);
        //Diagonal movement
        translateTransition.setByX(100);
        translateTransition.setByY(100);
        translateTransition.setCycleCount(1);
        //translateTransition is used here and dies.
        goRight.getChildren().
            addAll(translateTransition);
    }

    //Synchronous movement to the left
    ParallelTransition goLeft  =
                  new ParallelTransition();
    //All circles are moved
     for (int i = 0; i < nCircles; i++)
    {
```

```
                //Translation
                TranslateTransition translateTransition =
                     new TranslateTransition(d, Circle[i]);
                //Diagonal movement
                translateTransition.setByX(-100);
                translateTransition.setByY(100);
                translateTransition.setCycleCount(1);
                //translateTransition is used here and dies.
                goLeft.getChildren().
                    addAll(translateTransition);
            }

            zigzag.setAutoReverse(true);
            zigzag.getChildren().addAll(goRight,goLeft);
        }
    }


    //A circle ready to be used is built
    private Circle  createCircle(int x)
    {
        //Center at (x,100)
        //Radius = 10
        Circle circle = new Circle(x, 100, 10);
        //Color is defined as a random
        //mix of red, green and blue
        int rr,gg,bb;
        rr = r.nextInt(226);
        gg = r.nextInt(226);
        bb = r.nextInt(226);
//The color is prepared
        javafx.scene.paint.Color c =
                javafx.scene.paint.Color.rgb(rr,gg,bb);
        //The circle is given a color
        circle.setFill(c);

        return circle;
    }
```

```
    public void play()
    {
        zigzag.play();
    }

    @Override public void stop()
    {
        zigzag.stop();
    }

    //This method is automatically called at the beginnig.
    @Override public void start(Stage primaryStage)
            throws Exception
    {
        init(primaryStage);
        primaryStage.show();
        play();
    }


    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {
        launch(args);
    }

}//End of main class L22 FiveCircles
```

25, page 47. A movie with two rectangles that move synchronically along the same timeline:

```
//Program L25 Move2Rect
//A movie with two rectangles
```

```
//that move synchronically
//along a timeline.


package ejvolu12p;

import java.util.Random;
import javafx.animation.Interpolator;
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.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.util.Duration;


public class Move2Rect extends Application
{

    //A movie needs a clock and a schedule
    private final Timeline timeline = new Timeline();

    //Turn on  the random generator
    static Random r  = new Random();
    //Starting the film
    private void init(Stage primaryStage)
    {
        //Tree of space-like JavaFX objects.
        Group root = new Group();
        primaryStage.setResizable(false);
        //Dimensions of the screen
        primaryStage.setScene(new Scene(root, 300, 300));
```

```
        //Create rectangles that can move.
        //The type of movement is dictated by
        //the interpolator: it is linear.
        Rectangle rect1 =
          createMovingRectangle(Interpolator.LINEAR,20,20);
        Rectangle rect2 =
          createMovingRectangle(Interpolator.LINEAR,20,140);
        //Rects are attached to the root
        root.getChildren().addAll(rect1, rect2);
    }

    //How a moving rectangle is built.
    //Interpolator: the law for movement.
    private Rectangle
        createMovingRectangle(Interpolator interpolator,
                              int x, int y)
    {
        //Upper left corner at (x,y)
        //width = 100, hight = 100
        Rectangle rect = new Rectangle(x,y, 100, 100);
        //Make round corners with span 20
        rect.setArcHeight(20);
        rect.setArcWidth(20);
        //Color is defined as a random
        //mix of red, green and blue
        int rr,gg,bb;
        rr = r.nextInt(226);
        gg = r.nextInt(226);
        bb = r.nextInt(226);
//The color is prepared
        Color c = Color.rgb(rr,gg,bb);
        //The rect is painted
        rect.setFill(c);

        //Create a timeline for moving the rectangle:
        //cycle over and over
        timeline.setCycleCount(Timeline.INDEFINITE);
        //Go forth and back
```

```
        timeline.setAutoReverse(true);

        //Define in Keyvalue what to do with rect:
        //move it  along the X axis 155 units
        //according to the law given by the interpolator
        KeyValue keyValue =
                new KeyValue(rect.translateXProperty(),
                                    155, interpolator);

        //Execute the keyValue using prescribed duration
        KeyFrame keyFrame =
              new KeyFrame(Duration.seconds(4), keyValue);

        //Add the keyframe to the timeline
        timeline.getKeyFrames().add(keyFrame);

        return rect;
    }

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

    @Override public void stop()
    {
        timeline.stop();
    }

    //This method is automatically called at the beginnig.
    @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 L25 Move2Rect
```

27, page 51. A choreography with 3 rectangles and one circle:
Answer:

```
/*
 * Program L27 Move4Figures
 * A timeline choreography with 3 rects and one circle.
 * Processes that happen at the same time are attached
 * to the same timeline.
 * Processes that happen sequencially are attached to
 * a sequentialTransition.
 */

package ejvolu12p;

import java.util.Random;
import javafx.animation.Interpolator;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.SequentialTransition;
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;

import javafx.scene.Group;
import javafx.scene.Scene;

import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.util.Duration;
```

```
public class Move4Figures extends Application
{

    //Our movie has three processes or scenes
    //and three schedules
    private final Timeline timeline1 = new Timeline();
    private final Timeline timeline2 = new Timeline();
    private final Timeline timeline3 = new Timeline();
    //This is the time marshall:
    //it dictates how scenes appear along the film
    private final SequentialTransition animation  =
                          new SequentialTransition();
    //Turn on  the random generator
    static Random r  = new Random();
    //Starting the movie
    private void init(Stage primaryStage)
    {
        //Root of the tree of space-like objects.
        Group root = new Group();
        primaryStage.setResizable(false);
        //Dimensions of the screen
        primaryStage.setScene(new Scene(root, 300, 600));

        //Create rectangles.
        Rectangle rect1 =
                createRectangle(20,20);
        Rectangle rect2 =
                createRectangle(20,140);
        Rectangle rect3 =
                createRectangle(20,260);
        //Create a circle
        Circle circle1 = createCircle();
        //Figures are booked
        root.getChildren().addAll(rect1, rect2,
                                  rect3, circle1);
```

```
//Define in Keyvalue what to do with rect1:
//move it  along the X axis   155 pixels
//according to the law given by the interpolator
KeyValue keyValue1 =
        new KeyValue(rect1.translateXProperty(),
                     155, Interpolator.LINEAR);
KeyValue keyValue2 =
        new KeyValue(rect2.translateXProperty(),
                     155, Interpolator.LINEAR);

//Execute  synchronically keyValues using prescribed time
KeyFrame keyFrame1 =
        new KeyFrame(Duration.seconds(4),
                     keyValue1,keyValue2);
//Set timeline1 with two cycles
timeline1.setCycleCount(2);
//Go forth and back
timeline1.setAutoReverse(true);

//Add keyframe1 to  timeline1.
//All keyValues attached to keyFrame1
//are executed synchronically because they
//belong in the same timeline.
timeline1.getKeyFrames().add(keyFrame1);

//Set timeline2
timeline2.setCycleCount(2);
timeline2.setAutoReverse(true);

//Define in Keyvalue what to do with rect3:
//move it  along the X axis   155 pixels
//according to the law given by the interpolator
KeyValue keyValue3 =
        new KeyValue(rect3.translateXProperty(),
                     155, Interpolator.LINEAR);

//Execute  keyValue3 using prescribed time
KeyFrame keyFrame2 =
```

```
            new KeyFrame(Duration.seconds(4), keyValue3);

        //Add   keyframe2 to timeline2
        timeline2.getKeyFrames().add(keyFrame2);



        //Define the movement of circle1:
        //shift it upwards 450 pixels.
        KeyValue keyValue4 =
                new KeyValue(circle1.translateYProperty(),
                             -450, Interpolator.LINEAR);

        //Execute  keyValue4 using prescribed time
        KeyFrame keyFrame3 =
              new KeyFrame(Duration.seconds(4), keyValue4);

        //Set timeline3 with two cycles
        timeline3.setCycleCount(2);
        //Go forth and back
        timeline3.setAutoReverse(true);
        //Add keyframe3 to   timeline3
        timeline3.getKeyFrames().add(keyFrame3);


        //Organization of  scenes along the movie:
        //animation is not a timeline,
        //it is a sequenticalTransition so,
        //listed timelines are executed in sequence.
        animation.getChildren().addAll(timeline1,
                timeline2, timeline3);
        //Repetition mode
        animation.setCycleCount(2);
    }

    //A rectangle is returned.
    private Rectangle  createRectangle( int x, int y)
    {
```

```
        //Upper left corner at (x,y)
        //width = 100, hight = 100
        Rectangle rect = new Rectangle(x,y, 100, 100);
        //Make round corners with span 20
        rect.setArcHeight(20);
        rect.setArcWidth(20);
        //Color is defined as a random
        //mix of red, green and blue
        int rr,gg,bb;
        rr = r.nextInt(226);
        gg = r.nextInt(226);
        bb = r.nextInt(226);
//The color is prepared
        Color c = Color.rgb(rr,gg,bb);
        //The rect is painted
        rect.setFill(c);

        return rect;
    }

    //A circle ready to be used is built
    private Circle  createCircle()
    {
        //Center at (70,520)
        //Radious = 50
        Circle circle = new Circle(70,520, 50);

//A color is prepared
        Color c = Color.FUCHSIA;
        //The circle is given a color
        circle.setFill(c);

        return circle;
    }

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

```
    }

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

    //This method is automatically called at the beginnig.
    @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 L27 Move4Figures
```

33, page 61. The next code presents some actors that are rectangles, then introduces a new one and then removes the formers.

```
//Program L33 SixRects
//Five rectangles   move in zigzag and
//on finished a new rectangle   appears
//and moves downwards.
//To end, the newcomer is left alone in the stage.

//Since the sixth rectangle is created and booked
//at the beginning,
//it is given width 0 until the moment
//when it is scheduled to appear:
//at that time it is given a positive width
//and its performance begins.
//The five former rectangles are scaled down to zero
```

```
//and in that way they disappear.


package ejvolu12p;


import javafx.scene.shape.Rectangle;
import java.util.Random;
import javafx.animation.Interpolator;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.ParallelTransition;
import javafx.animation.PauseTransitionBuilder;
import javafx.animation.SequentialTransition;
import javafx.animation.Timeline;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.event.ActionEvent;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.util.Duration;


public class SixRects extends Application
{

    //Everything is organized sequentially.
    private static SequentialTransition all;
    private static SequentialTransition zigzag;

    //Turn on  the random generator
    private static final Random r  = new Random();
    //Duration of events
    private static final Duration d = Duration.millis(2000);
```

```
private void init(Stage primaryStage)
{
    Group root = new Group();
    primaryStage.setResizable(false);
    //Dimensions of the screen
    primaryStage.setScene(new Scene(root, 400, 600));
    //We will work with rectangles
    Rectangle[] Rectangles = new Rectangle[20];
    int nRects = 5;
    //Rectangles are created, positioned and booked
    for(int i = 0; i < nRects; i++)
    {
        Rectangles[i] = createRectangle((i+1)* 40);
        root.getChildren().addAll(Rectangles[i]);
    }
    //A new actor is created
    Rectangle rect6 = createRectangle(300);
    //The new actor is hidden
    rect6.setHeight(0);
    rect6.setWidth(0);
    //Actor is booked
    root.getChildren().addAll(rect6);


    //Choreography: general organization
    zigzag = new SequentialTransition();
    //Two cycles of zigzag
    for(int j = 0; j < 2; j++)
    {
        //Synchronous movement to the right
        ParallelTransition goRight  =
                new ParallelTransition();
        //All rects are moved
         for (int i = 0; i < nRects; i++)
        {
        //Translation
        TranslateTransition translateTransition =
```

```
            new TranslateTransition(d, Rectangles[i]);
        //Diagonal movement
        translateTransition.setByX(100);
        translateTransition.setByY(100);
        translateTransition.setCycleCount(1);
        //translateTransition is used here and dies.
        goRight.getChildren().
                addAll(translateTransition);
        }

        //Synchronous movement to the left
        ParallelTransition goLeft  =
                       new ParallelTransition();
        //All rects are moved
         for (int i = 0; i < nRects; i++)
        {
        //Translation
        TranslateTransition translateTransition =
            new TranslateTransition(d, Rectangles[i]);
        //Diagonal movement
        translateTransition.setByX(-100);
        translateTransition.setByY(100);
        translateTransition.setCycleCount(1);
        //translateTransition is used here and dies.
        goLeft.getChildren().
                addAll(translateTransition);
        }

        zigzag.setAutoReverse(true);
        zigzag.getChildren().addAll(goRight,goLeft);

    }

    //Hidden rect6 is shown
    SequentialTransition show = showST(rect6);
    //Visible rect6 is translated
    TranslateTransition translateTransition =
                 new TranslateTransition(d, rect6);
```

```
        translateTransition.setByY(400);
                translateTransition.setCycleCount(1);


        //Some actors are removed from the stage
        SequentialTransition remove = removeST(Rectangles);

        all = new SequentialTransition();
        //translateTransition is used here and dies.
        all.getChildren().
            addAll(zigzag,show,translateTransition,remove);
    }

    //A rectangle is created
    private static  Rectangle
           createRectangle(int x )
    {
        //Upper left corner at (x,100)
        //width = 10, hight = 10.
        //Hidden rect is shown later.
        Rectangle rect = new Rectangle(x, 100, 10, 10);
        //Make round corners with span 20
        //rect.setArcHeight(20);
        //rect.setArcWidth(20);
        //Color is defined as a random
        //mix of red, green and blue
        int rr,gg,bb;
        rr = r.nextInt(226);
        gg = r.nextInt(226);
        bb = r.nextInt(226);
//The color is prepared
        javafx.scene.paint.Color c
                = javafx.scene.paint.Color.rgb(rr,gg,bb);
        //The rect is painted
        rect.setFill(c);


    return rect;
```

```
}

//The new actor rect6 is shown:
//Its width is scaled from 0 to 100
private SequentialTransition showST(Rectangle rect6)
{
    Duration d2 =   Duration.millis(1);

    //Rect6 must be shown at true scale
    KeyValue keyValue1 =
            new KeyValue(rect6.scaleXProperty(),
                         1, Interpolator.LINEAR);
    KeyValue keyValue2 =
            new KeyValue(rect6.scaleYProperty(),
                         1, Interpolator.LINEAR);
    Timeline b = new Timeline(
            new KeyFrame(d2,keyValue1,keyValue2 ));

    SequentialTransition s = new SequentialTransition();
    //Width of  rect6 is enlarged and after this
    //it is shown according to the
    //instructions given by timeline b.
    //To do this, a pause of one millisecond is added.
    s.getChildren().add(
            new SequentialTransition(

                PauseTransitionBuilder.create().
                    duration(Duration.ONE).onFinished
                    ((ActionEvent t) ->
                    {rect6.setWidth(100);
                    rect6.setHeight(100);}).build(),b));
    return s;
}

//Five former rectangles are removed:
//they are scaled down to zero.
private SequentialTransition
    removeST(Rectangle[] Rectangles)
```

```
{
    Duration d2 =   Duration.millis(1);
    SequentialTransition ss = new SequentialTransition();
    for(int i = 0; i < 5; i++)
    {
        //Rectangles are scaled down to zero
        KeyValue keyValue1 =
            new KeyValue(Rectangles[i].scaleXProperty(),
                         0, Interpolator.LINEAR);
        KeyValue keyValue2 =
            new KeyValue(Rectangles[i].scaleYProperty(),
                         0, Interpolator.LINEAR);
        //KeyValue2 is inserted into a timeline
        Timeline b = new Timeline(
            new KeyFrame(d2,keyValue1,keyValue2 ));
        //Timeline is inserted into a transition
        ss.getChildren().addAll(b);
    }
    return ss;
}



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

@Override public void stop()
{
    all.stop();
}

//This method is automatically called at the beginnig.
@Override public void start(Stage primaryStage)
        throws Exception
{
```

```
        init(primaryStage);
        primaryStage.show();
        play();
    }


    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {
        launch(args);
    }

}//End of main class L33 SixRects
```

35, page 63. The next code adds some texts to a dynamics with rectangles.

```
//Program L35 Textual2
//Some informative text is added to a scene:
//Five rectangles   move in zigzag and
//on finished a new rectangle   appears
//and moves downwards.
//To end, the newcomer is left alone in the stage.

//Since the sixth rectangle is created and booked
//at the beginning,
//it is given width 0 until the moment
//when it is scheduled to appear:
//at that time it is given a positive width
//and its role begins.
//The five former rectangles are scaled down to zero
//and it that way they disappear.


package textual2;
```

```
import javafx.scene.shape.Rectangle;
import java.util.Random;
import javafx.animation.AnimationTimer;
import javafx.animation.Interpolator;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.ParallelTransition;
import javafx.animation.PauseTransitionBuilder;
import javafx.animation.SequentialTransition;
import javafx.animation.Timeline;
import javafx.animation.TranslateTransition;
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.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.text.Text;
import javafx.stage.Stage;
import javafx.util.Duration;


public class Textual2 extends Application
{

    //Everything is organized sequentially.
    private static SequentialTransition all;
    private static SequentialTransition zigzag;

    //Turn on  the random generator
    private static final Random r  = new Random();
    //Duration of events
    private static final Duration d = Duration.millis(2000);
    //Counter (of instants of time)
    private Integer counter = 0;
```

```
//Starting the film
private void init(Stage primaryStage)
{
    Group root = new Group();
    primaryStage.setResizable(false);
    //Dimensions of the screen
    primaryStage.setScene(new Scene(root, 400, 600));
    //A circle at the upper left corner
    final Circle circle = new Circle(20,  Color.BISQUE);

    //Constant text
    final Text text = new Text ("Welcome!" );
    text.setStroke(Color.BLACK);
    text.setLayoutX(20);
    text.setLayoutY(20);

    //A changing text that simulates
    //a chronometer
    final Text watch = new Text ("");
    watch.setLayoutX(40);
    watch.setLayoutY(40);
    //Watch is bound to a chronometer in arbitrary units
    AnimationTimer timer = new AnimationTimer()
    {
        @Override
        public void handle(long l)
        {
            watch.setText("Time  = " + counter.toString());
            counter++;
        }
    };
    timer.start();


    root.getChildren().addAll(circle, text, watch);

    //Moving rectangles
```

```
Rectangle[] Rectangles = new Rectangle[20];
int nRects = 5;
//Rectangles are created, positioned and booked
for(int i = 0; i < nRects; i++)
{
    Rectangles[i] = createRectangle((i+1)* 40);
    root.getChildren().addAll(Rectangles[i]);
}
//A new actor is created
Rectangle rect6 = createRectangle(300);
//The new actor is hidden
rect6.setHeight(0);
rect6.setWidth(0);
//Actor is booked
root.getChildren().addAll(rect6);


//Choreography: general organization of scenes
zigzag = new SequentialTransition();
//Two cycles of zigzag
for(int j = 0; j < 2; j++)
{
    //Synchronous movement to the right
    ParallelTransition goRight  =
            new ParallelTransition();
    //All rects are moved
     for (int i = 0; i < nRects; i++)
    {
    //Translation
    TranslateTransition translateTransition =
        new TranslateTransition(d, Rectangles[i]);
    //Diagonal movement
    translateTransition.setByX(100);
    translateTransition.setByY(100);
    translateTransition.setCycleCount(1);
    //translateTransition is used here and dies.
    goRight.getChildren().
            addAll(translateTransition);
```

```
    }

    //Synchronous movement to the left
    ParallelTransition goLeft  =
                  new ParallelTransition();
    //All rects are moved
     for (int i = 0; i < nRects; i++)
    {
    //Translation
    TranslateTransition translateTransition =
        new TranslateTransition(d, Rectangles[i]);
    //Diagonal movement
    translateTransition.setByX(-100);
    translateTransition.setByY(100);
    translateTransition.setCycleCount(1);
    //translateTransition is used here and dies.
    goLeft.getChildren().
            addAll(translateTransition);
    }

    zigzag.setAutoReverse(true);
    zigzag.getChildren().addAll(goRight,goLeft);

}

//Hidden rect6 is shown
SequentialTransition show = showST(rect6);
//Visible rect6 is translated
TranslateTransition translateTransition =
             new TranslateTransition(d, rect6);
translateTransition.setByY(400);
        translateTransition.setCycleCount(1);


//Some actors are removed from the stage
SequentialTransition remove = removeST(Rectangles);

all = new SequentialTransition();
```

```
        //translateTransition is used here and dies.
        all.getChildren().
            addAll(zigzag,show,translateTransition,remove);
    }

    //A rectangle is created
    private static  Rectangle
           createRectangle(int x )
    {
        //Upper left corner at (x,100)
        //width = 10, hight = 10.
        //Hidden rect is shown later.
        Rectangle rect = new Rectangle(x, 100, 10, 10);
        //Make round corners with span 20
        //rect.setArcHeight(20);
        //rect.setArcWidth(20);
        //Color is defined as a random
        //mix of red, green and blue
        int rr,gg,bb;
        rr = r.nextInt(226);
        gg = r.nextInt(226);
        bb = r.nextInt(226);
//The color is prepared
        javafx.scene.paint.Color c
                = javafx.scene.paint.Color.rgb(rr,gg,bb);
        //The rect is painted
        rect.setFill(c);


    return rect;
    }

    //The new actor rect6 is shown:
    //Its width is scaled from 0 to 100
    private SequentialTransition showST(Rectangle rect6)
    {
        Duration d2 =   Duration.millis(1);
```

```
    //Rect6 must be shown at true scale
    KeyValue keyValue1 =
            new KeyValue(rect6.scaleXProperty(),
                         1, Interpolator.LINEAR);
    KeyValue keyValue2 =
            new KeyValue(rect6.scaleYProperty(),
                         1, Interpolator.LINEAR);
    Timeline b = new Timeline(
            new KeyFrame(d2,keyValue1,keyValue2 ));

    SequentialTransition s = new SequentialTransition();
    //Width of  rect6 is enlarged and after this
    //it is shown according to the
    //instructions given by timeline b.
    //To do this, a pause of one millisecond is added.
    s.getChildren().add(
            new SequentialTransition(

                PauseTransitionBuilder.create().
                    duration(Duration.ONE).onFinished
                    ((ActionEvent t) ->
                    {rect6.setWidth(100);
                    rect6.setHeight(100);}).build(),b));
    return s;
}

//Five former rectangles are removed:
//they are scaled down to zero.
private SequentialTransition
    removeST(Rectangle[] Rectangles)
{
    Duration d2 =   Duration.millis(1);
    SequentialTransition ss = new SequentialTransition();
    for(int i = 0; i < 5; i++)
    {
        //Rectangles are scaled down to zero
        KeyValue keyValue1 =
            new KeyValue(Rectangles[i].scaleXProperty(),
```

```
                            0, Interpolator.LINEAR);
            KeyValue keyValue2 =
                new KeyValue(Rectangles[i].scaleYProperty(),
                            0, Interpolator.LINEAR);
            //KeyValue2 is inserted into a timeline
            Timeline b = new Timeline(
                new KeyFrame(d2,keyValue1,keyValue2 ));
            //Timeline is inserted into a transition
            ss.getChildren().addAll(b);
        }
        return ss;
    }



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

    @Override public void stop()
    {
        all.stop();
    }

    //This method is automatically called at the beginnig.
    @Override public void start(Stage primaryStage)
            throws Exception
    {
        init(primaryStage);
        primaryStage.show();
        play();
    }


    /**
     * @param args the command line arguments
```

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

}//End of main class L35 Textual2
```

40, page 68. The next code shows tics + jumps of tree actors, two permanent and one transient.

```
//Program L40 Jumps2
//For every tic of a clock,
//random jumps happen.
//There are tree actors: two permanent
//but a third is transient:
//he enters at some time,
//makes a performance and leaves.

package ejvolu12p;

import java.util.Random;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.util.Duration;

public class Jumps2 extends Application
{
```

```
//Clock
private Timeline timeline;
//Generator of pseudo random numbers
private final Random r = new Random();
//Counter of tics
private static Integer nTics = 0;


@Override public void start(Stage stage)
{
    Group p = new Group();
    Scene scene = new Scene(p);
    stage.setScene(scene);
    stage.setWidth(200);
    stage.setHeight(200);

    //first actor
    Rectangle rect1 = new Rectangle(0,0,20,20);
    rect1.setFill(Color.BURLYWOOD);

    //Second actor
    Rectangle rect2 = new Rectangle(30,30,20,20);
    rect2.setFill(Color.CHOCOLATE);

    //The third actor is hidden:
    //it has no width nor height
    Rectangle rect3 = new Rectangle(50,50,0,0);
    rect3.setFill(Color.DEEPPINK);

    //Actors are booked
    p.getChildren().addAll(rect1, rect2, rect3);
    stage.show();

    //Instanciate the clock
    timeline = new Timeline();
    //Automatic repetition of tics
    timeline.setCycleCount(Timeline.INDEFINITE);
```

```
//Duration in within tics
Duration duration = Duration.millis(2000);
//A jump is programmed at random
EventHandler jumpTo;
jumpTo = (EventHandler<ActionEvent>)
    (ActionEvent t) ->
    {
        //Jump of rect1
        int toX1 =  r.nextInt(100);
        int toY1 =  r.nextInt(100);
        rect1.setTranslateX(toX1);
        rect1.setTranslateY(toY1);
        //Jump of rect2
        int toX2 =  r.nextInt(100);
        int toY2 =  r.nextInt(100);
        rect2.setTranslateX(toX2);
        rect2.setTranslateY(toY2);
        //Performance of the third actor
        if ((5 < nTics) & (nTics < 10))
        {
            //Appearance
            rect3.setHeight(20);
            rect3.setWidth(20);
            //Jump
            int toX3 =  r.nextInt(100);
            int toY3 =  r.nextInt(100);
            rect3.setTranslateX(toX3);
            rect3.setTranslateY(toY3);
        }
        else
        {
            //Disappearance
            rect3.setHeight(0);
            rect3.setWidth(0);
        }
        //Tic is counted
        nTics++;
```

```
            };
        //Tic and jump  are synchronized
        KeyFrame keyFrame = new KeyFrame(duration, jumpTo);
        //The new ensemble is booked
        timeline.getKeyFrames().add(keyFrame);
        timeline.play();

    }


    public static void main(String[] args)
    {
        Application.launch(args);
    }
  } //End of main class L40 Jumps2
```

43, page 72. A planet orbits along an elliptic orbit around a sun.

```
//Program L43 Jumps4
//A planet spins along an elliptical orbit
//around a sun.
//An ellipse is a deformation of a circle:
//it is enlarged a times in the x-direction
//and b times in the y-direction.
//The sun is at the left focus.
//x-coordinate of the left focus
//c= sqrt(a^2- b^2)

package ejvolu12p;

import java.util.Random;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
```

```
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;
import javafx.util.Duration;

public class Jumps4 extends Application
{

    //Clock
    private Timeline timeline;
    //Generator of pseudo random numbers
    private final Random r = new Random();
    //Counter of tics
    private static Integer nTics = 0;
    //The radius of the sun
    private static Integer rSun = 10;
    //The radius of the mother circular orbit
    private static Integer radiusOrbit = 100;
    //Angular velocity
    private static double omega = 0.005;
    //Initial position of the planet
    private static double x = 0;
    private static double y = radiusOrbit;
    //Paramters of the ellipse
    private static double a = 2;
    private static double b = 1;
    private static double c = Math.sqrt(a+a-b*b);
    //Time is discrete
    private static Integer t = 0;

    @Override public void start(Stage stage)
    {
        Group p = new Group();
        Scene scene = new Scene(p, 520, 420);
        stage.setScene(scene);
```

```
//The sun is centered
Circle sun = new Circle(0, c,rSun);
sun.setFill(Color.GOLD);

//The planet begins down under the sun
Circle planet = new Circle(0,radiusOrbit,3);
planet.setFill(Color.LIGHTSTEELBLUE);

//Pane with ensemble sun + planet
final StackPane stack = new StackPane();
stack.getChildren().addAll(sun, planet);
//Centering
int shiftX = 300;
int shiftY = 200;
stack.setLayoutX(shiftX);
stack.setLayoutY(shiftY);
//This was found by trial and error
y = y - shiftY/2;
x = x + shiftX/2;


p.getChildren().add(stack);



stage.show();

//Instanciate the clock
timeline = new Timeline();
//Automatic repetition of tics
timeline.setCycleCount(Timeline.INDEFINITE);
//Duration in within tics = dt = 1 = one unit of time
Duration duration = Duration.millis(1);
//A jump is programmed along an ellipse
EventHandler jumpTo;
jumpTo = (EventHandler<ActionEvent>)
    (ActionEvent e) ->
    {
```

```
                //tic
                //Jump of the planet
                x = x - a*omega*radiusOrbit
                        * Math.sin(omega* t);
                y = y + b*omega*radiusOrbit
                        * Math.cos(omega* t);
                planet.setTranslateX(x);
                planet.setTranslateY(y);
                //The tic is counted
                t++;
            };
        //Tic and jump  are synchronized
        KeyFrame keyFrame = new KeyFrame(duration, jumpTo);
        //The new ensemble is booked
        timeline.getKeyFrames().add(keyFrame);
        timeline.play();
    }


    public static void main(String[] args)
    {
        Application.launch(args);
    }
  } //End of main class L43 Jumps4
```

44, page 72. A planet orbits along an elliptic orbit around a sun. Jumps were changed by soft transitions.

```
//An ellipse is a deformation of a circle:
//it is elarged a times in the x-direction
//and b times in the y-direction.
//x-coordinate of the left focus
//c= sqrt(a^2- b^2).
//Jumps were changed by soft transitions.

package jumps5;


import static javafx.animation.Animation.INDEFINITE;
```

```
import javafx.animation.SequentialTransition;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;
import javafx.util.Duration;

public class Jumps5 extends Application
{

    //Counter of tics
    private static final Integer nTics = 0;
    //The radius of the sun
    private static final Integer rSun = 10;
    //The radius of the mother circular orbit
    private static final Integer radiusOrbit = 100;
    //Angular velocity
    private static final double omega = 0.001;
    //Initial position of the planet
    private static double x = 0;
    private static double y = radiusOrbit;
    //Paramters of the ellipse
    private static final double a = 2;
    private static final double b = 1;
    private static final double c = Math.sqrt(a+a-b*b);
    //Time is discrete
    private static Integer t = 0;

    @Override public void start(Stage stage)
    {
        //The scene
        Group p = new Group();
        Scene scene = new Scene(p, 520, 420);
        stage.setScene(scene);
```

```
//The sun is centered
Circle sun = new Circle(0, c,rSun);
sun.setFill(Color.GOLD);

//The planet begins down under the sun
Circle planet = new Circle(0,radiusOrbit,3);
planet.setFill(Color.LIGHTSTEELBLUE);

//Ensemble sun + planet
final StackPane stack = new StackPane();
stack.getChildren().addAll(sun, planet);
//Centering
int shiftX = 300;
int shiftY = 200;
stack.setLayoutX(shiftX);
stack.setLayoutY(shiftY);
//This was found by trial and error
y = y - shiftY/2;
x = x + shiftX/2;
planet.setTranslateX(x);
planet.setTranslateY(y);
//Booking
p.getChildren().add(stack);
stage.show();

//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);
            t++;
        }
        s.play();
    }


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

    }
} //End of main class L44 Jumps5
```

53, page 104. The next code in JavaFX instructs links of worms to move at random. Repetitions are allowed. To begin anew, worms are repositioned at the startline but information to move links is reset.

```
//Program L53 WormsRandJFX3
//A population of worms is programmed
//by randomness to
//climb the screen.
```

```
//Repetitions are allowed. To that aim,
//worms are repositioned at the
//start line to begin a new try.
//Program is written in JavaFX.

package ejvolu12p;

import java.util.Random;
import javafx.animation.ParallelTransition;
import javafx.animation.SequentialTransition;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.util.Duration;



public class WormsRandJFX3 extends Application
{
    private static WormsRandJFX3 w = new WormsRandJFX3();
    //Overall process
    private static  SequentialTransition nRepetitionsST
                    = new SequentialTransition();
    //Number of repetitions
    private static int nRepetitions = 5;
    private static int repCounter = 0;


    private static   Group p = new Group();

    //Number of steps
    int nSteps = 3;
    //Number of worms
    private static final int nWorms = 5;
```

```
    //Worms are kept in an array
    private static Worm[] Worms = new Worm[nWorms];
    //Number of links of each worm
    private static final int nLinks = 9;
    //Initial position  of the head
    private static int initialRow = 180;
    //Generator of pseudo random numbers
    private static final Random r = new Random();
    //Each worm has nMovers movers,
    //instructions to move.
    private static int  nMovers = 3;
    //Width of the worm = width of each link
    private static int width = 4;
    //Height of each link
    private static int height = 12;
    //Vector with 9 possible moves
    private static Move  vectMoves[ ] = new Move[9];

    private static final boolean print1 = false;
    //Initial position of worms
    private static int[][][]  xy = new int[nWorms][nLinks][2];


    /* Definition of inner classes.
    *
    The Move class:
    A Move says in which direction
    a link of the worm must move.
    */

 private  class Move
 {
    int x;
    int y;


    //Default initialization
    Move()
```

```
    {
        x= 0;
        y= 0;
    }

    // Initialize with two coordinates
    Move(int a, int b)
    {
        x= a;
        y= b;
    }

    // Copy a move
    Move(Move move)
    {
        x= move.x;
        y= move.y;
    }
}//End of class Move


// Definition of the class Mover:
// A Mover keeps two types of information:
// a) the link of the worm to be moved and
// b) the direction or move to be chosen.
//
private   class Mover
{
    int link ;
    Move   move;

    //  Movers can acquired information
    // in three ways, they are  overloaded.
    // Form 1: by a list
    private Mover(int l, int mx, int my)
     {
        link= l;
        move.x = mx;
```

```
        move.y = my;
     }

    // Form 2, by structured data
    private Mover(int l, Move m)
    {
        link= l;
        move = m;
        move.x = m.x;
        move.y = m.y;
    }

    // Form 3, by structured data
    private Mover(Mover mover)
    {
        link= mover.link;
        move =  new Move(mover.move);
    }
  }//End of class Mover



    private static  Rectangle
           createRectangle()
    {
        //Upper left corner at (0,0)
        Rectangle rect = new Rectangle(0,0,width,height);
        //Color is defined as a random
        //mix of red, green and blue
        int rr,gg,bb;
        rr = r.nextInt(226);
        gg = r.nextInt(226);
        bb = r.nextInt(226);
//The color is prepared
        Color c = Color.rgb(rr,gg,bb);
        //The rect is painted
        rect.setFill(c);
```

```
    return rect;
    }

/* Here we define the class worm.
    A worm is composite: it has links.
    Each link has a rectangular shape
    and  its own color,
    which is defined by three numbers.
    The vector Mover[ ]
     keeps the information
    on how to move its links:
    which  link and in which order
    (in the order that they appear in
    the vector Mover[]), and
    in which direction.
    Each worm keeps record of
    the actual position of each
    one of its links,
    in the associated ractangle. */


private class Worm
{
    //Number of links
    int numberOfLinks = nLinks;
    //Number of movers = instructions to move
    int numberOfMovers = nLinks;
    //mover[] keeps movers
    Mover mover[ ] = new Mover[nLinks];
    //The shape of each link
    Rectangle rect[] = new Rectangle[20];
    //move right (side = 1) else left (side = -1)
    int side = 2*r.nextInt(2) -1;
```

```
  //Default initialization
   private Worm( )
   {
//Rects are created.
        // A color is assigned to each link
        // but different than white
        for(int i = 0; i < numberOfLinks; i++)
            rect[i] = createRectangle();
        //Instructions to move at random
for(int m = 0; m < nMovers; m++)
        {
            //Select at random  which link to move:
            int l = r.nextInt(nLinks) ;
            //Select at random  which direction to move on
            Move  move = w.new Move (vectMoves[r.nextInt(9)]);
            //Memorization
            mover[m]= w.new Mover(l,move);
        }
 }//End of constructor Worm()

}//End of class Worm



@Override public void start(Stage stage)
    {
        //The scene
        Scene scene = new Scene(p, 500,500);
        stage.setScene(scene);
        stage.setTitle("Links move at random");
        stage.show();
        //Definition of possible moves
        moveDef();

        nRepetitionsST.setCycleCount(1);
        for(int i = 0; i < nRepetitions; i++)
        {
```

```
            repCounter = i;
            //Worms are posited at the start line
            startLine();
            //Worms climb the screen
            SequentialTransition oneTrialST  =  climb();
            nRepetitionsST.getChildren().add(oneTrialST);
        }
    //Film is played
    nRepetitionsST.play();
    }


 //Nine basic moves = coordinates where to move in
static private void moveDef()
{
 final Move move0 =  w.new Move(0,1);
   vectMoves[0]   = move0 ;
 final Move move1 =  w.new Move(1,1);
   vectMoves[1]   = move1 ;
 final Move move2 =  w.new Move(1,0);
   vectMoves[2]   = move2 ;
 final Move move3 =  w.new Move(1,-1);
   vectMoves[3]   = move3 ;
 final Move move4 =  w.new Move(0,-1);
   vectMoves[4]   = move4 ;
 final Move move5 =  w.new Move(-1,-1);
   vectMoves[5]   = move5 ;
 final Move move6 =  w.new Move(-1,0);
   vectMoves[6]   = move6 ;
 final Move move7 =  w.new Move(-1,1);
   vectMoves[7]   = move7 ;
 final Move move8 =  w.new Move(0,0);
   vectMoves[8]   = move8 ;
}


// All worms are posited at the same initial row
// in vertical position.
```

```
static public void startLine()
{
    for (int W = 0; W < nWorms; W++)
    {
        int  x,y;
        //Auxiliary worm
        Worm   worm;
        if (repCounter == 0)
            worm  = w.new Worm();
        else
            worm = Worms[W];
        //Each link is positioned
        for(int  link = 0; link < nLinks; link++)
        {
            //Horizontal position
            x = 30 + 50 * W;
            //Vertical position
            y = initialRow + height*link;

           //Position is updated
            worm.rect[link].setX(x);
            worm.rect[link].setY(y);
            //Link is booked for the   0-th trial
            if (repCounter == 0)
            {
                p.getChildren().add(worm.rect[link]);
                xy[W][link][0] = x;
                xy[W][link][1] = y;
            }
            else //link is repositioned
            {
                //Duration
                Duration d = Duration.millis(1);
                //A soft jump is programmed
                TranslateTransition translateTransition =
                    new TranslateTransition(d,
                                            worm.rect[link]);
                //Repositioning at initial position.
```

```
                //Instruction found by serendipity.
                translateTransition.setToX(0);
                translateTransition.setToY(0);



                //Number of repetitions
                translateTransition.setCycleCount(1);
                //Repositioning is booked
                nRepetitionsST.getChildren()
                        .add(translateTransition);
            }
            //Moves are redefined
            worm = resetMoves(worm);
        }
        Worms[W]= worm;
    }
}


//Moves at random are created
private static Worm resetMoves(Worm worm)
{
    //Instructions to move at random
    for(int m = 0; m < nMovers; m++)
    {
        //Select at random  which link to move:
        int l = r.nextInt(nLinks) ;
        //Select at random  which direction to move on
        Move  move = w.new Move (vectMoves[r.nextInt(9)]);
        //Memorization
        worm.mover[m]= w.new Mover(l,move);
    }
    return worm;
}

//To climb the screen the worm W
//translates each one of its links
```

```
//according to the information
//consigned in mover[].
//Move m is executed: it moves link m.
private static TranslateTransition
                    translate( int W, int m)
{
    if (print1)
        System.out.println("\n Worm = " + W
                           + " move = " + m);
    //Read the link to be moved
    int link = Worms[W].mover[m].link;
    //Read position
    int x = (int) Worms[W].rect[link].getX();
    int y = (int) Worms[W].rect[link].getY();
    if (print1)
        System.out.println(" x = " + x
                           + " y = " + y);
    //The move to be done is read
    Move  move1;
    move1 = Worms[W].mover[m].move;
    //Move is interpreted
    int byX = 4* move1.x;
    int byY = -6* move1.y;
    // A new place is defined:
    x = x + byX;
    y = y + byY;
    if (print1)
        System.out.println(" byX = " + byX
                           + " byY = " + byY);

    //Address is updated
    Worms[W].rect[link].setX(x);
    Worms[W].rect[link].setY(y);

    //Duration in within tics
    Duration d = Duration.millis(500);
    //A soft jump is programmed
    TranslateTransition translateTransition =
```

```
            new TranslateTransition(d, Worms[W].rect[link]);
        //Jump is executed
        translateTransition.setByX(byX);
        translateTransition.setByY(byY);
        //Number of repetitions
        translateTransition.setCycleCount(1);

        return translateTransition;
    }


//Worms climb the screen.
//Various transitions are nested to achieve that.
public    SequentialTransition  climb()
{
        //Overall process
        SequentialTransition oneTrialST
                        = new SequentialTransition();
         oneTrialST.setCycleCount(1);
         //worms advance nSteps steps
        for (int i = 0; i < nSteps; i++)
        {
            //Worms advance one step:
            //To move all links, each worm must execute
            //instructions to move in mover[]
            SequentialTransition oneStepST
                            = new SequentialTransition();
            oneStepST.setCycleCount(1);
            //Information is read and decoded
            for (int m = 0; m < nMovers; m++)
            {
                //All worms execute move m
                ParallelTransition moveMPT
                            = new ParallelTransition();
                moveMPT.setCycleCount(1);
                for (int W = 0; W < nWorms; W++)
                {
                    //A translation of some link
```

```
                //is programmed for worm W
                //according to move m
                TranslateTransition tT = translate(W,m);
                tT.setCycleCount(1);
                //Translation is booked
                moveMPT.getChildren().add(tT);
            }
            //The ensamble of translations is booked
            oneStepST.getChildren().add(moveMPT);
        }
    //The movement of worms in one step is booked
    oneTrialST.getChildren().add(oneStepST);
    }
    return oneTrialST;
}



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


} //End of main class L53 WormsRandJFX3
```

54, page 104. The next code in JavaFX instructs links of worms to move at random. Repetitions are allowed. To that aim, we begin an almost completely new world for each repetition. The problem is that JavaFX books all objects at the pure beginning so, they must be hidden at birth to be shown when it is needed.

```
//Program L54 WormsRandJFX4
//A population of worms is programmed
//by randomness to
//climb the screen.
//Repetitions are allowed for a certain
```

```
//number of generations.
//Each generations has its own
//population of worms.
//Program is written in JavaFX.

package ejvolu12p;

import java.util.Random;
import javafx.animation.Interpolator;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.ParallelTransition;
import javafx.animation.PauseTransitionBuilder;
import javafx.animation.SequentialTransition;
import javafx.animation.Timeline;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.util.Duration;



public class WormsRandJFX4 extends Application
{
    private static WormsRandJFX4 w = new WormsRandJFX4();
    //Overall process
    private static  SequentialTransition nGenerationsST
                    = new SequentialTransition();
    //Number of repetitions or generations
    private static int nGenerations = 5;
    private static int gen = 0;
```

```
    //The scene
    private static   Group p = new Group();


    //Number of steps
    int nSteps = 3;
    //Number of worms
    private static final int nWorms = 5;
    //Worms are kept in an array
    private static Worm[] Worms = new Worm[nWorms];
    //Number of links of each worm
    private static final int nLinks = 9;
    //Initial position   of the head
    private static int initialRow = 180;
    //Generator of pseudo random numbers
    private static final Random r = new Random();
    //Each worm has nMovers movers,
    //instructions to move.
    private static int  nMovers = 3;
    //Width of the worm = width of each link
    private static int width = 4;
    //Height of each link
    private static int height = 12;
    //Vector with 9 possible moves
    private static Move  vectMoves[ ] = new Move[9];

    private static final boolean print1 = false;



    /* Definition of inner classes.
    *
    The Move class:
    A Move says in which direction
    a link of the worm must move.
    */

private  class Move
```

```
{
   int x;
   int y;


   //Default initialization
   Move()
   {
       x= 0;
       y= 0;
   }

   // Initialize with   two coordinates
   Move(int a, int b)
   {
       x= a;
       y= b;
   }

   // Copy a move
   Move(Move move)
   {
       x= move.x;
       y= move.y;
   }
}//End of class Move


// Definition of the class Mover:
// A Mover keeps two types of information:
// a) the link of the worm to be moved and
// b) the direction or move to be chosen.
//
private   class Mover
{
   int link ;
   Move   move;
```

```
    //  Movers can acquired information
    // in three ways, they are  overloaded.
    // Form 1: by a list
    private Mover(int l, int mx, int my)
     {
        link= l;
        move.x = mx;
        move.y = my;
     }

    // Form 2, by structured data
    private Mover(int l, Move m)
    {
        link= l;
        move = m;
        move.x = m.x;
        move.y = m.y;
    }

    // Form 3, by structured data
    private Mover(Mover mover)
    {
        link= mover.link;
        move =  new Move(mover.move);
    }
  }//End of class Mover



//Rects that belongs in the worms of the first generation
 //are ready to move. Others are hidden.
private static  Rectangle
          createRectangle(int gen)
{
    //Upper left corner at (0,0)
    Rectangle rect = new Rectangle(0,0,width,height);
    //Color is defined as a random
    //mix of red, green and blue
```

```
    int rr,gg,bb;
    rr = r.nextInt(226);
    gg = r.nextInt(226);
    bb = r.nextInt(226);
    //The color is prepared
    Color c = Color.rgb(rr,gg,bb);
    //The rect is painted
    rect.setFill(c);
    //Rects of the first generation (gen = 0)
    //are ready to move,
    //otherwise they got hidden
    if (gen > 0)
    {
        rect.setHeight(0);
        rect.setWidth(0);
    }
    return rect;
}


/*  Here we define the class worm.
    A worm is composite: it has links.
    Each link has a rectangular shape
    and  its own color,
    which is defined by three numbers.
    The vector Mover[ ]
    keeps the information
    on how to move its links:
    which  link and in which order
    (in the order that they appear in
    the vector Mover[]), and
    in which direction.
    Each worm keeps record of
    the actual position of each
    one of its links,
    in the associated ractangle. */
```

```
private class Worm
{
    //Number of links
    int numberOfLinks = nLinks;
    //Number of movers = instructions to move
    int numberOfMovers = nLinks;
    //mover[] keeps movers
    Mover mover[ ] = new Mover[nLinks];
    //The shape of each link
    Rectangle rect[] = new Rectangle[20];
    //move right (side = 1) else left (side = -1)
    int side = 2*r.nextInt(2) -1;



    //Default initialization:
    //worms of the first generation are
    //ready to move. Others get hidden.
    private Worm(int gen )
    {

        // Rects are created.
        // A color is assigned to each link
        // but different than white
        for(int i = 0; i < numberOfLinks; i++)
            rect[i] = createRectangle(gen);
        //Instructions to move at random
for(int m = 0; m < nMovers; m++)
        {
            //Select at random  which link to move:
            int l = r.nextInt(nLinks) ;
            //Select at random  which direction to move on
            Move  move = w.new Move (vectMoves[r.nextInt(9)]);
            //Memorization
            mover[m]= w.new Mover(l,move);
        }
    }//End of constructor Worm(int  gen)
```

```
}//End of class Worm



@Override public void start(Stage stage)
    {
        //The scene
        Scene scene = new Scene(p, 500,500);
        stage.setScene(scene);
        stage.setTitle("Links move at random");
        stage.show();
        //Definition of possible moves
        moveDef();

        nGenerationsST.setCycleCount(1);
        for(int i = 0; i < nGenerations; i++)
        {
            gen = i;
            //Worms are posited at the start line
            startLine(gen);
            //Worms climb the screen
            SequentialTransition oneGenST  =  climb();
            nGenerationsST.getChildren().add(oneGenST);
            //Worms get hidden
            ParallelTransition hideST  =  hideST();
            if (gen < nGenerations-1)
            nGenerationsST.getChildren().add(hideST);
        }
    //Film is played
    nGenerationsST.play();
    }


 //Nine basic moves = coordinates where to move in
static private void moveDef()
{
 final Move move0 =  w.new Move(0,1);
```

```
   vectMoves[0]   = move0 ;
 final Move move1 =  w.new Move(1,1);
   vectMoves[1]   = move1 ;
 final Move move2 =  w.new Move(1,0);
   vectMoves[2]   = move2 ;
 final Move move3 =  w.new Move(1,-1);
   vectMoves[3]   = move3 ;
 final Move move4 =  w.new Move(0,-1);
   vectMoves[4]   = move4 ;
 final Move move5 =  w.new Move(-1,-1);
   vectMoves[5]   = move5 ;
 final Move move6 =  w.new Move(-1,0);
   vectMoves[6]   = move6 ;
 final Move move7 =  w.new Move(-1,1);
   vectMoves[7]   = move7 ;
 final Move move8 =  w.new Move(0,0);
   vectMoves[8]   = move8 ;
}


// All worms are posited at the same initial row
// in vertical position.
static public void startLine(int gen)
{
    for (int W = 0; W < nWorms; W++)
    {
        int  x,y;
        //Auxiliary worm
        Worm   worm = w.new Worm(gen);


        //Each link is positioned
        for(int  link = 0; link < nLinks; link++)
        {
            //Horizontal position
            x = 30 + 50 * W;
            //Vertical position
            y = initialRow + height*link;
```

```
            //Position is updated
             worm.rect[link].setX(x);
             worm.rect[link].setY(y);
             //Link is booked
             p.getChildren().add(worm.rect[link]);
             //Link is shown
             SequentialTransition show
                         = showST(worm.rect[link]);
             //Show transition is booked
             nGenerationsST.getChildren().add(show);
        }
        Worms[W]= worm;
    }

}



//To climb the screen the worm W
//translates each one of its links
//according to the information
//consigned in mover[].
//Move m is executed: it moves link m.
private static TranslateTransition
                    translate( int W, int m)
{
    if (print1)
        System.out.println("\n Worm = " + W
                           + " move = " + m);
    //Read the link to be moved
    int link = Worms[W].mover[m].link;
    //Read position
    int x = (int) Worms[W].rect[link].getX();
    int y = (int) Worms[W].rect[link].getY();
    if (print1)
        System.out.println(" x = " + x
                           + " y = " + y);
```

```
    //The move to be done is read
    Move  move1;
    move1 = Worms[W].mover[m].move;
    //Move is interpreted
    int byX = 4* move1.x;
    int byY = -6* move1.y;
    // A new place is defined:
    x = x + byX;
    y = y + byY;
    if (print1)
        System.out.println(" byX = " + byX
                           + " byY = " + byY);

    //Address is updated
    Worms[W].rect[link].setX(x);
    Worms[W].rect[link].setY(y);

    //Duration in within tics
    Duration d = Duration.millis(500);
    //A soft jump is programmed
    TranslateTransition translateTransition =
         new TranslateTransition(d, Worms[W].rect[link]);
    //Jump is executed
    translateTransition.setByX(byX);
    translateTransition.setByY(byY);
    //Number of repetitions
    translateTransition.setCycleCount(1);

    return translateTransition;
 }


//Worms climb the screen.
//Various transitions are nested to achieve that.
public    SequentialTransition  climb()
{
    //Overall process
    SequentialTransition oneGenST
```

```
                    = new SequentialTransition();
     oneGenST.setCycleCount(1);
     //worms advance nSteps
    for (int i = 0; i < nSteps; i++)
    {
        //Worms advance one step:
        //To move all links, each worm must execute
        //instructions to move in mover[]
        SequentialTransition oneStepST
                        = new SequentialTransition();
        oneStepST.setCycleCount(1);
        //Information is read and decoded
        for (int m = 0; m < nMovers; m++)
        {
            //All worms execute move m
            ParallelTransition moveMPT
                        = new ParallelTransition();
            moveMPT.setCycleCount(1);
            for (int W = 0; W < nWorms; W++)
            {
                //A translation of some link
                //is programmed for worm W
                //according to move m
                TranslateTransition tT = translate(W,m);
                tT.setCycleCount(1);
                //Translation is booked
                moveMPT.getChildren().add(tT);
            }
            //The ensamble of translations is booked
            oneStepST.getChildren().add(moveMPT);
        }
    //The movement of worms in one step is booked
    oneGenST.getChildren().add(oneStepST);
    }
    return oneGenST;
}
```

```
//A hidden rect gets shown
private static SequentialTransition showST(Rectangle rect)
{
    Duration d2 =   Duration.millis(1);

    //Rect must be shown at true scale
    KeyValue keyValue1 =
            new KeyValue(rect.scaleXProperty(),
                         1, Interpolator.LINEAR);
    KeyValue keyValue2 =
            new KeyValue(rect.scaleYProperty(),
                         1, Interpolator.LINEAR);
    Timeline b = new Timeline(
            new KeyFrame(d2,keyValue1,keyValue2 ));

    SequentialTransition s = new SequentialTransition();
    //Width of  rect is enlarged
    //and after this
    //it is shown according to the
    //instructions given by timeline b.
    //To do this, a pause of one millisecond is added.
    s.getChildren().add(
            new SequentialTransition(

                PauseTransitionBuilder.create().
                    duration(Duration.ONE)
                        .onFinished
                        ((ActionEvent t) ->
                        {
                            if (gen > 0)//Hidden -> show
                            {
                                rect.setWidth(width);
                                rect.setHeight(height);
                            }
                        })
                        .build(),b));
    return s;
}
```

```
//Worms are hidden after their performance.
private static ParallelTransition  hideST()
{

    ParallelTransition hidePT
                = new ParallelTransition();
    hidePT.setCycleCount(1);
    //Hide all worms
    for (int W = 0; W < nWorms; W++)
    {
        //Hide each link
        for(int  link = 0; link < nLinks; link++)
        {
            Duration d2 =   Duration.millis(1);
            SequentialTransition ss
                = new SequentialTransition();
            //Rectangles are scaled down to zero
            KeyValue keyValue1 =
                new KeyValue(Worms[W].rect[link]
                    .scaleXProperty(),
                         0, Interpolator.LINEAR);
            KeyValue keyValue2 =
                new KeyValue(Worms[W].rect[link]
                    .scaleYProperty(),
                         0, Interpolator.LINEAR);
            //KeyValues are  inserted into a timeline
            Timeline b = new Timeline(
                new KeyFrame(d2,keyValue1,keyValue2 ));
            //Timeline is inserted into a transition
            ss.getChildren().addAll(b);
            hidePT.getChildren().add(ss);
        }
    }
    return hidePT;
}
```

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


} //End of main class L54 WormsRandJFX4
```

56, page 117. Program L53 WormsRandJFX3 is rewritten using Lava List instead of Array. The program allows for repetitions by repositioning old worms at the startLine but with new information to move links and with the addition of one link per generation.

```
//Program L56 WormsRandJFX6
//A population of worms is programmed
//by randomness to
//climb the screen.
//Repetitions are allowed for a certain
//number of generations.
//For each new generation:
//worms are repositioned at startLine,
//the information to move is reset and
//one link is added.
//Program is written in JavaFX.

package ejvolu12p;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javafx.animation.Interpolator;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.ParallelTransition;
import javafx.animation.PauseTransitionBuilder;
import javafx.animation.SequentialTransition;
```

```
import javafx.animation.Timeline;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.util.Duration;



public class WormsRandJFX6 extends Application
{
    private static WormsRandJFX6 w = new WormsRandJFX6();
    //Overall process
    private static  SequentialTransition nGenerationsST
                    = new SequentialTransition();
    //Number of repetitions or generations
    private static int nGenerations = 3;
    private static int gen = 0;

    //The tree of spacelike JavaFX objects
    private static   Group p = new Group();


    //Number of steps
    int nSteps = 3;
    //Number of worms
    private static  int nWorms = 3;
    //Worms are kept in a list
    private static List<Worm> Worms = new ArrayList<>();
    //Number of links of each worm
    private static  int nLinks = 6;
    //Initial position over the screen of the head
    private static int initialRow = 180;
```

```
    //Generator of pseudo random numbers
    private static final Random r = new Random();
    //Each worm has nMovers movers,
    //instructions to move.
    private static int  nMovers = 3;
    //Width of the worm = width of each link
    private static int width = 4;
    //Height of each link
    private static int height = 12;
    //Array with 9 possible moves
    private static Move  vectMoves[ ] = new Move[9];

    private static final boolean print1 = false;




    /* Definition of inner classes.
    *
    The Move class:
    A Move says in which direction
    a link of the worm must move.
    */

private  class Move
{
    int x;
    int y;


    //Default initialization
    Move()
    {
        x= 0;
        y= 0;
    }

    // Initialize with  two coordinates
```

```
   Move(int a, int b)
   {
       x= a;
       y= b;
   }

   // Copy a move
   Move(Move move)
   {
       x= move.x;
       y= move.y;
   }
}//End of class Move


// Definition of the class Mover:
// A Mover keeps two types of information:
// a) the link of the worm to be moved and
// b) the direction or move to be chosen.
//
private   class Mover
{
   int link ;
   Move   move;

   //  Movers can acquired information
   // in three ways, they are  overloaded.
   // Form 1: by a list
   private Mover(int l, int mx, int my)
    {
       link= l;
       move.x = mx;
       move.y = my;
    }

   // Form 2, by structured data
   private Mover(int l, Move m)
   {
```

```
        link= l;
        move = new Move(m);
    }

    // Form 3, by structured data
    private Mover(Mover mover)
    {
        link= mover.link;
        move =  new Move(mover.move);
    }
  }//End of class Mover



//Rects that belongs in the worms of the first generation
//are ready to move. Others are hidden.
private static  Rectangle
          createRectangle(int gen)
{
    //Upper left corner at (0,0)
    Rectangle rect = new Rectangle(0,0,width,height);
    //Color is defined as a random
    //mix of red, green and blue
    int rr,gg,bb;
    rr = r.nextInt(226);
    gg = r.nextInt(226);
    bb = r.nextInt(226);
    //The color is prepared
    Color c = Color.rgb(rr,gg,bb);
    //The rect is painted
    rect.setFill(c);
    //Rects of the first generation (gen = 0)
    //are ready to move,
    //otherwise they got hidden
    if (gen > 0)
    {
        rect.setHeight(0);
        rect.setWidth(0);
```

```
    }
    return rect;
}


/*  Here we define the class worm.
    A worm is composite: it has links.
    Each link has a rectangular shape
    and  its own color,
    which is defined by three numbers.
    The vector Mover[ ]
    keeps the information
    on how to move its links:
    which  link and in which order
    (in the order that they appear in
    the vector Mover[]), and
    in which direction.
    Each worm keeps record of
    the actual position of each
    one of its links,
    in the associated ractangle. */



private class Worm
{
    //Number of links
    int numberOfLinks = nLinks;
    //Number of movers = instructions to move
    int numberOfMovers = nLinks;
    //List mover  keeps movers
    List<Mover> Mover = new ArrayList<>();
    //The shape of each link
    List<Rectangle> Rect = new ArrayList<>();
    //move right (side = 1) else left (side = -1)
    int side = 2*r.nextInt(2) -1;
```

```
    //Default initialization:
    //worms of the first generation are
    //ready to move. Others get hidden.
    private Worm()
    {
//Rects are created.
        // A color is assigned to each link
        // but different than white
        for(int i = 0; i < numberOfLinks; i++)
            Rect.add(createRectangle(gen));
        //Instructions to move at random
for(int m = 0; m < nMovers; m++)
        {
            //Select at random  which link to move:
            int l = r.nextInt(nLinks) ;
            //Select at random  which direction to move on
            Move  move = w.new Move (vectMoves[r.nextInt(9)]);
            //Memorization
            Mover.add(w.new Mover(l,move));
        }
    }//End of constructor Worm(int  gen)

}//End of class Worm



 //Nine basic moves = directions where to move in
static private void moveDef()
{
 final Move move0 =  w.new Move(0,1);
   vectMoves[0]   = move0 ;
 final Move move1 =  w.new Move(1,1);
   vectMoves[1]   = move1 ;
 final Move move2 =  w.new Move(1,0);
   vectMoves[2]   = move2 ;
 final Move move3 =  w.new Move(1,-1);
   vectMoves[3]   = move3 ;
```

```
 final Move move4 =  w.new Move(0,-1);
   vectMoves[4]   = move4 ;
 final Move move5 =  w.new Move(-1,-1);
   vectMoves[5]   = move5 ;
 final Move move6 =  w.new Move(-1,0);
   vectMoves[6]   = move6 ;
 final Move move7 =  w.new Move(-1,1);
   vectMoves[7]   = move7 ;
 final Move move8 =  w.new Move(0,0);
   vectMoves[8]   = move8 ;
}


@Override public void start(Stage stage)
    {
        Scene scene = new Scene(p, 500,400);
        stage.setScene(scene);
        stage.setTitle("Links move at random");
        stage.show();
        //Definition of possible moves
        moveDef();
        //Run the overall process only once
        nGenerationsST.setCycleCount(1);
        //This functions well
        nLinks--;
        for(int i = 0; i < nGenerations; i++)
        {
            gen = i;
            //Worms are posited at the start line
            startLine();
            //One link is added to all worms
            addLink();
            //Worms climb the screen
            SequentialTransition oneGenST  =  express();
            nGenerationsST.getChildren().add(oneGenST);
        }
    //Film is played
    nGenerationsST.play();
```

```
    }


//The head of all worms are posited at the same initial row
//in vertical position.
//There is one procedure for the first time
//(all links must be booked),
//and another procedure for later times:
//no booking but worms must be repositioned
//at the start line.
static public void startLine()
{
    for (int W = 0; W < nWorms; W++)
    {
        int  x,y;
        //Auxiliary worm
        Worm   worm;
        if (gen == 0) //initial generation
            worm  = w.new Worm();
        else //Subsequent generations
            worm = Worms.get(W);
        //Each link is positioned
        for(int  link = 0; link < nLinks; link++)
        {
            //Horizontal position
            x = 30 + 100 * W;
            //Vertical position
            y = initialRow + height*link;

            //Position is updated
            worm.Rect.get(link).setX(x);
            worm.Rect.get(link).setY(y);
            //Link is booked for the   0-th generation
            if (gen == 0)
            {
                p.getChildren().add(worm.Rect.get(link));
```

```
            }
            else //link is repositioned at startLine
            {
                //Duration
                Duration d = Duration.millis(1);
                //A soft jump is programmed
                TranslateTransition translateTransition =
                    new TranslateTransition(d,
                                            worm.Rect.get(link));
                //Repositioning at initial position.
                //Instruction found by serendipity.
                translateTransition.setToX(0);
                translateTransition.setToY(0);
                //Number of repetitions
                translateTransition.setCycleCount(1);
                //Repositioning is booked
                nGenerationsST.getChildren()
                        .add(translateTransition);
                //Invent new moves
                worm = resetMoves(worm);
            }
        }
        //worm is added at position W in list Worms
        Worms.add(worm);
    }
}


//Moves at random are created
private static Worm resetMoves(Worm worm)
{
    //Instructions to move at random
    for(int m = 0; m < nMovers; m++)
    {
        //Select at random  which link to move:
        int l = r.nextInt(nLinks) ;
        //Select at random  which direction to move on
        Move  move = w.new Move (vectMoves[r.nextInt(9)]);
```

```
        //Memorization
        worm.Mover.set(m, w.new Mover(l,move));
    }
    return worm;
}



//Move m of worm W is executed.
private static TranslateTransition
                    translate( int W, int m)
{
    if (print1)
        System.out.println();
    //Read the link to be moved
    int link = Worms.get(W).Mover.get(m).link;
    //Read position
    int x = (int) Worms.get(W).Rect.get(link).getX();
    int y = (int) Worms.get(W).Rect.get(link).getY();

    //The move to be done is read
    Move  move1;
    move1 = Worms.get(W).Mover.get(m).move;
    //Move is interpreted
    int byX = 4* move1.x;
    int byY = -6* move1.y;
    // A new place is defined:
    x = x + byX;
    y = y + byY;
    if (print1)
        System.out.println("\n Worm = " + W
                        + " move = " + m
                        + " link = " + link
                        + " \n x = " + x
                        + " y = " + y
                        + " \nbyX = " + byX
                        + " byY = " + byY);
```

```
    //Address is updated
    Worms.get(W).Rect.get(link).setX(x);
    Worms.get(W).Rect.get(link).setY(y);

    //Duration in within tics
    Duration d = Duration.millis(500);
    //A soft jump is programmed
    TranslateTransition translateTransition =
         new TranslateTransition(d, Worms.get(W).Rect.get(link));
    //Jump is executed
    translateTransition.setByX(byX);
    translateTransition.setByY(byY);
    //Number of repetitions
    translateTransition.setCycleCount(1);

    return translateTransition;
 }



//Worms express their information to move.
public    SequentialTransition  express()
{
    //Overall process
    SequentialTransition oneTrialST
                    = new SequentialTransition();
     oneTrialST.setCycleCount(1);
     //worms advance nSteps
    for (int i = 0; i < nSteps; i++)
    {
        //Worms advance one step:
        //To move all links, each worm must execute
        //instructions to move
        SequentialTransition oneStepST
                        = new SequentialTransition();
        oneStepST.setCycleCount(1);
        //Information is read and decoded
        for (int m = 0; m < nMovers; m++)
```

```
        {
            //All worms execute move m
            ParallelTransition moveMPT
                        = new ParallelTransition();
            moveMPT.setCycleCount(1);
            for (int W = 0; W < nWorms; W++)
            {
                //A translation of some link
                //is programmed for worm W
                //according to move m
                TranslateTransition tT = translate(W,m);
                tT.setCycleCount(1);
                //Translation is booked
                moveMPT.getChildren().add(tT);
            }
            //The ensamble of translations is booked
            oneStepST.getChildren().add(moveMPT);
        }
    //The movement of worms in one step is booked
    oneTrialST.getChildren().add(oneStepST);
    }
    return oneTrialST;
}


    //One link is added to all worms
    public static void addLink()
    {
        //All worms are visited
        for (int W = 0; W < nWorms; W++)
        {
            //New rect is created and added to the worm
            Worms.get(W).Rect.add(createRectangle(gen));
            //Rect is booked
            p.getChildren().add(Worms.get(W)
                                .Rect.get(nLinks));
            //Horizontal position
            int x = 30 + 150 * W;
```

```
        //Vertical position
        int y = initialRow + height*nLinks;

        //Position is updated
        Worms.get(W).Rect.get(nLinks).setX(x);
        Worms.get(W).Rect.get(nLinks).setY(y);

        //Link is shown
        SequentialTransition show
                    = showST(Worms.get(W)
                            .Rect.get(nLinks));
        //Show transition is booked
        nGenerationsST.getChildren().add(show);
    }
    //Bookkeeping
    nLinks++;
}


//A hidden rect gets shown
private static SequentialTransition showST(Rectangle rect)
{
    Duration d1 =   Duration.millis(1);

    //Rect must be shown at true scale
    KeyValue keyValue1 =
            new KeyValue(rect.scaleXProperty(),
                        1, Interpolator.LINEAR);
    KeyValue keyValue2 =
            new KeyValue(rect.scaleYProperty(),
                        1, Interpolator.LINEAR);
    Timeline b = new Timeline(
            new KeyFrame(d1,keyValue1,keyValue2 ));

    SequentialTransition s = new SequentialTransition();
    Duration d2 =   Duration.millis(1);
    //Width of  rect is enlarged  to get it visible
    //and after this
```

```
        //it is shown according to the
        //instructions given by timeline b.
        //To do this, a pause  is added.
        s.getChildren().add(
                new SequentialTransition(

                    PauseTransitionBuilder.create().
                        duration(d2)
                            .onFinished
                            ((ActionEvent t) ->
                            {
                                if (gen > 0)//Hidden -> show
                                {
                                    rect.setWidth(width);
                                    rect.setHeight(height);
                                }
                            })
                            .build(),b));
        return s;
    }


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


} //End of main class L56 WormsRandJFX6
```

69, page 186. The next code proposes a hint to help evolution. It is gene amplification. The reason is that the winning strategy is to move all links northwest (or northeast), the tail first and then the subsequent links. It is the very same instruction. So, if it found by mere randomness, to copy it to other movers might help evolution. The relevant piece of code follows but no change in performance was noticed by the Author. Nevertheless, if

move (1,1) is given as a hint, an acceleration of learning rate is readily seen.

```
//Program L69 WormsEvoJFX2
//...



public class WormsEvoJFX2 extends Application
...


//Chosen moves replace others
// in within the same worm
private static void geneAmplification()
{
    //  System.out.println("geneAmplification");
    int    m,    mm;
    int    link;
    //All worms are scanned
    for(int W = 0; W < nWorms; W++)
    {
        //nSourceMovers are amplified
        for (int i = 0; i < nSourceMovers; i++)
        {
            //Select at random which Mover to be read
            m = rd.nextInt(nMovers);
            //Move to be amplified is read
            Move move =  Worms.get(W).Mover.get(m).move;
            //Try also this
            //Move move = w.new Move(1,1);
            //The Move is amplified
            for(int k = 0; k< nCopies; k++)
            {
                //Select at random the place for replacement:
                //throw a number from 1 to nMovers.
                mm = rd.nextInt(nMovers) ;
                //Link is read
                link = Worms.get(W).Mover.get(mm).link;
```

```
                    //Mover mm is assembled from link + move
                    Worms.get(W).Mover
                        .set(mm, w.new Mover(link,move));
                }
            }
        }
}


//Information to move is changed
private static void mutation(   )
{
    geneAmplification( );
    moverMutation();
}
```

71, page 186. The next code includes a fossil study to support the idea that there is no evolution without a terrible, battle against imperfection of both function and structure.

```
//Program L71 WormsEvoJFX3
//A population of worms is programmed
//by evolution to
//climb the screen.
//All moves are executed
//but  nondisruptive moves are
//not undone.
//Each generations has its own
//population of worms.
//All worms have two links at the start.
//One link is added when some worm is shown
//to climb the screen.
//A fossil study is included.
//Program is written in JavaFX.
```

```
package ejvolu12p;

import static java.lang.Math.pow;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javafx.animation.Interpolator;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.ParallelTransition;
import javafx.animation.SequentialTransition;
import javafx.animation.Timeline;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Text;
import javafx.stage.Stage;
import javafx.util.Duration;



public class WormsEvoJFX3 extends Application
{
    private static WormsEvoJFX3 w = new WormsEvoJFX3();
    //Overall drawing process
    private static  SequentialTransition allST
                    = new SequentialTransition();
    //Drawing process for evolution
    private static  SequentialTransition evolutionST
                    = new SequentialTransition();
    //Number of repetitions or generations
```

```
private static int nGenerations = 5;
//Actual generation
private static int gen = 0;
//Duration of graphical events
private static final Duration d = Duration.millis(100);
//The scene
private static Group p = new Group();
//Number of steps given for each try
private static int nSteps = 8;
//Number of worms
private static  int nWorms = 8;
//Worms are kept in a list
private static List<Worm> Worms = new ArrayList<>();
//Helping files
private static List<Worm> WormsTwo = new ArrayList<>();
private static Worm[] OldWorms ;
//Every worm leaves a fossil
private  static List<Fossil> Fossils = new ArrayList<>();
//Number of fossils
private static int nFossils = 0;
//Number of sampled fossils
private static int nSampledFossils = 4;
//Number of links of each worm
private static  int nLinks = 2;
//Initial position over the screen of the head
private static int initialRow = 280;
//Generator of pseudo random numbers
private static final Random rd = new Random();
//Each worm has nMovers movers,
//instructions to move.
private static int  nMovers = 5;
//Width of the worm = width of each link
private static int width = 6;
//Height of each link
private static int height = 6;
//Array with 9 possible moves,
//9 possible directions of movement
private static Move  vectMoves[ ] = new Move[9];
```

```
private static final boolean print1 = false;
private static final boolean print2 = false;
private static final boolean printReproduction = true;
//Text to be drawn
private static  Text genCount = new Text ("");
//Two links cannot occupy the same place
static boolean isOccupied[][]  = new boolean[600][300];
//Is a worm capable of climbing the screen?
private static  int Fitness[] =  new int[21];
//Was climbing good enough?
private static boolean bullseye  = false;
//List of generations at which success was made
private static String record = "";
//Measures of the screen
private static int xWin = 1260;
private static int yWin = 850;
//Grouping of links in order to ease erasing
private static List<Rectangle> allRects
                        = new ArrayList<>();
private static List<Rectangle> lastRects
                        = new ArrayList<>();
//The number of point mutations of the offspring
private static int numbPointMutations = 5;


//Gene amplification follows:
//nSourceMovers is the number of different Movers
//to be copied.
private static int nSourceMovers = 2;
//ncopies is the number of new copies
//of each copied Mover in
 // the gene amplification method.
private static int   nCopies = 2;
```

```
    /* Definition of inner classes.
    *
    The Move class:
    A Move says in which direction
    a link of the worm must be moved.
    */

 private  class Move
 {
    int x;
    int y;


    //Default initialization
    Move()
    {
        x= 0;
        y= 0;
    }

    // Initialize with a list of two coordinates
    Move(int a, int b)
    {
        x= a;
        y= b;
    }

    // Copy a move
    Move(Move move)
    {
        x= move.x;
        y= move.y;
    }
 }//End of class Move


 // Definition of the class Mover:
 // A Mover keeps two types of information:
```

```
// a) the link of the worm to be moved and
// b) the direction or move to be chosen.
//
private   class Mover
{
   int link ;
   Move   move;

   //  Movers can acquired information
   // in three ways, they are  overloaded.
   // Form 1: by a list of numbers
   private Mover(int l, int mx, int my)
    {
       link= l;
       move.x = mx;
       move.y = my;
    }

   // Form 2, by structured data: link + move
   private Mover(int l, Move m)
   {
       link= l;
       move = new Move(m);
   }

   // Form 3, by structured data: mover
   private Mover(Mover mover)
   {
       link= mover.link;
       move =  new Move(mover.move);
   }
 }//End of class Mover


//A random color is synthesized
private static Color randomColor()
{
   //Color is defined as a random
```

```
    //mix of red, green and blue
    int r,g,b;
    r = rd.nextInt(226);
    g = rd.nextInt(226);
    b = rd.nextInt(226);
    //The color is prepared
    Color c = Color.rgb(r,g,b);
    return c;
 }


//Rects that belong in the worms of the first generation
 //are visible. Others are hidden.
private static  Rectangle
           createRectangle(int gen)
{
    //Upper left corner at (0,0)
    Rectangle rect = new Rectangle(0,0,width,height);

    //The color is prepared
    Color c = randomColor();
    //The rect is painted
    rect.setFill(c);
    //Rects of the first generation (gen = 0)
    //are visible, otherwise they are not
    if (gen > 0)
    {
        rect.setHeight(0);
        rect.setWidth(0);
    }
    return rect;
}


/*  Here we define the class worm.
    A worm is composite: it has links.
    Each link has a rectangular shape
    and  its own color.
```

```
    The vector Mover[ ]
     keeps the information
    on how to move its links:
    which  link and in which order
    (in the order that they appear in
    the vector Mover[]), and
    in which direction.
    Each worm keeps record of
    the actual position of each
    one of its links,
    in the associated rectangle. */



private class Worm
{
    //Number of links
    int numberOfLinks = nLinks;
    //Number of movers = instructions to move
    int numberOfMovers = nLinks;
    //List mover  keeps movers
    List<Mover> Mover = new ArrayList<>();
    //The shape of each link
    List<Rectangle> Rect = new ArrayList<>();

    //Default initialization:
    //worms of the first generation are
    //visible. Others not.
    private Worm()
    {
//Visible rects are created.
        // A color is assigned to each link
        // but different than white
        for(int i = 0; i < numberOfLinks; i++)
            Rect.add(createRectangle(0));
        //Instructions to move at random
for(int m = 0; m < nMovers; m++)
        {
```

```
            //Select at random  which link to move:
            int l = rd.nextInt(nLinks) ;
            //Select at random  which direction to move on
            Move  move
                    = w.new Move (vectMoves[rd.nextInt(9)]);
            //Memorization
            Mover.add(w.new Mover(l,move));

        }
    }//End of constructor Worm()


    //Constructor from worm W of the old generation.
    private Worm(int W)
    {
        numberOfLinks = nLinks;
        numberOfMovers = nMovers;
        //Fake initialization
        for(int i = 0; i < numberOfLinks; i++)
               Rect.add(createRectangle(1));
        for(int m = 0; m < nMovers; m++)
        {
            //Select at random  which link to move:
            int l = rd.nextInt(nLinks) ;
            //Select at random  which direction to move on
            Move  move
                    = w.new Move (vectMoves[rd.nextInt(9)]);
            //Memorization
            Mover.add(w.new Mover(l,move));
        }
        //Read true color
        for(int i = 0; i <  OldWorms[W].numberOfLinks; i++)
        {
           Rect.get(i).setFill( OldWorms[W]
                                .Rect.get(i).getFill());
        }
        //Read true movers
        for(int m = 0; m < OldWorms[W].numberOfMovers; m++)
```

```
            Mover.set(m, OldWorms[W].Mover.get(m));
    }

    //Constructor from a fossil.
    private Worm(Fossil fossil)
    {
        numberOfLinks = fossil.numberOfLinks;
        numberOfMovers = fossil.numberOfMovers;
        //Rects are initialized, 1 = unvisible
        for(int i = 0; i < numberOfLinks; i++)
               Rect.add(createRectangle(1));
        //Symbolic initalization of movers
        for(int m = 0; m < numberOfMovers; m++)
        {
            //Select at random  which link to move:
            int l = rd.nextInt(nLinks) ;
            //Select at random  which direction to move on
            Move  move
                    = w.new Move (vectMoves[rd.nextInt(9)]);
            //Memorization
            Mover.add(w.new Mover(l,move));
        }
        //Get true movers
        for(int m = 0; m < numberOfMovers; m++)
           Mover.set(m, fossil.Mover.get(m));
    }

}//End of class Worm



//Inner class Fossil.
//Fossils keep relevant information about worms
private class Fossil
{
    int numberOfLinks;
    int numberOfMovers;
    //The shape of each link
```

```
    List<Rectangle> Rect = new ArrayList<>();
    //List mover  keeps movers
    List<Mover> Mover = new ArrayList<>();



    public Fossil(Worm worm)
    {
        numberOfLinks = worm.numberOfLinks;
        numberOfMovers = worm.numberOfMovers;
        //Rects are created
         for(int i = 0; i < numberOfLinks; i++)
               Rect.add(createRectangle(1));
         //Symbolic initialziation of movers
         for(int m = 0; m < numberOfMovers; m++)
        {
            //Select at random  which link to move:
            int l = rd.nextInt(numberOfLinks) ;
            //Select at random  which direction to move on
            Move  move
                    = w.new Move (vectMoves[rd.nextInt(9)]);
            //Memorization
            Mover.add(w.new Mover(l,move));
        }
        //True initialization of movers
        for(int m = 0; m < numberOfMovers; m++)
            Mover.set(m, worm.Mover.get(m));
 }
}//End of Fossil



//General setting
@Override public void start(Stage stage)
    {
        //The scene
        Scene scene = new Scene(p, xWin,yWin);
        stage.setScene(scene);
```

```
stage.setTitle("Evolution teaches worms "
                + "to climb the screen");
stage.show();
//Definition of possible moves
moveDef();
//Text to draw
genCount = new Text ("");
genCount.setLayoutX(40);genCount.setLayoutY(40);
p.getChildren().addAll(genCount);
evolutionST.setCycleCount(1);

//Evolution for nGenerations
for(int i = 0; i < nGenerations; i++)
{
    //The scene is void
    voidOccupationMatrix();
    //Generation
    gen = i;
    if (print2)
        System.out.println("\nGen = " + gen);
    //Generation number is drawn
    Timeline timeline = drawGen(gen);
    evolutionST.getChildren().addAll( timeline );
    //Worms are created and posited at the start line
    startLine();
    //Worms express their information to move
    SequentialTransition oneGenST = oneGeneration();
    evolutionST.getChildren().add(oneGenST);
    //Drawn worms get hidden
    ParallelTransition hideST  =  hideST();
    evolutionST.getChildren().add(hideST);
    //Helping array
    Worm[] file1 = new Worm[50];
    //Information is recorded in an array
    OldWorms = Worms.toArray(file1);
    //Every worm leaves a fossil
    fossilization();
    //Destroy everything
```

```
            //for the sake of a new beginning
            Worms.clear();
            WormsTwo.clear();

            //Success?
            Integer genI = gen;
            if (bullseye)
            {
                //If climbing was successful, add a new link
                 nLinks = nLinks+1;
                 //List of generations with sucess
                record = record + "  " + genI.toString();
            }
        }
        //EvolutionST is booked
        allST.getChildren().add(evolutionST);
        System.out.println("Successful gens = "
            + record);
        //Fossil study
        SequentialTransition fsST = fossilSampling();
        allST.getChildren().add(fsST);
        //Film is played
        allST.play();


    }


// To avoid overwriting of worms,
// an occupation record is kept:
// The screen is divided in cells width x hight.
// At the beginning every cell is empty=false.
// Occupancy is recorded in isOccupied[][],
// which is a boolean matrix.
// A boolean  variable can be
// true= occupied,  else false= not occupied.
private static void  voidOccupationMatrix()
{
```

```
   //System.out.println("occupationMatrix");
    for(int i = 0; i < 600; i++)
    for(int j = 0; j < 300; j++)
        isOccupied[i][j]= false;
}

//Nine basic moves = irections where to move in
static private void moveDef()
{
    final Move move0 =  w.new Move(0,1);
       vectMoves[0]   = move0 ;
    final Move move1 =  w.new Move(1,1);
       vectMoves[1]   = move1 ;
    final Move move2 =  w.new Move(1,0);
       vectMoves[2]   = move2 ;
    final Move move3 =  w.new Move(1,-1);
       vectMoves[3]   = move3 ;
    final Move move4 =  w.new Move(0,-1);
       vectMoves[4]   = move4 ;
    final Move move5 =  w.new Move(-1,-1);
       vectMoves[5]   = move5 ;
    final Move move6 =  w.new Move(-1,0);
       vectMoves[6]   = move6 ;
    final Move move7 =  w.new Move(-1,1);
       vectMoves[7]   = move7 ;
    final Move move8 =  w.new Move(0,0);
       vectMoves[8]   = move8 ;
}


//All worms are posited at the same initial row
//in vertical position.
static public void startLine()
{
    for (int W = 0; W < nWorms; W++)
    {
        int  x,y;
        //Auxiliary worm
```

```
        Worm   worm;

        if (gen == 0)//Create worm
            worm = w.new Worm();
        else //Copy worm W from old generation
            worm = w.new Worm(W);

       //Each link is positioned
       for(int  link = 0; link < nLinks; link++)
       {
           //Horizontal position
           x = 140 + 140 * W;

           //Vertical position
           y = initialRow + height*link;

           //Position is updated
           worm.Rect.get(link).setX(x);
           worm.Rect.get(link).setY(y);

           //Link is shown
           ParallelTransition show
                       = showPT(worm.Rect.get(link));
           //Link is booked
           p.getChildren().add(worm.Rect.get(link));
           //Link is grouped
           allRects.add(worm.Rect.get(link));
           //Show-transition is booked
           evolutionST.getChildren().add(show);
       }
       //worm is located at position W in lists Worms
       Worms.add(worm);

       if (print1)
           System.out.println("W = " + W + " nLinks = "
               + WormsTwo.get(W).numberOfLinks  );
   }
```

```
}



//Draw the generation number
private static Timeline drawGen(int gen)
{
    Integer genI = gen;
    String message =  "Generation  = " +
                   genI.toString();
    //A JavaFX event committed to print a text is generated
    EventHandler printText =
        (EventHandler<ActionEvent>)
            (ActionEvent t ) ->
                {
                    genCount.setText(message);
                };
    //Presentation of printText must be
    //synchronized with the overall film so,
    //duration d is specified
    KeyFrame keyFrame = new KeyFrame(d, printText);

    Timeline timeline = new Timeline();
    timeline.setCycleCount(1);
    //add the keyframe to the timeline
    timeline.getKeyFrames().add(keyFrame);

    return timeline;
}



//Is move m disruptive for the worm?
//(A worm is and must remain as a whole,
//nevertheless, some elasticity is allowed.)
private static boolean isDisruptive(int m, Worm worm )
```

```
{
    //A move is disruptive if the target address
    //is farther than certain distance
    //from the neighboring links.
    boolean isDisruptive = false;
    int  link;
    int  x,y,  x1,y1, x2,y2, x3,y3;
    double d1;

    double  d2;

    //Read the link to be moved
    link = worm.Mover.get(m).link;
    //Read the   address of the link;
    x  = (int) worm.Rect.get(link).getX();
    y  = (int) worm.Rect.get(link).getY();
    //The   move to be done is read
    Move move = w.new Move(worm.Mover.get(m).move);

    //The target address is calculated
    x1 = x + width*move.x;
    y1 = y -  width*move.y;

    //The addresses of the two neighboring links are read
    //The disruption is calculated
    //Case: intermediate link
    if ((0 < link) & (link < nLinks-1))
    {
        x2 = (int) worm.Rect.get(link-1).getX();

        y2 = (int) worm.Rect.get(link-1).getY();

         d1 =    pow(  ( pow( (x2 - x1),2)
                              +  pow((y2 - y1),2)),0.5);

        x3 = (int) worm.Rect.get(link+1).getX();
        y3 = (int) worm.Rect.get(link+1).getY();
        d2 =  pow(  (pow( (x3 - x1),2)
```

```
                                + pow((y3 - y1),2)),0.5);


        // Here we decide whether or not
        // a move is disruptive.
        //To augment elasticity, augment the threshold

        if ((d1 > 1.415*width) || (d2 > 1.415*width))
                    isDisruptive = true;

     }

    //First link
    if (link == 0)
    {
        x2 = (int) worm.Rect.get(1).getX();
        y2 = (int) worm.Rect.get(1).getY();
        d1 = pow( (pow( (x2 - x1),2) + pow((y2 - y1),2)),0.5);
        if( d1 > 1.1*width) isDisruptive = true;
     }
     //Last link
    if( link == nLinks-1 )
    {
        x2 = (int) worm.Rect.get(nLinks - 2).getX();
        y2 = (int) worm.Rect.get(nLinks - 2).getY();
        d2 = pow( (pow( (x2 - x1),2) + pow((y2 - y1),2)),0.5);
        if ( d2 > 1.1*width ) isDisruptive =   true;
    }
    return isDisruptive;
 }



//Execute move m for worm W.
//If it is  disruptive, undo it.
private static SequentialTransition
                    exeIf( Worm worm, int m)
```

```
{
    if (print1)
        System.out.println(" exeIf");
    //Reads the link to be moved
    int link = worm.Mover.get(m).link;
    //Reads position
    int x = (int) worm.Rect.get(link).getX();
    int y = (int) worm.Rect.get(link).getY();

    //The move to be done is read
    Move  move;
    move = worm.Mover.get(m).move;
    //Move is interpreted
    int byX = width* move.x;
    int byY = -height* move.y;
    //Target address is calculated:
    int x1 = x + byX;
    int y1 = y + byY;
    if (x1 < 0 ) x1 = 0;
    if (y1 < 0 ) y1 = 0;
    if (print1)
        System.out.println(
            " move = " + m   + " link = " + link
            + " \n x = " + x + " y = " + y
            + " \nbyX = " + byX  + " byY = " + byY);


    //Occupied cell
    int i = (int)  x / width;
    int j = (int) y / height;


     //Target cell
    int k =(int) x1/width;
    int l = (int) y1/height;


    //Move is always executed
```

```
SequentialTransition moveRect
        = new SequentialTransition();

TranslateTransition translateTransition =
     new TranslateTransition(d, worm.Rect.get(link));
//Jump is executed
translateTransition.setByX(byX);
translateTransition.setByY(byY);
//Number of repetitions
translateTransition.setCycleCount(1);
moveRect.getChildren().add(translateTransition);



//Is the move disruptive?
boolean   isMoveDisruptive =  isDisruptive(m,worm);
if (print1)
{
    if (isMoveDisruptive)
        System.out.println("move is Disruptive");
    else
        System.out.println("move is not  Disruptive");
}
//Disruptive moves are undone
if  (((isOccupied[k][l])) || ((isMoveDisruptive)))
{
    byX = -byX;
    byY = -byY;
}
else //move is not disruptive -> it is not undone
{
    byX = 0;
    byY = 0;
    worm.Rect.get(link).setX(x1);
    worm.Rect.get(link).setY(y1);
    isOccupied[i][j]= false;
    isOccupied[k][l]= true;
}
```

```
    //Move is possibly undone
    TranslateTransition translateTransition2 =
         new TranslateTransition(d, worm.Rect.get(link));
    //Jump is programmed
    translateTransition2.setByX(byX);
    translateTransition2.setByY(byY);
    //Number of repetitions
    translateTransition2.setCycleCount(1);
    moveRect.getChildren().add(translateTransition2);

    return moveRect;
 }


// Every worm is assigned a fitness
//The nearer to the top, the fitter.
//If climbing has been significative,
//the complexity of the game rises up one level:
//a true bullseye is reported (to add
//next a new link to all worms).
private static void assessFitness()
{
    int yOfTail;
    double yOfHead;

    bullseye  = false;
    for(int W = 0; W <  nWorms; W++)
    {
        // We get the y-coordinate of the tail
        yOfTail = (int) Worms.get(W)
                              .Rect.get(nLinks-1).getY();
        yOfHead =  Worms.get(W).Rect.get(0).getY();
        //Fitness measures how much the head has  been lifted
        Fitness[W] = (yWin - yOfTail);
```

```
        if(print1)
            System.out.println("Fitness of worm "
                    + W + " = " + Fitness[W]
                    + " head = " + yOfHead);
        //If worm W is good for climbing
        //its head must be high enough
        if (yOfHead < initialRow - 2 * height)
        {
            //Worms that climb are rewarded
            Fitness[W] = 100 * Fitness[W];
            bullseye = true;
            if (printReproduction)
            System.out.println(
                    "There was advance W = " + W );
        }
    }
}


//Worms express their information to move.
public    SequentialTransition  express()
{
    //Overall process
    SequentialTransition expressST
                    = new SequentialTransition();
     expressST.setCycleCount(1);
     //Worms advance nSteps
    for (int i = 0; i < nSteps; i++)
    {
        //Worms advance one step:
        //To move all links, each worm must execute
        //instructions to move
        SequentialTransition oneStepST
                        = new SequentialTransition();
        oneStepST.setCycleCount(1);
        //Information to move is read and decoded
        for (int m = 0; m < nMovers; m++)
        {
```

```
            if (print1)
                System.out.println("nMovers = " + nMovers);
            //All worms execute move m
            ParallelTransition moveMPT
                        = new ParallelTransition();
            moveMPT.setCycleCount(1);
            //This is a parallel loop for a list
            for (Worm worm: Worms)
            {
                //A translation of some link
                //is programmed for worm W
                //according to move m.
                //If it is not disruptive,
                //it is executed
                SequentialTransition tT = exeIf(worm,m);
                tT.setCycleCount(1);
                //Translation is booked
                moveMPT.getChildren().add(tT);
            }
            //The ensamble of translations is booked
            oneStepST.getChildren().add(moveMPT);
        }
    //The movement of worms in one step is booked
    expressST.getChildren().add(oneStepST);
    }
    return expressST;
}



    //A hidden rect gets shown
    private static
        ParallelTransition showPT(Rectangle rect)
    {
        ParallelTransition s = new ParallelTransition();
        Duration d2 =   Duration.millis(1);
        //Rect is enlarged
```

```
    KeyValue keyValue3 =
        new KeyValue(rect.widthProperty(),width,
            Interpolator.LINEAR);
    KeyValue keyValue4 =
        new KeyValue(rect.heightProperty(),height,
            Interpolator.LINEAR);
    //KeyValues are  inserted into a timeline
    Timeline b = new Timeline(
        new KeyFrame(d2,keyValue3,keyValue4 ));
    //Timeline is inserted into a transition
    s.getChildren().addAll(b);

    return s;
}




//Worms are hidden after their performance.
private static ParallelTransition  hideST()
{

    ParallelTransition hidePT
                = new ParallelTransition();
    hidePT.setCycleCount(1);
    Duration d2 =   Duration.millis(1);
    //Hide all rects
    for(Rectangle rect: allRects)
    {
       //Rects are made unvisible
        KeyValue keyValue =
                new KeyValue(rect
                        .visibleProperty(), false
                        ,Interpolator.LINEAR);
        Timeline b = new Timeline(
                new KeyFrame(d2,keyValue));
        hidePT.getChildren().add(b);
```

```
        }

        return hidePT;
    }




    //====================================
    //=========Evolutionary part==========
    //====================================


    //One generation = expression + assessement
    //+ reproduction, recombination and mutation.
    public    SequentialTransition  oneGeneration()
    {
        SequentialTransition oneGenST
                = new SequentialTransition();
        //Worms express their information to move
        SequentialTransition expressST = express();
        oneGenST.getChildren().addAll(expressST);
        //Fitness is measured
        assessFitness();
        //Reproduction is proportional to fitness
        reproduction();
        //Population is replaced by a recombinat one
        recombination();
        //Information to move suffers mutation
        mutation();

        return oneGenST;
    }
```

```
/*  To generate a worm of the next generation,
    an individual is sampled at random
    from the old population
    and cloned
    with a probability that is
    proportional to its fitness.
    */
private static void reproduction()
{
    if (printReproduction)
        System.out.println( "\nGeneration = " + gen);
    //Cumulative distribution
    Double cumulate[] = new Double[nWorms];
    // Counters
    int count[] = new int[nWorms];
    //Number of trials
    int nTrials = nWorms;

    //Initialization
    WormsTwo.clear();
    WormsTwo.addAll(Worms);
    for (int i=0; i< nWorms; i++)
        count[i]=0;

    //Cumulative distribution
    cumulate[0] = (double) Fitness[0];
    for (int i=1; i< nWorms; i++)
        cumulate[i]= cumulate[i-1] + Fitness[i];

    //Report
    if (printReproduction)
    {
        System.out.println( "\ni \tfitness cumulate ");

    for (int i = 0; i < nWorms; i++)
        System.out.println( i + "\t" + Fitness[i] + " \t "
                + cumulate[i]);
    }
```

```
//Maximal value
double max = cumulate[nWorms-1];
//Counter of chosen worms
int W = 0;
//System.out.println("Generated numbers: ");
for (int i = 0; i < nTrials; i++)
{
    //A random integer less than max is generated
    //with a uniform distribution
    int n = rd.nextInt((int) max);

    //System.out.println("n =  " + n);
    //Probability of cloning is made
    //in proportion to fitness.
    //Is worm 0 to be chosen?
    if (n < cumulate[0])
    {
        /*
        if (printReproduction)
            System.out.println( "chosen individual  = 0 ");
        */
        WormsTwo.set(W, Worms.get(0));
        count[0]++;
        W++;
    }
    //Is worm l to be chosen?
    for (int l = 1; l < nWorms; l++)
    {

        if ( ( cumulate[l-1] <= n) & (n < cumulate[l]) )
        {
            /*
            if (printReproduction)
                System.out.println(
                    "Chosen individual = " + l);
            */
            count[l]++;
```

```
                WormsTwo.set(W, Worms.get(l));
                W++;
            }
        }

    }
    if (printReproduction)
    {
        System.out.println("\nEvents and their observed "
            + "and expected frequencies: ");
        for (int i=0; i<nWorms;i++)
            System.out.println(i + "\t" + count[i] + "\t"
                +  Fitness[i]*nTrials/cumulate[nWorms-1]);
    }

    //WormsTwo is copied into Worms
    Worms.clear();
    Worms.addAll(WormsTwo);

}



//Recombination is implemented
private static void recombination()
{

    //Every worm is replaced by a recombinant
    for(int W = 0; W < nWorms; W++ )
    {
        //A  binary recombinant operator
        //is assembled at random.

        int rec[] = new int[210];
        for (int j = 0; j< nMovers; j++)
  rec[j]= rd.nextInt(2);
//Another worm is chosen at random for recombination
        int m = rd.nextInt(nWorms);
```

```
   //Basal initialization of the recombinant
        Worm  worm = Worms.get(W);
for(int j = 0; j< nMovers; j++)
        {
            // Recombination is executed.
            //For 0, take from the partner worm m
            //For 1, take from worm W.
            if (rec[j] == 0)
            worm.Mover.set(j, Worms.get(m).Mover.get(j));
        }

        //Recombinant is memorized
        WormsTwo.set(W, worm);
        if(print2)
        {
            System.out.println("Old move.x = " + Worms.get(W).Mo

            System.out.println("New Rec move.x = " + worm.Mover.
        }

    }
    //Old population is replaced by the recombinated one
    Worms.clear();
    Worms.addAll(WormsTwo);
}




//Blind mutations of infomation to move
private static void moverMutation()
{
    int   m, n;
    //Let us try out blind  mutations
    //in specific Movers,
    //in the  link  to be moved
    //and in the direction of movement.
```

```
    for(int W = 0; W < nWorms; W++)
    {
        //Each offspring receives
        //numbPointMutation mutations
        for(int k = 0; k< numbPointMutations; k++)
        {
            //Select at random  which mover to mutate:
            m =  rd.nextInt(nMovers );
            //Select at random which link  to move:
            n = rd.nextInt( nLinks) ;
            if(print2)
                System.out.println("Old move.x = "
                     + Worms.get(W).Mover.get(m).move.x);
            //Select at random which direction to move on.
            Move move = vectMoves[rd.nextInt(9) ];
            //Mover m is assembled from link + move
            Worms.get(W).Mover.set(m, w.new Mover(n,move));
            if(print2)
                System.out.println("New Mutation move.x = "
                    + Worms.get(W).Mover.get(m).move.x);
            //Color of link n is changed
            //(possibly various times)
            Worms.get(W).Rect.get(n).setFill(randomColor());
        }
    }
}


//Chosen moves replace others
// in within the same worm
private static void geneAmplification()
{
    //  System.out.println("geneAmplification");
    int    m,    mm;
    int    link;
    //All worms are scanned
    for(int W = 0; W < nWorms; W++)
    {
```

```
        //nSourceMovers are amplified
        for (int i = 0; i < nSourceMovers; i++)
        {
            //Select at random which Mover to be read
            m = rd.nextInt(nMovers);
            //Move to be amplified is read
            Move move =  Worms.get(W).Mover.get(m).move;
            //Try also this
            //Move move = w.new Move(1,1);
            //The Move is amplified
            for(int k = 0; k< nCopies; k++)
            {
                //Select at random the place for replacement:
                //throw a number from 1 to nMovers.
                mm = rd.nextInt(nMovers) ;
                //Link is read
                link = Worms.get(W).Mover.get(mm).link;
                //Mover mm is assembled from link + move
                Worms.get(W).Mover
                    .set(mm, w.new Mover(link,move));
            }
        }
    }
}



//Information to move is changed
private static void mutation(   )
{
    geneAmplification( );
    moverMutation();
}



//All worms leave their fossils
private static void fossilization()
```

```
{
    for(int W = 0; W < nWorms; W++)
    {
//Worm W leaves its fossil
        Fossil fossil = w.new Fossil(Worms.get(W));
        Fossils.add(fossil);
if (print1)
            System.out.println("Fossil number = "
    + nFossils
    + " nLinks = " + Fossils.get(nFossils).numberOfLinks);
        //Counter
nFossils++;
    }
}

//Fitness of worm is measured.
private static double fitness(Worm worm)
{
    double yOfTail;
    double fitness;
    // We get the y-coordinate of the tail
    yOfTail =  worm.Rect.get(nLinks-1).getY();
    //The nearer to the top of the screen (yOfTail = 0),
    //the fitter:
    fitness  = 100*(yWin - yOfTail);
    if(print1)
        System.out.println("Fitness of fossil = "
                           + fitness);
    return fitness;
}

//Draw the fossil number
private static Timeline drawFossilNumber(int f)
{
    Integer F = f;
    String message =  "Fossil  = " +
                    F.toString();
    //A JavaFX event committed to print a text is generated
```

```
    EventHandler printText =
        (EventHandler<ActionEvent>)
            (ActionEvent t ) ->
                {
                    genCount.setText(message);
                };
    //Presentation of printText must be
    //synchronized with the overall film so,
    //duration d is specified
    KeyFrame keyFrame = new KeyFrame(d, printText);

    Timeline timeline = new Timeline();
    timeline.setCycleCount(1);
    //add the keyframe to the timeline
    timeline.getKeyFrames().add(keyFrame);

    return timeline;
}

//Reconstructed worm is given an initial position
private static ParallelTransition startLine(Worm worm)
{
    voidOccupationMatrix();
    nLinks = worm.numberOfLinks ;
    int x,y;
    ParallelTransition showWorm = new ParallelTransition();
    showWorm.setCycleCount(1);
    for(int  link = 0; link < nLinks; link++)
    {
        //Horizontal position
        x = 140;
        //Vertical position
        y = initialRow + height*link;
        //Position is updated
        worm.Rect.get(link).setX(x);
        worm.Rect.get(link).setY(y);
        //Link is shown
        ParallelTransition showPT
```

```
                    = showPT(worm.Rect.get(link));
        //Show-transition is booked
        showWorm.getChildren().add(showPT);
        //Link is booked
        p.getChildren().add(worm.Rect.get(link));
        //Link is grouped to ease hidding
        lastRects.add(worm.Rect.get(link));
    }
    return showWorm;
}


//Worm expresses its movers
private static SequentialTransition expressMovers(Worm worm)
{
    SequentialTransition expressMoversST
                    = new SequentialTransition();
    expressMoversST.setCycleCount(1);
    // d is the number of steps.
    // m is the index of the mover;
    for(int d = 0; d < nSteps ; d++)
    {
        for(int m  = 0 ; m <  worm.numberOfMovers; m ++)
        {
            //Movers a re xpressed
            SequentialTransition exeMovers = exeIf(worm,m);
            exeMovers.setCycleCount(1);
            //Execution of movers is booked
            expressMoversST.getChildren().add(exeMovers);
        }
    }
    return expressMoversST;
}


//Reconstructed fossil  is  hidden after its performance.
    private static ParallelTransition  hideLastRectsST()
    {
```

```
        ParallelTransition hidePT
                    = new ParallelTransition();
        hidePT.setCycleCount(1);
        Duration d2 =   Duration.millis(1);
        //Hide all rects
        for(Rectangle rect: lastRects)
        {
           //Rects are made unvisible
            KeyValue keyValue =
                    new KeyValue(rect
                            .visibleProperty(), false
                            ,Interpolator.LINEAR);
            Timeline b = new Timeline(
                    new KeyFrame(d2,keyValue));
            hidePT.getChildren().add(b);
        }
        return hidePT;
    }


//Fossils are sampled and reonstructed
//to measure their functionality
private static  SequentialTransition fossilSampling()
{


    System.out.println("\nFOSSIL SAMPLING\n");
    System.out.println("Perfection "
            + "= number of links *fitness");
    System.out.println("\nFossil number, number of links,"
  + "fitness, perfection\n");
    double fitness;

    SequentialTransition fossilSamplingST
          = new SequentialTransition();
    for(int i = 0; i < nSampledFossils; i++)
    {
```

```
        //A fossil is chosen at random
        int f = rd.nextInt(nFossils);
        //Fossil is reconstructed into a worm
        Worm worm =   w.new  Worm(Fossils.get(f));
        //Worm is positioned at the startline
        ParallelTransition showWorm = startLine(worm);
        fossilSamplingST.getChildren().addAll( showWorm );
        //Generation number is drawn
        Timeline timeline = drawFossilNumber(f);
        fossilSamplingST.getChildren().addAll( timeline );
        //Movers of worm are unfolded
        SequentialTransition expressMoversST
                = expressMovers(worm);
        fossilSamplingST.getChildren().add(expressMoversST);
        fitness = fitness(worm);
        //Drawn rects of worm get hidden
        ParallelTransition hideLastRectsST
                =  hideLastRectsST();
        fossilSamplingST.getChildren().add(hideLastRectsST);

        System.out.println(f + "\t" + nLinks
            + "\t" + fitness + "\t\t" + fitness*nLinks);

    }
    return fossilSamplingST;
}


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


} //End of main class L71 WormsEvoJFX3
```

79, page 199. The next code includes division of numbers of type double.

```
//Program L79 RestrictedCapturing2
//A comboBox is used to display several options
//from which the User must pick up one.
//A console is also included to show
//a large amount of results.
//This program reads two integers,
//the type of arithmetic operation,
//executes it
//and prints a report to a console.


package ejvolu12p;

import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.event.ActionEvent;
import javafx.event.Event;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;


public class RestrictedCapturing2 extends Application
{

//Declarations
final Button OKBt = new Button ("OK");
//Console to write output
final TextArea console = new TextArea ("");
//Captured strings
//First number
private static String s1 = "";
//Second number
private static String s2 = "";
```

```
//
//private static String s3 = "";
private static String operation = "Sum";
//Associated numbers
private static Integer n1 = 0;
private static Integer n2 = 1;
private static double output = 0;



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



@Override public void start(Stage stage)
{
    stage.setTitle("RestrictedCapturing");
    Scene scene = new Scene(new Group(), 600, 270);


    //First number is captured
    final ComboBox firstNumber = new ComboBox();
    //Options
    firstNumber.getItems()
            .addAll("1", "2", "3", "4", "5");

    //Prompt
    firstNumber.setPromptText("Choose a number");
    firstNumber.setEditable(false);
    //Option is picked up
    firstNumber.setOnAction((Event ev) ->
    {
        s1 =
        firstNumber.getSelectionModel()
```

```
            .getSelectedItem().toString();
    n1 = Integer.parseInt(s1);
    System.out.println(n1);
});


//Second number is captured
final ComboBox secondNumber = new ComboBox();
secondNumber.getItems()
        .addAll("6", "7", "8", "9", "0");

secondNumber.setPromptText("Choose a number");
secondNumber.setEditable(false);
secondNumber.setOnAction((Event ev) ->
{
    s2 =
    secondNumber.getSelectionModel()
            .getSelectedItem().toString();
    n2 = Integer.parseInt(s2);
});


//A mathematical operation is chosen
final ComboBox operationComboBox = new ComboBox();
operationComboBox.getItems().addAll(
"Sum", "Subtraction",
        "Multiplication", "Division");
operationComboBox.setValue("Sum");
operationComboBox.setEditable(false);

operationComboBox.setOnAction((Event ev) ->
{
    operation =
    operationComboBox.getSelectionModel()
            .getSelectedItem().toString();


});
```

```
//Listener of OK button
OKBt.setOnAction((ActionEvent e) ->
{
    if (firstNumber.getValue() != null &&
        !firstNumber.getValue().toString().isEmpty())
    {
        if (operation != null)
            {
                //setText(item);
                if (operation.contains("Sum"))
                {
                    output = n1 + n2;
                    operation = "Sum";
                }
                else if
                    (operation.contains("Subtraction"))
                {
                    output = n1 - n2;
                    operation = "Subtraction";
                }
                else if
                    (operation.contains("Multiplication"))
                {
                    output = n1 * n2;
                    operation = "Multiplication";
                }
                else if
                        (operation.contains("Division"))
                {
                    double n11 = n1;
                    double n22 = n2;
                    output = n11 / n22;
                    operation = "Division";
                }
            }
```

```
            int intOutput = (int) output;
            double difference = output - intOutput;
            if ( difference == 0 )
                console.setText( "Inputs = " + s1 + " and " +  s
                    + "\nOperation  = " + operation
                    + "\nOuput = " + intOutput);
            else
                console.setText( "Inputs = " + s1 + " and " +  s
                    + "\nOperation  = " + operation
                    + "\nOuput = " + output);
        }

    });

    //Grid is initialized
    GridPane grid = new GridPane();
    grid.setVgap(4);
    grid.setHgap(10);
    grid.setPadding(new Insets(5, 5, 5, 5));
    //Row   0
    grid.add(new Label("First number: "), 0, 0);
    grid.add(firstNumber, 1, 0);
    grid.add(new Label("Operation: "), 2, 0);
    grid.add(operationComboBox, 3, 0);
    //Row 1
    grid.add(new Label("Second number: "), 0, 1);
    grid.add(secondNumber, 1, 1);
    //Row 2
    grid.add(console, 0, 2, 4, 1);
    //Row 3
    grid.add(OKBt, 0, 3);


    Group root = (Group)scene.getRoot();
    root.getChildren().add(grid);
    stage.setScene(scene);
    stage.show();
}
```

```
}//End of main class L79 RestrictedCapturing2
```

94, page 220. The next code presents a watch with two hands such as usual ones.

```
//Program L94 StopFX2
//A tutorial that shows how to
//control start/stop/pause/resume of an animation.
//Watch with two handles is simulated.
//JavaFX version.


package ejvolu12p;


import javafx.animation.Animation;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
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.*;
import javafx.scene.shape.Line;
import javafx.scene.transform.Rotate;
import javafx.stage.Stage;
import javafx.util.Duration;



public class StopFX2 extends Application
{

 //Timeline for watch
 private static final Timeline watch = new Timeline();
 //A rotation is initialized with angle 0 degrees
```

```
 //and position (200,200)
 final Rotate rotationTransform = new Rotate(0, 200,200);
 final Rotate rotationTransform2 = new Rotate(0, 200,200);




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



@Override
public void start(Stage stage)
{

    Group p = new Group();
    Scene scene = new Scene(p, 400, 350);
    stage.setTitle("Watch");
    //Hand of the watch
    Line hand = new Line(200, 200, 200, 100);
    Line hand2 = new Line(200, 200, 200, 70);
    p.getChildren().addAll(hand, hand2);

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

    // --- Menu General
    Menu menuGeneral = new Menu("Control");
    //Menu item declaration and initialization
    MenuItem start = new MenuItem("Start");

    //Listener
    start.setOnAction((ActionEvent t) ->
    {
```

```
    //Initial angle
    rotationTransform.setAngle(0);
    rotationTransform2.setAngle(0);
    watch.playFromStart();

});
//MenuItems added to menu
menuGeneral.getItems().addAll(start);


//Menu item declaration and initialization
MenuItem pause = new MenuItem("Pause");
//Listener
pause.setOnAction((ActionEvent t) ->
{
    watch.pause();
});
//MenuItems added to menu
menuGeneral.getItems().addAll(pause);


//Menu item declaration and initialization
MenuItem resume = new MenuItem("Resume");
//Listener
resume.setOnAction((ActionEvent t) ->
{
    watch.play();
});
//MenuItems added to menu
menuGeneral.getItems().addAll(resume);

//Menu item declaration and initialization
MenuItem stop = new MenuItem("Stop");
//Listener
stop.setOnAction((ActionEvent t) ->
{
    watch.stop();
});
```

```
//MenuItems added to menu
menuGeneral.getItems().addAll(stop);


//MenuItem exit
MenuItem exit = new MenuItem("Exit");
exit.setOnAction((ActionEvent t) ->
{
    //Finish everything and exit
    System.exit(0);
});

//Insertions
menuGeneral.getItems()
        .addAll( new SeparatorMenuItem(), exit);

//Booking
menuBar.getMenus()
        .addAll(menuGeneral);
p.getChildren().addAll(menuBar);
stage.setScene(scene);
stage.show();

//Rotations are assigned to hands
hand.getTransforms().add(rotationTransform);
hand2.getTransforms().add(rotationTransform2);

//Timeline is specified:
//hand gives  a round tour ( 360 degrees)
//for each  10  seconds
//while hand2 makes 5 round tours
//during the same time.
watch.getKeyFrames().addAll(
    new KeyFrame(Duration.seconds(10),
        new KeyValue(
            rotationTransform.angleProperty(),360) ),
    new KeyFrame(Duration.seconds(10),
        new KeyValue(
```

```
                rotationTransform2.angleProperty(),1800) )

            );


    //Repeat tours over and over
    watch.setCycleCount(Animation.INDEFINITE);

    //Enable start  only when animation is stopped
    start.disableProperty().bind(watch.statusProperty()
            .isNotEqualTo(Animation.Status.STOPPED));
    //Enable pause only when animation is running
    pause.disableProperty().bind(watch.statusProperty()
            .isNotEqualTo(Animation.Status.RUNNING));
    //Enable resume only when animation is paused
    resume.disableProperty().bind(watch.statusProperty()
            .isNotEqualTo(Animation.Status.PAUSED));
    //Enable stop only when animation is not stopped
    stop.disableProperty().bind(watch.statusProperty()
            .isEqualTo(Animation.Status.STOPPED));
}



}//End of main class L94 StopFX2
```

99, page 293. The Author claims that the price that evolution must pay for an increase in complexity is correctly programmed as well as the sampling of fossils and their classification as perfect else imperfect. The overall reason is this: the program has been developed after too much work to correct bugs that created more bugs. That was made thanks to normative use of instructions to print local results. At the end, the program produces what it is expected. In effect, the program correctly displays the expected movement of worms under intelligent design and on the other hand, the number of generations to add a new link sensibly decreases after inserting the hint that commands to move each link northeast (move (1,1)). Such as expected. So, the Author considers that the program is correct in its

overall design. Nevertheless, it is impossible to confidently test everything. Henceforth, the Author predicts that the code has many bugs, some of which might be mortal. This is the very reason because of which all software is always plagued with bugs. Darwinian evolution of software (the genome included) is not and cannot be an exception.

100, page 293. The next code represents an improvement of the encoding of our program of JavaFX worms.

```
//Program L100 WormsEvoJFX5
//A population of worms is programmed
//by evolution to
//climb the screen.
//All moves are executed
//but  nondisruptive moves are
//not undone.
//All worms have two links at the start.
//One link is added when some worm is shown
//to climb the screen.
//A fossil study is included.
//The  simulation includes the accepted by us fact that
//evolution can create complexity and is
//addressed to tackle two questions:
// 1. Which is the price that evolution
//      must pay for complexity?
// 2. Does imperfect fossils appear
//      in fossil sampling
//      and in which proportion?
//A suitable GUI is provided.
//Program is written in JavaFX.
//It makes some improvements over the previous version.
```

```
package ejvolu12p;
```

```
import java.awt.AWTException;
import java.io.IOException;
import static java.lang.Math.pow;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.animation.Interpolator;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.ParallelTransition;
import javafx.animation.SequentialTransition;
import javafx.animation.Timeline;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.event.Event;
import javafx.event.EventHandler;
import javafx.geometry.Bounds;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.chart.BarChart;
import javafx.scene.chart.CategoryAxis;
import javafx.scene.chart.LineChart;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.PieChart;
import javafx.scene.chart.XYChart;
import javafx.scene.control.Button;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;
```

```
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.control.Pagination;
import javafx.scene.control.RadioButton;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.ScrollPane.ScrollBarPolicy;
import javafx.scene.control.SeparatorMenuItem;
import javafx.scene.control.Slider;
import javafx.scene.control.Tab;
import javafx.scene.control.TabPane;
import javafx.scene.control.TextArea;
import javafx.scene.control.Toggle;
import javafx.scene.control.ToggleGroup;
import javafx.scene.image.Image;
import javafx.scene.input.Clipboard;
import javafx.scene.input.ClipboardContent;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.FlowPane;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Font;
import javafx.scene.text.Text;
import javafx.stage.Stage;
import javafx.util.Duration;
import javax.imageio.ImageIO;



public class WormsEvoJFX5 extends Application
{
    //Instantiation
    private static WormsEvoJFX5 w = new WormsEvoJFX5();
    //Overall drawing process
```

```
private static  SequentialTransition allST
                = new SequentialTransition();
//Drawing process for evolution
private static  SequentialTransition evolutionST
                = new SequentialTransition();
//Number of repetitions or generations
private static int nGenerations = 5;
//Actual generation
private static int gen = 0;

//The scene
private static Group p = new Group();
//Number of steps given for each try
private static int nSteps = 7;
//Number of worms
private static  int nWorms = 8;
//Worms are kept in a list
private static List<Worm> Worms = new ArrayList<>();
//Helping files
private static List<Worm> WormsTwo = new ArrayList<>();
private static Worm[] OldWorms ;
//Every worm leaves a fossil
private  static List<Fossil> Fossils = new ArrayList<>();
//Counter  of fossils
private static int nFossils = 0;
//Number of   fossils to be sampled
private static int nSampledFossils = 5;
//Number of links of each worm
private static  int nLinks = 2;
//Elasticicty of inner links:
//inner links can get separated without geting disrupted
private static double innerElasticity = 1.415;
//Elasticity of head and tail
private static double borderElasticity = 1.1;
//Width of the worm = width of each link
private static int widthOfLinks = 6;
//Height of each link
private static int heightOfLinks = 6;
```

```
//Each worm has nMovers movers,
//instructions to move.
private static int  nMovers = 5;
//Initial vertical position of the head over the screen
private static int initialRow = 280;

//Generator of pseudo random numbers
private static final Random rd = new Random();
//Array with 9 possible moves
//that are possible directions of movement
private static Move  vectMoves[ ] = new Move[9];
private static final boolean print1 = false;
private static final boolean print2 = false;
private static final boolean print3 = false;
private static final boolean printReproduction = true;
//Text to be drawn
private static  Text infoText = new Text ("");
//Two links cannot occupy the same place
static boolean isOccupied[][]  = new boolean[600][300];
//Is a worm capable of climbing the screen?
private static  int Fitness[] =  new int[21];
//Was climbing good enough?
private static boolean bullseye  = false;
//Measures of the screen
private static int xWin = 1260;
private static int yWin = 1000;
//Grouping of links in order to ease erasing
private static List<Rectangle> allRects
                        = new ArrayList<>();
private static List<Rectangle> lastRects
                        = new ArrayList<>();
//The number of point mutations of the offspring
private static int nPointMutations = 5;
//Gene amplification follows:
//nSourceMovers is the number of different Movers
//to be copied.
private static int nSourceMovers = 2;
//ncopies is the number of new copies
```

```
//of each copied Mover in
 //the gene amplification method.
private static int   nCopies = 2;

//List of generations at which a worm were found
//to climb the screen
private static List<Integer> bullseyes = new ArrayList<>();

//Slider for shifting the position of heads of worms:
//declaration and symbolic initialization
final Slider verticalShiftSlider = new Slider(-100, 100, 0);

//Slider that regulates the separation in within worms
final Slider separationOfWormsSlider = new Slider(0,200,140);
 private static double separationOfWorms = 100;

//Slider that duration of translation events
final Slider durationSlider = new Slider(0, 200, 100);
//Duration of graphical events
private static  Duration duration = Duration.millis(100);


//Slider for the elasticity of inner links
final Slider innerElasticitySlider = new Slider(0.5, 2, 0);

//Slider for the elasticity of
//bordering links    (head and tail)
final Slider borderElasticitySlider
                    = new Slider(0.5, 2, 0);
//Dynamics can be fueled by randomness,
//by evolution without hints,
//or by evolution with hints
private static String typeOfProcess
                    = "Evolution without hints";
//Blackboard to write a report
private static TextArea console
                    = new TextArea ("Use copy-paste");
```

```
//Click over this button to begin simulation
final Button goBt = new Button ("Go");

//Click over this button to take a screenshot of worms
private static Button wormsShotBt = new Button ("Screenshot

//This is the outer pane to place tabs
BorderPane outerPane = new BorderPane();

//Pane to place a console
private static FlowPane flowPane;
//Declaration of line chart
private static LineChart newLinkChart;

//Declaration of bar chart of perfection of fossils
private static BarChart qualityChart;

//Declaration of pie chart of perfection of fossils
private static PieChart qualityPieChart;

//Number of perfect fossils
private static int nPerfect;
//Number of imperfect fossils
private static int nImperfect;
//Quality of fossils
private static String quality = "";

//Tool for information display
private Pagination pagination;
//Information to be displayed.
//Pages are separated by a comma.
final String[] textPages = new String[]
{
    "A population of worms is programmed "
    + "by evolution to "
    + "climb the screen."
    ,"Evolution must synthesize an ordered set "
    + "of instructions that say what link to move "
```

```
        + "and in which direction."
        , "Constraint : "
        + "Worms cannot be disrupted."
        , "All moves are executed: "
        + "disruptive moves are undone "
        + "but  nondisruptive moves remain."
        ,"One link is added when some worm is shown "
        + "to climb the screen."
        ,"A fossil study is included."
        ,"The  simulation departs from the accepted by us"
        + " fact that "
        + "evolution can create complexity and is"
        + " addressed to tackle two questions:"
        + " \n   1. Which is the price that evolution "
        + "must pay for complexity?"
        + "\n   2. Does imperfect fossils appear "
        + "in fossil sampling "
        + "and in which proportion?"

    };

    /* Definition of inner classes.
    *
    The Move class:
    A Move says in which direction
    a link of the worm must be moved.
    */

private  class Move
{
    int x;
    int y;


    //Default initialization
    Move()
    {
        x= 0;
```

```
        y= 0;
    }

    // Initialize with a list of two coordinates
    Move(int a, int b)
    {
        x= a;
        y= b;
    }

    // Copy a move
    Move(Move move)
    {
        x= move.x;
        y= move.y;
    }
}//End of class Move


// Definition of the class Mover:
// A Mover keeps two types of information:
// a) the link of the worm to be moved and
// b) the direction or move to be chosen.
//
private   class Mover
{
    int link ;
    Move   move;

    //  Movers can acquired information
    // in three ways, they are  overloaded.
    // Form 1: by a list of numbers
    private Mover(int l, int mx, int my)
     {
        link= l;
        move.x = mx;
        move.y = my;
     }
```

```
    // Form 2, by structured data: link + move
    private Mover(int l, Move m)
    {
        link= l;
        move = new Move(m);
    }

    // Form 3, by structured data: mover
    private Mover(Mover mover)
    {
        link= mover.link;
        move =  new Move(mover.move);
    }
  }//End of class Mover


 //A random color is synthesized
 private static Color randomColor()
 {
    //Color is defined as a random
    //mix of red, green and blue
    int r,g,b;
    r = rd.nextInt(226);
    g = rd.nextInt(226);
    b = rd.nextInt(226);
    //The color is prepared
    Color c = Color.rgb(r,g,b);
    return c;
 }


//Rects are created
private static  Rectangle
           createRectangle(int gen)
{
    //Upper left corner at (0,0)
    Rectangle rect =
```

```
        new Rectangle(0,0,widthOfLinks,heightOfLinks);

    //The color is prepared
    Color c = randomColor();
    //The rect is painted
    rect.setFill(c);
    //Rects of the first generation (gen = 0)
    //are visible, otherwise they are not
    if (gen > 0)
    {
        rect.setHeight(0);
        rect.setWidth(0);
    }
    return rect;
}


/*  Here we define the class worm.
    A worm is composite: it has links.
    Each link has a rectangular shape
    and  its own color.
    The vector Mover[ ]
     keeps the information
    on how to move its links:
    which  link and in which order
    (in the order that they appear in
    the vector Mover[]), and
    in which direction.
    Each worm keeps record of
    the actual position of each
    one of its links,
    in the associated rectangle. */



private class Worm
{
    //Number of links
```

```
    int numberOfLinks = nLinks;
    //Number of movers = instructions to move
    int numberOfMovers = nLinks;
    //List Mover  keeps movers
    List<Mover> Mover = new ArrayList<>();
    //The shape of each link
    List<Rectangle> Rect = new ArrayList<>();
    //move right (side = 1) else left (side = -1)
    int side = 2*rd.nextInt(2) -1;

    //Default initialization: at random.
    //Worms of the first generation are
    //visible. Others not.
    private Worm()
    {
        //Visibility of links: 0 = visible; 1 = unvisible
        int k;
        if ( (typeOfProcess.equals("Randomness"))
            && (gen == 0))    k = 0;
        else  k = 1;

        //A color is assigned to each link
        //but different than white
        for(int i = 0; i < numberOfLinks; i++)
            Rect.add(createRectangle(k));

        //Instructions to move at random
for(int m = 0; m < nMovers; m++)
        {
            //Select at random  which link to move:
            int l = rd.nextInt(nLinks) ;
            //Select at random  which direction to move on
            Move  move
                    = new Move (vectMoves[rd.nextInt(9)]);
            //Memorization
            Mover.add(new Mover(l,move));

        }
```

```
    }//End of constructor Worm()


    //Constructor from worm W of the old generation.
    private Worm(int W)
    {
        numberOfLinks = nLinks;
        numberOfMovers = nMovers;
        //Symbolic initialization
        for(int i = 0; i < numberOfLinks; i++)
               Rect.add(createRectangle(1));
        for(int m = 0; m < nMovers; m++)
        {
            //Select at random  which link to move:
            int l = rd.nextInt(nLinks) ;
            //Select at random  which direction to move on
            Move  move
                    = new Move (vectMoves[rd.nextInt(9)]);
            //Memorization
            Mover.add(new Mover(l,move));
        }
        //Read true color
        for(int i = 0; i <  OldWorms[W].numberOfLinks; i++)
        {
            Rect.get(i).setFill( OldWorms[W]
                                 .Rect.get(i).getFill());
        }
        //Read true movers
        for(int m = 0; m < OldWorms[W].numberOfMovers; m++)
           Mover.set(m, OldWorms[W].Mover.get(m));
    }

    //Human design initialization
    private Worm(String HD )
    {
numberOfLinks = nLinks;
        numberOfMovers = nLinks;
        //Rects
```

```
        for(int i = 0; i < numberOfLinks; i++)
               Rect.add(createRectangle(1));
        //The know how to climb the screen
int link;
for(int l = 0; l  < nLinks; l++)
        {
            //Move the tail first and then the others links.
            link = nLinks-l-1;
            Mover mover;

            //Move each link in the y-direction
            if (side == 1)
                mover = new Mover(link,new Move(1,1));
            else
                mover =  new Mover(link,new Move (-1,1));
            Mover.add(mover);
            if (print3)
                System.out.println(l + "WormHD link = "
                + mover.link + " move.x = " + mover.move.x
                + " move.y= " + mover.move.y);
        }
    }//End of Worm()


    //Constructor from a fossil
    private Worm(Fossil fossil)
    {
        numberOfLinks = fossil.numberOfLinks;
        numberOfMovers = fossil.numberOfMovers;
        //Rects are initialized, 1 = unvisible
        for(int i = 0; i < numberOfLinks; i++)
               Rect.add(createRectangle(1));

        //Movers are copied
        for(int m = 0; m < numberOfMovers; m++)
        {
            //Link is read:
            int link = fossil.Mover.get(m).link;
```

```
            //Move is read
            Move  move
                    = w.new Move (fossil.Mover.get(m).move);
            //Mover is built
            Mover mover = w.new Mover(link,move);
            //Memorization
            Mover.add(mover);
            if (print3)
                System.out.println(m
                        + "Fossil -> worm link = "
                + mover.link + " move.x = " + mover.move.x
                + " move.y= " + mover.move.y);
        }
    }

}//End of class Worm



//Inner class Fossil.
//Fossils keep relevant information about worms
private class Fossil
{
    int numberOfLinks;
    int numberOfMovers;
    //The shape of each link
    List<Rectangle> Rect = new ArrayList<>();
    //List mover  keeps movers
    List<Mover> Mover = new ArrayList<>();


    //Constructor of a fossil from a worm
    //Worm -> fossil
    private Fossil(Worm worm)
    {
        numberOfLinks = worm.numberOfLinks;
        numberOfMovers = worm.numberOfMovers;
```

```
        //Rects are created
         for(int i = 0; i < numberOfLinks; i++)
               Rect.add(createRectangle(1));

        //Movers are cloned
        for(int m = 0; m < numberOfMovers; m++)
            Mover.add(new Mover(worm.Mover.get(m)));
    }
}//End of Fossil




//General setting
@Override public void start(Stage stage)
{
    //The scene
    Scene scene = new Scene(p, xWin,yWin);
    stage.setScene(scene);
    stage.setTitle("Evolution teaches worms "
                   + "to climb the screen");
    stage.show();
    //Definition of possible moves
    moveDef();



    //========BEGINNING OF GUI============

    //========MENUBAR===========

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

    // --- Menu Program
    Menu menuProgram = new Menu("Program");
```

```
//Instantiation and initialization = 0th page
pagination = new Pagination(textPages.length, 0);
//Colored border
pagination.setStyle("-fx-border-color:red;");
//Pagination master
pagination.setPageFactory((Integer pageIndex) ->
{
    if (pageIndex >= textPages.length)
    {
        return null;
    }
    else
    {
        return createPage(pageIndex);
    }
});

//Pane to place pages
AnchorPane anchor = new AnchorPane();

//Margins
AnchorPane.setTopAnchor(pagination, 20.0);
AnchorPane.setRightAnchor(pagination, 20.0);
AnchorPane.setBottomAnchor(pagination, 10.0);
AnchorPane.setLeftAnchor(pagination, 10.0);
//Booking
anchor.getChildren().addAll(pagination);

//Pane
FlowPane s = new FlowPane();
s.getChildren().add(anchor);
//Pane is  hidden
s.setOpacity(0);
s.setLayoutX(100);
s.setLayoutX(100);

//Click over this button to begin simulation
final Button closeBt = new Button ("Close");
```

```
s.getChildren().add(closeBt);

p.getChildren().add(s);
//Listener of OK button
closeBt.setOnAction((ActionEvent e) ->
{
    s.setOpacity(0);

});


//Menu item declaration and initialization
MenuItem displayInfo = new MenuItem("Information");
//Listener
displayInfo.setOnAction((ActionEvent t) ->
{
 s.setOpacity(1);
});

//MenuItems added to menu
menuProgram.getItems().addAll(displayInfo);


//MenuItem exit
MenuItem exit = new MenuItem("Exit");
exit.setOnAction((ActionEvent t) ->
{
    //Finish everything and exit
    System.exit(0);
});
//Insertions
menuProgram.getItems()
        .addAll(
                new SeparatorMenuItem(), exit);

    menuBar.getMenus()
        .addAll(menuProgram);
 p.getChildren().add(menuBar);
```

```
//======== Tabs=========


//Position of outerPane

outerPane.setLayoutX(100);
outerPane.setLayoutY(500);
outerPane.setOpacity(0);

outerPane.getChildren().add(wormsShotBt);
//Screenshot of worms or fossils
wormsShotBt.setLayoutX(50);
wormsShotBt.setLayoutY(70);
wormsShotBt.setOpacity(0);
wormsShotBt.setText("Take screenshot of worms");
p.getChildren().addAll(wormsShotBt);

//Listener of wormsShot button
wormsShotBt.setOnAction((ActionEvent t) ->
{
    //Rectangle for screenshot is gotten
    int x = (int)Math.round(stage.getX() );
    int y = (int)Math.round(stage.getY() + 145);
    int ww = (int)Math.round(outerPane.getWidth() );
    int h = (int)Math.round(outerPane.getLayoutY() - 120);
    //Rect is catured to clipboard
    captureRect(x,y,ww,h);
});



//Pane to host tabs
TabPane tabPane = new TabPane();


//=====Worms Tab
```

```
Tab wormsTab = new Tab();
wormsTab.setText("Worms");

//Number of worms is captured
final ComboBox nWormsCB = new ComboBox();
//Options
nWormsCB.getItems()
        .addAll("1",  "2", "3", "4", "5","6",
                "7", "8", "9", "10");
nWormsCB.setEditable(false);
nWormsCB.setValue(nWorms);
//Listener
nWormsCB.setOnAction((Event ev) ->
{
    String s1 =
    nWormsCB.getSelectionModel()
            .getSelectedItem().toString();
    nWorms = Integer.parseInt(s1);
    System.out.println(nWorms);
});


//Number of links is captured
final ComboBox nLinksCB = new ComboBox();
//Options
nLinksCB.getItems()
        .addAll(  "2", "3", "4", "5","6",
                "7", "8", "9", "10");
nLinksCB.setEditable(false);
nLinksCB.setValue(nLinks);
//Listener
nLinksCB.setOnAction((Event ev) ->
{
    String s1 =
    nLinksCB.getSelectionModel()
            .getSelectedItem().toString();
    nLinks = Integer.parseInt(s1);
    System.out.println(nLinks);
```

```
});


//Width of links
final ComboBox widthOfLinksCB = new ComboBox();
//Options
widthOfLinksCB.getItems()
        .addAll(  "2", "4", "6", "8", "10", "12");
widthOfLinksCB.setEditable(false);
widthOfLinksCB.setValue(widthOfLinks);
widthOfLinksCB.setOnAction((Event ev) ->
{
    String s1 =
    widthOfLinksCB.getSelectionModel()
            .getSelectedItem().toString();
    widthOfLinks = Integer.parseInt(s1);
    System.out.println(widthOfLinks);
});

//Height of links
final ComboBox heightOfLinksCB = new ComboBox();
//Options
heightOfLinksCB.getItems()
        .addAll(  "2", "4", "6", "8", "10", "12");
heightOfLinksCB.setEditable(false);
heightOfLinksCB.setValue(heightOfLinks);
heightOfLinksCB.setOnAction((Event ev) ->
{
    String s1 =
    heightOfLinksCB.getSelectionModel()
            .getSelectedItem().toString();
    heightOfLinks = Integer.parseInt(s1);
    System.out.println(heightOfLinks);
});



//Slider for the elasticity of inner links
```

```
innerElasticitySlider.setMin(0.5);
innerElasticitySlider.setMax(2);
innerElasticitySlider.setValue(innerElasticity);
innerElasticitySlider.setShowTickLabels(true);
innerElasticitySlider.setShowTickMarks(true);
innerElasticitySlider.setMajorTickUnit(0.5);
innerElasticitySlider.setMinorTickCount(1);
innerElasticitySlider.setBlockIncrement(0.1);

innerElasticitySlider.valueProperty().addListener((
ObservableValue<? extends Number> ov, Number old_val,
Number new_val) ->
{
    innerElasticity =    new_val.doubleValue();
    System.out.println(innerElasticity);
});


//Slider for the elasticity of bordering links,
//head and tail
borderElasticitySlider.setMin(0.5);
borderElasticitySlider.setMax(2);
borderElasticitySlider.setValue(borderElasticity);
borderElasticitySlider.setShowTickLabels(true);
borderElasticitySlider.setShowTickMarks(true);
borderElasticitySlider.setMajorTickUnit(0.5);
borderElasticitySlider.setMinorTickCount(1);
borderElasticitySlider.setBlockIncrement(0.1);

borderElasticitySlider.valueProperty().addListener((
ObservableValue<? extends Number> ov, Number old_val,
Number new_val) ->
{
    borderElasticity =    new_val.doubleValue();
    System.out.println(borderElasticity);
});
```

```
//Number of steps is captured
final ComboBox nStepsCB = new ComboBox();
//Options
nStepsCB.getItems()
        .addAll( "1", "2", "3", "4", "5","6",
                "7", "8", "9", "10");
nStepsCB.setEditable(false);
nStepsCB.setValue(nSteps);
nStepsCB.setOnAction((Event ev) ->
{
    String s1 =
    nStepsCB.getSelectionModel()
            .getSelectedItem().toString();
    nSteps = Integer.parseInt(s1);
    System.out.println(nSteps);
});


//Number of steps is captured
final ComboBox nMoversCB = new ComboBox();
//Options
nMoversCB.getItems()
        .addAll( "1", "2", "3", "4", "5","6",
                "7", "8", "9", "10");
nMoversCB.setEditable(false);
nMoversCB.setValue(nMovers);
nMoversCB.setOnAction((Event ev) ->
{
    String s1 =
    nMoversCB.getSelectionModel()
            .getSelectedItem().toString();
    nMovers = Integer.parseInt(s1);
    System.out.println(nMovers);
});


//Slider for Position of heads
```

```
verticalShiftSlider.setMin(-100);
verticalShiftSlider.setMax(100);
verticalShiftSlider.setValue(0);
verticalShiftSlider.setShowTickLabels(true);
verticalShiftSlider.setShowTickMarks(true);
verticalShiftSlider.setMajorTickUnit(100);
verticalShiftSlider.setMinorTickCount(50);
verticalShiftSlider.setBlockIncrement(50);

verticalShiftSlider.valueProperty().addListener((
ObservableValue<? extends Number> ov, Number old_val,
Number new_val) ->
{
    initialRow = (int) (280 +  new_val.doubleValue());
    System.out.println(" initialRow = " + initialRow);
});

//Slider for the separation in within worms
separationOfWormsSlider.setMin(0);
separationOfWormsSlider.setMax(200);
separationOfWormsSlider.setValue(140);
separationOfWormsSlider.setShowTickLabels(true);
//separationOfWormsSlider.setShowTickMarks(true);
separationOfWormsSlider.setMajorTickUnit(100);
separationOfWormsSlider.setMinorTickCount(50);
separationOfWormsSlider.setBlockIncrement(50);


separationOfWormsSlider.valueProperty().addListener((
ObservableValue<? extends Number> ov, Number old_val,
Number new_val) ->
{
    separationOfWorms = new_val.doubleValue();
    System.out.println("separation Of Worms = "
            + separationOfWorms);
});

//Slider for the separation in within worms
```

```
durationSlider.setMin(0);
durationSlider.setMax(200);
durationSlider.setValue(100);
durationSlider.setShowTickLabels(true);
//durationSlider.setShowTickMarks(true);
durationSlider.setMajorTickUnit(100);
durationSlider.setMinorTickCount(50);
durationSlider.setBlockIncrement(50);


durationSlider.valueProperty().addListener((
ObservableValue<? extends Number> ov, Number old_val,
Number new_val) ->
{
    int durationReading = (int) (new_val.doubleValue());
    duration = Duration.millis(durationReading);
    System.out.println("Duration of translation"
            + " events = "  + duration);
});




//GridPane  of wormsTab
GridPane gridWT = new GridPane();
gridWT.setVgap(4);
gridWT.setHgap(10);
gridWT.setPadding(new Insets(5, 5, 5, 5));
//Row   0
gridWT.add(new Label("Choose the number of "
        + "worms per generation: "), 0, 0);

gridWT.add(nWormsCB, 1, 0);

//Row   1
gridWT.add(new Label("Choose the number"
        + " of links: "), 0, 1);
```

```
gridWT.add(nLinksCB, 1,1);

gridWT.add(new Label("Choose the width "
        + "of links: "), 2, 1);
gridWT.add(widthOfLinksCB, 3,1);

gridWT.add(new Label("Choose the height "
        + "of links: "), 4, 1);
gridWT.add(heightOfLinksCB, 5,1);

//Row   2
gridWT.add(new Label("Choose the inner "
        + "elasticity: "), 0, 2);
gridWT.add(innerElasticitySlider, 1,2);

gridWT.add(new Label("Choose the border "
        + "elasticity: "), 2, 2);
gridWT.add(borderElasticitySlider, 3,2);

//Row   3
gridWT.add(new Label("Choose the number "
        + "of steps: "), 0, 3);
gridWT.add(nStepsCB, 1,3);

//Row   4
gridWT.add(new Label("Choose the number "
        + "of movers: "), 0, 4);
gridWT.add(nMoversCB, 1,4);

//Row   5
gridWT.add(new Label("Vertical shift "
        + "of the head: "), 0, 5);
gridWT.add(verticalShiftSlider, 1,5);

//Row   6
gridWT.add(new Label("Separation "
        + "in within worms: "), 0, 6);
gridWT.add(separationOfWormsSlider, 1,6);
```

```
//Row   7
gridWT.add(new Label("Duration of "
        + "translating events: "), 0, 7);
gridWT.add(durationSlider, 1,7);

wormsTab.setContent(gridWT);
tabPane.getTabs().add(wormsTab);



//============Evolution tab
Tab evolutionTab = new Tab();
evolutionTab.setText("Evolution");




//Number of generations is captured
final ComboBox nGenerationsCB = new ComboBox();
//Options
nGenerationsCB.getItems()
        .addAll("2",  "5", "10", "20", "50","100",
                "200", "300", "400", "500");
nGenerationsCB.setEditable(false);
nGenerationsCB.setValue(nGenerations);
nGenerationsCB.setOnAction((Event ev) ->
{
    String s1 =
    nGenerationsCB.getSelectionModel()
            .getSelectedItem().toString();
    nGenerations = Integer.parseInt(s1);
    System.out.println(nGenerations);
});


//Number of point mutation per child
```

```
final ComboBox nPointMutationsCB = new ComboBox();
//Options
nPointMutationsCB.getItems()
        .addAll( "1", "2", "3", "4", "5","6",
                 "7", "8", "9", "10");
nPointMutationsCB.setEditable(false);
nPointMutationsCB.setValue(nMovers);
nPointMutationsCB.setOnAction((Event ev) ->
{
    String s1 =
    nPointMutationsCB.getSelectionModel()
            .getSelectedItem().toString();
    nPointMutations = Integer.parseInt(s1);
    System.out.println(nPointMutations);
});

//Number of nSourceMovers =
//number of movers that serve as pattern to be cloned
final ComboBox nSourceMoversCB = new ComboBox();
//Options
nSourceMoversCB.getItems()
        .addAll( "1", "2", "3", "4", "5","6",
                 "7", "8", "9", "10");
nSourceMoversCB.setEditable(false);
nSourceMoversCB.setDisable(true);
nSourceMoversCB.setValue(nSourceMovers);
nSourceMoversCB.setOnAction((Event ev) ->
{
    String s1 =
    nSourceMoversCB.getSelectionModel()
            .getSelectedItem().toString();
    nSourceMovers = Integer.parseInt(s1);
    System.out.println(nSourceMovers);
});

 //Number of copies of each clone
final ComboBox nCopiesCB = new ComboBox();
//Options
```

```
nCopiesCB.getItems()
        .addAll( "1", "2", "3", "4", "5","6",
                "7", "8", "9", "10");
nCopiesCB.setEditable(false);
nCopiesCB.setDisable(true);
nCopiesCB.setValue(nCopies);
nCopiesCB.setOnAction((Event ev) ->
{
    String s1 =
    nCopiesCB.getSelectionModel()
            .getSelectedItem().toString();
    nCopies = Integer.parseInt(s1);
    System.out.println(nCopies);
});


//Type of process
final ToggleGroup group = new ToggleGroup();

//First radio button
RadioButton rb1 = new RadioButton("Randomness");
rb1.setToggleGroup(group);
rb1.setUserData("Randomness");
//Second radio button
RadioButton rb2 =
        new RadioButton("Evolution without hints");
rb2.setToggleGroup(group);
rb2.setUserData("Evolution without hints");
typeOfProcess = "Evolution without hints";
rb2.setSelected(true);
//Third radio button
RadioButton rb3 =
        new RadioButton("Evolution with hint1 "
        + "= gene amplification");
rb3.setToggleGroup(group);
rb3.setUserData("Evolution with hint1"
                + " = gene amplification");
//Fourth radio button
```

```
RadioButton rb4 =
        new RadioButton("Evolution with hint2"
                            + " = move(1,1)");
rb4.setToggleGroup(group);
rb4.setUserData("Evolution with hint2"
                            + " = move(1,1)");
//Fifth radio button
RadioButton rb5 =
        new RadioButton("Evolution "
                + "+ hint 1 +  hint2 ");
rb5.setToggleGroup(group);
rb5.setUserData("Evolution "
                + "+ hint 1 +  hint2 ");
//Sixth radio button
RadioButton rb6 =
        new RadioButton("Comparison");
rb6.setToggleGroup(group);
rb6.setUserData("Comparison");
rb6.setDisable(true);
//Sixth radio button
RadioButton rb7 =
        new RadioButton("Human design");
rb7.setToggleGroup(group);
rb7.setUserData("Human design");



//Listener
group.selectedToggleProperty()
    .addListener((ObservableValue<?
        extends Toggle> ov, Toggle t1, Toggle t2) ->
    {
        RadioButton chk = (RadioButton)t2
            .getToggleGroup().getSelectedToggle();
        if (print3)
            System.out.println("Selected Radio Button = "
            + chk.getText());
```

```
if (chk.getText().contains("Randomness"))
{
    System.out.println("Randomness");
    typeOfProcess = "Randomness";
}
else if
   (chk.getText().contains(
          "Evolution without hints"))
{
    System.out.println(
           "Evolution without hints");
    typeOfProcess = "Evolution without hints";

}
else if
    (chk.getText().contains(
           "Evolution with hint1 "
           + "= gene amplification"))
{
    System.out.println(
           "Evolution with hint1 "
           + "= gene amplification");
    typeOfProcess =  "Evolution with hint1 "
           + "= gene amplification";


}
else if
    (chk.getText().contains(
           "Evolution with hint2"
                  + " = move(1,1)"))
{
    System.out.println(
           "Evolution with hint2 "
           + "=  move(1,1)");
    typeOfProcess =  "Evolution with hint2"
                  + " = move(1,1)";
}
```

```
else if
    (chk.getText().contains(
           "Evolution "
        + "+ hint 1 +  hint2 "))
{
    System.out.println(
            "Evolution "
        + "+ hint 1 +  hint2 ");
    typeOfProcess =  "Evolution "
        + "+ hint 1 +  hint2 ";

}
else if
    (chk.getText().contains(
           "Comparison"))
{
    System.out.println(
            "Comparison");
    typeOfProcess =  "Comparison";
}
else if
    (chk.getText().contains(
           "Human design"))
{
    System.out.println(
            "Human design");
    typeOfProcess =  "Human design";
}


//Subordination of nSourceMoversCB and nCopiesCB
//to hint 1
if
    ((chk.getText().contains(
            "Evolution with hint1 "
            + "= gene amplification")
        ||
        (chk.getText().contains("Evolution "
```

```
                + "+ hint 1 +  hint2 "))))
        {
            nSourceMoversCB.setDisable(false);
            nCopiesCB.setDisable(false);
        }
        else
            {
                nSourceMoversCB.setDisable(true);
                nCopiesCB.setDisable(true);
            }
    });


//Booking
VBox vbox = new VBox();
vbox.getChildren().add(rb1);
vbox.getChildren().add(rb2);
vbox.getChildren().add(rb3);
vbox.getChildren().add(rb4);
vbox.getChildren().add(rb5);
vbox.getChildren().add(rb6);
vbox.getChildren().add(rb7);
vbox.setSpacing(5);
//Presentation
vbox.setPadding(new Insets(20, 10, 10, 20));


/*
//Alternate disposition
        HBox hbox = new HBox();
hbox.getChildren().add(rb1);
hbox.getChildren().add(rb2);
hbox.getChildren().add(rb3);
hbox.getChildren().add(rb4);
hbox.getChildren().add(rb5);
hbox.getChildren().add(rb6);
//Presentation
```

```
hbox.setSpacing(5);
hbox.setPadding(new Insets(20, 10, 10, 20));
*/



//GridPane  of wormsTab
GridPane gridET = new GridPane();
gridET.setVgap(4);
gridET.setHgap(10);
gridET.setPadding(new Insets(5, 5, 5, 5));
//Column 0, Row   0
gridET.add(new Label("Choose the number "
        + "of  generations: "), 0, 0);
//Column 1, Row   0
gridET.add(nGenerationsCB, 1, 0);

//Column 0, Row   1
gridET.add(new Label("Choose the number "
        + "of  point mutations: "), 0, 1);
//Column 1, Row   1
gridET.add(nPointMutationsCB, 1, 1);



//Row   2
gridET.add(new Label("Choose the type "
        + "of process: "), 0, 2);
//Row 3
gridET.add(vbox, 0, 3);


//Column 0, Row   4
gridET.add(new Label("Choose the number "
        + "of  source movers  for hint 1: "), 0, 4);
//Column 1, row 4
gridET.add(nSourceMoversCB, 1, 4);
```

```
//Row   5
gridET.add(new Label("Choose the number "
        + "of  copies for hint 1: "), 0, 5);
gridET.add(nCopiesCB, 1, 5);


evolutionTab.setContent(gridET);
tabPane.getTabs().add(evolutionTab);



//============Fossils tab
Tab fossilsTab = new Tab();
fossilsTab.setText("Fossils");



//Number of sampled fossils is captured
final ComboBox nSampledFossilsCB = new ComboBox();
//Options
nSampledFossilsCB.getItems()
        .addAll(  "5", "10", "20", "50"
                ,"100", "200", "300");
nSampledFossilsCB.setEditable(false);
nSampledFossilsCB.setValue(nSampledFossils);
nSampledFossilsCB.setOnAction((Event ev) ->
{
    String s1 =
    nSampledFossilsCB.getSelectionModel()
            .getSelectedItem().toString();
    nSampledFossils = Integer.parseInt(s1);
    System.out.println(nSampledFossils);
});


//GridPane  of fossilsTab
GridPane gridFT = new GridPane();
```

```
gridFT.setVgap(4);
gridFT.setHgap(10);
gridFT.setPadding(new Insets(5, 5, 5, 5));
//Row   0
gridFT.add(new Label("Choose the number "
        + "of  fossils: "), 0, 0);
gridFT.add(nSampledFossilsCB, 1, 0);



fossilsTab.setContent(gridFT);
tabPane.getTabs().add(fossilsTab);


//Report tab
//Contains the blackboard of the application
Tab reportTab = new Tab();
reportTab.setText("Go + Report ");

ScrollPane scrollPane = new ScrollPane();

//Show scrollBars?
scrollPane.setHbarPolicy(ScrollBarPolicy.NEVER);
scrollPane.setVbarPolicy(ScrollBarPolicy.ALWAYS);
//Height of scrollPane
scrollPane.setPrefHeight(400);

//Pane to place a console
flowPane = new FlowPane();
flowPane.setPadding(new Insets(5, 0, 5, 0));
flowPane.setVgap(4);
flowPane.setHgap(4);
//One column
flowPane.setPrefWrapLength(100);
//Color for background = Pale blue
flowPane.setStyle("-fx-background-color: DAE6F3;");

 Button copyPasteBtn = new Button();
```

```
copyPasteBtn.setText("You can copy  output to"
        + " the clipboard and paste it into a document."
        + "\nText: click + control + A (or E) + C."
        + "\nGraphics: Accommodate each one "
        + "and click corresponding button.");
//position
copyPasteBtn.setLayoutX(400);
copyPasteBtn.setLayoutY(40);
copyPasteBtn.setOpacity(0);
flowPane.getChildren().add(copyPasteBtn);


//Listener of OK button
goBt.setOnAction((ActionEvent e) ->
{
    //Date of experiment is reported
    Calendar cal = Calendar.getInstance();
    String format = "yyyy-MM-dd HH:mm:ss";
    SimpleDateFormat sdf = new SimpleDateFormat(format);
    String startTime =  sdf.format(cal.getTime());
    console.appendText("Program L92 WormsEvoJFX4 "
            + "\nDate of experiment = " + startTime);
    //CopyPaste button is made visible
    copyPasteBtn.setOpacity(1);
    //OK button gets unvisible
    goBt.setOpacity(0);
    //Parameters are reported
    appendParameters();
    //Simulation begins
    go();
    Calendar cal2 = Calendar.getInstance();
    String endTime =  sdf.format(cal2.getTime());
    console.appendText("\n\nExperiment ended by = "
            + endTime);
});
//OK button is booked
flowPane.getChildren().add(goBt);
```

```
//Console to write output
console = new TextArea ("");
//Unmodifiable
console.setEditable(false);
//Booking
flowPane.getChildren().add(console);


//Line chart for the generation at which
//a new link was added.
//Declaration of x and y axes
CategoryAxis xAxis1 = new CategoryAxis();
NumberAxis yAxis1 = new NumberAxis();
xAxis1.setLabel("New link");
yAxis1.setLabel("Generation");
//Instantiation
newLinkChart = new LineChart(xAxis1, yAxis1);
//Size
newLinkChart.setPrefSize(500, 330);




//Trigger of screeshot
Button saveLinkBtn = new Button("Save newLinkChart "
        + "to clipboard");
//Listener
saveLinkBtn.setOnAction((ActionEvent t) ->
{

    //Rectangle for screenshot is gotten
        Bounds b = newLinkChart.getBoundsInParent();
        int x = (int)Math.round(stage.getX()
                + outerPane.getLayoutX()   );
        int y = (int)Math.round(stage.getY()
                + outerPane.getLayoutY() + 61);
```

```
            int ww = (int)Math.round(b.getWidth());
            int h = (int)Math.round(b.getHeight() + 28);
        //Rect is catured to clipboard
        captureRect(x,y,ww,h);


});
//Booking
flowPane.getChildren().add(newLinkChart);
//vertical space in withing items
flowPane.setVgap(25);
//Booking
flowPane.getChildren().addAll( saveLinkBtn);



//BarChart for the quality of fossils:
//Perfect fossils can climb, imperfect not.
CategoryAxis xAxis2 = new CategoryAxis();
NumberAxis yAxis2 = new NumberAxis();
xAxis2.setLabel("Quality of reconstructed fossils ("
        + nSampledFossils + ")");
yAxis2.setLabel("Absolute frequency");
//Instantiation
qualityChart = new BarChart(xAxis2, yAxis2);

//Size
qualityChart.setPrefSize(500, 330);

flowPane.getChildren().add(qualityChart);



//Screenshot for qualityChart
Button saveQualityBtn
        = new Button("Save qualityChart to clipboard");
```

```
    saveQualityBtn.setOnAction((ActionEvent t) ->
    {

//Rectangle for screenshot is gotten
            Bounds b = newLinkChart.getBoundsInParent();
            int x = (int)Math.round(stage.getX()
                    + outerPane.getLayoutX()   );
            int y = (int)Math.round(stage.getY()
                    + outerPane.getLayoutY() + 61);
            int ww = (int)Math.round(b.getWidth());
            int h = (int)Math.round(b.getHeight() + 28);
//Rect is catured to clipboard
        captureRect(x,y,ww,h);


    });
    flowPane.getChildren().addAll( saveQualityBtn);




    ObservableList<PieChart.Data> qualityPieChartData =
                FXCollections.observableArrayList(
                new PieChart.Data("Perfect", 20),
                new PieChart.Data("Imperfect", 20));

    qualityPieChart = new PieChart(qualityPieChartData);
    qualityPieChart.setPrefSize(500, 330);
    flowPane.getChildren().add(qualityPieChart);
    final Label caption = new Label("");
    caption.setTextFill(Color.DARKORANGE);
    caption.setStyle("-fx-font: 24 arial;");



    //Screenshot for qualityChart
```

```
Button saveQualityPieBtn
    = new Button("Save qualityPieChart to clipboard");
saveQualityPieBtn.setOnAction((ActionEvent t) ->
{

    //Rectangle for screenshot is gotten
        Bounds b = newLinkChart.getBoundsInParent();
        int x = (int)Math.round(stage.getX()
                + outerPane.getLayoutX()   );
        int y = (int)Math.round(stage.getY()
                + outerPane.getLayoutY() + 61);
        int ww = (int)Math.round(b.getWidth());
        int h = (int)Math.round(b.getHeight() + 28);
        //Rect is catured to clipboard
    captureRect(x,y,ww,h);
});
flowPane.getChildren().addAll( saveQualityPieBtn);




//Filling and booking
scrollPane.setContent(flowPane);
reportTab.setContent(scrollPane);
tabPane.getTabs().add(reportTab);


//======

//Welcome button
Button welcomeBtn = new Button();
welcomeBtn.setText("Set parameters and "
        + "clik  over the Go button, "
        + "under the Go + Report tab, "
        + "to begin simulation."
        + "\nTo confirm reading, click over here.");
```

```
    //Position
    welcomeBtn.setLayoutX(400);
    welcomeBtn.setLayoutY(40);
    //Listener
    welcomeBtn.setOnAction((ActionEvent event) ->
    {
        welcomeBtn.setOpacity(0);
        //GUI is made visible
        outerPane.setOpacity(1);
    });
    //Booking
    p.getChildren().add(welcomeBtn);

    //=========TABPANE
    //TabPane is booked and  positioned
    outerPane.setCenter(tabPane);
    //OuterPane is booked
    p.getChildren().add(outerPane);



    //================================
    //======== END OF GUI=============
    //================================


}


//Start simulation
private static void go()
{
    //Text to be drawn
    infoText = new Text ("");
    infoText.setLayoutX(40);infoText.setLayoutY(60);
    p.getChildren().addAll(infoText);
    evolutionST.setCycleCount(1);
```

```
//Evolution for nGenerations
for(int i = 0; i < nGenerations; i++)
{
    //The scene is void
    voidOccupationMatrix();
    //Generation
    gen = i;
    //if (print2)
        System.out.println("\nGen = " + gen);
    //Generation number is drawn
    Timeline timeline = drawGen(gen);
    evolutionST.getChildren().addAll( timeline );
    //Worms are created and posited at the start line
    startLine(typeOfProcess);
    //Every worm leaves a fossil
    fossilization();
    //Worms express their information to move
    SequentialTransition oneGenST = oneGeneration();
    evolutionST.getChildren().add(oneGenST);
    //Drawn worms get hidden
    ParallelTransition hideST  =  hideST();
    evolutionST.getChildren().add(hideST);
    //Helping array
    Worm[] file1 = new Worm[50];
    //Information is recorded in an array
    OldWorms = Worms.toArray(file1);

    //Destroy everything
    //for the sake of a new beginning
    Worms.clear();
    WormsTwo.clear();

    //Success?
    Integer genI = gen;
    if (bullseye)
    {
        //If climbing was successful, add a new link
         nLinks = nLinks+1;
```

```
            //List of successful generations
            bullseyes.add(genI);
        }

    }
    //EvolutionST is booked
    allST.getChildren().add(evolutionST);

    console.appendText("\n\n====OUTPUT===== ");
    console.appendText("\n\n=SUCCESSFUL GENERATIONS= ");
    Integer size = bullseyes.size();
            for(Integer index = 0; index < size; index++)
            {
    console.appendText("\n" + bullseyes.get(index));

            }
    //Data for newLinkChart is provided
    newLinkChart.setData(getNewLinkChartData());
newLinkChart.setTitle("Cost of complexity under \n"
            + typeOfProcess);



  //Fossil study
    SequentialTransition fsST = fossilSampling();
    //Data for qualityChart of fossils is provided
    qualityChart.setData(getQualityChartData() );
    qualityChart.setTitle("Quality of fossils under \n"
            + typeOfProcess);


    ObservableList<PieChart.Data> qualityPieChartData =
        FXCollections.observableArrayList(
        //1 is added in case nPerfect = 0
        new PieChart.Data("Perfect", 10000*nPerfect+1),
        new PieChart.Data("Imperfect", 10000*nImperfect));
    qualityPieChart.setData(qualityPieChartData);
```

```
    qualityPieChart.setTitle("Quality of fossils under \n"
            + typeOfProcess
    + "\nPerfect = " + nPerfect
    + "    Imperfect = " + nImperfect);

    allST.getChildren().add(fsST);
    //Film is played
    allST.play();
    wormsShotBt.setOpacity(1);
    memoryUsage();

}

//Number of items per page
private int itemsPerPage()
{
    return 1;
}

//Pieces of information are associated to pages
private VBox createPage(int pageIndex)
{
    VBox box = new VBox(1);
    int page = pageIndex * itemsPerPage();
    for (int i = page; i < page + itemsPerPage(); i++)
    {
        TextArea text = new TextArea(textPages[i]);
        text.setPrefSize(20, 270);
        text.setFont(Font.font(18-i));
        text.setWrapText(true);
        box.getChildren().add(text);
    }
return box;
}
```

```
//Data for a newLinkChart is fabricated.
//It must be a map that associates a number
//to a string.
private static ObservableList<XYChart.Series<String, Double>>
    getNewLinkChartData()
    {
        //Data is packed into an ObservableList
        ObservableList<XYChart.Series<String, Double>>
            map = FXCollections.observableArrayList();
        XYChart.Series<String, Double> series
                        = new XYChart.Series<>();

        series.setName("Generation at which a new link "
                + "was added  during " + nGenerations
                + " generations");

        //Fabrication of data:
        //It is a series of couplets.
        //First item = index of new link.
        //Second item = generation at which link was added
        Integer size = bullseyes.size();
        for(Integer index = 0; index < size; index++)
        {

            series.getData().add(new XYChart.Data
                    (Integer.toString(index),
                            bullseyes.get(index)));
        }
        map.addAll(series);
        return map;
    }

//Data for qualityChart is fabricated.
//It must be a map that associates a number
//to a string.
private static ObservableList<BarChart.Series<String, Double>>
    getQualityChartData()
    {
```

```
        //Data is packed into an ObservableList
        ObservableList<BarChart.Series<String, Double>>
            map = FXCollections.observableArrayList();
        BarChart.Series<String, Double> series
                        = new BarChart.Series<>();

        series.setName("Absolute frequency");

        //Fabrication of data:
        //It is a series of couplets.
        //First item = quality descriptor.
        //Second item = absolute frequency

        series.getData().add(new BarChart.Data
                ("Perfect",
                        nPerfect));
        series.getData().add(new BarChart.Data
                ("Imperfect",
                        nImperfect));
        System.out.println("\nnPerfect = " + nPerfect
        + "\n nImperfect = " + nImperfect);
        map.addAll(series);
        return map;
    }



//Parameters are published on the console
private static void appendParameters()
{
    console.appendText("\n\n====PARAMETERS====");

    console.appendText("\n\n=WORMS= ");
    console.appendText("\nNumber of worms "
            + "per generation = " + nWorms);
    console.appendText("\nNumber of links = " + nLinks);
    console.appendText("\nWidth of links = "
            + widthOfLinks);
```

```
    console.appendText("\nHeight of links = "
            + heightOfLinks);
    console.appendText("\nInner elasticity = "
            + innerElasticity);
    console.appendText("\nBorder elasticity = "
            + borderElasticity);
    console.appendText("\nNumber of steps = " + nSteps);
    console.appendText("\nNumber of movers = " + nMovers);
    console.appendText("\nInitial row of the head = "
            + initialRow);

    console.appendText("\n\n=EVOLUTION=");
    console.appendText("\nNumber of  generations = "
            + nGenerations);
    console.appendText("\nNumber of point mutations = "
            + nPointMutations);
    console.appendText("\nNumber of source movers "
            + "in gene amplification = " + nSourceMovers);
    console.appendText("\nNumber of  copies "
            + "in gene amplification = " + nCopies);
    console.appendText("\nType of process = "
            + typeOfProcess);

    console.appendText("\n\n =FOSSILS=");
    console.appendText("\nNumber of fossils = "
            + nSampledFossils);
}



// To avoid overwriting of worms,
// an occupation record is kept:
// The screen is divided in cells width x hight.
// At the beginning every cell is empty=false.
// Occupancy is recorded in isOccupied[][],
// which is a boolean matrix.
// A boolean  variable can be
// true= occupied,  else false= not occupied.
```

```
private static void  voidOccupationMatrix()
{
   //System.out.println("occupationMatrix");
    for(int i = 0; i < 600; i++)
    for(int j = 0; j < 300; j++)
        isOccupied[i][j]= false;
}

//Nine basic moves = directions where to move in
static private void moveDef()
{
    final Move move0 =  w.new Move(0,1);
       vectMoves[0]   = move0 ;
    final Move move1 =  w.new Move(1,1);
       vectMoves[1]   = move1 ;
    final Move move2 =  w.new Move(1,0);
       vectMoves[2]   = move2 ;
    final Move move3 =  w.new Move(1,-1);
       vectMoves[3]   = move3 ;
    final Move move4 =  w.new Move(0,-1);
       vectMoves[4]   = move4 ;
    final Move move5 =  w.new Move(-1,-1);
       vectMoves[5]   = move5 ;
    final Move move6 =  w.new Move(-1,0);
       vectMoves[6]   = move6 ;
    final Move move7 =  w.new Move(-1,1);
       vectMoves[7]   = move7 ;
    final Move move8 =  w.new Move(0,0);
       vectMoves[8]   = move8 ;
}


static private void captureRect(int x,  int y, int w, int h)
{
        try
        {
            //Multipurpose tool
            java.awt.Robot robot = new java.awt.Robot();
```

```
            //Rectangle for screenshot
            java.awt.image.BufferedImage bi
                    = robot.createScreenCapture
                        (new java.awt.Rectangle(x,y,w,h));
            //Declaration of type of data
            java.io.ByteArrayOutputStream stream
                    = new java.io.ByteArrayOutputStream();
            //Formatting
            ImageIO.write(bi, "png", stream);
            //Image is created as image of screenshot
            Image image = new Image(new java
                .io
                .ByteArrayInputStream(
                    stream.toByteArray()), w, h, true, true);
            //Declaration of clipboard
            ClipboardContent cc = new ClipboardContent();
            //Clipboard is loaded with image
            cc.putImage(image);
            //Loading is finished
            Clipboard.getSystemClipboard().setContent(cc);
        }   //In case of problems
        catch (AWTException | IOException ex)
            {
                Logger.getLogger(WormsEvoJFX5
                .class.getName()).log(Level.SEVERE, null, ex);
            }

}

static private Worm chooseGenerator(int W, String process)
{
    //Auxiliary worm
    Worm   worm = null;

    //Generate a worm at random
    if (process.equals("Randomness"))
        worm = w.new Worm();
    //Copy worm W from old generation
```

```
    if (process.equals("Evolution without hints"))
        worm = w.new Worm(W);
    //Copy worm W from old generation
    if (process.equals("Evolution with hint1 "
                       + "= gene amplification"))
        worm = w.new Worm(W);
    //Copy worm W from old generation
    if (process.equals("Evolution with hint2"
                               + " = move(1,1)"))
        worm = w.new Worm(W);
    //Copy worm W from old generation
    if (process.equals("Evolution "
                   + "+ hint 1 +  hint2 "))
        worm = w.new Worm(W);
    //Moves are designed by human intelligence
    if (process.equals("Human design"))
    {
        String HD = "HD";
        nGenerations = 5;
        nMovers = nLinks;
        worm = w.new Worm(HD);
    }
    return worm;
}


//All worms are posited at the same initial row
//in vertical position.
static private void startLine(String process)
{
    for (int W = 0; W < nWorms; W++)
    {
        int  x,y;
        //Auxiliary worm
         Worm   worm;
         //Worms are created at generation 0
         if (gen == 0)
         {
```

```
            if (process.equals("Human design"))
               worm = chooseGenerator(W, process);
            else worm = w.new Worm();
         }
         else //worms are inherited
            worm = chooseGenerator(W, process);

        //Each link is positioned
        for(int  link = 0; link < nLinks; link++)
        {
            //Horizontal position
            x = (int) (separationOfWorms
                    + separationOfWorms * W);

            //Vertical position
            y = initialRow + heightOfLinks*link;

            //Position is updated
            worm.Rect.get(link).setX(x);
            worm.Rect.get(link).setY(y);

            //Link is shown
            ParallelTransition show
                        = showPT(worm.Rect.get(link));
            //Link is booked
            p.getChildren().add(worm.Rect.get(link));
            //Link is grouped
            allRects.add(worm.Rect.get(link));
            //Show-transition is booked
            evolutionST.getChildren().add(show);
        }
        //worm is located at position W in lists Worms
        Worms.add(worm);

        if (print1)
            System.out.println("W = " + W + " nLinks = "
                + WormsTwo.get(W).numberOfLinks  );
    }
```

```
}



//Draw the generation number
private static Timeline drawGen(int gen)
{
    Integer genI = gen;
    String message =  "Generation  = " +
                   genI.toString();
    //A JavaFX event committed to print a text is generated
    EventHandler printText =
        (EventHandler<ActionEvent>)
            (ActionEvent t ) ->
                {
                    infoText.setText(message);
                };
    //Presentation of printText must be
    //synchronized with the overall film so,
    //duration d is specified
    KeyFrame keyFrame = new KeyFrame(duration, printText);

    Timeline timeline = new Timeline();
    timeline.setCycleCount(1);
    //add the keyframe to the timeline
    timeline.getKeyFrames().add(keyFrame);

    return timeline;
}



//Is move m disruptive for the worm?
//(A worm is and must remain as a whole,
//nevertheless, some elasticity is allowed.)
```

```
private static boolean isDisruptive(int m, Worm worm )
{
    //A move is disruptive if the target address
    //is farther than certain distance
    //from the neighboring links.
    boolean isDisruptive = false;
    int  link;
    int  x,y,  x1,y1, x2,y2, x3,y3;
    double d1;

    double  d2;

    //Read the link to be moved
    link = worm.Mover.get(m).link;
    //Read the   address of the link;
    x  = (int) worm.Rect.get(link).getX();
    y  = (int) worm.Rect.get(link).getY();
    //The   move to be done is read
    Move move = w.new Move(worm.Mover.get(m).move);

    //The target address is calculated
    x1 = x + widthOfLinks*move.x;
    y1 = y -  widthOfLinks*move.y;

    //The addresses of the two neighboring links are read
    //The disruption is calculated
    //Case: intermediate link
    if ((0 < link) & (link < nLinks-1))
    {
        x2 = (int) worm.Rect.get(link-1).getX();

        y2 = (int) worm.Rect.get(link-1).getY();

         d1 =    pow(  ( pow( (x2 - x1),2)
                              +  pow((y2 - y1),2)),0.5);

        x3 = (int) worm.Rect.get(link+1).getX();
        y3 = (int) worm.Rect.get(link+1).getY();
```

```
        d2 =  pow(  (pow( (x3 - x1),2)
                              + pow((y3 - y1),2)),0.5);


        // Here we decide whether or not
        // a move is disruptive.
        if ((d1 > innerElasticity * widthOfLinks)
                || (d2 > innerElasticity * widthOfLinks))
                   isDisruptive = true;

     }

    //First link
    if (link == 0)
    {
        x2 = (int) worm.Rect.get(1).getX();
        y2 = (int) worm.Rect.get(1).getY();
        d1 = pow( (pow( (x2 - x1),2) + pow((y2 - y1),2)),0.5);
        if( d1 > borderElasticity
                * widthOfLinks) isDisruptive = true;
     }
     //Last link
    if( link == nLinks-1 )
    {
        x2 = (int) worm.Rect.get(nLinks - 2).getX();
        y2 = (int) worm.Rect.get(nLinks - 2).getY();
        d2 = pow( (pow( (x2 - x1),2) + pow((y2 - y1),2)),0.5);
        if ( d2 > borderElasticity
                *widthOfLinks ) isDisruptive =   true;
    }
    return isDisruptive;
 }



//Execute move m for worm W.
//If it is  disruptive, undo it.
```

```
private static SequentialTransition
                    exeIf( Worm worm, int m)
{
    if (print1)
        System.out.println(" exeIf");
    //Read the link to be moved
    int link = worm.Mover.get(m).link;
    //Read position
    int x = (int) worm.Rect.get(link).getX();
    int y = (int) worm.Rect.get(link).getY();

    //The move to be done is read
    Move  move;
    move = worm.Mover.get(m).move;
    //Move is interpreted
    int byX = widthOfLinks* move.x;
    int byY = -heightOfLinks* move.y;
    //Target address is calculated:
    int x1 = x + byX;
    int y1 = y + byY;
    if (x1 < 0 ) x1 = 0;
    if (y1 < 0 ) y1 = 0;
    if (print1)
        System.out.println(
            " move = " + m   + " link = " + link
            + " \n x = " + x + " y = " + y
            + " \nbyX = " + byX  + " byY = " + byY);


    //Occupied cell
    int i = (int)  x / widthOfLinks;
    int j = (int) y / heightOfLinks;


     //Target cell
    int k =(int) x1/widthOfLinks;
    int l = (int) y1/heightOfLinks;
```

```
//Move is always executed
SequentialTransition moveRect
        = new SequentialTransition();

TranslateTransition translateTransition =
     new TranslateTransition(duration
              , worm.Rect.get(link));
//Jump is executed
translateTransition.setByX(byX);
translateTransition.setByY(byY);
//Number of repetitions
translateTransition.setCycleCount(1);
moveRect.getChildren().add(translateTransition);



//Is the move disruptive?
boolean   isMoveDisruptive =  isDisruptive(m,worm);
if (print1)
{
    if (isMoveDisruptive)
        System.out.println("move is Disruptive");
    else
        System.out.println("move is not  Disruptive");
}
//Disruptive moves are undone
if  (((isOccupied[k][l])) || ((isMoveDisruptive)))
{
    byX = -byX;
    byY = -byY;
}
else //move is not disruptive -> it is not undone
{
    byX = 0;
    byY = 0;
    worm.Rect.get(link).setX(x1);
    worm.Rect.get(link).setY(y1);
```

```
        isOccupied[i][j]= false;
        isOccupied[k][l]= true;
    }



    //Move is possibly undone
    TranslateTransition translateTransition2 =
         new TranslateTransition(duration
                 , worm.Rect.get(link));
    //Jump is programmed
    translateTransition2.setByX(byX);
    translateTransition2.setByY(byY);
    //Number of repetitions
    translateTransition2.setCycleCount(1);
    moveRect.getChildren().add(translateTransition2);

    return moveRect;
 }


// Every worm is assigned a fitness
//The nearer to the top, the fitter.
//If climbing has been significative,
//the complexity of the game rises up one level:
//a true bullseye is reported (to add
//next a new link to all worms).
private static void assessFitness()
{
    int yOfTail;
    double yOfHead;

    bullseye  = false;
    for(int W = 0; W <  nWorms; W++)
    {
        // We get the y-coordinate of the tail
        yOfTail = (int) Worms.get(W)
```

```
                                .Rect.get(nLinks-1).getY();
            yOfHead =  Worms.get(W).Rect.get(0).getY();
            //Fitness measures how much the head has  been lifted
            Fitness[W] = (yWin - yOfTail);
            if(print1)
                System.out.println("Fitness of worm "
                        + W + " = " + Fitness[W]
                        + " head = " + yOfHead);
            //If worm W is good for climbing
            //its head must be high enough
            if (yOfHead < initialRow - 2 * heightOfLinks)
            {
                //Worms that climb are rewarded
                Fitness[W] = 100 * Fitness[W];
                bullseye = true;
                if (printReproduction)
                System.out.println(
                        "There was advance W = " + W );
            }
        }
    }



    //Worms express their information to move.
    private  static  SequentialTransition  express()
    {
        //Overall process
        SequentialTransition expressST
                        = new SequentialTransition();
         expressST.setCycleCount(1);
         //Worms advance nSteps
        for (int i = 0; i < nSteps; i++)
        {
            //Worms advance one step:
            //To move all links, each worm must execute
            //instructions to move
            SequentialTransition oneStepST
```

```
                    = new SequentialTransition();
        oneStepST.setCycleCount(1);
        //Information to move is read and decoded
        for (int m = 0; m < nMovers; m++)
        {
            if (print1)
                System.out.println("nMovers = " + nMovers);
            //All worms execute move m
            ParallelTransition moveMPT
                        = new ParallelTransition();
            moveMPT.setCycleCount(1);
            //This is a parallel loop for a list
            for (Worm worm: Worms)
            {
                //Direction is flipped in human design
                if (typeOfProcess.contains("Human design"))
                    worm.Mover.get(m).move.x
                            = - worm.Mover.get(m).move.x;
                //A translation of some link
                //is programmed for worm W
                //according to move m.
                //If it is not disruptive,
                //it is executed
                SequentialTransition tT = exeIf(worm,m);
                tT.setCycleCount(1);
                //Translation is booked
                moveMPT.getChildren().add(tT);
            }
            //The ensamble of translations is booked
            oneStepST.getChildren().add(moveMPT);
        }
    //The movement of worms in one step is booked
    expressST.getChildren().add(oneStepST);
    }
    return expressST;
}
```

```
//A hidden rect gets shown
private static
    ParallelTransition showPT(Rectangle rect)
{
    ParallelTransition s = new ParallelTransition();
    Duration d2 =   Duration.millis(1);
    //Rect is enlarged
    KeyValue keyValue3 =
        new KeyValue(rect.widthProperty(),widthOfLinks,
            Interpolator.LINEAR);
    KeyValue keyValue4 =
        new KeyValue(rect.heightProperty()
                ,heightOfLinks, Interpolator.LINEAR);
    //KeyValues are  inserted into a timeline
    Timeline b = new Timeline(
        new KeyFrame(d2,keyValue3,keyValue4 ));
    //Timeline is inserted into a transition
    s.getChildren().addAll(b);

    return s;
}




//Worms are hidden after their performance.
private static ParallelTransition  hideST()
{

    ParallelTransition hidePT
                = new ParallelTransition();
    hidePT.setCycleCount(1);
    Duration d2 =   Duration.millis(1);
    //Hide all rects
    for(Rectangle rect: allRects)
```

```
    {
       //Rects are made unvisible
        KeyValue keyValue =
                new KeyValue(rect
                        .visibleProperty(), false
                        ,Interpolator.LINEAR);
        Timeline b = new Timeline(
                new KeyFrame(d2,keyValue));
        hidePT.getChildren().add(b);
    }

    return hidePT;
}




//=====================================
//=========Evolutionary part===========
//=====================================


//One generation = expression + assessement
//+ reproduction, recombination and mutation.
private static   SequentialTransition  oneGeneration()
{
    SequentialTransition oneGenST
            = new SequentialTransition();
    //Worms express their information to move
    SequentialTransition expressST = express();
    oneGenST.getChildren().addAll(expressST);
    //Fitness is measured
    assessFitness();
    //Reproduction is proportional to fitness
    reproduction();
    //Population is replaced by a recombinat one
    recombination();
```

```
        //Information to move suffers mutation
        mutation();

        return oneGenST;
    }


/*  To generate a worm of the next generation,
an individual is sampled at random
from the old population
and cloned
with a probability that is
proportional to its fitness.
*/
private static void reproduction()
{
    if (printReproduction)
        System.out.println( "\nGeneration = " + gen);
    //Cumulative distribution
    Double cumulate[] = new Double[nWorms];
    // Counters
    int count[] = new int[nWorms];
    //Number of trials
    int nTrials = nWorms;

    //Initialization
    WormsTwo.clear();
    WormsTwo.addAll(Worms);
    for (int i=0; i< nWorms; i++)
        count[i]=0;

    //Cumulative distribution
    cumulate[0] = (double) Fitness[0];
    for (int i=1; i< nWorms; i++)
        cumulate[i]= cumulate[i-1] + Fitness[i];

    //Report
    if (printReproduction)
```

```
{
    System.out.println( "\ni \tfitness cumulate ");

for (int i = 0; i < nWorms; i++)
    System.out.println( i + "\t" + Fitness[i] + " \t "
            + cumulate[i]);
}

//Maximal value
double max = cumulate[nWorms-1];
//Counter of chosen worms
int W = 0;
//System.out.println("Generated numbers: ");
for (int i = 0; i < nTrials; i++)
{
    //A random integer less than max is generated
    //with a uniform distribution
    int n = rd.nextInt((int) max);

    //System.out.println("n =  " + n);
    //Probability of cloning is made
    //in proportion to fitness.
    //Is worm 0 to be chosen?
    if (n < cumulate[0])
    {
        /*
        if (printReproduction)
            System.out.println( "chosen individual  = 0 ");
        */
        WormsTwo.set(W, Worms.get(0));
        count[0]++;
        W++;
    }
    //Is worm l to be chosen?
    for (int l = 1; l < nWorms; l++)
    {

        if ( ( cumulate[l-1] <= n) & (n < cumulate[l]) )
```

```
            {
                /*
                if (printReproduction)
                    System.out.println(
                        "Chosen individual = " + l);
                */
                count[l]++;
                WormsTwo.set(W, Worms.get(l));
                W++;
            }
        }

    }
    if (printReproduction)
    {
        System.out.println("\nEvents and their observed "
            + "and expected frequencies: ");
        for (int i=0; i<nWorms;i++)
            System.out.println(i + "\t" + count[i] + "\t"
                +  Fitness[i]*nTrials/cumulate[nWorms-1]);
    }

    //WormsTwo is copied into Worms
    Worms.clear();
    Worms.addAll(WormsTwo);

}


//Recombination is implemented
private static void recombination()
{

    //Every worm is replaced by a recombinant
    for(int W = 0; W < nWorms; W++ )
    {
        //A  binary recombinant operator
        //is assembled at random.
```

```
        int rec[] = new int[210];
        for (int j = 0; j< nMovers; j++)
  rec[j]= rd.nextInt(2);
//Another worm is chosen at random for recombination
        int m = rd.nextInt(nWorms);
   //Basal initialization of the recombinant
        Worm  worm = Worms.get(W);
for(int j = 0; j< nMovers; j++)
        {
            // Recombination is executed.
            //For 0, take from the partner worm m
            //For 1, take from worm W.
            if (rec[j] == 0)
            worm.Mover.set(j, Worms.get(m).Mover.get(j));
        }

        //Recombinant is memorized
        WormsTwo.set(W, worm);
        if(print2)
        {
            System.out.println("Old move.x = "
                    + Worms.get(W).Mover.get(0).move.x);

            System.out.println("New Rec move.x = "
                    + worm.Mover.get(0).move.x);
        }

    }
    //Old population is replaced by the recombinated one
    Worms.clear();
    Worms.addAll(WormsTwo);
}
```

```
//Blind mutations of infomation to move
private static void moverMutation()
{
    int   m, n;
    //Let us try out blind  mutations
    //in specific Movers,
    //in the  link  to be moved
    //and in the direction of movement.
    for(int W = 0; W < nWorms; W++)
    {
        //Each offspring receives
        //numbPointMutation mutations
        for(int k = 0; k< nPointMutations; k++)
        {
            //Select at random  which mover to mutate:
            m =  rd.nextInt(nMovers );
            //Select at random which link  to move:
            n = rd.nextInt( nLinks) ;
            if(print2)
                System.out.println("Old move.x = "
                     + Worms.get(W).Mover.get(m).move.x);
            //Select at random which direction to move on.
            Move move = vectMoves[rd.nextInt(9) ];
            //Mover m is assembled from link + move
            Worms.get(W).Mover.set(m, w.new Mover(n,move));
            if(print2)
                System.out.println("New Mutation move.x = "
                    + Worms.get(W).Mover.get(m).move.x);
            //Color of link n is changed
            //(possibly various times)
            Worms.get(W).Rect.get(n).setFill(randomColor());
        }
    }
}


//Hint1 = some chosen moves replace others
//in within the same worm
```

```
private static void geneAmplification()
{
    //  System.out.println("geneAmplification");
    int    m,    mm;
    int    link;
    //All worms are scanned
    for(int W = 0; W < nWorms; W++)
    {
        //nSourceMovers are amplified
        for (int i = 0; i < nSourceMovers; i++)
        {
            //Select at random which Mover to be read
            m = rd.nextInt(nMovers);
            //Move to be amplified is read
            Move move =  Worms.get(W).Mover.get(m).move;
            //The Move is amplified
            for(int k = 0; k< nCopies; k++)
            {
                //Select at random the place for replacement:
                //throw a number from 1 to nMovers.
                mm = rd.nextInt(nMovers) ;
                //Link is read
                link = Worms.get(W).Mover.get(mm).link;
                //Mover mm is assembled from link + move
                //and amplified
                Worms.get(W).Mover
                    .set(mm, w.new Mover(link,move));
            }
        }
    }
}


//Hint2: move = Move(1,1) = move northeast
private static void move11()
{
    //  System.out.println("move11");
    int    m,    mm;
```

```
    int    link;
    //All worms are scanned
    for(int W = 0; W < nWorms; W++)
    {
        //nSourceMovers are amplified
        for (int i = 0; i < nSourceMovers; i++)
        {
            //Select at random which Mover to be read
            m = rd.nextInt(nMovers);
            //Move11
            Move move = w.new Move(1,1);
            //The Move is amplified
            for(int k = 0; k< nCopies; k++)
            {
                //Select at random the place for replacement:
                //throw a number from 1 to nMovers.
                mm = rd.nextInt(nMovers) ;
                //Link is read
                link = Worms.get(W).Mover.get(mm).link;
                //Mover mm is assembled from link + move
                //and cloned
                Worms.get(W).Mover
                    .set(mm, w.new Mover(link,move));
            }
        }
    }
}


//Information to move is changed
private static void mutation(   )
{

    //Mutation of movers
    moverMutation();
    //Mutations on request
    if (typeOfProcess.contains("Evolution with hint1 "
                        + "= gene amplification"))
```

```
        geneAmplification( );
    if (typeOfProcess.contains("Evolution with hint2"
                                + " = move(1,1)"))
        move11();
    if (typeOfProcess.contains("Evolution "
                  + "+ hint 1 +  hint2 "))
    {
      geneAmplification( );
      move11();
    }
}


//Worm ->fossil
//All worms leave their fossils
private static void fossilization()
{
    for(int W = 0; W < nWorms; W++)
    {
//Worm W leaves its fossil
        Fossil fossil = w.new Fossil(Worms.get(W));
        Fossils.add(fossil);
if (print1)
            System.out.println("Fossil number = "
    + nFossils
    + " nLinks = "
            + Fossils.get(nFossils).numberOfLinks);
        //Counter
nFossils++;
    }
}

//Fitness of reconstructed fossil is measured.
private static double fitness(Worm worm)
{
    double yOfTail;
    double fitness;
    // We get the y-coordinate of the tail
```

```
    yOfTail =  worm.Rect.get(nLinks-1).getY();
    //The nearer to the top of the screen (yOfTail = 0),
    //the fitter:
    fitness  = 100*(yWin - yOfTail);
    if(print1)
        System.out.println("Fitness of fossil = "
                           + fitness);
    //Classification according to perfection
    //Perfect = can climb
    //Imperfect = cannot climb
    double yOfHead =  worm.Rect.get(0).getY();
    if (yOfHead < initialRow - 2 * heightOfLinks)
    {
       nPerfect++;
       quality = "Perfect";
        if (print3)
        System.out.println( "perfect " );
    }
    else
    {
        nImperfect++;
        quality = "Imperfect";
    }

    return fitness;
}

//Draw the fossil number
private static Timeline drawFossilNumber(int f)
{
    Integer F = f;
    String message =  "Fossil  = " +
                    F.toString();
    //A JavaFX event committed to print a text is generated
    EventHandler printText =
        (EventHandler<ActionEvent>)
            (ActionEvent t ) ->
                {
```

```
                    infoText.setText(message
                            + " " + quality);
                };
        //Presentation of printText must be
        //synchronized with the overall film so,
        //duration d is specified
        KeyFrame keyFrame = new KeyFrame(duration, printText);

        Timeline timeline = new Timeline();
        timeline.setCycleCount(1);
        //add the keyframe to the timeline
        timeline.getKeyFrames().add(keyFrame);

        return timeline;
    }

    //Reconstructed worm is given an initial position
    private static ParallelTransition startLine(Worm worm)
    {
        voidOccupationMatrix();
        nLinks = worm.numberOfLinks ;
        int x,y;
        ParallelTransition showWorm = new ParallelTransition();
        showWorm.setCycleCount(1);
        for(int  link = 0; link < nLinks; link++)
        {
            //Horizontal position
            x = 140;
            //Vertical position
            y = initialRow + heightOfLinks*link;
            //Position is updated
            worm.Rect.get(link).setX(x);
            worm.Rect.get(link).setY(y);
            //Link is shown
            ParallelTransition showPT
                        = showPT(worm.Rect.get(link));
            //Show-transition is booked
            showWorm.getChildren().add(showPT);
```

```
        //Link is booked
        p.getChildren().add(worm.Rect.get(link));
        //Link is grouped to ease hidding
        lastRects.add(worm.Rect.get(link));
    }
    return showWorm;
}


//Worm expresses its movers
private static SequentialTransition expressMovers(Worm worm)
{
    SequentialTransition expressMoversST
                    = new SequentialTransition();
    expressMoversST.setCycleCount(1);
    // d is the number of steps.
    // m is the index of the mover;
    for(int d = 0; d < nSteps ; d++)
    {
        for(int m  = 0 ; m <  worm.numberOfMovers; m ++)
        {
            //Direction is flipped in human design
                if (typeOfProcess.contains("Human design"))
                    worm.Mover.get(m).move.x
                            = - worm.Mover.get(m).move.x;
            //Movers a re expressed
            SequentialTransition exeMovers = exeIf(worm,m);
            exeMovers.setCycleCount(1);
            //Execution of movers is booked
            expressMoversST.getChildren().add(exeMovers);
        }
    }
    return expressMoversST;
}


//Reconstructed fossil  is  hidden after its performance.
    private static ParallelTransition  hideLastRectsST()
```

```
    {

        ParallelTransition hidePT
                    = new ParallelTransition();
        hidePT.setCycleCount(1);
        Duration d2 =   Duration.millis(1);
        //Hide all rects
        for(Rectangle rect: lastRects)
        {
            //Rects are made unvisible
            KeyValue keyValue =
                    new KeyValue(rect
                            .visibleProperty(), false
                            ,Interpolator.LINEAR);
            Timeline b = new Timeline(
                    new KeyFrame(d2,keyValue));
            hidePT.getChildren().add(b);
        }
        return hidePT;
    }


//==============FOSSILS================

//Fossils are sampled and reconstructed
//to measure their functionality
private static  SequentialTransition fossilSampling()
{
    if (print3)
        System.out.println("\nFOSSIL SAMPLING\n");
    console.appendText("\n\n=FOSSIL SAMPLING=\n");
    if (print3)
        System.out.println("Perfection "
            + "= number of links *fitness");
    console.appendText("\nPerfection "
            + "= number of links *fitness");
    if (print3)
        System.out.println("\nFossil number, "
```

```
                + "number of links,"
  + "fitness, perfection\n");
    console.appendText("\nFossil number, number of links, "
  + "fitness, perfection");
    console.appendText("\nTotal number of fossils = "
            + nFossils);
    double fitness;

    SequentialTransition fossilSamplingST
          = new SequentialTransition();
    for(int i = 0; i < nSampledFossils; i++)
    {
        //A fossil is chosen at random
        int f = rd.nextInt(nFossils);
        //Fossil is reconstructed to a worm
        Worm worm =   w.new  Worm(Fossils.get(f));
        //Worm is positioned at the startline
        ParallelTransition showWorm = startLine(worm);
        fossilSamplingST.getChildren().addAll( showWorm );

        //Movers of worm are unfolded
        SequentialTransition expressMoversST
                = expressMovers(worm);

        fitness = fitness(worm);
        //Generation number is drawn
        Timeline timeline = drawFossilNumber(f);
        fossilSamplingST.getChildren().addAll( timeline );
        fossilSamplingST.getChildren().add(expressMoversST);

        //Drawn rects of worm get hidden
        ParallelTransition hideLastRectsST
                =  hideLastRectsST();
        //Booking of hidding process
        //But  the lst fossil is not erased
        if ( i < nSampledFossils - 1)
            fossilSamplingST.getChildren()
                    .add(hideLastRectsST);
```

```
        if (print3)
            System.out.println(f + "\t" + nLinks
            + "\t" + fitness + "\t\t" + fitness*nLinks);

        console.appendText("\n" + f + "\t" + nLinks
            + "\t" + fitness + "\t\t" + fitness*nLinks
            + "\t" + quality);
    }

    double nPerf = nPerfect;
    double nImperf = nImperfect;
    double propPerfect = 100* nPerf/(nPerf + nImperf);
    double propImperfect = 100* nImperf/(nPerf + nImperf);
    console.appendText("\nPerfect fossils = " + nPerfect
            + " (" + propPerfect + ")%" );
    console.appendText("\nImperfect fossils = " + nImperfect
            + " (" + propImperfect + "%)" );

    return fossilSamplingST;
}


private static void memoryUsage()
{
    //Memory usage
    long fm = Runtime.getRuntime().freeMemory();
    long tm = Runtime.getRuntime().totalMemory();
    long um = tm-fm;
    System.out.println("Total  memory = " + tm
        +"\nFree  memory = " + fm
        + "\nUsed memory = " + um );
    //For 20 gens, Used memory = 79601432
}


public static void main(String[] args)
{
        Application.launch(args);
```

```
}


} //End of main class L100 WormsEvoJFX5
```

103, page 302. The following code shows how the general formalism of timelines accepts recurrent movement:

```
//Program L103 ColorfulCircles3

//The program shows how a timeline can host parallel,
//independent  movement
//of many objects.
//It also illustrates how JavaFX allows things to be
//made wonderful besides being functional.
//General formalism of a timeline.

//Original: ColorfulCircles
//Modified  by JR ,
//the EvolJava Community (2014)


/*Note of the Owner:
 *
 * Copyright (c) 2008, 2012 Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under
 * the following license:
 *
 * Redistribution and use in source and binary forms,
 * with or without
 * modification, are permitted provided that the
 * following conditions
 * are met:
```

```
*
*  - Redistributions of source code must retain the
*    above copyright
*    notice, this list of conditions and the following
*    disclaimer.
*  - Redistributions in binary form must reproduce the
*    above copyright
*    notice, this list of conditions and the following
*    disclaimer in
*    the documentation and/or other materials provided
*    with the distribution.
*  - Neither the name of Oracle Corporation nor
*    the names of its
*    contributors may be used to endorse or promote
*    products derived
*    from this software without specific prior written
*    permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT
* HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
* INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
* AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL
* THE COPYRIGHT
* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
* BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
* LOSS OF USE,
* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
* AND ON ANY
* THEORY OF LIABILITY, WHETHER IN CONTRACT,
* STRICT LIABILITY, OR TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
* IN ANY WAY OUT OF THE USE
```

```
 * OF THIS SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 */



package colorfulcircles3;

import static java.lang.Math.random;
import java.util.ArrayList;
import java.util.List;
import javafx.animation.Animation;
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.Group;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.effect.BlendMode;
import javafx.scene.effect.BoxBlur;
import javafx.scene.paint.Color;
import javafx.scene.paint.CycleMethod;
import javafx.scene.paint.LinearGradient;
import javafx.scene.paint.Stop;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Rectangle;
import javafx.scene.shape.StrokeType;
import javafx.stage.Stage;
import javafx.util.Duration;

/**
 * A sample that demonstrates how to draw
 * and paint shapes, apply visual
 * effects, blend colors in overlapping
 * objects, and animate objects.
 *
```

```
 * @see javafx.scene.effect.BlendMode
 * @see javafx.scene.effect.BoxBlur
 * @see javafx.scene.shape.Circle
 * @see javafx.scene.Group
 * @see javafx.scene.paint.LinearGradient
 * @see javafx.animation.Timeline
 */
public class ColorfulCircles3 extends Application
{
    private static final double WIDTH = 495, HEIGHT = 480;
    private Timeline animation;

    private void init(Stage primaryStage)
    {
        Group root = new Group();
        primaryStage.setResizable(false);
        primaryStage.setScene(new Scene(root, 495,480));
        primaryStage.setTitle("Heavenly dance");
        Group layer1 = new Group();
        for(int i=0; i<15;i++)
        {
            Circle circle = new Circle(200,Color
                    .web("white",0.05f));
            circle.setStrokeType(StrokeType.OUTSIDE);
            circle.setStroke(Color.web("white",0.2f));
            circle.setStrokeWidth(4f);
            layer1.getChildren().add(circle);
        }
        // create second list of circles
        Group layer2 = new Group();
        for(int i=0; i<20;i++)
        {
            Circle circle = new Circle(70,Color
                    .web("white",0.05f));
            circle.setStrokeType(StrokeType.OUTSIDE);
            circle.setStroke(Color.web("white",0.1f));
            circle.setStrokeWidth(2f);
            layer2.getChildren().add(circle);
```

```
}
// create third list of circles
Group layer3 = new Group();
for(int i=0; i<10;i++)
{
    Circle circle = new Circle(150,Color
            .web("white",0.05f));
    circle.setStrokeType(StrokeType.OUTSIDE);
    circle.setStroke(Color.web("white",0.16f));
    circle.setStrokeWidth(4f);
    layer3.getChildren().add(circle);
}
// Set a blur effect on each layer
layer1.setEffect(new BoxBlur(30,30,3));
layer2.setEffect(new BoxBlur(2,2,2));
layer3.setEffect(new BoxBlur(10,10,3));
// create a rectangle size
//of window with colored gradient
Rectangle colors = new Rectangle(WIDTH, HEIGHT,
        new LinearGradient(0f,1f,1f,0f,true
            , CycleMethod.NO_CYCLE
            , new Stop(0,Color.web("#f8bd55")),
            new Stop(0.14f,Color.web("#c0fe56")),
            new Stop(0.28f,Color.web("#5dfbc1")),
            new Stop(0.43f,Color.web("#64c2f8")),
            new Stop(0.57f,Color.web("#be4af7")),
            new Stop(0.71f,Color.web("#ed5fc2")),
            new Stop(0.85f,Color.web("#ef504c")),
            new Stop(1,Color.web("#f2660f")))
);
colors.setBlendMode(BlendMode.OVERLAY);
// create main content
Group group = new Group(
        new Rectangle(WIDTH, HEIGHT, Color.BLACK),
        layer1,
        layer2,
        layer3,
        colors
```

```
);
Rectangle clip = new Rectangle(WIDTH, HEIGHT);
clip.setSmooth(false);
group.setClip(clip);
root.getChildren().add(group);

// create list of all circles
List<Node> allCircles = new ArrayList<>();
allCircles.addAll(layer1.getChildren());
allCircles.addAll(layer2.getChildren());
allCircles.addAll(layer3.getChildren());
// Create an animation to parallely and
//independently
//move every circle in allCircles
animation = new Timeline();
//This can be done with lists only.
//Otherwise, use a for loop
allCircles.stream().forEach((circle) ->
{
    int i = 0;
    double xZero  = i*WIDTH;
    double yZero  = 0;
    //Make recurrent transitions, one after the other.
    //From time 10000*i
    //to time  10000*(i+1)
    //1 second = 1000;
    int n = 40;//n < 2
    for(i = 0; i < n; i++)
    {
        //Initial and final times
        Duration dZero = Duration.millis(10000*i);
        Duration dFinal
                = Duration.millis(10000*(i+1));
        //Initial and final positions
        double xFinal  = 0;
        double yFinal  = 0;
        if (i < n-1)
        {
```

```
    xFinal  = random()*WIDTH;
    yFinal  = random()*HEIGHT;
}
else
{
    xFinal  = WIDTH;
    yFinal  = 0;
}


//Translation from intial position
//at initial time
//to
//final position at final time
animation.getKeyFrames().addAll
    (
        // set start position at time dZero
        new KeyFrame(dZero,
            new KeyValue(circle
                .translateXProperty()
                    ,xZero),
            new KeyValue(circle
                .translateYProperty(),yZero)
        ),
        // set end position at time dFinal
        new KeyFrame(dFinal,
            new KeyValue(circle
                .translateXProperty()
                ,xFinal),
            new KeyValue(circle
                .translateYProperty()
                ,yFinal)
        )
    );
//Recurrence is here
xZero  = xFinal;
yZero  = yFinal;
//Feedback
```

```
            System.out.println("xZero = " + xZero);
        }
    });
    //Eternal repeating
    animation.setCycleCount(Animation.INDEFINITE);
    animation.setAutoReverse(true);

    long fm = Runtime.getRuntime().freeMemory();
    long tm = Runtime.getRuntime().totalMemory();
    long um = tm-fm;
    System.out.println("Total  memory = " + tm
        +"\nFree  memory = " + fm
        + "\nUsed memory = " + um);


}

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

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

@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 L103 ColorfulCircles3
```

105, page 307. The StartStopRestart program with the possibility to reset parameters and with a full fledged progress bar indicator.

```
//Program L105 StartStopRestart2
//This program shows how to deal in JavaFX
//with long-running computing tasks
//that may cause unresponsiveness of the application.
//The program provides the possibility
//to start a task, to cancel it,
//and to give a fresh start to the task.
//The program uses the
//JavaFx Service Class that can start, stop and restart
//from the beginning a task.
//But it cannot resume a stopped, cancelled task.
//Long-run task and UI-management are execcuted in isolation
//one from the other.
//A progress bar has been added together
//with the possibility to reset working parameter.

//Next line is for NetBeans. Delete it for Eclipse.
package startstoprestart2;


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.event.Event;
import javafx.scene.Group;
import javafx.scene.Scene;
```

```
import javafx.scene.control.Button;
import javafx.scene.control.ComboBox;
import javafx.scene.control.ProgressBar;
import javafx.scene.control.ProgressIndicator;
import javafx.stage.Stage;



public class StartStopRestart2 extends Application
{

    private static int max = 100;

//Inner class that contains the long task
private static class LongTask extends Service<String>
{

    @Override //createTask() must return a task
    protected Task<String> createTask()
    {

        return new Task<String>()
        {
            @Override //If startButton is clicked
            protected String call()
            {
                 counter++;
                Integer iterations;

                for (iterations = 0; iterations < max;
                        iterations++)
                {
                    System.out.println("Run  " + counter
                            + " Iteration " + iterations);
                    //If cancelButton is clicked
                    if (isCancelled())
                    {
                       break;
```

```
                    }
                    updateProgress(iterations, max);
                }
                return iterations.toString();
            }
        };
    }
}//End of inner class LongTask



//UI management

private static Group p = new Group();
private static  int counter = 0;

@Override
public void start(Stage stage) throws Exception
{

    LongTask service = new LongTask();

    //Progress indicators
    final ProgressIndicator progressIndicator
            = new ProgressIndicator();
    ProgressBar progressBar = new ProgressBar();
    progressBar.setLayoutX(200);

    progressBar.progressProperty()
            .bind(service.progressProperty());


    //Booking
    p.getChildren().addAll(progressIndicator, progressBar);


    Scene scene = new Scene(p, 400, 350);
    stage.setTitle("StartStopRestart");
```

```
//Declaration of buttons
Button startButton = new Button("Start");
Button cancelButton = new Button("Cancel");
Button newRunButton = new Button("newRun");
Button exitButton = new Button("Exit");
//Maximal number of iterations
final ComboBox maxNIterationsCB = new ComboBox();

//Properties of buttons
cancelButton.setOpacity(0);
newRunButton.setOpacity(0);

startButton.setLayoutX(100);
startButton.setLayoutY(100);

cancelButton.setLayoutX(100);
cancelButton.setLayoutY(130);

newRunButton.setLayoutX(100);
newRunButton.setLayoutY(160);

exitButton.setLayoutX(100);
exitButton.setLayoutY(190);

//Listeners
startButton.setOnAction((ActionEvent actionEvent) ->
{

    service.start();
    startButton.setOpacity(0);
    cancelButton.setOpacity(1);
    maxNIterationsCB.setDisable(true);

});


cancelButton.setOnAction((ActionEvent actionEvent) ->
```

```
    {
        service.cancel();
        newRunButton.setOpacity(1);
        cancelButton.setOpacity(0);
        maxNIterationsCB.setDisable(false);

    });



    newRunButton.setOnAction((ActionEvent actionEvent) ->
    {
        service.restart();
        newRunButton.setOpacity(0);
        cancelButton.setOpacity(1);
        maxNIterationsCB.setDisable(true);
        progressBar.progressProperty().unbind();
        progressBar.progressProperty().setValue(0);
        progressBar.progressProperty()
            .bind(service.progressProperty());

    });

    exitButton.setOnAction((ActionEvent actionEvent) ->
    {
        //Finish everything and exit
        System.exit(0);
    });

    //What to do if task is accomplished
    service.setOnSucceeded((WorkerStateEvent t) ->
    {
        System.out.println("done:"
                        + t.getSource().getValue());
        newRunButton.setOpacity(1);
        cancelButton.setOpacity(0);
        maxNIterationsCB.setDisable(false);
```

```
});



//Options
maxNIterationsCB.getItems()
        .addAll( "1000", "2000", "30000",  "50000","100000",
                "500000", "1000000");
maxNIterationsCB.setEditable(false);

maxNIterationsCB.setValue(max);
maxNIterationsCB.setLayoutX(100);
maxNIterationsCB.setLayoutY(50);
maxNIterationsCB.setOnAction((Event ev) ->
{
    String s1 =
    maxNIterationsCB.getSelectionModel()
            .getSelectedItem().toString();
    max = Integer.parseInt(s1);
    System.out.println("Max N Iterations = " + max);
});

//Booking
p.getChildren().addAll(startButton, cancelButton,
                        newRunButton, exitButton,
                        maxNIterationsCB);

//Stage filling and showing
stage.setScene(scene);
stage.show();

}//End of start()


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

```
    }

}//End of main class L105 StartStopRestart2
```

112, page 331. The next code shows that twisted randomness can program worms to climb the screen. Each time a worm is shown to rise the screen, a new link is added to all worms of the next generation. The code illustrates how to implement maximal reuse of JavaFX objects.

```
//Program L112 WormsRandJFXa10
//A population of worms is programmed
//by randomness to
//climb the screen.
//All moves are executed
//but  nondisruptive moves are
//not undone.
//We claim maximal reuse of JavaFX objects.
//All worms have two links at the start.
//One link is added when some worm is shown
//to climb the screen.
//Program is written in JavaFX.



package ejvolu12p;


import static java.lang.Math.pow;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javafx.animation.Interpolator;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.ParallelTransition;
import javafx.animation.SequentialTransition;
```

```
import javafx.animation.Timeline;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Text;
import javafx.stage.Stage;
import javafx.util.Duration;



public class WormsRandJFXa10 extends Application
{
    //Measures of the screen
    private static int xWin = 1200;
    private static int yWin = 850;
    private static WormsRandJFXa10 w = new WormsRandJFXa10();
    //Overall animation
    private static  SequentialTransition nGenerationsST
                    = new SequentialTransition();
    //Number of repetitions or generations
    private static int nGenerations = 20;
    //Generation counter
    private static int gen = 0;

    //The scene
    private static   Group p = new Group();
    //Worms are kept in a list
    private static List<Worm> Worms = new ArrayList<>();
    //Initial position over the screen of the head
    private static int initialRow = 180;
    //Generator of pseudo random numbers
```

```
private static final Random r = new Random();
 //Number of worms
private static  int nWorms = 8;
//Number of links of each worm
private static  int nLinks = 2;
//Each worm has nMovers movers,
//instructions to move.
private static int  nMovers = 3;
//Number of steps allowed for each try
int nSteps = 8;
//Width of the worm = width of each link
private static int width = 6;
//Height of each link
private static int height = 6;
//Array with 9 possible moves
private static Move  vectMoves[ ] = new Move[9];
//Number of ms of translation events.
private static final Duration d =  Duration.millis(200);
//Pane to ease coordinate management:
//It is seem what you  expect
private static StackPane stackPane = new StackPane();
//Text to be drawn
private static  Text genCount = new Text ("");
//Two links cannot occupy the same place
static boolean isOccupied[][]  = new boolean[600][300];
//Is a worm capable of climbing the screen?
private static  int Fitness[] =  new int[21];
//Was climbing good enough?
private static boolean bulleye  = false;
//List of generations at which success was made
private static String record = "";

private static final boolean print1 = false;
private static final boolean print2 = false;


/* Definition of inner classes
```

```
   The Move class:
   A Move says in which direction
   a link of the worm must move.
   */

private  class Move
{
   int x;
   int y;


   //Default initialization
   Move()
   {
       x= 0;
       y= 0;
   }

   // Initialize wth a list of two coordinates
   Move(int a, int b)
   {
       x= a;
       y= b;
   }

   // Copy a move
   Move(Move move)
   {
       x= move.x;
       y= move.y;
   }
}//End of class Move


// Definition of the class Mover:
// A Mover keeps two types of information:
// a) the link of the worm to be moved and
// b) the direction or move to be chosen.
```

```
 //
 private   class Mover
 {
    int link ;
    Move   move;

    //  Movers can acquired information
    // in three ways, they are  overloaded.
    // Form 1: by a list
    private Mover(int l, int mx, int my)
     {
        link= l;
        move.x = mx;
        move.y = my;
     }

    // Form 2, by structured data
    private Mover(int l, Move m)
    {
        link= l;
        move = new Move(m);
    }

    // Form 3, by structured data
    private Mover(Mover mover)
    {
        link= mover.link;
        move =  new Move(mover.move);
    }
  }//End of class Mover



//Rects that belongs in the worms of the first generation
 //are visible and ready to move.
 //Others are hidden.
private static  Rectangle
           createRectangle(boolean visible)
```

```
{
    //Upper left corner at (0,0)
    Rectangle rect = new Rectangle(0,0,width,height);
    //Color is defined as a random
    //mix of red, green and blue
    int rr,gg,bb;
    rr = r.nextInt(226);
    gg = r.nextInt(226);
    bb = r.nextInt(226);
    //The color is prepared
    Color c = Color.rgb(rr,gg,bb);
    //The rect is painted
    rect.setFill(c);
    //Rects of the first generation (gen = 0)
    //are ready to move,
    //otherwise they got hidden
    if ( !visible)   rect.setOpacity(0);
    stackPane.getChildren().addAll(rect);

    return rect;
}


/*  Here we define the class worm.
    A worm is composite: it has links.
    Each link has a rectangular shape
    and  its own color,
    which is defined by three numbers.
    The vector Mover[ ]
     keeps the information
    on how to move its links:
    which  link and in which order
    (in the order that they appear in
    the vector Mover[]), and
    in which direction.
    Each worm keeps record of
    the actual position of each
    one of its links,
```

```
    in the associated ractangle. */


private class Worm
{
    //Number of links
    int numberOfLinks = nLinks;
    //Number of movers = instructions to move
    int numberOfMovers = nLinks;
    //List mover  keeps movers
    List<Mover> Mover = new ArrayList<>();
    //The shape of each link
    List<Rectangle> Rect = new ArrayList<>();

    //Default initialization:
    //worms of the first generation are
    //ready to move. Others get hidden.
    private Worm()
    {
//Rects are created.
        // A color is assigned to each link
        // but different than white
        for(int i = 0; i < numberOfLinks; i++)
            Rect.add(createRectangle(true));
        //Instructions to move at random
for(int m = 0; m < nMovers; m++)
        {
            //Select at random  which link to move:
            int l = r.nextInt(nLinks) ;
            //Select at random  which direction to move on
            Move  move
                = w.new Move (vectMoves[r.nextInt(9)]);
            //Memorization
            Mover.add(w.new Mover(l,move));
        }
    }//End of constructor Worm(int  gen)
```

```
}//End of class Worm


 //Nine basic moves = directions where to move in
static private void moveDef()
{
 final Move move0 =  w.new Move(0,1);
   vectMoves[0]   = move0 ;
 final Move move1 =  w.new Move(1,1);
   vectMoves[1]   = move1 ;
 final Move move2 =  w.new Move(1,0);
   vectMoves[2]   = move2 ;
 final Move move3 =  w.new Move(1,-1);
   vectMoves[3]   = move3 ;
 final Move move4 =  w.new Move(0,-1);
   vectMoves[4]   = move4 ;
 final Move move5 =  w.new Move(-1,-1);
   vectMoves[5]   = move5 ;
 final Move move6 =  w.new Move(-1,0);
   vectMoves[6]   = move6 ;
 final Move move7 =  w.new Move(-1,1);
   vectMoves[7]   = move7 ;
 final Move move8 =  w.new Move(0,0);
   vectMoves[8]   = move8 ;
}


@Override public void start(Stage stage)
    {

        //The scene
        Scene scene = new Scene(p, xWin, yWin);
        stage.setScene(scene);
        stage.setTitle("Links move at random");
        stage.show();
        //Definition of possible moves
        moveDef();
```

```
        genCount.setLayoutX(40);
        genCount.setLayoutY(40);
        p.getChildren().addAll(genCount);
        //Run the overall animation only once
        nGenerationsST.setCycleCount(1);
        //Initial worms are created
        createWorms();
        for(int i = 0; i < nGenerations; i++)
        {
            //Voidness
            voidOccupationMatrix();
            gen = i;
            System.out.println("Gen = " + gen);
            //Generation number is drawn
            Timeline timeline = drawGen(gen);
            nGenerationsST.getChildren().addAll( timeline );
            //Worms are posited at the start line
            startLine();
            //Worms express their information to move
            SequentialTransition oneGenST  =  express();
            nGenerationsST.getChildren().add(oneGenST);
            //Is some worm able to climb?
            assessFitness();
            //Success?
            Integer genI = gen;
            if (bulleye)
            {
                //If climbing was successful, add a new link
                addLink();
                //List of generations with sucess
                record = record + "  " + genI.toString();
            }
            memoryUsage();
        }
    //Film is played
    nGenerationsST.play();
    p.getChildren().add(stackPane);
    System.out.println("Successful generations = "
```

```
            + record);
    memoryUsage();
    }

//Initial worms are created
static public void createWorms()
{
    for (int W = 0; W < nWorms; W++)
    {

        Worm   worm = w.new Worm();



        //worm is added at position W in list Worms
        Worms.add(worm);
    }

}


//The head of all worms are posited at the same initial row
//in vertical position.

static public void startLine()
{
    // Create an animation to parallely and
    //independently
    //move every link
    ParallelTransition positionZeroPT
            = new ParallelTransition();
    for (int W = 0; W < nWorms; W++)
    {
        int  x,y;
        //Auxiliary worm
        Worm   worm  = Worms.get(W);

        //Each link is positioned
```

```
        for(int  link = 0; link < nLinks; link++)
        {
            //Horizontal position
            x = 30 + 100 * W;
            //Vertical position
            y = initialRow + height*link;

            //Address is updated
            worm.Rect.get(link).setX(x);
            worm.Rect.get(link).setY(y);

            //Link gets initial position
            Duration d = Duration.millis(1 );
            //A soft jump is programmed
            TranslateTransition translateTransition =
                new TranslateTransition(d,
                                        worm.Rect.get(link));
            //Repositioning at initial position.
            translateTransition.setToX(x);
            translateTransition.setToY(y);
            //Number of repetitions
            translateTransition.setCycleCount(1);
            //Repositioning is booked
            positionZeroPT.getChildren()
                    .add(translateTransition);
        }
        //Invent new moves
                Worms.set(W,resetMoves(worm));
    }
    //Booking
    nGenerationsST.getChildren()
                        .add(positionZeroPT);

    //Recently added links, if any, are  shown
    if (bulleye)
        {
            for(Worm worm: Worms)
            {
```

```
                ParallelTransition show
                        = showPT(worm
                                .Rect.get(nLinks-1));
                //Show transition is booked
                nGenerationsST.getChildren().add(show);
            }
        }
    //Run garbage collector to clean trash
    System.gc();
}


// To avoid overwriting of worms,
// an occupation record is kept:
// The screen is divided in cells 6x6.
// At the beginning every cell is empty=false.
// Occupancy is recorded in isOccupied[][],
// which is a boolean matrix.
// A boolean  variable can be
// true= occupied,  else false= not occupied.
private static void  voidOccupationMatrix()
{
   //System.out.println("occupationMatrix");
    for(int i = 0; i < 600; i++)
    for(int j = 0; j < 300; j++)
        isOccupied[i][j]= false;
}


//Moves at random are created
private static Worm resetMoves(Worm worm)
{
    //Instructions to move at random
    for(int m = 0; m < nMovers; m++)
    {
        //Select at random  which link to move:
        int l = r.nextInt(nLinks) ;
        //Select at random  which direction to move on
```

```
        Move  move = w.new Move (vectMoves[r.nextInt(9)]);
        //Memorization
        worm.Mover.set(m, w.new Mover(l,move));
    }
    return worm;
}



//Draws the generation number
private static Timeline drawGen(int gen)
{
    Integer genI = gen;
    String message =  "Generation  = " +
                    genI.toString();
    //A JavaFX event committed to print a text is generated
    EventHandler printText =
        (EventHandler<ActionEvent>)
            (ActionEvent t ) ->
                {
                    genCount.setText(message);
                };
    //Presentation of printText must be
    //synchronized with the overall film so,
    //duration d is specified
    KeyFrame keyFrame = new KeyFrame(d, printText);

    Timeline timeline = new Timeline();
    timeline.setCycleCount(1);
    //add the keyframe to the timeline
    timeline.getKeyFrames().add(keyFrame);

    return timeline;
}



//Is move m disruptive?
```

```
//(A worm is and must remain as a whole,
//nevertheless, some elasticity is allowed.)
private static boolean isDisruptive(int m, Worm worm )
{

    // A move is disruptive if the target address
    // is farther than certain distance
    // from the neighboring links.
    boolean isDisruptive = false;
    int  link;
    int  x,y,  x1,y1, x2,y2, x3,y3;
    double d1;

    double  d2;

    //Reads the link to be moved
    link = worm.Mover.get(m).link;
    //Reads the   address of the link;
    x  = (int) worm.Rect.get(link).getX();
    y  = (int) worm.Rect.get(link).getY();
    //The   move to be done is read
    Move move = w.new Move(worm.Mover.get(m).move);

    //The target address is calculated
    x1 = x + width*move.x;
    y1 = y -  width*move.y;

    //The addresses of the two neighboring links are read
    //The disruption is calculated
    //Case: intermediate link
    if ((0 < link) & (link < nLinks-1))
     {
      x2 = (int) worm.Rect.get(link-1).getX();

      y2 = (int) worm.Rect.get(link-1).getY();

      d1 =    pow(  ( pow( (x2 - x1),2)
                          +  pow((y2 - y1),2)),0.5);
```

```
      x3 = (int) worm.Rect.get(link+1).getX();
      y3 = (int) worm.Rect.get(link+1).getY();
      d2 =  pow(  (pow( (x3 - x1),2)
                              + pow((y3 - y1),2)),0.5);


      // Here we decide whether or not
      // a move is disruptive.
      //To augment elasticity, augment the threshold

      if ((d1 > 1.415*width) || (d2 > 1.415*width))
                   isDisruptive = true;

     }

    //First link
    if (link == 0)
    {
        x2 = (int) worm.Rect.get(1).getX();
        y2 = (int) worm.Rect.get(1).getY();
        d1 = pow( (pow( (x2 - x1),2)
                + pow((y2 - y1),2)),0.5);
        if( d1 > 1.1*width) isDisruptive = true;
     }
     //Last link
     if( link == nLinks-1 )
      {
        x2 = (int) worm.Rect.get(nLinks - 2).getX();
        y2 = (int) worm.Rect.get(nLinks - 2).getY();
        d2 = pow( (pow( (x2 - x1),2)
                + pow((y2 - y1),2)),0.5);
        if ( d2 > 1.1*width ) isDisruptive =   true;
       }
      return isDisruptive;
}
```

```
//Is the move m of worm W not disruptive?
//If it is not, execute it.
private static SequentialTransition
                    exeIf( Worm worm, int m)
{
    if (print1)
        System.out.println(" exeIf");
    //Reads the link to be moved
    int link = worm.Mover.get(m).link;
    //Reads position
    int x = (int) worm.Rect.get(link).getX();
    int y = (int) worm.Rect.get(link).getY();

    //The move to be done is read
    Move  move;
    move = worm.Mover.get(m).move;
    //Move is interpreted
    int byX = width* move.x;
    int byY = -height* move.y;
    //Target address is calculated:
    int x1 = x + byX;
    int y1 = y + byY;
    if (x1 < 0 ) x1 = 0;
    if (y1 < 0 ) y1 = 0;
    if (print1)
        System.out.println(
            " move = " + m   + " link = " + link
            + " \n x = " + x + " y = " + y
            + " \nbyX = " + byX  + " byY = " + byY);


    //Occupied cell
    int i = (int)  x / width;
    int j = (int) y / height;
```

```
 //Target cell
int k =(int) x1/width;
int l = (int) y1/height;


//Move is always executed
SequentialTransition a =
        new   SequentialTransition();

TranslateTransition translateTransition =
     new TranslateTransition(d, worm.Rect.get(link));
//Jump is executed
translateTransition.setByX(byX);
translateTransition.setByY(byY);
//Number of repetitions
translateTransition.setCycleCount(1);
a.getChildren().add(translateTransition);



//Is the move disruptive?
boolean   isMoveDisruptive =  isDisruptive(m,worm);
if (print2)
{
    if (isMoveDisruptive)
        System.out.println("move is Disruptive");
    else
        System.out.println("move is not  Disruptive");
}
//Disruptive moves are undone
if  (((isOccupied[k][l])) || ((isMoveDisruptive)))
{
    byX = -byX;
    byY = -byY;
}
else //move is not disruptive -> it is not undone
{
    byX = 0;
```

```
        byY = 0;
        worm.Rect.get(link).setX(x1);
        worm.Rect.get(link).setY(y1);
        isOccupied[i][j]= false;
        isOccupied[k][l]= true;
    }



    //Move is possibly undone
    TranslateTransition translateTransition2 =
         new TranslateTransition(d, worm.Rect.get(link));
    //Jump is programmed
    translateTransition2.setByX(byX);
    translateTransition2.setByY(byY);
    //Number of repetitions
    translateTransition2.setCycleCount(1);
    a.getChildren().add(translateTransition2);

return a;
 }



//Worms express their information to move.
public    SequentialTransition  express()
{
    //Overall process
    SequentialTransition expressST
                    = new SequentialTransition();
     expressST.setCycleCount(1);
     //Worms advance nSteps
    for (int i = 0; i < nSteps; i++)
    {
        //Worms advance one step:
        //To move all links, each worm must execute
        //instructions to move
```

```
        SequentialTransition oneStepST
                        = new SequentialTransition();
        oneStepST.setCycleCount(1);
        //Information is read and decoded
        for (int m = 0; m < nMovers; m++)
        {
            if (print1)
                System.out.println("nMovers = " + nMovers);
            //All worms execute move m
            ParallelTransition moveMPT
                        = new ParallelTransition();
            moveMPT.setCycleCount(1);
            for (Worm worm: Worms)
            {
                //A translation of some link
                //is programmed for worm W
                //according to move m.
                //If it is not disruptive,
                //it is executed
                SequentialTransition tT = exeIf(worm,m);
                tT.setCycleCount(1);
                //Translation is booked
                moveMPT.getChildren().add(tT);
            }
            //The ensamble of translations is booked
            oneStepST.getChildren().add(moveMPT);
        }
    //The movement of worms in one step is booked
    expressST.getChildren().add(oneStepST);
    }
    return expressST;
}



    //One link is added to all worms
    public static void addLink()
    {
```

```
        //All worms are visited
        for (Worm worm: Worms)
        {
            //New unvisible rect is created
            //and added to the worm
            worm.Rect.add(createRectangle(false));
        }
        nLinks++;
    }



    //A hidden rect gets shown
    private static ParallelTransition showPT(Rectangle rect)
    {
        ParallelTransition s = new ParallelTransition();
        Duration d2 =   Duration.millis(1);

        //Rect gets visible
        KeyValue keyValue =
            new KeyValue(rect.opacityProperty(),1,
                Interpolator.LINEAR);
        //KeyValue is  inserted into a timeline
        Timeline b = new Timeline(
            new KeyFrame(d2,keyValue ));
        //Timeline is inserted into a transition
        s.getChildren().addAll(b);
        return s;
    }

// Every worm is assigned a fitness
//The nearer to the top, the fitter.
//If climbing has been significative,
//the complexity of the game rises up one level:
//a true bulleye is reported (to add
//next a new link to all worms).
private static void assessFitness()
{
```

```
    int yOfTail;
    double yOfHead;

    bulleye  = false;
    for(int W = 0; W <  nWorms; W++)
    {
        // We get the y-coordinate of the tail
        yOfTail = (int) Worms.get(W)
                .Rect.get(nLinks-1).getY();
        yOfHead =  Worms.get(W).Rect.get(0).getY();
        //Climbing up is worth  a point
        Fitness[W] = 100*(yWin - yOfTail);
        if(print1)
            System.out.println("Fitness of worm "
                    + W + " = " + Fitness[W]
                    + " head = " + yOfHead);

        //if (headPosition < initialRow - width*nLinks)
        if (yOfHead < initialRow - 2 * height)
        {
            bulleye = true;
            //if (print1)
            System.out.println(
                    "There was advance W = " + W );
        }
    }
}


    private static void memoryUsage()
{
    //Memory usage
    long fm = Runtime.getRuntime().freeMemory();
    long tm = Runtime.getRuntime().totalMemory();
    long um = tm-fm;
    System.out.println("Total  memory = " + tm
        +"\nFree  memory = " + fm
        + "\nUsed memory = " + um );
```

```
}

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


} //End of main class L112 WormsRandJFXa10
```

118, page 340. A bill that updates itself:

```
//Program L118 PropertyDemo2
//A bill that updates itself is presented.
//Numbers are declared as properties.


package ejvolu12p;

import javafx.application.Application;


import static javafx.application.Application.launch;
import javafx.beans.binding.Bindings;
import javafx.beans.binding.NumberBinding;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.IntegerProperty;
import javafx.beans.property.ObjectProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleObjectProperty;
import javafx.event.Event;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
```

```
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;



public class PropertyDemo2 extends Application
{

    //Declarations of fields (global variables)

    //Button for subTotal A is declared
    final Button btnSubTotalA = new Button();
    //Button for subTotal is declared
    final Button btnSubTotalB = new Button();
    //Button for Total is declared
    final Button btnTotal = new Button();
    //Console to write output
    final TextArea console = new TextArea ("");


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




@Override public void start(Stage stage)
{
    stage.setTitle("Bill");
    Scene scene = new Scene(new Group(), 1000, 270);

    //Parameters are declared as properties
    //and are initialized
    final IntegerProperty numA
            = new SimpleIntegerProperty(1);
```

```
  final DoubleProperty priceA
          = new SimpleDoubleProperty(2.3);
  final IntegerProperty numB
          = new SimpleIntegerProperty(6);
  final DoubleProperty priceB
          = new SimpleDoubleProperty(9.75);


  //Subtotals are bound to input data

  //SubTotal1 = numA * priceA
  NumberBinding subTotalA =
    Bindings.multiply(numA, priceA);
  //SubTotal2 = numB * priceB
  NumberBinding subTotalB =
    Bindings.multiply(numB, priceB);

  //Total = subTotalA + subTotalB
  NumberBinding Total =
    Bindings.add(subTotalA, subTotalB);

ObjectProperty<NumberBinding> theA
          = new SimpleObjectProperty<>
              (subTotalA);

  //Number of objects type A is captured
  final ComboBox numACB = new ComboBox();
  //Options
  numACB.getItems()
          .addAll("1", "2", "3", "4", "5");
  //Initial value
  numACB.setValue("1");
  numACB.setEditable(false);
  //Option is picked up
  numACB.setOnAction((Event ev) ->
  {
      //Input is read as string
      String s1 =
```

```
    numACB.getSelectionModel()
            .getSelectedItem().toString();
    //String -> number
    numA.set(Integer.parseInt(s1));
    System.out.println(numA);
    //Updating
    btnSubTotalA.setText("" + subTotalA.getValue());
    btnTotal.setText ("" + Total.getValue());
});



//Number of objects type B is captured
final ComboBox numBCB = new ComboBox();
numBCB.getItems()
        .addAll("6", "7", "8", "9", "10");
numBCB.setValue("6");
numBCB.setEditable(false);
numBCB.setOnAction((Event ev) ->
{
    String s2 =
    numBCB.getSelectionModel()
            .getSelectedItem().toString();
    numB.set(Integer.parseInt(s2));

    btnSubTotalB.setText("" + subTotalB.getValue());
    btnTotal.setText ("" + Total.getValue());
});


//Button for price A is declared
Button btnPriceA = new Button();
//Button is initialized
btnPriceA.setText(""+ priceA.get() );


//Buttons are initialized
Button btnPriceB = new Button();
```

```
    btnPriceB.setText(""+ priceB.get()  );
    btnSubTotalA.setText(""+ subTotalA.getValue() );
    btnSubTotalB.setText(""+ subTotalB.getValue()  );
    btnTotal.setText(""+ Total.getValue()  );

    //Container is declared and filled
    GridPane grid = new GridPane();
    //Vertical and horizontal units
    grid.setVgap(4);
    grid.setHgap(10);
    //Margings
    grid.setPadding(new Insets(5, 5, 5, 5));
    //Row   0
    grid.add(new Label("Units of A: "), 0, 0);
    grid.add(numACB, 1, 0);
    grid.add(new Label("Unitary price of A: "), 2, 0);
    grid.add(btnPriceA, 3, 0);
    grid.add(new Label("SubTotal A: "), 4, 0);
    grid.add(btnSubTotalA, 5, 0);

    //Row 1
    grid.add(new Label("Units of B: "), 0, 1);
    grid.add(numBCB, 1, 1);
    grid.add(new Label("Unitary price of B: "), 2, 1);
    grid.add(btnPriceB, 3,1);
        grid.add(new Label("SubTotal B: "), 4, 1);
    grid.add(btnSubTotalB, 5, 1);

    //Row 2
    grid.add(new Label("Total = "), 4,2);
    grid.add(btnTotal, 5, 2);

    //Booking
    Group root = (Group)scene.getRoot();
    root.getChildren().add(grid);
    stage.setScene(scene);
    stage.show();
}
```

```
}//End of main class L118 PropertyDemo2
```

129, page 353. The next code shows how to use JavaFX inbuilt properties to control the separation in within two rectangles by means of a slider:

```
//Program L129 SliderBinding2
//A slider is used to control the distance
//in within two rectangles.
//The code lacks listeners and  timelines.
//The code relies on inbuilt JavaFX properties.

package ejvolu12p;



import java.util.Random;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.util.Duration;


public class SliderBinding2 extends Application
{

//Slider declaration and symbolic initialization
final Slider distanceSlider = new Slider(0, 200, -1);


//Message for the User
```

```
final Label promptMessage
        = new Label("Drag the thumb of the slider!");
final static Color messageColor = Color.BLUE;

//Width of rects
private static final int width = 63;
//Height of each rect
private static final int height = 63;
//Generator of pseudo random numbers
private static final Random rd = new Random();


//Duration of graphical events
private static final Duration d
            = Duration.millis(100);

//A random color is synthesized
private static Color randomColor()
 {
    //Color is defined as a random
    //mix of red, green and blue
    int r,g,b;
    r = rd.nextInt(226);
    g = rd.nextInt(226);
    b = rd.nextInt(226);
    //The color is prepared
    Color c = Color.rgb(r,g,b);
    return c;
 }


//A rect is created.
private static  Rectangle
           createRectangle(int x, int y)
{
    //Upper left corner at (x,y)
    Rectangle rect = new Rectangle(x,y,width,height);
    //The color is prepared
```

```
    Color c = randomColor();
    //The rect is painted
    rect.setFill(c);
    return rect;
}



@Override
public void start(Stage stage)
{
    Group root = new Group();
    Scene scene = new Scene(root, 700, 200);
    stage.setScene(scene);
    stage.setTitle("Slider controls separation "
            + "in within two rects."
            + " Drag thumb of the slider");

    //Setting of parameters of the slider
    distanceSlider.setMin(0);
    distanceSlider.setMax(200);
    distanceSlider.setValue(0);
    distanceSlider.setShowTickLabels(true);
    distanceSlider.setShowTickMarks(true);
    distanceSlider.setMajorTickUnit(100);
    distanceSlider.setMinorTickCount(5);
    distanceSlider.setBlockIncrement(10);
    //Message is given a color
    promptMessage.setTextFill(messageColor);

   //Slider is given a position
    distanceSlider.setLayoutX(273);

    //Two rects are declared, initialized  and positioned
    Rectangle rect1 =  createRectangle(217,55);
    Rectangle rect2 =  createRectangle(280, 55);
    root.getChildren().addAll(distanceSlider, rect1, rect2);
```

```
    //A helping double property is declared
    DoubleProperty value = new SimpleDoubleProperty();
    //Helping property gets bound to the slider reading
    //value = 0.625*SliderReading + 280
    value.bind(distanceSlider.valueProperty()
                    .multiply(0.625).add(280));
    //X-position of rect2 is bound to value
    rect2.xProperty().bind(value);

    stage.show();

}

public static void main(String[] args)
{
    launch(args);
}
}//End of main class L129 SliderBinding2
```

142, page 365. Natural evolution includes randomness, so it can explain anything. The problem is that some events need by far more resources than those allowed by the whole Universe along its complete history. Do we have here one of those cases? The answer is uncertain for the Author. Nevertheless something is quite sure: were evolution the developer of Program L140 WormsEvoJFXa10 with all its duplications, a huge mountain of bugged, poorly operative programs and variants developed during million of years should had left tracks along a would-be evolutionary processes. That is not the case here: the code was found to be highly evolvable in this regard and so, some few hundreds of bugs appeared along the restructuring and were easily corrected by the Author during some 6 hours of work.

143, page 365. To achieve a complete and successful virtual animation, the JavaFX code that was in LongTask was put aside as a method:

```
//Program L143 WormsEvoJFXa11
```

```
...
...

private static void longTaskFinish()
{

    console.appendText("\n\n====OUTPUT===== ");
    console.appendText("\n\n=SUCCESSFUL GENERATIONS= ");
    Integer size = bullseyes.size();
            for(Integer index = 0; index < size; index++)
            {
                console.appendText("\n"
                    + bullseyes.get(index));

            }
    //Data for newLinkChart is provided
    newLinkChart.setData(getNewLinkChartData());
    newLinkChart.setTitle("Cost of complexity under \n"
            + typeOfProcess);
    //Worms are hidden
    if (gen < nGenerations -1) allST.getChildren().add(hide());


    //Fossil study
    fossilStudy();


    //ScreenShot of worms is enabled
    wormsShotBt.setOpacity(1);
    //Data for qualityChart of fossils is provided
    qualityChart.setData(getQualityChartData() );
    qualityChart.setTitle("Quality of fossils under \n"
            + typeOfProcess);


    ObservableList<PieChart.Data> qualityPieChartData =
        FXCollections.observableArrayList(
        //1 is added in case nPerfect = 0
```

```
        new PieChart.Data("Perfect", 10000*nPerfect+1),
        new PieChart.Data("Imperfect", 10000*nImperfect));
    qualityPieChart.setData(qualityPieChartData);

    qualityPieChart.setTitle("Quality of fossils under \n"
            + typeOfProcess
    + "\nPerfect = " + nPerfect
    + "    Imperfect = " + nImperfect);


}
```

This longTaskFinish() method was called from the next piece of code that was inserted in line 2529 below the listener of the goBt button:

```
    //What to do if task is accomplished
    goTask.setOnSucceeded((WorkerStateEvent t) ->
    {
        System.out.println("done:"
                        + t.getSource().getValue());
        longTaskFinish();
        //goTask.setOpacity(1);
        //cancelButton.setOpacity(0);
        //maxNIterationsCB.setDisable(false);
    });
```

146, page 366. To fix the bug that draws a little square near to the top at right hand of the screen, JavaFX worms were created invisible but were make visible at their appropriate time. When trying to run the program for other number of worms different than 8, the program collapsed. This bug got fixed when loops for Worms were governed by an ordinary loop for arrays instead if using a parallel loop appropriate for lists. Our code can be found in program L146 WormsEvoJFXa13, not included.

Bibliography

[1] EFXCLIPSE 2014 Efxclipse addons
http://efxclipse.bestsolution.at/install.html
Verified 22/X/2014 vi

[2] IRINA FEDORTSOVA, ORACLE 2014 Concurrency in JavaFX
http://docs.oracle.com/javafx/2/threads/jfxpub-threads.htm
Verified 23/X/2014 302

[3] JONI GORDON, ORACLE 2014 Working with Layouts in JavaFX.
Release 2.1 E20476-05 Oracle, June 2013.
http://docs.oracle.com/javafx/2/layout/jfxpub-layout.htm
Verified 23/X/2014 363

[4] ORACLE 2014 JDK 8u25 with NetBeans 8.0.1
http://www.oracle.com/technetwork/java/javase/downloads/jdk-netbeans-jsp-142931.html
Verified 22/X/2014 v

[5] ORACLE 2014 Java SE Development Kit 8 Downloads
http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
Verified 22/X/2014 21

[6] ORACLE 2014 JavaFX: Working with JavaFX UI Components (Java SE Documentation).
http://docs.oracle.com/javase/8/javafx/user-interface-tutorial/ui_controls.htm#JFXUI336
Verified 22/X/2014 189