JAVA FOR THE STUDY OF EVOLUTION

VOLUME IV

CONVICTIONS

José del Carmen Rodríguez Santamaría
The EvolJava Community

Contents

1 Starting up **1**
- 1.1 Installing Java and Eclipse . . . 1
- 1.2 Purposes and projects . . . 2
- 1.3 Our first program . . . 4
- 1.4 Review . . . 5

2 Cubes **7**
- 2.1 3d graphics . . . 7
- 2.2 A cube . . . 9
- 2.3 To become an expert . . . 34
- 2.4 Touching at consciousness . . . 34
- 2.5 Conclusion . . . 44

3 Drawing a hand **45**
- 3.1 Drawing a 3d hand . . . 45
- 3.2 The hand . . . 79
- 3.3 Conclusion . . . 146

4 Rationalizing creationism **147**
- 4.1 Let us reinvent gravity . . . 147
- 4.2 Creationism vs chance . . . 160
- 4.3 Conclusion . . . 218

5 The hand of a tiny god **221**
- 5.1 Detecting the mouse . . . 221
- 5.2 Pop-up menus . . . 227
- 5.3 A hand with reflexes . . . 251
- 5.4 Almost human gods . . . 289
- 5.5 Conclusion . . . 291

6 Rationalizing the evolutionary theory 293
6.1 A vision over evolution 294
6.2 Evolution is science 298
6.3 The morphs of Dawkins 299
6.4 Evolution produces art 358
6.5 Conclusion 401

7 Evolution is quantitative 403
7.1 Chance has a personality 404
7.2 Chance vs Retro-engineering 405
7.3 The evolutionary theory 432
7.4 Conclusion 458

8 Evolution vs complexity 461
8.1 What is complexity? 464
8.2 SAT as a model of complexity 466
8.3 Evolution vs the disjunctive subclass 475
8.4 A nice challenge 512
8.5 In hindsight 590
8.6 Conclusion 592

9 Two visions over evolution 593
9.1 Our experience 594
9.2 Our claim 597
9.3 Example: vertebrates 599
9.4 Conclusion 651

10 Conclusion 653
10.1 Beliefs 653
10.2 World views 656
10.3 The role of Java 660
10.4 Evolution and computing science 663

11 Glossary 665

12 Answers to exercises 669

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.

This is the fourth volume: we present some discussions that can be illuminated with Java and that we hope are directly related to the question about how a person rationalizes his or her belief in evolution or in a Creator. In regard with Java, we include a soft introduction to 3D (three dimensional) graphics and also we dress previous material in GUI (graphic user interfaces). We show how Java can help everybody to study his or her beliefs but we find that it is direct and merciless with scientific proposals: our studies allow us to claim that the modern evolutionary theory is disguised lamarckism.

Bogotá, Colombia, *José Rodríguez*
Dec 2009

Introduction

Evolution is an important concern for most persons and they use a complex mix of instinct + **convictions** + facts + logic to judge it. We do know that the overall consequence has been a terrible explosion as evidenced by the evolutionist & creationist struggle.

Our purpose in this volume is to show that we can provide a proper environment in which the energy of instinct and the power of **convictions** can be used to forge the history of the universe in a very constructive way. Our strategy is simple and obeys the following directive: the existence of instincts and the capability of carrying convictions are wonderful facts of the human nature that can be the object of admiration and of scientific study. More to the point, the problems that we find in the study of evolution are so daunting that without the endurance given by a conviction they would remain largely unexplored. So, we firmly believe that convictions are the very source of energy to work very hard pursuing elusive and complex targets. If you do not have a conviction, get one: you will need it if you want to survive in this deal. Believe or perish!

We have tried to make the material self-contained and easy to handle for newcomers, so the needed material from previous volumes has been pasted directly or sometimes recycled when needed. So, our *chapter 1, Starting up* is how to install Java to run an existing program. The background philosophy in our community is that Java adds salt to the discussion of problems.

Chapter 2, Cubes, contains the first exercise in three dimensional graphics: the code for a cube that flies while rotates. Cubes lend themselves for the study of visual illusions, a phenomenon that we consider related to the problem of the arising of beliefs: we contend that beliefs are fabricated by the brain by the very same mechanism that creates visual illusions and so they are biology induced. Cubes are also used in *chapter 3, Drawing a hand,* as the building block to model a left hand that can close its fingers.

Chapter 4, Rationalizing creationism, shows that it is elementary to reinvent and justify creationism as a viable alternative against chance: all we need is to look at our left hand. In *Chapter 5, The hand of a tiny god,* we learn how to

implement the magic of the mouse to design a program that simulates a hand with reflexes. We propose a relation with theology that helps us to understand why the God of Moses is believed to be almighty.

Four additional chapters are dedicated to the study of evolution as a belief. First we see why evolution is easily reinvented and why it is preferred to chance. Then we invite evolution to measure its strength in a battle against complexity. Our work encourages us to belief that evolution can solve every kind of complex problems but a price must be paid: the path along any developing process must be paved with every kind of half cooked products. One implication is that we can easily distinguish between evolution by random mutation and lamarckism, in which improving mutations readily happen without a struggle with less fitted proposals. Our final conclusion is that the evolutionary explanations of modern science are indeed explanations proper of lamarckism.

Chapter 1

Starting up

`Running a first program.`

1 ***Purpose.*** *To enable the use of this material by beginners, we see here how to acquire and install **Java** and **Eclipse** and to run an already existing Java program. If you want to solve the exercises and to enjoy the material to the full, you must go through the first volume of this series, where you can learn the fundamentals of Java, science and evolution.*

1.1 Installing Java and Eclipse

Sun Microsystems composed Java as a great symphony and distributes it without fees. Please, use Google to contact that company and to load and install Java on your desktop. You can prompt Google with `Java Sun Microsystems` or you can contact Sun Microsystems directly (Sun Microsystems, 2009). Java comes with a lot of tools, which together form a run time environment. Follow the instructions to install Java and forget about it because Java is accessed by the User through an interface, ECLIPSE, which is also freely distributed. Please, use Google again to localize Eclipse, next load and install it. Else, contact Eclipse directly (Eclipse, 2009). Eclipse offers various options, you can try

Eclipse Classic.

The duo JAVA + ECLIPSE will be referred simply as Java.

The first window that ECLIPSE opens for you is the WELCOME window. In *Tutorials* → *Create a Hello world application*, follow the instructions and try to create your first program. If you fail, you might try again. To that end, you can rescue the Welcome window from the help at the main menu of ECLIPSE. Else, use our methodology, which comes below.

2 ***Guarantee.*** *To your disposal, we present many programs that are guaranteed to work perfectly on Java Runtime Environment jre1.6 and Eclipse SDK 3.5.1. Memory might become important: we used 2 GB of RAM. Otherwise, express your discomfort to*

jose@evoljava.com

Please, visit http://www.evoljava.com the site of the evolJava community.

Let us learn how to copy a program into Eclipse that it could be run. The first step is opening a project.

1.2 Purposes and projects

In Java, before copying or designing a program, one must make clear what the purpose one has in mind is. Our present purpose is to study volume 4, so the name of our project is Vol4. Then, we must communicate to Java our intention.

3 ***Opening a project***

To open a project, please, follow the sequence of options beginning from the main menu: $File \rightarrow new \rightarrow project \rightarrow Java \rightarrow Project \rightarrow Next$. A window will pop up for you with an invitation to fill the relevant data about your project. Give it a name: Vol4. Click Next. In the new window, click Finish.

Now that we have a purpose, a project, we can learn to append a program to our project into **ECLIPSE**, the interface of Java with the User.

4 ***Copying a program***

The simplest programs in Java have the next basic structure, where comments begin with the marker // and instructions have no initial marker. Officially, a program in Java is called **class**.

```
// This is the structure of the simplest program.
//The name in Java of this program is structure
public class structure
{
 //The next is the head of the program
public static void main(String[] args)
```

```
{
   //Here are the instructions
}
}
```

Programs are not typed except when they are created. Instead, they are transcribed by a copy-paste procedure, for which you have three possibilities:

1. To copy each program from the present pdf document. In this case, you must resolve a little problem: the copy operation also copies navigating signals of the document, headers or page numbers, so they must be identified and deleted. The identification is done automatically by Eclipse after the paste operation is complete: extraneous lines of text appear with a red mark at the right margin. They must be deleted.

2. To get the Latex source of the present document, which can be opened with any text processor. You can copy any program from it without suffer anyhow. Additionally, you are invited to use our Latex source to devise your own material for yourself or for your pupils. In general, feel free to do anything good for forming a healthy and strong **evolJava** community.

3. To get a file, in plain text, with all programs, from which you can copy the desired program immediately. To get the file, go to

   http://www.evoljava.com

Now, search the desired program in the Latex source or in the file with the programs and:

1. Highlight the text that you want to copy.
2. Copy the highlighted text to the clipboard.
3. Open Eclipse, if it is not.
4. We must acquire now the right to append a program into a project. In JAVA a program is called class. So, let us click on $File \rightarrow new \rightarrow class$ and a window will pop up, whose name is New Java Class. Let us fill in the name of the source project: Vol4 (else browse for it). Let us give a name to our new class: printing. If desired, we could activate the Public static void main property. Click Finish. The editor window will filled in the seed of a new program. In its upper bar, one can see the name that we gave to the class, say printing.java.
5. We can now copy and run our first program, as explained below.

1.3 Our first program

We have below a a program whose purpose is to print a message to the console. The **console** is a special window where Eclipse reports results and errors. If whatever cause hides the console, one can restore its visibility in any one of the two following options:

a) In the main menu follow the path: *Windows* → *ShowView* → *Console*.

b) Click Alt+Shift + Q, then click C.

5 ***Printing in the console.*** *The next code contains a program, whose purpose is to print a message to the console.*

```
// Program   printing, D5
// This is my first program or class.
// It prints a message to the console.
// The name of this class is printing.
// The  name of this class must coincide
// with the name with which the class was
// declared and opened as a  new Java class.
// The editor distinguishes upper case from lower case.

public class printing
{

 //This is the main method
 public static void main(String[] args)
 {
  // This is a comment.

  // Comments help to understand programs.

  // The next line contains an instruction
  // that prints a message on the console:
     System.out.println("I am starting up! ");
 }
}
```

6 ***Running our program***

To copy into Eclipse and run this class, please, do the following:

1. Copy this program from the Latex source into the clipboard.

2. Clear from the editor or main window of Eclipse the code that was automatically written by Eclipse for you. Paste the enclosure of the clipboard to the blanked document. Warning: Eclipse for Windows is not based on Windows but on DOS: the command for **delete** is control + x, for **copy** is control + c, for **pasting** is control + v, for **select all** is control + a.

3. Save the new program: click on the corresponding icon.

4. Right-click on the editor window that contains the text of the new class.

5. In the sub-menu that popped up, click on *Run as* $\rightarrow$ *Java* $\rightarrow$ *Application*.

We shall see the output of our first program that writes in the console the phrase: *I am starting up!*

7 ***License.*** *Dear Reader: as part of the exercises, you must modify the programs. So, you have the* ***license*** *to use all programs written in this book in whatever form you want even to get rich and very specially to modify them as you are asked in the exercises. This freedom might help the formation of a strong evolJava community. Please, visit http://www.evoljava.com*

1.4 Review

We have learned how to acquire and install Java and its companion Eclipse. We have created a project to which we appended our first program and we have run it. With minor variations, any other program can be run just like this.

Chapter 2

Cubes

`From 3d (three dimensional) graphics to consciousness`

8 ***Purpose.*** *We devise our own system of three dimensional graphics and test it with a cube that can be rotated and translated and that can be observed in vivo. This simple exercise opens the door for an interesting discussion in relation with consciousness and the nature of man: what a human being calls reality is no more than an artificially synthesized world that might be an unfair copy of the true reality and that can be different from person to person and even from moment to moment.*

2.1 3d graphics

The fundamental idea of the technology of the 3d graphics is as follows: one draws an object in a three dimensional space that resides in the memory of the computer. This object can be modified, say, it can be rotated. To see a 3d object, one calculates the shade or projection of the object along a given axis and then one renders the projection over the display: the only thing one can see is that 2d shade. Holographic 3d imagines might be synthesized in a future.

9 ***Technical detail***

Everyone has the right to understand the magic behind 3d graphics, which is entirely based on linear algebra. Let us review at this moment the main ideas of the procedure to calculate rotations and projections (José-José, 2009).

The matrix of a rotation with counterclockwise angle θ around the z axis is:

$$R = \begin{pmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The role of z-axis can be played by any axis w if only we use an orthonormal basis B, whose z-axis is precisely w. Actually, w is generated by a point along it, say, $w = (a,b,c)$. A plane that passes through the origin with normal vector w is $ax + by + cz = 0$. In this plane, we found a vector b_1, different than (0,0,0), say, $(b,-a,0)$, or $(c,0,-a)$ or $(0,c,-b)$. Once we have the vector in the plane, we form the crux product between w and b_1. We form a basis with these three vectors, which is next normalized: we have basis B that generates an orthogonal matrix: its transpose is the inverse.

The matrix of rotation around axis w of angle θ is then:
$R = BRB^T$.

Example: Let us find the matrix of rotation by an angle $\theta = \pi/3$ around the axis that passes through $(0,0,0)$ and $(1,1,1)$:

The plane that passes through the origin with normal vector (1,1,1) is
$x + y + z = 0$. A point in that plane could be $(1,0,-1)$. A third element is $(1,0,-1) \times (1,1,1) = (1,-2,1)$.

The set $\{(1,0,-1),(1,-2,1),(1,1,1)\}$ forms a basis that could be normalized to get an orthonormal basis that in matrix notation is:

$$B = \begin{pmatrix} \sqrt{2}/2 & \sqrt{6}/6 & \sqrt{3}/3 \\ 0 & -2\sqrt{6}/6 & \sqrt{3}/3 \\ -\sqrt{2}/2 & \sqrt{6}/6 & \sqrt{3}/3 \end{pmatrix}$$

The inverse is the transpose

$$B^{-1} = B^T = \begin{pmatrix} \sqrt{2}/2 & 0 & -\sqrt{2}/2 \\ \sqrt{6}/6 & -2\sqrt{6}/6 & \sqrt{6}/6 \\ \sqrt{3}/3 & \sqrt{3}/3 & \sqrt{3}/3 \end{pmatrix}$$

The determinant of B is $+1$, so we can think of B as a natural basis around $(1,1,1)$. Henceforth, the rotation of $\pi/3$ around axis $(1,1,1)$ as seen form basis B is:

$$R_B^B = \begin{pmatrix} cos(\pi/3) & -sin(\pi/3) & 0 \\ sin(\pi/3) & cos(\pi/3) & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1/2 & -\sqrt{3}/2 & 0 \\ \sqrt{3}/2 & 1/2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The matrix of rotation around (1,1,1) is now:

$$R = BR_B^B B^T = \begin{pmatrix} \sqrt{2}/2 & \sqrt{6}/6 & \sqrt{3}/3 \\ 0 & -2\sqrt{6}/6 & \sqrt{3}/3 \\ -\sqrt{2}/2 & \sqrt{6}/6 & \sqrt{3}/3 \end{pmatrix}$$

$$\begin{pmatrix} 1/2 & -\sqrt{3}/2 & 0 \\ \sqrt{3}/2 & 1/2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \sqrt{2}/2 & 0 & -\sqrt{2}/2 \\ \sqrt{6}/6 & -2\sqrt{6}/6 & \sqrt{6}/6 \\ \sqrt{3}/3 & \sqrt{3}/3 & \sqrt{3}/3 \end{pmatrix}$$

Example: Let us find the matrix of projection along the axis generated by the point (a,b,c).

The projection of point (x,y,z) along the z-axis over the plane XY is (x,y). So, the matrix of this projection is

$$J = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

The role of the z-axis can be played by anyone axis $w = (a,b,c)$. We found an orthonormal basis B whose z axis is w as in the previous example. Then, the matrix of projection along axis w is:

$P = BJB^T$.

Example, the projection along the axis $w = (1,1,1)$ is

$$P = \begin{pmatrix} \sqrt{2}/2 & \sqrt{6}/6 & \sqrt{3}/3 \\ 0 & -2\sqrt{6}/6 & \sqrt{3}/3 \\ -\sqrt{2}/2 & \sqrt{6}/6 & \sqrt{3}/3 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \sqrt{2}/2 & 0 & -\sqrt{2}/2 \\ \sqrt{6}/6 & -2\sqrt{6}/6 & \sqrt{6}/6 \\ \sqrt{3}/3 & \sqrt{3}/3 & \sqrt{3}/3 \end{pmatrix}$$

2.2 A cube

The philosophy of 3d-graphics is captured at once by looking at a rotating cube, so this is as a rule the first exercise in every course of 3d graphics. The corners of the cube are numbered as in the next figure:

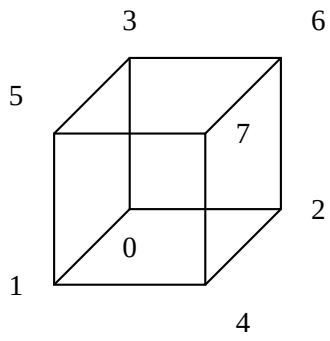


Figure 2.1. The cube has numbered corners, whose coordinates are as follows:

$$0 \to (0,0,0),$$
$$1 \to (1,0,0),$$
$$2 \to (0,1,0),$$
$$3 \to (0,0,1),$$
$$4 \to (1,1,0),$$
$$5 \to (1,0,1),$$
$$6 \to (0,1,1),$$
$$7 \to (1,1,1)$$

10 ***The code*** *The code for a 3d cube that rotates follows:*

```
/* Program D10, cube
   A 3d cube that rotates.
   */

import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;

public class cube  extends Applet
 {
  private static final long serialVersionUID = 1L;


  /*
   Our basic cube is the unitary cube at the origin.
   It is composed of 8 points or vertexes = points3d
   and 12 edges.
   Points are numbered from 0 to 7.
```

```
 These numbers are used in
 edges, which  records the pairs of
  points that are joined by a line. */


private static   double points3d[][]=
                          { {0,0,0},
                        {1,0,0},
                        {0,1,0},
                        {0,0,1},
                        {1,1,0},
                        {1,0,1},
                        {0,1,1},
                        {1,1,1}
                       };

private static int edges[][] =
                             {{0,1},
                          {0,2},
                          {0,3},
                          {1,4},
                          {1,5},
                          {2,6},
                          {2,4},
                          {3,5},
                          {3,6},
                          {4,7},
                          {5,7},
                          {6,7}};

/* A 3d cube exists only in the imagination:
 * it is not seen.
 * One sees a shade or projection of it over the  screen.
 * The shade is defined by 2d points. The edges of
 * the 2d shade are inherited from the edges of the
 * 3d cube.
 */
private static int shade2d[][] = new int[8][3];

//All cubes are derived from the basic cube  by rotations
```

```
 //and translations
 private static double cube[][]  = new double[8][3];
 private static double  cubeProj[][] = new double[8][3];

 private static double axisRot[] = new double[3];
 private static double axisProj[]  = new double[3];
 private static double theta  = 0;
 private static double deltaTheta;

 private static double basisRot[][], basisProj[][]
                                        = new double[3][3];
 private static double matrixRotz[][],
                       matrixRot[][],
                       matrixProj[][] = new double[3][3];
 //The matrix of a projection along the z-axis
 private static   double J[][]=  { {1,0,0},
                                  {0,1,0},
                                  {0,0,0}
                                 };
 private static  double scale;
 private static int delay ;
 private static int nScenes ;

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

 //The multiplication of two matrixes
 private static  double[][] matrixMult(double p[][],
                                 double q[][])
 {
 double[][] helpM = new double[3][3];

 for(int i=0; i<3; i++)
```

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

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

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

 //------------------------------

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

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

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

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

  //The first vector is the crux product of the other 2.
  private static  double[] firstVector(double[] v,
                                  double[] w)
  {
  double a,b,c;
```

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

  }


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

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

return helpM;
  }

 //------------------------------


//Rotation about the z-axis.
private static  double[][] mRot(double theta)
 {
  double[][]  mRotz = new double[3][3];
  mRotz[0][0] = Math.cos(theta);
```

```
 mRotz[0][1] = Math.sin(theta);
 mRotz[0][2] = 0;
 mRotz[1][0] = -Math.sin(theta);
 mRotz[1][1] = Math.cos(theta);
 mRotz[1][2] = 0;
 mRotz[2][0] = 0;
 mRotz[2][1] = 0;
 mRotz[2][2] = 1;
 return mRotz;
}

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

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

 //The 3d cube is projected into a 2d plane.
 //The resultant figure must be rendered to the screen.
 //A projection is done along  an axis that crosses
 //through 0.
 //Our projections are observed from infinite:
 //we use no perspective corrections.
 private  static  double[][] mProj(double w[] )
```

```
  {
  basisProj = basisB(w);
  double[][]  helpM = new double[3][3];
  helpM = matrixMult(J,matrixTranspose(basisProj));
  helpM = matrixMult(basisProj,helpM);
  //printMatrix(helpM, "Proj");
  //Test: Proj*Proj must be equal to Proj
  /*helpM = matrixMult(helpM,helpM);
   printMatrix(helpM, "Proj* Proj");*/
  return helpM;
  }

  //The cube c is rotated around axis v an angle theta
  private  static   double[][] rotCube(double c[][],
                        double v[], double theta )
  {
double[]  helpV = new double[3];
double cc[][] = new double[8][3];
matrixRot = mRotation( v,theta);
//printMatrix(matrixRot, "matrixRot");
//The 8 points of the cube must be rotated
for(int i=0; i<8; i++)
{
 for(int j=0; j<3; j++)
   helpV[j] = c[i][j];
 helpV = matrixVector(matrixRot, helpV);
 for(int j=0; j<3; j++)
 cc[i][j] = helpV[j];
}
//printCube(cc, " after rot");
return cc;
  }

  //The cube c is projected along axis w
  private static double[][] projCube(double c[][],
                                double w[])
  {
double[]  helpV = new double[3];
double cc[][] = new double[8][3];
matrixProj = mProj(w);
```

```
for(int i=0; i<8; i++)
{
 for(int j=0; j<3; j++)
   helpV[j] = c[i][j];
 helpV = matrixVector(matrixProj, helpV);
 for(int j=0; j<3; j++)
 cc[i][j] = helpV[j];
}
return cc;
  }

  //The rendering procedure accepts integer values
  private  static  int[][] round(double c[][] )
  {
  int[][] helpM = new int [8][3];
  for(int i=0;i<8;i++)
   for(int j=0; j<3; j++)
  helpM[i][j] = (int) Math.round(c[i][j]);
  return helpM;
  }

  private static void printCube(double c[][], String a)
  {
   System.out.println("This is the cube" +a);
   for(int i=0;i<8;i++)
    System.out.println(c[i][0] + " " + c[i][1]+ " "
                                      + c[i][2]);
  }

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

  private void printShade()
  {
   System.out.println("This is the shade");
```

```
   for(int i=0;i<8;i++)
    System.out.println(shade2d[i][1] + " " + shade2d[i][2]);
  }



  private static  void initialize()
  {
scale = 50;
for(int i=0; i<8;i++)
 for(int j=0; j<3;j++)
  points3d[i][j] = scale*points3d[i][j];
//printCube();
//The 3d-graphic is rotated along the axisRot
//an angle theta that is incremented by deltaTheta
theta = 0;
deltaTheta = 0.01;
axisRot[0] = 1;
axisRot[1]=1;
axisRot[2]=1;
  //The 3d-graphic is projected over the screen
  // which is perpendicular to the x-axis.
axisProj[0] = 1;
axisProj[1]=0;
axisProj[2]=0;
nScenes = 500;
//delay between two scenes
delay =  10;
  }

public void paint (Graphics g)
{
 System.out.println("Cube rotates");
 initialize();
 int a,b,c,d=0;
 //backgound color
 g.setColor(Color.pink);
 g.fillRect(0,0,200,200);
 //Loop for the scenes
```

```
 for(int k = 0; k<nScenes; k++)
 {
  theta = theta + deltaTheta;

  cube = rotCube(points3d,axisRot,theta);
  //printCube(cube, " after rotation");
  cubeProj = projCube(cube,axisProj);
  //printCube(cubeProj, " after projection");
  shade2d =  round(cubeProj );
  //printShade();
  //draw cube
  g.setColor(Color.blue);
   for(int i=0;i<12;i++)
   {
a =   100-shade2d[ edges[i][0]  ] [1];
b =   100-shade2d[ edges[i][0]  ] [2];
c =   100-shade2d[ edges[i][1]  ] [1];
d =   100-shade2d[ edges[i][1]  ] [2];

/*System.out.println(edges[i][0] + " " + edges[i][1]);
System.out.println( "i= " +i + " " +a + " "
                                  +b+ " " + c+" " +d);*/
g.drawLine(a,b,c,d);
   }
   //delay
   if (k < nScenes-1)
    {
     try
      {
   Thread.sleep( delay );
      }
      catch (InterruptedException e)
      {
   e.printStackTrace();
      }

     //erase cube: we repaint the cube in
     //the background color.
     g.setColor(Color.pink);
```

```
    for(int i=0;i<12;i++)
     {
  a =   100-shade2d[ edges[i][0]  ] [1];
  b =   100-shade2d[ edges[i][0]  ] [2];
  c =   100-shade2d[ edges[i][1]  ] [1];
  d =   100-shade2d[ edges[i][1]  ] [2];
  g.drawLine(a,b,c,d);
     }//end of for i
  }//end of if
 }//end of for nScenes
}//end of paint
}//end of class
```

11 ***Exercise*** *Play with the code. Modify it to include translation of the cube.*

12 ***A cube with faces*** *Our cube is until this moment made of wires, so faces must be imagined. Let us include real faces: a face is a polygon determined by its vertexes and that can be colored on its own. The code follows:*

```
/* Program D12,  cubeFaces
   A flying 3d cube with faces that  rotates.
   */

import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Polygon;
import java.util.Random;

public class cubeFaces extends Applet
 {
  private static final long serialVersionUID = 1L;


  /*
   Our basic cube is the unitary cube at the origin.
```

```
 It is composed of 8 points or vertexes = points3d
 and 12 edges.
 Points are numbered from 0 to 7.
 These numbers are used in
 edges, which  records the pairs of
 points that are joined by a line.
 A face is determined by its 4 corners */


private static   double points3d[][]=
                         { {0,0,0},
                        {1,0,0},
                        {0,1,0},
                        {0,0,1},
                        {1,1,0},
                        {1,0,1},
                        {0,1,1},
                        {1,1,1}
                       };

private static int edges[][] =
                            {{0,1},
                           {0,2},
                           {0,3},
                           {1,4},
                           {1,5},
                           {2,6},
                           {2,4},
                           {3,5},
                           {3,6},
                           {4,7},
                           {5,7},
                           {6,7}};

private static int faces[][] =
                            {{0,1,4,2},
                           {0,2,6,3},
                           {0,3,5,1},
                               {3,5,7,6},
                               {1,4,7,5},
```

```
                                {2,6,7,4}};


  /*A 3d cube exists only in the imagination:
   * it is not seen.
   * One sees a shade or projection of it over the  screen.
   * The shade is defined by 2d points. The edges of
   * the 2d shade are inherited from the edges of the
   * 3d cube.
   */
  private static int shade2d[][] = new int[8][3];

  //All cubes are derived from the basic cube by rotations
  //and translations
  private static double cube[][]  = new double[8][3];
  private static double  cubeProj[][] = new double[8][3];

  private static double axisRot[] = new double[3];
  private static double axisProj[]  = new double[3];
  private static double deltaTrans[]  = new double[3];
  private static double vTrans[]  = new double[3];
  private static double theta  = 0;
  private static double deltaTheta;

  private static double basisRot[][], basisProj[][]
                                        = new double[3][3];
  private static double matrixRotz[][],
                        matrixRot[][],
                        matrixProj[][] = new double[3][3];
  //The matrix of a projection along the z-axis
  private static   double J[][]=  { {1,0,0},
                                    {0,1,0},
                                    {0,0,0}
                                   };
  private static  double scale;
  private static int delay ;
  private static int nScenes ;

  static int colors[][] = new int[6][3];
  private static int pointsX[]  = new int[6];
```

```
  private static int pointsY[]  = new int[6];

  static Random r  = new Random();


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

  //The multiplication of two matrixes
  private static  double[][] matrixMult(double p[][],
                                double q[][])
  {
  double[][] helpM = new double[3][3];

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

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

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

```
for(int j=0; j<3; j++)
 helpV[i]  =helpV[i]  + m[i][j]*v[j];
 return helpV;
  }

 //-----------------------------

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

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

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

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

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

  }


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

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

return helpM;
  }

 //------------------------------


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

  //A rotation is defined by an axis v that crosses
  //through 0,
  //and a counterclockwise angle theta of rotation
  private static  double[][] mRotation(double v[],
                                double theta)
  {
  matrixRotz= mRot(theta);
```

```
  basisRot = basisB(v);
  //printMatrix(matrixRotz, "Rotz");
  //printMatrix(basisRot, "BasisRot");
  double[][]  helpM = new double[3][3];
  helpM = matrixMult(matrixRotz,
              matrixTranspose(basisRot));
  helpM = matrixMult(basisRot,helpM);

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

  //The 3d cube is projected into a 2d plane.
  //The resultant figure must be rendered to the screen.
  //A projection is done along  an axis that crosses
  //through 0.
  //Our projections are observed from infinite:
  //we use no perspective corrections.
  private  static  double[][] mProj(double w[] )
  {
  basisProj = basisB(w);
  double[][]  helpM = new double[3][3];
  helpM = matrixMult(J,matrixTranspose(basisProj));
  helpM = matrixMult(basisProj,helpM);
  //printMatrix(helpM, "Proj");
  //Test: Proj*Proj must be equal to Proj
  /*helpM = matrixMult(helpM,helpM);
   printMatrix(helpM, "Proj* Proj");*/
  return helpM;
  }

  //The cube c is rotated around axis v an angle theta
  private  static   double[][] rotCube(double c[][],
                        double v[], double theta)
  {
double[]  helpV = new double[3];
double cc[][] = new double[8][3];
```

```
matrixRot = mRotation( v,theta);
//printMatrix(matrixRot, "matrixRot");
//The 8 points of the cube must be rotated
for(int i=0; i<8; i++)
{
 for(int j=0; j<3; j++)
   helpV[j] = c[i][j];
 helpV = matrixVector(matrixRot, helpV);
 for(int j=0; j<3; j++)
 cc[i][j] = helpV[j];
}
//printCube(cc, " after rot");
return cc;
  }



//The cube c is translated   deltaV
  private static double[][] translateCube(double c[][],
                                     double deltaV[])
  {

   double cc[][] = new double[8][3];
   for(int i=0; i<8; i++)
    for(int j=0; j<3; j++)
   cc[i][j] = c[i][j] + deltaV[j];
   return cc;
  }


  //The cube c is projected along axis w
  private static double[][] projCube(double c[][],
                                double w[])
  {
double[]  helpV = new double[3];
double cc[][] = new double[8][3];
matrixProj = mProj(w);
for(int i=0; i<8; i++)
{
```

```
 for(int j=0; j<3; j++)
   helpV[j] = c[i][j];
 helpV = matrixVector(matrixProj, helpV);
 for(int j=0; j<3; j++)
 cc[i][j] = helpV[j];
}
return cc;
  }

  //The rendering procedure accepts integer values
  private  static  int[][] round(double c[][] )
  {
  int[][] helpM = new int [8][3];
for(int i=0;i<8;i++)
 for(int j=0; j<3; j++)
helpM[i][j] = (int) Math.round(c[i][j]);
return helpM;
  }

  private static void printCube(double c[][], String a)
  {
   System.out.println("This is the cube" +a);
   for(int i=0;i<8;i++)
    System.out.println(c[i][0] + " " + c[i][1]+ " "
                                     + c[i][2]);
  }

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

  private void printShade()
  {
System.out.println("This is the shade");
    for(int i=0;i<8;i++)
     System.out.println(shade2d[i][1] + " "
```

```
                              + shade2d[i][2]);
  }

  //Colors to paint the faces are defined
  static private void defineColors()
  {
  for(int l = 0; l<6;l++)
{
 //We use random colors.
 //A color can be defined by three numbers,
     //each one from 0 to 255.
     //We has 6 faces, so we need
     //a matrix to keep colors.

  int c1,c2,c3;
  c1 = r.nextInt(226);
  c2 = r.nextInt(226);
  c3 = r.nextInt(226);
  //Colors are memorized
  colors[l][0]= c1;
  colors[l][1]= c2;
  colors[l][2]= c3;
}
  }


  //Parameters are set up.
  private static  void initialize()
  {
scale = 50;
defineColors();
for(int i=0; i<3;i++)
vTrans[i] = 0;
for(int i=0; i<8;i++)
 for(int j=0; j<3;j++)
  points3d[i][j] = scale*points3d[i][j];
//printCube();
//The 3d-graphic is rotated along the axisRot
//an angle theta that is incremented by deltaTheta
theta = 0;
```

```
deltaTheta = 0.1;
axisRot[0] = 1;
axisRot[1]=1;
axisRot[2]=1;
    //The 3d-graphic is projected over the screen
    // which is perpendicular to the x-axis.
axisProj[0] = 1;
axisProj[1]=0;
axisProj[2]=0;
//The 3d cube flies along vectTrans[]
deltaTrans[0] = 0;
deltaTrans[1] = -1;
deltaTrans[2] = 0;
nScenes = 500;
//delay between two scenes
delay =  500;
  }

  //This is the main method
public void paint (Graphics g)
{
 System.out.println("Cube with faces");
 initialize();


 //background color
 g.setColor(Color.pink);
 g.fillRect(0,0,400,400);
 //Loop for the scenes
 for(int k = 0; k<nScenes; k++)
 {
  theta = theta + deltaTheta;

  cube = rotCube(points3d,axisRot,theta);
  //printCube(cube, " after rotation");
  for(int i = 0; i<3;i++)
   vTrans[i] = vTrans[i] + deltaTrans[i];

  cube = translateCube(cube, vTrans);
```

```
  cubeProj = projCube(cube,axisProj);
  //printCube(cubeProj, " after projection");
  shade2d =  round(cubeProj );
  //printShade();

  //draw cube
  for(int i=0;i<6;i++)
   {
g.setColor( new Color( colors[i][0], colors[i][1],
                              colors[i][2]));



// A polygon is defined by two vectors and an integer
//number:
//The first vector lists the x-coordinates of the
//corners;
//the second lists  the y-coordinates. The integer is
//the number of points.
//If the last point is different than the first,
// the first point is closed with the last.
for(int l =0; l<4; l++)
 {
  pointsX[l] = 100-shade2d[ faces[i][l]  ] [1];
  pointsY[l] = 100-shade2d[ faces[i][l]  ] [2];
  //System.out.println(pointsX[l] + " " + pointsY[l]);
 }
 Polygon face = new Polygon(pointsX, pointsY, 4);
 g.fillPolygon(face);
   }
   //delay
   if (k < nScenes-1)
    {
     try
      {
   Thread.sleep( delay );
      }
      catch (InterruptedException e)
      {
   e.printStackTrace();
```

```
      }

     //erase cube: we repaint everything in
     //the background color.
      g.setColor(Color.pink);
      g.fillRect(0,0,200,400);
  }//end of if
 }//end of for nScenes
}//end of paint
}//end of class
```

13 ***Exercise*** *Run the code. Verify that the code builds a cube with faces but fails to color the right ones to show a solid opaque cube. Please, correct the bug.*

2.3 To become an expert

The modern standard of 3d graphics is quite demanding because 3d graphics is at the root of the industry of video games, which cause nowadays the flux of very important sums of money.

Nevertheless, graphics has been traditionally a late development in most programming languages. Precisely, the Java Community has recently acquired the possibility to use a bundle of refined tools for 3d graphics and multimedia which has been released in the **Java 3D API**. We have decided at this moment to save its study for another day because our approach is simpler (and less powerful). If you cannot resist the temptation of giving a glance to that project, please follow the next link (Cited 16 Dec 2009):

http://java.sun.com/javase/technologies/desktop/java3d/

We use cubes as the fundamental building blocks of our graphics, but triangulated surfaces also can be used and look more professional.

2.4 Touching at consciousness

Our game with cubes allows us to discuss a delicate theme that relates reality, perception and consciousness. Let us begin with a well known effect: a 2d figure that represents a cube is perceived by a human as a three dimensional one, but two options are possible, as in the next figure:

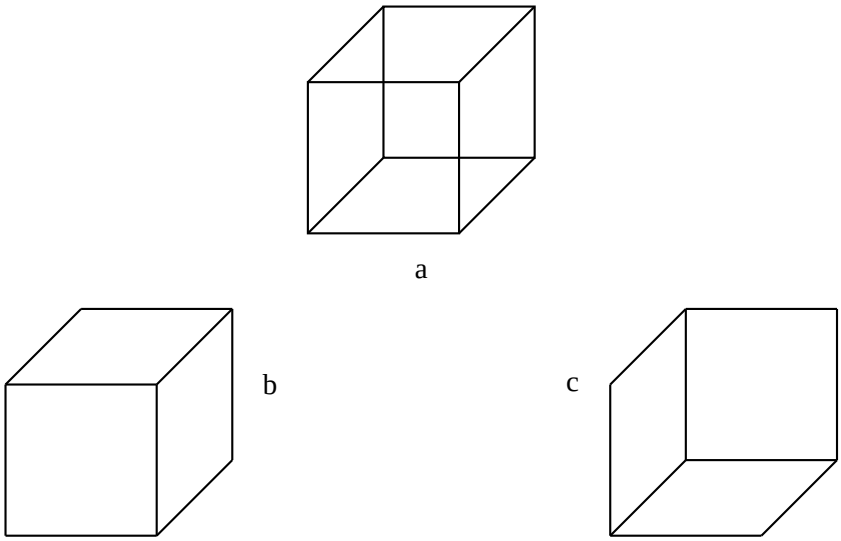


Figure 2.2. A two dimensional figure is seen by a human like a three dimensional one, as in b or c.

Why?

The reason of this illusion is the following: the mind is a processor that offers to the consciousness not the reality such as it is but a delicate elaboration of what senses perceive. What we see in this case is the general modus operandis of the mind: if data can be compressed and presented in a simple way, it is reedited and the consciousness perceives only the final output. Let us notice that consciousness is not at the retina, or, reality is not accessed by the consciousness: the inner artificial reality, a virtual reality synthesized by the mind, is all to the human form of knowing the world.

In this case, a three dimensional figure is more common and simpler than a two dimensional one: two dimensional depictions of a cube are seen only in art while cubes and blocks of complex geometry are seen all around. Moreover, the computations that predict how a 3d cube behaves are simpler in comparison with the computations that show how the 2d figures would evolve to represent the projection over a 2d plane of a 3d cube.

We can understand now the fundamental formulation of **philosophy** which is built upon a trinity composed of **consciousness, mind and reality**:

1. Consciousness is the perception of the self amidst a world and a society and it

is the source of free will. The consciousness is what observes and perceives. Let us notice that consciousness is real for every person but it is outside of the world it perceives. Nevertheless, consciousness is considered by modern science as an illusion created by the brain. A reason that supports this belief is that robots that function like a human being are under development but they have no consciousness apart from their programming.

2. Some part of the human perception corresponds to a reality which exists in itself without the concourse of any mind or consciousness.

3. Reality is distinguished because its perception is stable, looks as highly complex and can be verified by anyone person while ideas, dreams and hallucinations are unstable, have limited complexity, can be changed by mere desire and are not subject of direct perception by other persons.

4. The exterior reality is inaccessible for the consciousness.

5. The object of perception of the consciousness is an inner artificial reality synthesized by the mind and which has two main parts: a reedited and simplified inform of the outside reality and a perception of the self, which includes thoughts, dreams and beliefs. A **belief** is an idea that is accepted as true or real by somebody. There are some beliefs that are not arrived at after delicate logical reasoning but because they appear as such in the inner artificial reality. That is why we are irrational beings. Beliefs include simple proposals, as 'blood is red' and very delicate ones, say, 'that person is bad'.

Let us see some simple and powerful mechanisms as **beliefs** can be induced.

The first mechanism corresponds to black-out: some piece of information is ignored as of secondary importance and so a belief automatically arises. This is illustrated with the cube by diminishing the force of some lines, see figure 2.3. The black-out mechanisms is universal: no neuronal processing machine has enough power to take account of every detail, so many pieces are blacked out automatically.

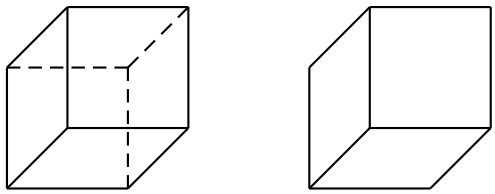


Figure 2.3. If some pieces of information are weakened, a given perception can be induced: the cube with dashed lines is interpreted with a slightly higher preference as in its companion figure.

The power of incomplete black-out is not total: by looking at figure 2.3, one sees that the cube with dashed lines can be interpreted in anyone of the two possible forms, although the suggested interpretation prevails.

Given that there are powerful ideological wars all around the world, more powerful mechanisms to catalyze the election of **beliefs** must exist. The next is a good option: it is called breaking of symmetry by contextualization: the mind tries to integrate details into great projects and so the general form dictates the perception of details. This is appreciated in the next figure:

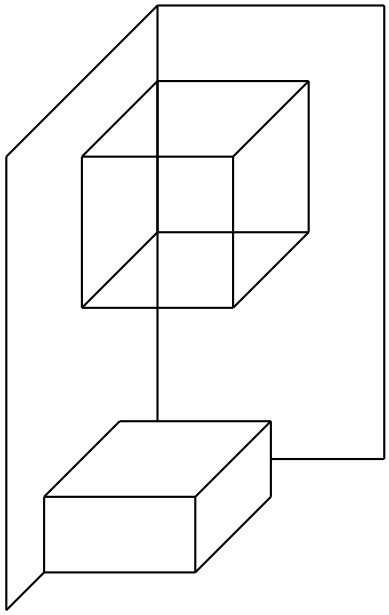


Figure 2.4. An interpretation of the whole dictates the form as details are perceived: a 2d representation of a cube is interpreted as a 3d cube in a preferred form which is dictated by the context.

This mechanism is of daily use in every rhetoric encounter: in ordinary conversations, in demagogy and in science.

The trinity composed of consciousness, artificial inner reality and (exterior) reality is organized in two main doctrines: **realism and idealism**. In realism, reality is considered to be exterior to the mind and independent of it. Idealism is the challenge to realism built upon the fact that reality is not accessible to consciousness and that the artificial inner reality is all what consciousness perceive and, therefore, that what we call reality cannot be no more than a product of our mind, an idea,

a construction that cannot exist outside of it. A recent and popular artistic explanation of the tension between realism and idealism is given by the film Matrix: people are induced to dream a reality but they are unaware of the fact that they are just dreaming.

14 ***Example: water vs wine***

A real person in his or her ordinary behavior swings among diverse flavors of philosophy. Let us consider, for instance, that one drinks some water to nourish the body, so one recognizes that one is reality dependent. But in a toast one drinks alcohol that causes happiness, a feeling that forever will differentiate us from robots, and one expresses aloud a good desire for somebody with the certainty that this world is moved by ideas, i.e., that consciousness can directly interfere with reality. This belief, that the mind can interfere with reality, is also present in some interpretations of quantum mechanics: the exact characteristics of matter are created when they are registered by consciousness and not otherwise.

15 ***Materialism***

Important for us is **materialism**, a realistic interpretation of the whole upon which contemporary science is built and that claims that

1. Reality is entirely composed of matter as depicted by our physical laboratories, consisting concretely of the atoms that are listed in the periodic table and by the elementary particles as in the standard model.

2. All minds and consciousness are byproducts of matter.

3. Matter is detectable by matter. So, if you think that something is real, let it influence the surrounding world or invent appropriate instrumentation to detect it in such a way that other persons and scientists could corroborate your description.

Materialism has inspired the difficult road of science that has driven its amazing development: matter can be studied by matter and no irremovable limit to the ensuing knowledge has been proved to exist.

Materialism, science and the scientific method are blamed by idealism as philosophical innocence and ignorance: any experiment is no more than an experiment of the consciousness and for the consciousness. In particular, no proof can be given that a reality exists. Or that if a reality exists, it is faithfully represented by the inner reality.

16 ***Example: ghosts***

Many people around the world claim that they share their houses with **ghosts**, faint creatures that can speak by telepathy, trespass walls, cause a sensation of cold and the hair of the skin to bristle. The capability to feel ghosts has a high variability ranging from people that never feel nothing to people that see and listen at them. The perception of ghosts is classified by those that feel them as objective, they agree on the close to them presence or absence of a ghost and every one is sure that their presence corresponds to something not created by their minds. The Author knows of one family that left a house because of the presence of ghosts and 'bad energies'. And some other houses remain void because of the same problem.

Let us look at the position of materialism about these perceptions:

- Materialism cannot deny the possibility that these perceptions could correspond to something objective, real or material.
- Materialism will accept that ghosts, demons and gods exist only if they can be reliably registered by physical instrumentation. If they are real, this is plainly possible: since they are perceived by human beings, they must be perceived by a given sense that is implemented with matter and so the physical principles that govern that matter can be used to devise registering instrumentation.
- Let us recall that matter is anything that can exist outside any mind. So, materialism cannot deny the possibility that ghosts could be another form of matter apart than that registered by our laboratories up to date. Materialism cannot deny the possibility that the man can have a spirit that after death becomes a ghost and that these entities could consists of an alien type of matter, let it be called spiritual matter. What materialism demands is to accept that spiritual matter must be registrable by ordinary matter in the lab because ghosts are perceived by humans: ghosts are possibly perceived by the spirit of living persons but how does that spirit manage to communicate others persons that a ghost has been seen? It is only because the spirit and ordinary matter can register one another. Therefore, spiritual matter and ordinary matter must register one another in the lab.
- You can gain a very honorable place in the history of the universe if you can devise a reliable protocol to register ghosts, auras, telepathy or whatever. Actually, some scientists have investigated these and related phenomena but all recognized results are disappointing. By the same token, many videos exist that show images that are claimed to belong to ghosts. But, how can we reliably repeat that registration to enable further investigation?

- In spite of the willingness to extend the ordinary physics, the controlled and systematic study of the fundamental laws that rule matter, we have no experimental reliable protocol to prove the existence of ghosts and we have received no help from those that believe in them. And so, studies about ghosts and related things do not appear in general in the international scientific literature.

- There are some people for which materialism is the dogma that provides the interpretation of the whole and that dictates how details are perceived. This is for them a **belief** that is certain by net perception and that needs no argumentation. One of such persons could simply say: "Ghosts? Did you say ghosts, God, creation, hell? Excuse me, I am very busy!".

The belief in ghosts is just one of the many items exhibited by those that adhere to **spiritualism**, who consider that reality is not entirely composed of matter but that there is a spiritual world that is well known since ancient times by every human group. In spite of the great and permanent effort made by scientists to combat the spiritualist beliefs of modern days, no visible progress has been made: religions and sorcery move more money than all scientific programs together. The accusation of spiritualism against materialism is simple: science is similar to a government that denies that alien aircrafts invades its skies until it can knock one of them down.

Spiritualism may be taken as part of reality for those people that can see or feel ghosts. Otherwise, it can be adopted as a dogma.

A **Dogma** is a belief adopted not by testing but because of authority. Dogmas are very important for surviving: they serve as world views and because they are received as inheritance, one must not worry because of testing of beliefs but instead ones receives truths already made in terminated form. Many mathematical results are taken as dogma for those that hate mathematics. Materialism is a dogma for many scientists that do not want to waste their time discussing questions that for them are ridiculous, say, why little children fear the darkness (they can feel ghosts but they experiment relief if they feel the light.)

17 ***Exercise*** *Consider the problem of the telepathic voices that some people listen and discuss how each philosophical position dictates a corresponding therapy. This problem can be described as follows:*

Many children look for counseling against voices that they listen from the interior of their heads, very commonly when they begin to fall asleep. These voices are also common among elders, drunker people, drug addicts, players of Ouija, practitioners of magic or Satanism, members of some religious and of some Christian movements. Those voices might come accompanied with hallucinations, lost

of consciousness and very strange psychological pathologies such as possession. Many religions have been founded by persons that saw visions 'from heaven' and that listen 'angelical' voices and prophecies that sometimes are spectacularly fulfilled. On the other hand, many suicides and homicides are believed to have happened by persons that follow instructions given by such voices. The voices have four peculiarities:

1. *The patient clearly differentiates among its own being and the voices.*
2. *These voices like to predict what will be in the future but predictions are many times very different than 'it will rain tomorrow' instead they are targeted to fit a lot of possible sceneries while producing tension and pain: 'weather will cause you to suffer'. Voices also say: worship me. Sometimes they promise something in response. For instance, the press reported the case of an Indian woman who killed her baby because she listened a voice that said: kill me your baby and I will show you where is a hidden treasure. No treasure was ever shown.*
3. *Patients pass over terrible crisis in which the voices seem to follow a simple rule: to fall with fury upon the fallen. Because of these crises, many adult patients must leave their works, while others go plainly mad.*
4. *Voices form sometimes part of a cocktail of illnesses that may include, say, epilepsy. But no neurological disorder is found by very modern examinations in many other cases. Nonetheless, molecular biology has been claiming some results.*

18 ***Doubts***

It is mandatory at this moment to return to figure 2.4: an appropriate background causes the 2d depiction of a cube to be perceived preferentially as one of the two possible 3d interpretations. But, if one drags the sight across the figure, one could observe that the contrary interpretation also appears by moments in the mind. The implication is simple: beliefs are not accompanied with certainty: the doubt always invades the consciousness of the believer from time to time. In other words, a belief is insecure by its very nature and to become certainty, it must be reinforced by a supporting feed back of the free will. The great conclusion is that everyone is responsible for his or her own **beliefs**. Additionally, permanent beliefs and credos are continuously nourished by active social networks and by personal rituals.

19 ***Compartmentalized knowledge***

There is a lot of suffering amidst doubt and certainty: the most humiliating fact about our nature is that any trace of certainty is deprived of universalism. To understand this, let us pay attention to the next figure:

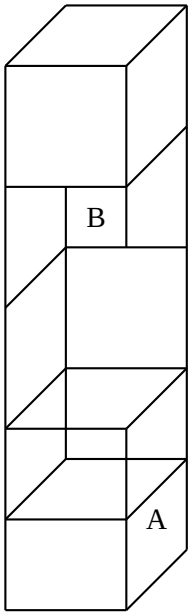


Figure 2.5. No background or brain washing is strong enough to generate a unique contextualized interpretation of the whole. Instead, different aspects of the contextualizing background can generate diverse and mutually contradictory interpretations of details. And this happens in one and the same mind. To verify this, please, fix you sight at point A and then switch it to B.

The universe we live in is very complex to be understood by our senses and mind and so we fabricate compartmentalized knowledge that cannot be integrated anyhow because of arising contradictions. According to my own observation of myself, this goes as far as to admit a double moral as constitutive part of my life. Another instance: a professional atheist had the kindness of telling me that he prayed the Lord for protection for his only son when late in the night he did not returned back to home.

20 ***Example: solving mysteries***

A mystery is a **belief** that can only be explained by a greater mystery. For instance, religion kills all mysteries by resorting to God. But everyone rapidly

arrives to a sort of questions about God that the religious system cannot answer. Science is not very different: the most popular scientific cosmogony is the Big Bang. It rests upon results ranging from general relativity to quadrupole studies of the infrared background radiation. So, it is very serious. But, what does it say about the origin of the universe?

It says that the origin of the universe was caused by an explosion in the quantum vacuum which furnished enough energy to create all galaxies and clusters together. Nevertheless, no one has ever registered a single event in the laboratory in which even an electron-positron pair had been synthesized from vacuum, be it a true vacuum (the actual one) or false (the original vacuum that was an exited state that next decayed into the actual one releasing the huge amounts of energy from which everything was synthesized). Quite to the contrary, the synthesis of electron-positron pairs demands high quantities of energy, so much that when such experiments are run, the electric system of the whole country gets stressed. In this way, modern science can explain everything but it rests on a **belief** that completely lacks support from experiment or observation.

21 *Test yourself*

All considered situations about **beliefs** are biology given. We can say that the philosophy behind this is to produce persons with beliefs but with wisdom. The philosophy is not to produce skeptics or people that are not eager to defend what they think or people that think that they are the truth. That is why we celebrate a congruence of five factors: express yourself, test yourself, help other persons to test themselves, let others persons to test you and try to be faithful to yourself.

Warning: we are proposing a mechanism to explain the irrationality of the human being, which is a situation in which no logical reason can cause a change of mind. This state of affairs is also biologically given: some beliefs might be arrived at not by logical reasoning but because the inner artificial reality presents them as facts to the consciousness and so they become as immanent part of the existence, a thing that consciousness does not like to reject. This knowledge might be used to devise peacemaking methods (Rodríguez, 2007). To say the least, if we keep this in mind, we will more tolerant with those that exercise beliefs that are contrary to what we think. So, creationists and evolutionists do not need to kill one another anymore. Instead, they could achieve a state of mind in which they would enjoy trying to understand what the other thinks. Such an approach will be cultivated in coming chapters.

2.5 Conclusion

We have had an encounter with 3d graphics and we have discovered that this theme is quite technical but not that impossible for an ordinary person. Between the frontier of 2d and 3d graphics, we were able to discuss some fundamental properties of our consciousness and the way it acquires and generates knowledge, questions that are studied by philosophy. In realism, a reality that does not depend on any consciousness can be known and studied. In Idealism, reality is not accessible to consciousness and all we can do is to be content with looking an artificial inner reality synthesized by the mind. The tension between realism and idealism has been illustrated with a discussion about ghosts. We contend that some beliefs of any ordinary person are not arrived at by reasoning but that they are presented to the consciousness by the artificial inner reality as facts and in this way consciousness acquires irrationality: no reason can make a change of mind. This is important for understanding why there are both creationists and evolutionists.

Chapter 3

Drawing a hand

```
Cubes are used to sketch a hand.
```

22 ***Purpose.*** *We use the technology developed for cubes in the previous chapter to making a 3d sketch of a hand.*

3.1 Drawing a 3d hand

Let us engage in the task of drawing a three dimensional left hand. This task can be very complicated if one aims at a finely terminated artistic work so let us restrict ourselves to a sketch in which we could understand that a hand is made of moving parts. They are slight modifications of our cube of the previous chapter but in ensemble amount to a complex object.

We have here an appropriate occasion to exercise the power of OOP (object oriented programming), whose philosophy is to devise an artificial reality in which objects could be manipulated as if they were in the real world. So, we might image a universe populated with various figures: cubes in diverse flavors that possibly form a hand, spheres, persons, animals, buildings, etc. Our first step in that direction is to reformulate our previous cube as an inner class, i.e., a class embedded in within another.

23 ***The code.*** *A first OOP version of a rotating flying solid cube.*

```
/*Program D23 CubeOOP

We present a second  OOP version of
a flying rotating solid cube:
```

```
the inner class Cube is
used as a prototype to build many cubes.
*/

import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Polygon;
import java.util.Random;

public class CubeOOP extends Applet
{

private   final long serialVersionUID = 1L;

// Class CubeOOP  is an archetype,  a plan or design,
// an idea.

// The next instruction
// constructs a real stuff, an instance, on the
// image and resemblance of the archetype.

 private static int nScenes ;

 static  CubeOOP CubeOOP   = new CubeOOP();

 //=============Inner class Cube==============
 public static class Cube
 {
  private   final long serialVersionUID = 1L;


  /*
   Our basic cube is the unitary cube at the origin.
   It is composed of 8 points or vertexes = points3d
   and 12 edges.
   Points are numbered from 0 to 7.
   These numbers are used in
   edges, which  records the pairs of
   points that are joined by a line.
```

```
   A face is determined by its 4 corners */


  private static   double points3d[][]=
  { {0,0,0},
    {1,0,0},
    {0,1,0},
    {0,0,1},
    {1,1,0},
    {1,0,1},
    {0,1,1},
    {1,1,1}
   };

private static int edges[][] =
      {{0,1},
       {0,2},
       {0,3},
       {1,4},
       {1,5},
       {2,6},
       {2,4},
       {3,5},
       {3,6},
       {4,7},
       {5,7},
       {6,7}};

private static int faces[][] =
      {{0,1,4,2},
       {0,2,6,3},
       {0,3,5,1},
       {3,5,7,6},
       {1,4,7,5},
       {2,6,7,4}};


/*A 3d cube exists only in the imagination:
* it is not seen.
* One sees a shade or projection of it over the  screen.
```

```
* The shade is defined by 2d points. The edges of
* the 2d shade are inherited from the edges of the
* 3d cube.
*/
private static int shade2d[][] = new int[8][3];

//All cubes are derived from the basic cube by rotations
//and translations
private static double cube[][]  = new double[8][3];
private static double  cubeProj[][] = new double[8][3];

private static double axisRot[] = new double[3];
private static double axisProj[]  = new double[3];
private static double deltaTrans[]  = new double[3];
private static double vTrans[]  = new double[3];
private static double theta  = 0;
private static double deltaTheta;

private static double basisRot[][], basisProj[][]
               = new double[3][3];
private static double matrixRotz[][],
matrixRot[][],
matrixProj[][] = new double[3][3];
//The matrix of a projection along the z-axis
private static   double J[][]=  { {1,0,0},
        {0,1,0},
        {0,0,0}
       };
private static  double scale;
private static int delay ;


static int colors[][] = new int[6][3];
private static int pointsX[]  = new int[6];
private static int pointsY[]  = new int[6];
//How to color faces
private static  double maxX;
private static  int maxXPoint;
```

```
private static int nExtreme;
private static boolean maxPoints[]  = new boolean[8];
private static boolean coloredFace[]  = new boolean[6];

static Random r  = new Random();

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

//The multiplication of two matrixes
private static  double[][] matrixMult(double p[][],
                                      double q[][])
{
double[][] helpM = new double[3][3];

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

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

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

```
for(int i=0; i<3; i++)
for(int j=0; j<3; j++)
helpV[i]  =helpV[i]  + m[i][j]*v[j];
return helpV;
}

//------------------------------

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

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

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

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

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

}


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

```
double[]   helpT, helpS, helpF = new double[3];

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

return helpM;
}

//-----------------------------


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

//A rotation is defined by an axis v that crosses
//through 0,
//and a counterclockwise angle theta of rotation
private static  double[][] mRotation(double v[],
                                     double theta)
{
```

```
matrixRotz= mRot(theta);
basisRot = basisB(v);
//printMatrix(matrixRotz, "Rotz");
//printMatrix(basisRot, "BasisRot");
double[][]  helpM = new double[3][3];
helpM = matrixMult(matrixRotz,
 matrixTranspose(basisRot));
helpM = matrixMult(basisRot,helpM);

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

//The 3d cube is projected into a 2d plane.
//The resultant figure must be rendered to the screen.
//A projection is done along  an axis that crosses
//through 0.
//Our projections are observed from infinite:
//we use no perspective corrections.
private  static  double[][] mProj(double w[] )
{
basisProj = basisB(w);
double[][]  helpM = new double[3][3];
helpM = matrixMult(J,matrixTranspose(basisProj));
helpM = matrixMult(basisProj,helpM);
//printMatrix(helpM, "Proj");
//Test: Proj*Proj must be equal to Proj
/*helpM = matrixMult(helpM,helpM);
printMatrix(helpM, "Proj* Proj");*/
return helpM;
}

//The cube c is rotated around axis v an angle theta
private  static   double[][] rotCube(double c[][],
                          double v[], double theta)
{
double[]  helpV = new double[3];
```

```
double cc[][] = new double[8][3];
matrixRot = mRotation( v,theta);
//printMatrix(matrixRot, "matrixRot");
//The 8 points of the cube must be rotated
for(int i=0; i<8; i++)
{
for(int j=0; j<3; j++)
helpV[j] = c[i][j];
helpV = matrixVector(matrixRot, helpV);
for(int j=0; j<3; j++)
cc[i][j] = helpV[j];
}
//printCube(cc, " after rot");
return cc;
}



//The cube c is translated   deltaV
private static double[][] translateCube(double c[][],
                                        double deltaV[])
{

double cc[][] = new double[8][3];
for(int i=0; i<8; i++)
for(int j=0; j<3; j++)
cc[i][j] = c[i][j] + deltaV[j];
return cc;
}


//The cube c is projected along axis w
private static double[][] projCube(double c[][],
                                   double w[])
{
double[]  helpV = new double[3];
double cc[][] = new double[8][3];
matrixProj = mProj(w);
for(int i=0; i<8; i++)
```

```
{
for(int j=0; j<3; j++)
helpV[j] = c[i][j];
helpV = matrixVector(matrixProj, helpV);
for(int j=0; j<3; j++)
cc[i][j] = helpV[j];
}
return cc;
}

//The rendering procedure accepts integer values
private  static  int[][] round(double c[][] )
{
int[][] helpM = new int [8][3];
for(int i=0;i<8;i++)
for(int j=0; j<3; j++)
helpM[i][j] = (int) Math.round(c[i][j]);
return helpM;
}

private static void printCube(double c[][], String a)
{
System.out.println("This is the cube" +a);
for(int i=0;i<8;i++)
System.out.println(c[i][0] + " " + c[i][1]+ " "
                  + c[i][2]);
}

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

private void printShade()
{
System.out.println("This is the shade");
for(int i=0;i<8;i++)
```

```
System.out.println(shade2d[i][1] + " "
           + shade2d[i][2]);
}

//Colors to paint the faces are defined
static private void defineColors()
{
for(int l = 0; l<6;l++)
{
//We use random colors.
//A color can be defined by three numbers,
//each one from 0 to 255.
//We has 6 faces, so we need
//a matrix to keep colors.

int c1,c2,c3;
c1 = r.nextInt(226);
c2 = r.nextInt(226);
c3 = r.nextInt(226);
//Colors are memorized
colors[l][0]= c1;
colors[l][1]= c2;
colors[l][2]= c3;
}
}

//what faces must be colored
private static  void marked()
{
for(int f = 0; f<6; f++)
coloredFace[f] = false;
for(int f = 0; f<6;f++)
{
for(int p = 0; p<4;p++)
{
for(int c = 0; c<8; c++)
{
if (maxPoints[c] == true)
{
if (faces[f][p] == c )
```

```
coloredFace[f] = true;
//System.out.println("coloring");
}
}
}
}
}


//The x coordinate of all points are measured
//and the extreme ones are recorded.
private static  void calculateColor()
{
//The maximum value of coordinate x
maxX =-1000;
for(int l = 0; l<8; l++)
{
if (cube[l][0] > maxX)
{
maxX = cube[l][0];
maxXPoint = l;
}
}
//System.out.println("maxX, maxZ "+ maxX + " " + maxZ);
//System.out.println("maxXPoint, maxZPoint "+
//                         maxXPoint + " " + maxZPoint);
nExtreme = 0;
//Number of extreme points
for(int l = 0; l<8;l++)
{
maxPoints[l] = false;
if (Math.abs(cube[l][0] - maxX )< 2)
{
nExtreme = nExtreme +1 ;
maxPoints[l] = true;
}
//faces to be colored
}
marked();
}
```

```
//Parameters are set up.
private static  void initialize()
{
scale = 50;
defineColors();
for(int i=0; i<3;i++)
vTrans[i] = 0;
for(int i=0; i<8;i++)
for(int j=0; j<3;j++)
points3d[i][j] = scale*points3d[i][j];
//printCube();
//The 3d-graphic is rotated along the axisRot
//an angle theta that is incremented by deltaTheta
theta = 0;
deltaTheta = 0.1;
axisRot[0] = 1;
axisRot[1]=1;
axisRot[2]=1;
//The 3d-graphic is projected over the screen
// which is perpendicular to the x-axis.
axisProj[0] = 1;
axisProj[1]=0;
axisProj[2]=0;
//The 3d cube flies along vectTrans[]
deltaTrans[0] = 0;
deltaTrans[1] = -1;
deltaTrans[2] = 0;
nScenes = 500;
//delay between two scenes
delay =  100;
}

 }

 //=============End of inner class Cube==============

 //The methods of the main application begin here.
```

```
 //Measures of the window
 private static  int   xWin = 500;
 private static  int   yWin = 400;


//This method is fired automatically
//at the beginning
 public void init()
  {
   setSize(xWin,yWin);
  }

 //The cube is rotated, translated and projected
 private static  void modifyCube()
 {
  Cube.theta = Cube.theta + Cube.deltaTheta;

  Cube.cube = Cube.rotCube(Cube.points3d,Cube.axisRot,
                                  Cube.theta);
  //printCube(cube, " after rotation");
  for(int i = 0; i<3;i++)
  Cube.vTrans[i] = Cube.vTrans[i] + Cube.deltaTrans[i];

  Cube.cube = Cube.translateCube(Cube.cube, Cube.vTrans);
  //printCube(cube, " after translation");
  Cube.cubeProj = Cube.projCube(Cube.cube,Cube.axisProj);
  //printCube(cubeProj, " after projection");
 }

 private static  void delay()
 {
 try
     {
   Thread.sleep(Cube.delay );
     }
     catch (InterruptedException e)
     {
   e.printStackTrace();
     }
```

```
 }

 //face ii of the cube is drawn
 private static  void drawface(Graphics g, int ii)
 {
 for(int l =0; l<4; l++)
  {
Cube.pointsX[l] = 100
             -Cube.shade2d[Cube.faces[ii][l]  ] [1];
 Cube.pointsY[l] = 100
            -Cube.shade2d[ Cube.faces[ii][l]  ] [2];
   //System.out.println(pointsX[l] + " " + pointsY[l]);
  }
  //A polygon is defined by two vectors and an integer
   //number:
   //The first vector lists the x-coordinates of the
   //corners;
   //the second lists  the y-coordinates. The integer is
   //the number of points.
   //If the last point is different than the first,
   //the first point is closed with the last.
   Polygon face = new Polygon(Cube.pointsX,
                       Cube.pointsY, 4);
   g.fillPolygon(face);
 }

 //This is the main method
 public void paint (Graphics g)
 {
  System.out.println("Solid flying rotating OOP cube");
  Cube.initialize();
  //background color
  g.setColor(Color.pink);
  g.fillRect(0,0,400,400);
  //Loop for the scenes
  for(int k = 0; k<nScenes; k++)
  {
   modifyCube();
   Cube.shade2d =  Cube.round(Cube.cubeProj );
   //printShade();
```

```
   //draw cube
   Cube.calculateColor();
   for(int i=0;i<6;i++)
   {
  if (Cube.coloredFace[i])
  {
   g.setColor( new Color( Cube.colors[i][0],
                  Cube.colors[i][1],
                  Cube.colors[i][2]));
   drawface(g,i);
   }
    }
    //delay
    if (k < nScenes-1)
     {
      delay();
      //erase cube: we repaint everything in
      //the background color.
       g.setColor(Color.pink);
       g.fillRect(0,0,xWin,yWin);
   }//end of if
  }//end of for nScenes
 }//end of paint

}//end of class CubeOOP
```

24 ***Exercise*** *Play with the code. Notice that the inner class Cube is an encapsulation of variables and methods all related with one purpose: to enable the drawing of a cube. The main method paint has been divided into various methods to make it more readable. Notice the form as the methods of the inner class are called from the main application.*

25 ***Cube as a prototype***

Every inner class can be used as a new type or prototype to create instances of the class. So, let us populate the world with some few cubes, a number that can be enlarged at will depending on the available memory. The code follows:

```
/*Program D25 CubesOOP

We present a second  OOP version of
a flying rotating solid cube:
the inner class Cube is
used as a prototype to build many cubes.
*/

import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Polygon;
import java.util.Random;

public class CubesOOP extends Applet
{

  private static final long serialVersionUID = 1L;


/*
Our basic cube is the unitary cube at the origin.
It is composed of 8 points or vertexes = oPoints3d[][],
12 edges = edges[][] and 6 faces = faces[][].
Points are numbered from 0 to 7.
These numbers are used in
edges, which  records the pairs of
points that are joined by a line.
A face is determined by its 4 corners.

All other cubes are transformations of this
fundamental cube. The form as this is done
appears in the class Cube.

The Cube as a prototype is defined as
an inner class in line 238.

*/

//The points of the fundamental cube
```

```
private static   double oPoints3d[][]=
{
 {0,0,0},
 {1,0,0},
 {0,1,0},
 {0,0,1},
 {1,1,0},
 {1,0,1},
 {0,1,1},
 {1,1,1}
};

/*
//The edges of the fundamental cube
private static int edges[][] =
   {{0,1},
    {0,2},
    {0,3},
    {1,4},
    {1,5},
    {2,6},
    {2,4},
    {3,5},
    {3,6},
    {4,7},
    {5,7},
    {6,7}};

*/

//The faces of the fundamental cube
private static int faces[][] =
   {{0,1,4,2},
    {0,2,6,3},
    {0,3,5,1},
    {3,5,7,6},
    {1,4,7,5},
    {2,6,7,4}};
```

```
 //Working instance of CubesOOP
 static  CubesOOP CubesOOP   = new CubesOOP();

 //Measures of the window
 private    int   xWin = 500;
 private    int   yWin = 400;
 private   int nScenes ;
 private   int delay ;

 //Up to 10 cubes can be built
 private   Cube[] vCube = new Cube[10];

//The window is enlarged.
//This method is fired automatically
//at the beginning
 public void init()
  {
   setSize(xWin,yWin);
  }

 //Each cube must have its initial position
 private   void rePosit(Cube cc, int m)
 {
  //System.out.println("rePosit(m)" + m);
  cc.oPosition[0] = 0;
  cc.oPosition[1] = 15 *m;
  cc.oPosition[2] = -60 *m;
  cc.cube = cc.locateCube(cc.cube, cc.oPosition);
 }

 //The cube is rotated, translated and projected
 private    void modifyCube(Cube cc, int m)
 {
  cc.theta = cc.theta + cc.deltaTheta;
  cc.cube = cc.rotCube(cc.points3d, cc.axisRot, cc.theta);
  //cc.printCube(cc.cube, " after rotation");
  rePosit(cc,m);
  //cc.printCube(cc.cube , " after relocation");
  for(int i = 0; i<3;i++)
```

```
  cc.vTrans[i] = cc.vTrans[i] + cc.deltaTrans[i];
  cc.cube = cc.translateCube(cc.cube,  cc.vTrans);
  //cc.printCube(cc.cube, " after translation");
  cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
  //cc.printCube(cc.cubeProj,  " after projection");
 }


 //Regulation of the velocity of movement
 private    void delay()
 {
 try
     {
   Thread.sleep(delay );
     }
     catch (InterruptedException e)
     {
   e.printStackTrace();
     }
 }

 //face ii of the cube is drawn
 private    void drawface(Graphics g, Cube cc, int ii)
 {
 for(int l =0; l<4; l++)
  {
cc.pointsX[l] = 100
             -cc.shade2d[faces[ii][l]  ] [1];
 cc.pointsY[l] = 100
            -cc.shade2d[ faces[ii][l]  ] [2];
   //System.out.println(pointsX[l] + " " + pointsY[l]);
  }
  //A polygon is defined by two vectors and an integer
   //number:
   //The first vector lists the x-coordinates of the
   //corners;
   //the second lists  the y-coordinates. The integer is
   //the number of points.
   //If the last point is different than the first,
   //the first point is closed with the last.
```

```
   Polygon face = new Polygon(cc.pointsX,
                      cc.pointsY, 4);
   g.fillPolygon(face);
 }

 //This is the main method
 public void paint (Graphics g)
 {
  //Number of scenes of the presentation.
  nScenes = 500;
  //Delay between two scenes
  delay =  120;
  //Number of cubes
  int nCubes = 5;
  System.out.println("Solid flying rotating OOP cubes");

  //backgound color
  g.setColor(Color.pink);
  g.fillRect(0,0,400,400);
  //Cubes are initialized
  for(int m =0; m < nCubes; m++)
   {
    final Cube cc =    new Cube(m);
    vCube[m] = cc;
   }
  //Loop for the scenes
  for(int k = 0; k<nScenes; k++)
  {
   //System.out.println("scene " + k);
   //nCubes are displayed
   for(int m = 0; m < nCubes; m++)
   {
//A working cube cc is initialized
Cube  cc = new Cube(m);
//The working cube is re-set.
    cc = vCube[m];
    //Movement begins
    modifyCube(cc,m);
    //Projection is calculated
    cc.shade2d =  cc.round(cc.cubeProj );
```

```
    //printShade();
    //The cube is drawn
    cc.calculateColor();
    for(int i=0;i<6;i++)
     {
    if (cc.coloredFace[i])
     {
      g.setColor( new Color( cc.colors[i][0],
                    cc.colors[i][1],
                    cc.colors[i][2]));
      drawface(g,cc,i);
     }
      }
    //Worked cube is kept to memory
    //Erase this instruction and explain the result.
    vCube[m] = cc;
    }
    //delay  + erase drawing
    if (k < nScenes-1)
     {
      delay();
      //erase cube: we repaint everything in
      //the background color.
       g.setColor(Color.pink);
       g.fillRect(0,0,xWin,yWin);
     }//end of if
 }//end of for nScenes
}//end of paint


//=============Inner class Cube==============

/*Variables and methods of the class cube are
  encapsulated in the same unit.
  Please, imagine a word populated with cubes, spheres,
  buildings, dogs and cats. Each class is defined
  following the pattern in here.
  All definitions are non static, i.e., they are
  used as erasable workbenches and resettable tools. */
public   class Cube
```

```
 {
  private double points3d[][] = new double[8][3];
  private   int shade2d[][] = new int[8][3];

//  All cubes are derived from the basic cube by rotations
//  and translations
  private double cube[][]  = new double[8][3];
  private   double  cubeProj[][] = new double[8][3];

  private double oPosition[] = new double[3];
  private   double axisRot[] = new double[3];
  private   double axisProj[]  = new double[3];
  private   double deltaTrans[]  = new double[3];
  private   double vTrans[]  = new double[3];
  private   double theta  = 0;
  private   double deltaTheta;

  private   double basisRot[][], basisProj[][]
               = new double[3][3];
  private   double matrixRotz[][],
  matrixRot[][],
  matrixProj[][] = new double[3][3];
//  The matrix of a projection along the z-axis
  private     double J[][]=
        { {1,0,0},
          {0,1,0},
          {0,0,0}};

  private    double scale;

  int colors[][] = new int[6][3];
  private   int pointsX[]  = new int[6];
  private   int pointsY[]  = new int[6];
//  How to color faces
  private    double maxX;
  private   int nExtreme;
  private   boolean maxPoints[]  = new boolean[8];
  private   boolean coloredFace[]  = new boolean[6];
```

```
 //Every implementation of Cube is initialized here.
 //Parameter m is used for the initial position
 //of cube number m
 Cube(int m)
 {
  scale = 50;
  defineColors();
  for(int i=0; i<3;i++)
   oPosition[i] = 0;
  oPosition[2] = 20*m;
  for(int i=0; i<3;i++)
  vTrans[i] = 0;
  for(int i=0; i<8;i++)
  for(int j=0; j<3;j++)
  points3d[i][j] = scale*oPoints3d[i][j];
//  printCube();
//  The 3d-graphic is rotated along the axisRot
//  an angle theta that is incremented by deltaTheta
  theta = 0;
  deltaTheta = 0.1;
  axisRot[0] = 1;
  axisRot[1]=1;
  axisRot[2]=1;
//  The 3d-graphic is projected over the screen
//  which is perpendicular to the x-axis.
  axisProj[0] = 1;
  axisProj[1]=0;
  axisProj[2]=0;
//  The 3d cube flies along vectTrans[]
  deltaTrans[0] = 0;
  deltaTrans[1] = -1;
  deltaTrans[2] = 0;

  }


 Random r  = new Random();

//The transpose of a matrix
private    double[][] matrixTranspose(double m[][])
```

```
{
double[][] helpM = new double[3][3];
for(int i=0; i<3; i++)
for(int j=0; j<3; j++)
helpM[i][j] =m[j][i];
return helpM;
}

//The multiplication of two matrixes
private  double[][] matrixMult(double p[][], double q[][])
{
 double[][] helpM = new double[3][3];

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

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

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



//To find a matrix of a rotation, one must find first
//an orthogonal bases with 3 vectors
```

```
//The third vector is the axis of rotation
private    double[] thirdVector(double[] v)
{
 double a = v[0];
 double b = v[1];
 double c = v[2];
 double norm = Math.sqrt(a*a + b*b + c*c);
 //System.out.println("norm = " + norm);
 double[] helpV = new double[3];
 helpV[0] = a/norm;
 helpV[1]= b/norm;
 helpV[2] = c/norm;
 /*System.out.println("Third vector " + helpV[0]+
          helpV[1]+helpV[2]);*/
 return helpV;
}

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

```
 double norm = Math.sqrt(x*x + y*y + z*z);

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

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


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

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

```
}


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

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

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

```
/*A 3d cube exists only in the imagination:
* it is not seen.
* One sees a shade or projection of it over the screen.
* The shade is defined by 2d points. The edges of
* the 2d shade are inherited from the edges of the
* 3d cube.
* The 3d cube is projected into a 2d plane.
* The resultant figure must be rendered to the screen.
* A projection is done along  an axis that crosses
* through 0.
* Our projections are observed from infinite:
* we use no perspective corrections. */
private     double[][] mProj(double w[] )
{
 basisProj = basisB(w);
 double[][]  helpM = new double[3][3];
 helpM = matrixMult(J,matrixTranspose(basisProj));
 helpM = matrixMult(basisProj,helpM);
 //printMatrix(helpM, "Proj");
 //Test: Proj*Proj must be equal to Proj
 /*helpM = matrixMult(helpM,helpM);
 printMatrix(helpM, "Proj* Proj");*/
 return helpM;
}

//The cube c is rotated around axis v an angle theta
private double[][] rotCube(double c[][],
                           double v[], double theta)
{
 double[]  helpV = new double[3];
 double cc[][] = new double[8][3];
 matrixRot = mRotation( v,theta);
 //printMatrix(matrixRot, "matrixRot");
 //The 8 points of the cube must be rotated
 for(int i=0; i<8; i++)
 {
  for(int j=0; j<3; j++)
  helpV[j] = c[i][j];
  helpV = matrixVector(matrixRot, helpV);
  for(int j=0; j<3; j++)
```

```
  cc[i][j] = helpV[j];
 }
 //printCube(cc, " after rot");
 return cc;
}


//The cube c is translated   deltaV
private double[][] locateCube(double c[][],
                                        double oPosition[])
{
 double cc[][] = new double[8][3];
 for(int i=0; i<8; i++)
 for(int j=0; j<3; j++)
 cc[i][j] = c[i][j] + oPosition[j];
 return cc;
}


//The cube c is translated   deltaV
private double[][] translateCube(double c[][],
                                        double deltaV[])
{
 double cc[][] = new double[8][3];
 for(int i=0; i<8; i++)
 for(int j=0; j<3; j++)
 cc[i][j] = c[i][j] + deltaV[j];
 return cc;
}


//The cube c is projected along axis w
private double[][] projCube(double c[][],  double w[])
{
 double[]  helpV = new double[3];
 double cc[][] = new double[8][3];
 matrixProj = mProj(w);
 for(int i=0; i<8; i++)
  {
   for(int j=0; j<3; j++)
```

```
   helpV[j] = c[i][j];
   helpV = matrixVector(matrixProj, helpV);
   for(int j=0; j<3; j++)
   cc[i][j] = helpV[j];
  }
 return cc;
}

//The rendering procedure accepts integer values
private     int[][] round(double c[][])
{
 int[][] helpM = new int [8][3];
 for(int i=0;i<8;i++)
 for(int j=0; j<3; j++)
 helpM[i][j] = (int) Math.round(c[i][j]);
 return helpM;
}

private   void printCube(double c[][], String a)
{
 System.out.println("This is the cube" +a);
 for(int i=0;i<8;i++)
 System.out.println(c[i][0] + " " + c[i][1]+ " "
                                        + c[i][2]);
}

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

private void printShade()
{
 System.out.println("This is the shade");
 for(int i=0;i<8;i++)
 System.out.println(shade2d[i][1] + " "
                                   + shade2d[i][2]);
```

```
}

//Colors to paint the faces are defined
private int[][] defineColors()
{
 for(int l = 0; l<6;l++)
  {
   //We use random colors.
   //A color can be defined by three numbers,
   //each one from 0 to 255.
   //We has 6 faces, so we need
   //a matrix to keep colors.

   int c1,c2,c3;
   c1 = r.nextInt(226);
   c2 = r.nextInt(226);
   c3 = r.nextInt(226);
   //Colors are memorized
   colors[l][0]= c1;
   colors[l][1]= c2;
   colors[l][2]= c3;
  }
 return colors;
}

//what faces must be colored
private    void marked()
{
 for(int f = 0; f<6; f++)
 coloredFace[f] = false;
 for(int f = 0; f<6;f++)
 {
  for(int p = 0; p<4;p++)
  {
   for(int c = 0; c<8; c++)
    {
     if (maxPoints[c] == true)
     {
       if (faces[f][p] == c )
       coloredFace[f] = true;
```

```
       //System.out.println("coloring");
      }
    }
   }
  }
}


//The x coordinate of all points are measured
//and the extreme ones are recorded.
private    void calculateColor()
{
 //The maximum value of coordinate x
 maxX =-1000;
 for(int l = 0; l<8; l++)
  {
   if (cube[l][0] > maxX)
    {
     maxX = cube[l][0];
    }
   }
 nExtreme = 0;
 //Number of extreme points
 for(int l = 0; l<8;l++)
  {
   maxPoints[l] = false;
   if (Math.abs(cube[l][0] - maxX )< 2)
   {
    nExtreme = nExtreme +1 ;
    maxPoints[l] = true;
   }
  }
  marked();
 }


 }//=============End of inner class Cube==============

}//end of class CubesOOP
```

26 ***Exercise*** *Play with the code. Please, pay attention to the form as our cubes are constructed: every cube is a modification ab initio of the fundamental cube. Observe that a working cube cc is defined as workbench and with it everything is done.*

27 ***Exercise*** *Modify the previous code to enable the cubes to fly each one with its own velocity, direction and size.*

3.2 The hand

Each human hand has 27 bones: 8 small bones that form the palm plus 19 other long bones that go in the metacarpals and the phalanges. The thumb has one metacarpal and 2 phalanges while the other fingers have one metacarpal and 3 phalanges. These bones are joined by muscles and tendons which are held in place by diverse packing structures.

We will pay attention to very simple questions about the structure of the hand that naturally relate anatomy with evolution and design, evolutionism and creationism. Let us try to figure out in first place the problem of devising a joint. This is a problem of design that has been very important in industry, engineering and medicine.

28 ***Joints***

A good approach is to begin with elementary proposals. So, we see in the next figure a plan for a joint that is both simple and effective:

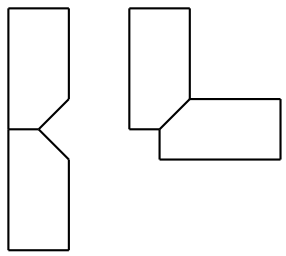


Figure 3.1. The purpose of a joint is to allow bending of the body. Here we have a bistable joint.

By looking at the figure, we see that the depicted joint offers two stable positions. This is not like the joints of our body that allow a continuous of positions and can be gradually bended. Since the freedom for gradual bending is more advantageous than bi-stability, we immediately recognize that joints must be curved. Anyway, we have too many choices, some of which appear in the next figure.

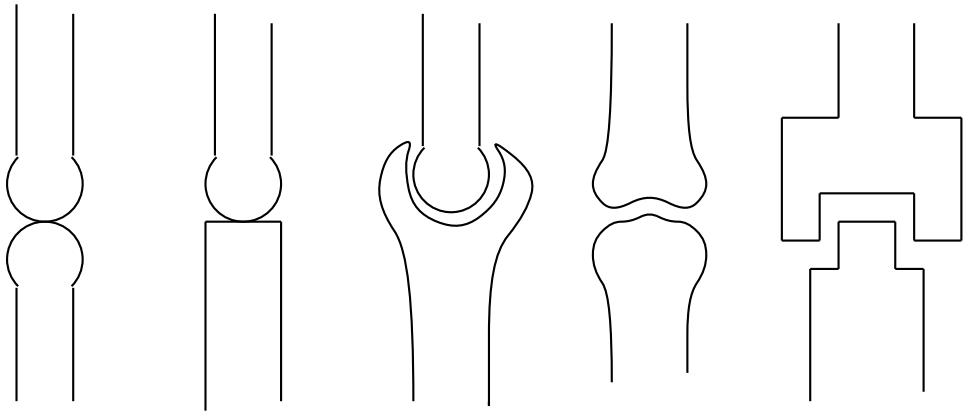


Figure 3.2. Sections of some circular motifs for possible artificial joints

The aesthetic simplicity of our proposals contrast with the delicate but amazing complexity of real bones: to begin with, no two bones of the body are equal and so each bone can be singled out from the rest.

Why are the bones of the body so complex and so distinct one from another?

An answer could be this: bones form part of a highly functional organism and organisms, natural and artificial, are such that they conform complex structures with complex functions and with difficulty one finds one piece equal to another one.

Let us notice that we explain structural complexity and diversity by functional complexity. This is a plainly scientific proposal that is immediately picked up by the evolutionary theory:

If we devise an evolutionary environment in which reproduction depends on functional complexity, we could expect the arising of entities with highly complex structures that consists of very diverse pieces.

We have formulated a very abstract evolutionary principle that is nevertheless reasonable. Can we accommodate some science between this belief and our left hand? Yes, we can but we must go rather slowly (our work and ideas are not for

those that want the pristine and whole truth all at a sudden). In a first step, let us draw a finger, which is our first coordinated entity made of blocks.

29 ***A finger***

The next code displays a finger that bends. It is composed of three modified cubes.

```
/*Program D29 finger

A finger is a   set of 3 modified cubes
that are posited in tandem and that move
coordinately.

Since we use finite precision, all our moves are
approximations. Therefore, a multiplication of errors
is expected if we use modifications in tandem.
That is why  every scene is built ab initio by
modification of constant entities at the origin.
The finger is located and moves at the origin
and points downwards over the screen.

*/

import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Polygon;
import java.util.Random;

public class finger extends Applet
{

  private static final long serialVersionUID = 1L;


/*
Our basic unit is    the unitary cube  at the origin.
It is composed of 8 points or vertexes = oPoints3d[][],
```

```
12 edges = edges[][] and 6 faces = faces[][].
Points are numbered from 0 to 7.
These numbers are used in
edges, which  records the pairs of
points that are joined by a line.
A face is determined by its 4 corners.

All other cubes are transformations of this
fundamental cube. The form as this is done
appears in the class Cube.

The Cube as a prototype is defined as
an inner class in line 300.

*/

//The points of the fundamental cube
private static   double oPoints3d[][]=
{
 {0,0,0},
 {1,0,0},
 {0,1,0},
 {0,0,1},
 {1,1,0},
 {1,0,1},
 {0,1,1},
 {1,1,1}
};

/*
//The edges of the fundamental cube
private static int edges[][] =
   {{0,1},
    {0,2},
    {0,3},
    {1,4},
    {1,5},
    {2,6},
    {2,4},
    {3,5},
```

```
    {3,6},
    {4,7},
    {5,7},
    {6,7}};

*/

//The faces of the fundamental cube
private static int faces[][] =
   {{0,1,4,2},
    {0,2,6,3},
    {0,3,5,1},
    {3,5,7,6},
    {1,4,7,5},
    {2,6,7,4}};



 //Measures of the window
 private    int   xWin = 500;
 private    int   yWin = 400;
 private   int nScenes ;
 private   int delay ;

 //Our finger is a composition of 3 cubes or phalanges
 private   Cube[] phalanges   = new Cube[3] ;

//The window is enlarged.
//This method is fired automatically
//at the beginning
 public void init()
  {
   setSize(xWin,yWin);
  }

 //The only allowed movement of a phalange is
 //inclination, forth and back, in within bounds.
 //Inclination of given phalange governs the
 //rest of the finger
```

```
private   void inclination(Cube cc, int m)
{
 //Inclination is equal to a rotation
 //Axis of rotation: the y axis.
cc.axisRot[0] = 0;
 cc.axisRot[1]=1;
 cc.axisRot[2]=0;
 switch(m)
 {
  case 0: cc.phi[0]  = cc.phi[0]  + cc.deltaPhi[0] ;
          break;
  case 1: cc.phi[1]  = cc.phi[0] + cc.phi[1]
                                   + cc.deltaPhi[1] ;
          break;
  case 2: cc.phi[2]  = cc.phi[1] + cc.phi[2]
                                   + cc.deltaPhi[2] ;
          break;
 }
 cc.cube = cc.rotCube(cc.points3d, cc.axisRot,
                                cc.phi[m] );
}

//For visualization, the finger is rotated
//around the z axis.
private   void rotation(Cube cc, int m)
{
 //Rotation about the z axis.
cc.axisRot[0] = 0;
 cc.axisRot[1]=0;
 cc.axisRot[2]=1;
 cc.theta = cc.theta + cc.deltaTheta;
 cc.cube = cc.rotCube(cc.cube, cc.axisRot, cc.theta);
}

//Each phalange must have its initial position
//that depends on the position of the phalanges below
private   void rePosit(Cube cc, int m)
{
 //System.out.println("rePosit(m)" + m);
cc.oPosition[0]  = 0;
```

```
   cc.oPosition[1]  = 0;
   cc.oPosition[2]  =0;
 switch(m)
  {
   case 0:   break;
   case 1: {
   cc.oPosition[0]  =  phalanges[0].cube[3][0];
   cc.oPosition[1]  =  phalanges[0].cube[3][1];
   cc.oPosition[2]  =  phalanges[0].cube[3][2];
   break;
       }
   case 2: {
   cc.oPosition[0]  =  phalanges[1].cube[3][0];
   cc.oPosition[1]  =  phalanges[1].cube[3][1];
   cc.oPosition[2]  =  phalanges[1].cube[3][2];
   break;
       }
  }

  cc.cube = cc.locateCube(cc.cube, cc.oPosition);

 }

 //The cube is inclined, rotated, translated and projected
 private    void modifyCube(Cube cc, int m)
 {
  inclination(cc,m);
  rotation(cc,m);

  //cc.printCube(cc.cube, " after rotation");
  rePosit(cc,m);
  //cc.printCube(cc.cube , " after relocation");
  for(int i = 0; i<3;i++)
  cc.vTrans[i] = cc.vTrans[i] + cc.deltaTrans[i];
  cc.cube = cc.translateCube(cc.cube,  cc.vTrans);
  //cc.printCube(cc.cube, " after translation");
  cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
  //cc.printCube(cc.cubeProj,  " after projection");
 }
```

```
 //Regulation of the velocity of movement
 private    void delay()
 {
 try
     {
   Thread.sleep(delay );
     }
     catch (InterruptedException e)
     {
   e.printStackTrace();
     }
 }

 //face ii of the cube is drawn
 private    void drawface(Graphics g, Cube cc, int ii)
 {
 for(int l =0; l<4; l++)
  {
cc.pointsX[l] = 100
             -cc.shade2d[faces[ii][l]  ] [1];
 cc.pointsY[l] = 100
            -cc.shade2d[ faces[ii][l]  ] [2];
   //System.out.println(pointsX[l] + " " + pointsY[l]);
  }
  //A polygon is defined by two vectors and an integer
   //number:
   //The first vector lists the x-coordinates of the
   //corners;
   //the second lists  the y-coordinates. The integer is
   //the number of points.
   //If the last point is different than the first,
   //the first point is closed with the last.
   Polygon face = new Polygon(cc.pointsX,
                       cc.pointsY, 4);
   g.fillPolygon(face);
 }

 //This is the main method
 public void paint (Graphics g)
```

```
 {
  //Number of scenes of the presentation.
  nScenes = 500;
  //Delay between two scenes
  delay =  1000;
  //Number of cubes in the finger
  int nCubes = 3;
  System.out.println("Bending finger");

  //backgound color
  g.setColor(Color.pink);
  g.fillRect(0,0,400,400);
  //Cubes are initialized
  for(int m =0; m < nCubes; m++)
   {
    final Cube cc =    new Cube(m);
    phalanges[m] = cc;
   }
  //Loop for the scenes
  for(int k = 0; k<nScenes; k++)
  {
   //System.out.println("scene " + k);
   //nCubes are displayed
   for(int m = 0; m < nCubes; m++)
   {
//A working cube cc is initialized
Cube  cc = new Cube(m);
//The working cube is re-set.
    cc = phalanges[m];
    //Movement begins
    modifyCube(cc,m);
    //Projection is calculated
    cc.shade2d =  cc.round(cc.cubeProj );
    //printShade();
    //The cube is drawn
    cc.calculateColor();
    for(int i=0;i<6;i++)
     {
    if (cc.coloredFace[i])
     {
```

```
      g.setColor( new Color( cc.colors[i][0],
                     cc.colors[i][1],
                     cc.colors[i][2]));
      drawface(g,cc,i);
     }
      }
    phalanges[m] = cc; //worked cube is kept to memory
    }
    //delay  + erase drawing
    if (k < nScenes-1)
     {
      delay();
      //erase cube: we repaint everything in
      //the background color.
       g.setColor(Color.pink);
       g.fillRect(0,0,xWin,yWin);
     }//end of if
  }//end of for nScenes
 }//end of paint


 //=============Inner class Cube==============

 /*Variables and methods of the class cube are
   encapsulated in the same unit.
   Please, imagine a word populated with cubes, spheres,
   buildings, dogs and cats. Each class is defined
   following the pattern in here.
   All definitions are non static, i.e., they are
   used as erasable workbenches and resettable tools. */
 public   class Cube
 {
  private double points3d[][] = new double[8][3];
  private   int shade2d[][] = new int[8][3];

//  All cubes are derived from the basic cube by rotations
//  and translations
  private double cube[][]  = new double[8][3];
  private   double  cubeProj[][] = new double[8][3];
```

```
  private double oPosition[] = new double[3];
  private   double axisRot[] = new double[3];
  private   double axisProj[]  = new double[3];
  private   double deltaTrans[]  = new double[3];
  private   double vTrans[]  = new double[3];
  private   double theta  = 0;
  private   double deltaTheta;

  private   double basisRot[][], basisProj[][]
                = new double[3][3];
  private   double matrixRotz[][],
  matrixRot[][],
  matrixProj[][] = new double[3][3];
//  The matrix of a projection along the z-axis
  private     double J[][]=
        { {1,0,0},
          {0,1,0},
          {0,0,0}};

  private    double scale[] = new double[3];

  int colors[][] = new int[6][3];
  private   int pointsX[]  = new int[6];
  private   int pointsY[]  = new int[6];
//  How to color faces
  private    double maxX;
  private   int nExtreme;
  private   boolean maxPoints[]  = new boolean[8];
  private   boolean coloredFace[]  = new boolean[6];
  //Angle of inclination of phalange
  private   double phi[] = new double[3];;
  private   double deltaPhi[] = new double[3];;

 //Every implementation of Cube is initialized here.
 //Parameter m is used for the initial position
 //of cube number m
 Cube(int m)
 {
  //Original  form
  scale[0] = 8;
```

```
scale[1] = 8;
scale[2] = 30;
for(int i=0; i<8;i++)
for(int j=0; j<3;j++)
points3d[i][j] =  -scale[j]*oPoints3d[i][j];
defineColors();
//Original position
for(int i=0; i<3;i++)
 oPosition[i] = 0;
//Each phalange has its own angle of inclination
//with respect to its support
for(int i=0; i<3;i++)
{
phi[i] = 0;
deltaPhi[i]  = -0.1*(m+1);
}
//  The 3d cube flies along vectTrans[]
//Each cube has its own direction of translation
for(int i=0; i<3;i++)
vTrans[i] = 0;
deltaTrans[0] =  0;
deltaTrans[1] =  0;
deltaTrans[2] =  0;
//  printCube();
//  The 3d-graphic is rotated along the axisRot
//  an angle theta that is incremented by deltaTheta.
//  Each cube has its own velocity and direction of
//  rotation.
theta = 0;
deltaTheta = 0.1;
axisRot[0] = 0;
axisRot[1]=0;
axisRot[2]=1;
//  The 3d-graphic is projected over the screen
//  which is perpendicular to the x-axis.
axisProj[0] = 1;
axisProj[1]=0;
axisProj[2]=0;
```

```
  }


 Random r  = new Random();

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

//The multiplication of two matrixes
private  double[][] matrixMult(double p[][], double q[][])
{
 double[][] helpM = new double[3][3];

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

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

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

```
 return helpV;
}



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

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

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

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

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


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

 helpT = thirdVector(v);
 helpS = secondVector(v);
 helpF = firstVector(helpT,helpS);
```

```
 for(int i=0; i<3;i++)
 {
  helpM[i][0] = helpF[i];
  helpM[i][1] = helpS[i];
  helpM[i][2] = helpT[i];
 }
return helpM;
}


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

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

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

/*A 3d cube exists only in the imagination:
* it is not seen.
* One sees a shade or projection of it over the screen.
* The shade is defined by 2d points. The edges of
* the 2d shade are inherited from the edges of the
* 3d cube.
* The 3d cube is projected into a 2d plane.
* The resultant figure must be rendered to the screen.
* A projection is done along  an axis that crosses
* through 0.
* Our projections are observed from infinite:
* we use no perspective corrections. */
private     double[][] mProj(double w[] )
{
 basisProj = basisB(w);
 double[][]  helpM = new double[3][3];
 helpM = matrixMult(J,matrixTranspose(basisProj));
 helpM = matrixMult(basisProj,helpM);
 //printMatrix(helpM, "Proj");
 //Test: Proj*Proj must be equal to Proj
 /*helpM = matrixMult(helpM,helpM);
 printMatrix(helpM, "Proj* Proj");*/
 return helpM;
}

//The cube c is rotated around axis v an angle theta
private double[][] rotCube(double c[][],
                           double v[], double theta)
{
 double[]  helpV = new double[3];
 double cc[][] = new double[8][3];
 matrixRot = mRotation( v,theta);
 //printMatrix(matrixRot, "matrixRot");
```

```
 //The 8 points of the cube must be rotated
 for(int i=0; i<8; i++)
 {
  for(int j=0; j<3; j++)
  helpV[j] = c[i][j];
  helpV = matrixVector(matrixRot, helpV);
  for(int j=0; j<3; j++)
  cc[i][j] = helpV[j];
 }
 //printCube(cc, " after rot");
 return cc;
}


//The cube c is translated   deltaV
private double[][] locateCube(double c[][],
                                        double oPosition[])
{
 double cc[][] = new double[8][3];
 for(int i=0; i<8; i++)
 for(int j=0; j<3; j++)
 cc[i][j] = c[i][j] + oPosition[j];
 return cc;
}


//The cube c is translated   deltaV
private double[][] translateCube(double c[][],
                                        double deltaV[])
{
 double cc[][] = new double[8][3];
 for(int i=0; i<8; i++)
 for(int j=0; j<3; j++)
 cc[i][j] = c[i][j] + deltaV[j];
 return cc;
}


//The cube c is projected along axis w
private double[][] projCube(double c[][],  double w[])
```

```
{
 double[]  helpV = new double[3];
 double cc[][] = new double[8][3];
 matrixProj = mProj(w);
 for(int i=0; i<8; i++)
  {
   for(int j=0; j<3; j++)
   helpV[j] = c[i][j];
   helpV = matrixVector(matrixProj, helpV);
   for(int j=0; j<3; j++)
   cc[i][j] = helpV[j];
  }
 return cc;
}

//The rendering procedure accepts integer values
private     int[][] round(double c[][])
{
 int[][] helpM = new int [8][3];
 for(int i=0;i<8;i++)
 for(int j=0; j<3; j++)
 helpM[i][j] = (int) Math.round(c[i][j]);
 return helpM;
}

private   void printCube(double c[][], String a)
{
 System.out.println("This is the cube" +a);
 for(int i=0;i<8;i++)
 System.out.println(c[i][0] + " " + c[i][1]+ " "
                                        + c[i][2]);
}

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

```
private void printShade()
{
 System.out.println("This is the shade");
 for(int i=0;i<8;i++)
 System.out.println(shade2d[i][1] + " "
                                  + shade2d[i][2]);
}

//Colors to paint the faces are defined
private int[][] defineColors()
{
 for(int l = 0; l<6;l++)
  {
   //We use random colors.
   //A color can be defined by three numbers,
   //each one from 0 to 255.
   //We has 6 faces, so we need
   //a matrix to keep colors.

   int c1,c2,c3;
   c1 = r.nextInt(226);
   c2 = r.nextInt(226);
   c3 = r.nextInt(226);
   //Colors are memorized
   colors[l][0]= c1;
   colors[l][1]= c2;
   colors[l][2]= c3;
  }
 return colors;
}

//what faces must be colored
private    void marked()
{
 for(int f = 0; f<6; f++)
 coloredFace[f] = false;
 for(int f = 0; f<6;f++)
 {
  for(int p = 0; p<4;p++)
```

```
  {
   for(int c = 0; c<8; c++)
    {
     if (maxPoints[c] == true)
     {
       if (faces[f][p] == c )
       coloredFace[f] = true;
       //System.out.println("coloring");
     }
   }
  }
 }
}


//The x coordinate of all points are measured
//and the extreme ones are recorded.
private    void calculateColor()
{
 //The maximum value of coordinate x
 maxX =-1000;
 for(int l = 0; l<8; l++)
  {
   if (cube[l][0] > maxX)
    {
     maxX = cube[l][0];
    }
   }
 nExtreme = 0;
 //Number of extreme points
 for(int l = 0; l<8;l++)
  {
   maxPoints[l] = false;
   if (Math.abs(cube[l][0] - maxX )< 2)
   {
    nExtreme = nExtreme +1 ;
    maxPoints[l] = true;
   }
  }
  marked();
```

```
 }

 }//=============End of inner class Cube==============

}//end of class finger
```

30 ***Exercise*** *Run the previous program and play with the code. Discover how to stop the movement just before self-penetration begins.*

31 ***An OOP version of a finger*** *Let us write the OOP version of the previous program that encodes for a finger. The code follows.*

```
/*Program D31 FingerOOP

This is an OOP version of a finger.

A finger is a   set of 3 modified cubes
that are posited in tandem and that move
coordinately.

Since we use finite precision, all our moves are
approximations. Therefore, a multiplication of errors
is expected if we use modifications in tandem.
That is why  every scene is built ab initio by
modification of constant entities at the origin.
The finger is located and moves at the origin
and points downwards over the screen.

The inner class Finger is presented in line 248.
A finger is composed of 3 cubes. The class Cube
appears in line 359

The main program follows.
```

```
*/

import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Polygon;
import java.util.Random;

public class FingerOOP extends Applet
{

  public static final long serialVersionUID = 1L;


 //This constructor is necessary when one have various
 //nested classes. In our case, we have a phalange,
 // which is used by the class Finger,
 // which is used by the main procedure in here.
 public FingerOOP()
 {

 }


/*
Our basic unit is    the unitary cube  at the origin.
It is composed of 8 points or vertexes = oPoints3d[][],
12 edges = edges[][] and 6 faces = faces[][].
Points are numbered from 0 to 7.
These numbers are used in
edges, which  records the pairs of
points that are joined by a line.
A face is determined by its 4 corners.

All other cubes are transformations of this
fundamental cube. The form as this is done
appears in the class Cube.


*/
```

```
//The points of the fundamental cube
private static   double oPoints3d[][]=
{
 {0,0,0},
 {1,0,0},
 {0,1,0},
 {0,0,1},
 {1,1,0},
 {1,0,1},
 {0,1,1},
 {1,1,1}
};

/*
//The edges of the fundamental cube
private static int edges[][] =
   {{0,1},
    {0,2},
    {0,3},
    {1,4},
    {1,5},
    {2,6},
    {2,4},
    {3,5},
    {3,6},
    {4,7},
    {5,7},
    {6,7}};

*/

//The faces of the fundamental cube
private static int faces[][] =
   {{0,1,4,2},
    {0,2,6,3},
    {0,3,5,1},
    {3,5,7,6},
    {1,4,7,5},
    {2,6,7,4}};
```

```
 //Measures of the window
 private    int   xWin = 500;
 private    int   yWin = 400;
 private   int nScenes ;
 private   int delay ;
 //Number of cubes in the finger
 private  static  int nPhalanges = 3;

 //The constructor of FingerOOP is invoked here.
 //A is now the instance of FingerOOP
 private static FingerOOP A = new FingerOOP();
 //Our finger is f1: it is an instance of Finger
 private static Finger f1 =  A.new Finger(nPhalanges);

//The window is enlarged.
//This method is fired automatically
//at the beginning

 public void init()
 {
  setSize(xWin,yWin);
 }

 //Regulation of the velocity of movement
 private    void delay()
 {
 try
     {
   Thread.sleep(delay );
     }
     catch (InterruptedException e)
     {
   e.printStackTrace();
     }
 }
```

```
 //face ii of the cube is drawn
 private    void drawface(Graphics g, Cube cc, int ii)
 {
 for(int l =0; l<4; l++)
  {
cc.pointsX[l] = 100
             -cc.shade2d[faces[ii][l]  ] [1];
 cc.pointsY[l] = 100
            -cc.shade2d[ faces[ii][l]  ] [2];
   //System.out.println(pointsX[l] + " " + pointsY[l]);
  }
  //A polygon is defined by two vectors and an integer
   //number:
   //The first vector lists the x-coordinates of the
   //corners;
   //the second lists  the y-coordinates. The integer is
   //the number of points.
   //If the last point is different than the first,
   //the first point is closed with the last.
   Polygon face = new Polygon(cc.pointsX,
                       cc.pointsY, 4);
   g.fillPolygon(face);
 }

 //This is the main method
 public void paint (Graphics g)
 {
  //Number of scenes of the presentation.
  nScenes = 20;
  //Delay between two scenes
  delay =  1000;

  System.out.println("Bending finger");

  //background color
  g.setColor(Color.pink);
  g.fillRect(0,0,400,400);
  //Cubes are initialized
```

```
  //Loop for the scenes
  for(int k = 0; k<nScenes; k++)
  {
   //System.out.println("scene " + k);
   //nCubes are displayed
   for(int m = 0; m < nPhalanges; m++)
   {
//A working cube cc is initialized
Cube  cc = new Cube(m);
//The working cube is re-set.
    cc = f1.phalanges[m];
    //Movement begins
    f1.modifyCube(cc,m);
    //Projection is calculated
    cc.shade2d =  cc.round(cc.cubeProj );
    //printShade();
    //The cube is drawn
    cc.calculateColor();
    for(int i=0;i<6;i++)
     {
    if (cc.coloredFace[i])
     {
      g.setColor( new Color( cc.colors[i][0],
                    cc.colors[i][1],
                    cc.colors[i][2]));
      drawface(g,cc,i);
     }
      }
    f1.phalanges[m] = cc; //worked cube is kept to memory
    }
    //delay  + erase drawing
    if (k < nScenes-1)
     {
      delay();
      //erase cube: we repaint everything in
      //the background color.
       g.setColor(Color.pink);
       g.fillRect(0,0,xWin,yWin);
     }//end of if
```

```
 }//end of for nScenes
}//end of paint



//=============Beginning of Inner class Finger============
/*Variables and methods of the class Finger are
encapsulated in the same unit.
Please, imagine a word populated with fingers, eyes,
hearts and so on. Each class is defined
following the pattern in here.
All definitions are non static, i.e., they are
used as erasable workbenches and resettable tools.

A finger is a coordinated set of 3 cubes.
The class cube is defined below.

All methods that are of exclusive use for fingers are
encapsulated here.
*/

public class Finger
{

//Our finger is a composition of 3 cubes or phalanges
private   Cube[] phalanges   = new Cube[3] ;


//When memory is alloted for a finger,
//all variables must be initialized
Finger(int nPhalanges)
{
for(int m =0; m < nPhalanges; m++)
  {
   final Cube cc =    new Cube(m);
   phalanges[m] = cc;
  }
}
// The only allowed movement of a phalange is
//inclination, forth and back, in within bounds.
```

```
 //Inclination of given phalange governs the
 //rest of the finger
 private   Cube inclination(Cube cc, int m)
 {
  //Inclination is equal to a rotation
  //Axis of rotation: the y axis.
 cc.axisRot[0] = 0;
  cc.axisRot[1]=1;
  cc.axisRot[2]=0;
  switch(m)
  {
   case 0: cc.phi[0]  = cc.phi[0]  + cc.deltaPhi[0] ;
           break;
   case 1: cc.phi[1]  = cc.phi[0] + cc.phi[1]
                                  + cc.deltaPhi[1] ;
           break;
   case 2: cc.phi[2]  = cc.phi[1] + cc.phi[2]
                                  + cc.deltaPhi[2] ;
           break;
  }
  cc.cube = cc.rotCube(cc.points3d, cc.axisRot,
                               cc.phi[m] );
  return cc;
 }

// For visualization, the finger is rotated
 //around the z axis.
 private   Cube rotation(Cube cc, int m)
 {
  //Rotation about the z axis.
 cc.axisRot[0] = 0;
  cc.axisRot[1]=0;
  cc.axisRot[2]=1;
  cc.theta = cc.theta + cc.deltaTheta;
  cc.cube = cc.rotCube(cc.cube, cc.axisRot, cc.theta);
  return cc;
 }

 //Each phalange must have its initial position
 //that depends on the position of the phalanges below it
```

```
private   void rePosit(Cube cc, int m)
{
 //System.out.println("rePosit(m)" + m);
cc.oPosition[0]  = 0;
  cc.oPosition[1]  = 0;
  cc.oPosition[2]  =0;
switch(m)
 {
  case 0:   break;
  case 1: {
  cc.oPosition[0]  =  f1.phalanges[0].cube[3][0];
  cc.oPosition[1]  =  f1.phalanges[0].cube[3][1];
  cc.oPosition[2]  =  f1.phalanges[0].cube[3][2];
  break;
      }
  case 2: {
  cc.oPosition[0]  =  f1.phalanges[1].cube[3][0];
  cc.oPosition[1]  =  f1.phalanges[1].cube[3][1];
  cc.oPosition[2]  =  f1.phalanges[1].cube[3][2];
  break;
      }
 }

 cc.cube = cc.locateCube(cc.cube, cc.oPosition);

}

//The cube is inclined, rotated, translated and projected
private    void modifyCube(Cube cc, int m)
{
 cc = inclination(cc,m);
 cc = rotation(cc,m);

 //cc.printCube(cc.cube, " after rotation");
 rePosit(cc,m);
 //cc.printCube(cc.cube , " after relocation");
 for(int i = 0; i<3;i++)
 cc.vTrans[i] = cc.vTrans[i] + cc.deltaTrans[i];
 cc.cube = cc.translateCube(cc.cube,  cc.vTrans);
 //cc.printCube(cc.cube, " after translation");
```

```
  cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
  //cc.printCube(cc.cubeProj,  " after projection");
 }

 }
 //=============End of Inner class Finger============

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

 //=============Beginning of Inner class Cube============


 public   class Cube
 {
  private double points3d[][] = new double[8][3];
  private   int shade2d[][] = new int[8][3];

//  All cubes are derived from the basic cube by rotations
//  and translations
  private double cube[][]  = new double[8][3];
  private   double  cubeProj[][] = new double[8][3];

  private double oPosition[] = new double[3];
  private   double axisRot[] = new double[3];
  private   double axisProj[]  = new double[3];
  private   double deltaTrans[]  = new double[3];
  private   double vTrans[]  = new double[3];
  private   double theta  = 0;
  private   double deltaTheta;

  private   double basisRot[][], basisProj[][]
               = new double[3][3];
  private   double matrixRotz[][],
  matrixRot[][],
  matrixProj[][] = new double[3][3];
//  The matrix of a projection along the z-axis
  private     double J[][]=
        { {1,0,0},
          {0,1,0},
          {0,0,0}};
```

```
  private    double scale[] = new double[3];

  int colors[][] = new int[6][3];
  private   int pointsX[]  = new int[6];
  private   int pointsY[]  = new int[6];
//  How to color faces
  private    double maxX;
  private   int nExtreme;
  private   boolean maxPoints[]  = new boolean[8];
  private   boolean coloredFace[]  = new boolean[6];
  //Angle of inclination of phalange
  private   double phi[] = new double[3];;
  private   double deltaPhi[] = new double[3];;

 //Every implementation of Cube is initialized here.
 //Parameter m is used for the initial position
 //of cube number m
 Cube(int m)
 {
  //Original  form
  scale[0] = 8;
  scale[1] = 8;
  scale[2] = 30;
  for(int i=0; i<8;i++)
  for(int j=0; j<3;j++)
  points3d[i][j] =  -scale[j]*oPoints3d[i][j];
  defineColors();
  //Original position
  for(int i=0; i<3;i++)
   oPosition[i] = 0;
  //Each phalange has its own angle of inclination
  //with respect to its support
  for(int i=0; i<3;i++)
  {
  phi[i] = 0;
  deltaPhi[i]  = -0.1*(m+1);
  }
  //  The 3d cube flies along vectTrans[]
  //Each cube has its own direction of translation
```

```
  for(int i=0; i<3;i++)
  vTrans[i] = 0;
  deltaTrans[0] =  0;
  deltaTrans[1] =  0;
  deltaTrans[2] =  0;
  //  printCube();
  //  The 3d-graphic is rotated along the axisRot
  //  an angle theta that is incremented by deltaTheta.
  //  Each cube has its own velocity and direction of
  //  rotation.
  theta = 0;
  deltaTheta = 0.1;
  axisRot[0] = 0;
  axisRot[1]=0;
  axisRot[2]=1;
  //  The 3d-graphic is projected over the screen
  //  which is perpendicular to the x-axis.
  axisProj[0] = 1;
  axisProj[1]=0;
  axisProj[2]=0;


  }


 Random r  = new Random();

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

//The multiplication of two matrixes
private  double[][] matrixMult(double p[][], double q[][])
{
```

```
 double[][] helpM = new double[3][3];

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

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

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



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

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

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

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

//The first vector is the crux product of the other 2.
private  double[] firstVector(double[] v, double[] w)
{
```

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


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

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


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

```
 mRotz[1][1] = Math.cos(theta);
 mRotz[1][2] = 0;
 mRotz[2][0] = 0;
 mRotz[2][1] = 0;
 mRotz[2][2] = 1;
 return mRotz;
}

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

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

/*A 3d cube exists only in the imagination:
* it is not seen.
* One sees a shade or projection of it over the screen.
* The shade is defined by 2d points. The edges of
* the 2d shade are inherited from the edges of the
* 3d cube.
* The 3d cube is projected into a 2d plane.
* The resultant figure must be rendered to the screen.
* A projection is done along  an axis that crosses
* through 0.
* Our projections are observed from infinite:
```

```
* we use no perspective corrections. */
private     double[][] mProj(double w[] )
{
 basisProj = basisB(w);
 double[][]  helpM = new double[3][3];
 helpM = matrixMult(J,matrixTranspose(basisProj));
 helpM = matrixMult(basisProj,helpM);
 //printMatrix(helpM, "Proj");
 //Test: Proj*Proj must be equal to Proj
 /*helpM = matrixMult(helpM,helpM);
 printMatrix(helpM, "Proj* Proj");*/
 return helpM;
}

//The cube c is rotated around axis v an angle theta
private double[][] rotCube(double c[][],
                           double v[], double theta)
{
 double[]  helpV = new double[3];
 double cc[][] = new double[8][3];
 matrixRot = mRotation( v,theta);
 //printMatrix(matrixRot, "matrixRot");
 //The 8 points of the cube must be rotated
 for(int i=0; i<8; i++)
 {
  for(int j=0; j<3; j++)
  helpV[j] = c[i][j];
  helpV = matrixVector(matrixRot, helpV);
  for(int j=0; j<3; j++)
  cc[i][j] = helpV[j];
 }
 //printCube(cc, " after rot");
 return cc;
}


//The cube c is translated   deltaV
private double[][] locateCube(double c[][],
                                          double oPosition[])
{
```

```
 double cc[][] = new double[8][3];
 for(int i=0; i<8; i++)
 for(int j=0; j<3; j++)
 cc[i][j] = c[i][j] + oPosition[j];
 return cc;
}


//The cube c is translated   deltaV
private double[][] translateCube(double c[][],
                                        double deltaV[])
{
 double cc[][] = new double[8][3];
 for(int i=0; i<8; i++)
 for(int j=0; j<3; j++)
 cc[i][j] = c[i][j] + deltaV[j];
 return cc;
}


//The cube c is projected along axis w
private double[][] projCube(double c[][],  double w[])
{
 double[]  helpV = new double[3];
 double cc[][] = new double[8][3];
 matrixProj = mProj(w);
 for(int i=0; i<8; i++)
  {
   for(int j=0; j<3; j++)
   helpV[j] = c[i][j];
   helpV = matrixVector(matrixProj, helpV);
   for(int j=0; j<3; j++)
   cc[i][j] = helpV[j];
  }
 return cc;
}

//The rendering procedure accepts integer values
private     int[][] round(double c[][])
{
```

```
 int[][] helpM = new int [8][3];
 for(int i=0;i<8;i++)
 for(int j=0; j<3; j++)
 helpM[i][j] = (int) Math.round(c[i][j]);
 return helpM;
}

private   void printCube(double c[][], String a)
{
 System.out.println("This is the cube" +a);
 for(int i=0;i<8;i++)
 System.out.println(c[i][0] + " " + c[i][1]+ " "
                                      + c[i][2]);
}

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

private void printShade()
{
 System.out.println("This is the shade");
 for(int i=0;i<8;i++)
 System.out.println(shade2d[i][1] + " "
                                  + shade2d[i][2]);
}

//Colors to paint the faces are defined
private int[][] defineColors()
{
 for(int l = 0; l<6;l++)
  {
   //We use random colors.
   //A color can be defined by three numbers,
   //each one from 0 to 255.
   //We has 6 faces, so we need
```

```
   //a matrix to keep colors.

   int c1,c2,c3;
   c1 = r.nextInt(226);
   c2 = r.nextInt(226);
   c3 = r.nextInt(226);
   //Colors are memorized
   colors[l][0]= c1;
   colors[l][1]= c2;
   colors[l][2]= c3;
  }
 return colors;
}

//what faces must be colored
private    void marked()
{
 for(int f = 0; f<6; f++)
 coloredFace[f] = false;
 for(int f = 0; f<6;f++)
 {
  for(int p = 0; p<4;p++)
  {
   for(int c = 0; c<8; c++)
    {
     if (maxPoints[c] == true)
     {
       if (faces[f][p] == c )
       coloredFace[f] = true;
       //System.out.println("coloring");
     }
   }
  }
 }
}


//The x coordinate of all points are measured
//and the extreme ones are recorded.
private    void calculateColor()
```

```
{
 //The maximum value of coordinate x
 maxX =-1000;
 for(int l = 0; l<8; l++)
  {
   if (cube[l][0] > maxX)
    {
     maxX = cube[l][0];
    }
   }
 nExtreme = 0;
 //Number of extreme points
 for(int l = 0; l<8;l++)
  {
   maxPoints[l] = false;
   if (Math.abs(cube[l][0] - maxX )< 2)
   {
    nExtreme = nExtreme +1 ;
    maxPoints[l] = true;
   }
  }
  marked();
 }


 }//=============End of inner class Cube==============


}//end of class FingerOOP
```

32 ***Exercise*** *In the previous program, the finger bends to simulate a closing movement. Modify the main program to have a simulation of a finger that closes and then reverses the movement.*

33 ***Challenge*** *Modify the main program to have a simulation of a finger that closes and then opens back while completing a round. Warning: to achieve a nice presentation, the faces of the phalanges and the phalanges themselves must be drawn in appropriate order. This introduces you to one fold of the problem of perspective in 3d graphics.*

34 ***A hand***

We already have developed the technology to devise a moving finger. Let us apply that technology to draw a cartoon of a moving hand in which all fingers are similar and an opposite thumb is lack. The code follows:

```
/*Program D34 protoHand

Our protohand has 5 equal fingers, a palm and an arm.
A finger has 3 equal phalanges.


Since we use finite precision, all our moves are
approximations. Therefore, a multiplication of errors
is expected if we use modifications in tandem.
That is why  every scene is built ab initio by
modification of a unique unmodifiable cube at the origin.
The hand is located and moves at the origin
and points downwards over the screen.

The inner class Hand is encoded beginning from line 377
The inner class Finger is presented in line 444.
A finger is composed of 3 cubes. The class Cube
appears in line 555

The main program follows.

*/

import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Polygon;
import java.util.Random;


public class protoHand extends Applet
{
```

```
 public static final long serialVersionUID = 1L;

 /*This constructor is necessary when one have various
  nested classes. In our case, we have the class Cube,
   which is used by the class Finger,
   which is used by the class Hand,
   which is used by the main class (protoHand) in here.*/
 public protoHand()
 {

 }


/*
Our basic unit is  the unitary cube at the origin.
It is composed of 8 points or vertexes = oPoints3d[][],
12 edges = edges[][] and 6 faces = faces[][].
Points are numbered from 0 to 7.
These numbers are used in
edges, which  records the pairs of
points that are joined by a line.
A face is determined by its 4 corners.

All other cubes are transformations of this
fundamental cube. The form as this is done
appears in the class Cube.


*/

//The points of the fundamental cube
private static   double oPoints3d[][]=
{
 {0,0,0},
 {1,0,0},
 {0,1,0},
 {0,0,1},
 {1,1,0},
 {1,0,1},
 {0,1,1},
```

```
 {1,1,1}
};

/*
//The edges of the fundamental cube
private static int edges[][] =
   {{0,1},
    {0,2},
    {0,3},
    {1,4},
    {1,5},
    {2,6},
    {2,4},
    {3,5},
    {3,6},
    {4,7},
    {5,7},
    {6,7}};

*/

//The faces of the fundamental cube
private static int faces[][] =
   {{0,1,4,2},
    {0,2,6,3},
    {0,3,5,1},
    {3,5,7,6},
    {1,4,7,5},
    {2,6,7,4}};



 //Measures of the window
 private    int   xWin = 600;
 private    int   yWin = 500;
 private   int nScenes ;
 private   int delay ;
 //Number of cubes in the finger
 private  static  int nPhalanges = 3;
```

```
 //The constructor of hand is invoked here.
 //A is now the instance of the main class hand
 private static protoHand A = new protoHand();
 //The inner class Hand is instantiated
 private static Hand  h = A.new Hand();
 //An auxiliary  instance of Finger
 private static Finger f1=  A.new Finger(nPhalanges);



//The window is enlarged.
//This method is fired automatically
//at the beginning
 public void init()
 {
  setSize(xWin,yWin);
 }

 //Regulation of the velocity of movement
 private    void delay(int delay)
 {
 try
     {
   Thread.sleep(delay );
     }
     catch (InterruptedException e)
     {
   e.printStackTrace();
     }
 }

 //face ii of the cube is drawn
 private    void drawface(Graphics g, Cube cc, int ii)
 {
 for(int l =0; l<4; l++)
  {
   cc.pointsX[l] = 200-cc.shade2d[faces[ii][l]][1];
   cc.pointsY[l] = 200-cc.shade2d[ faces[ii][l]][2];
   //System.out.println(pointsX[l] + " " + pointsY[l]);
```

```
  }
  //A polygon is defined by two vectors and an integer
   //number:
   //The first vector lists the x-coordinates of the
   //corners;
   //the second lists  the y-coordinates. The integer is
   //the number of points.
   //If the last point is different than the first,
   //the first point is closed with the last.
   Polygon face = new Polygon(cc.pointsX,
                      cc.pointsY, 4);
   g.fillPolygon(face);
 }

 private    void drawArm(Graphics g)
 {
// A working cube cc is initialized
Cube  cc = new Cube(1);
//The working cube is re-set.
   cc = h.arm ;
   //Movement begins
   f1.modifyCube(cc,0);
   cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
   //Projection is calculated
   cc.shade2d =  cc.round(cc.cubeProj );
   //printShade();
   //The cube is drawn
   cc.calculateColor();
   for(int i=0;i<6;i++)
    {
  if (cc.coloredFace[i])
   {
    g.setColor( new Color( cc.colors[i][0],
                   cc.colors[i][1],
                   cc.colors[i][2]));
    drawface(g,cc,i);
   }
     }
   //worked cube is kept to memory
   h.arm = cc;
```

```
 }

 private Cube repositPalm(Cube cc)
 {
  cc.oPosition[0]  =  h.arm.cube[3][0];
  cc.oPosition[1]  =  h.arm.cube[3][1]+5;
  cc.oPosition[2]  =   h.arm.cube[3][2];
  cc.cube = cc.translateCube(cc.cube, cc.oPosition);
  return cc;
}

 private    Cube updatePalm(Cube cc)
 {
 cc = f1.inclination(cc,1);
  cc = f1.rotation(cc,1);

  //cc.printCube(cc.cube, " after rotation");
   cc = repositPalm(cc);
  //cc.printCube(cc.cube , " after relocation");
  for(int i = 0; i<3;i++)
  cc.vTrans[i] = cc.vTrans[i] + cc.deltaTrans[i];
  cc.cube = cc.translateCube(cc.cube,  cc.vTrans);
  //cc.printCube(cc.cube, " after translation");
  cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
  //cc.printCube(cc.cubeProj,  " after projection");
 return cc;
 }

 private    void drawPalm(Graphics g)
 {
// A working cube cc is initialized
Cube  cc = new Cube(1);
//The working cube is re-set.
   cc = h.palm ;
   //Movement begins
   cc = updatePalm(cc);
   cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
   //Projection is calculated
   cc.shade2d =  cc.round(cc.cubeProj );
   //printShade();
```

```
   //The cube is drawn
   cc.calculateColor();
   for(int i=0;i<6;i++)
    {
  if (cc.coloredFace[i])
   {
    g.setColor( new Color( cc.colors[i][0],
                  cc.colors[i][1],
                  cc.colors[i][2]));
    drawface(g,cc,i);
   }
     }
   //worked cube is kept to memory
   h.palm = cc;
 }

 //Fingers are posited below the palm
 private    void locateFingers()
  {
   for(int i=0; i<5;i++)
for(int j=0; j<3;j++)
 h.finger[i].phalange[0].oPosition[j]
                                  =  h.palm.cube[3][j];
   for(int i=0; i<5;i++)
h.finger[i].phalange[0].oPosition[1]
        = h.finger[i].phalange[0].oPosition[1] -10*i ;
  }


 private    void drawFingers(Graphics g)
 {
  locateFingers();
  for(int j = 4; j>=0;j--)
  {
   for(int m = 0; m < nPhalanges; m++)
   {
//A working cube cc is initialized
Cube  cc = new Cube(m);
//The working cube is re-set.
    cc = h.finger[j].phalange[m];
```

```
    //Movement begins
    cc = h.finger[j].modifyCube(cc,m);
    //Projection is calculated
    cc.shade2d =  cc.round(cc.cubeProj );
    //printShade();
    //The cube is drawn
    cc.calculateColor();
    for(int i=0;i<6;i++)
     {
  if (cc.coloredFace[i])
   {
    g.setColor( new Color( cc.colors[i][0],
                  cc.colors[i][1],
                  cc.colors[i][2]));
    drawface(g,cc,i);
   }
     }
    //worked cube is kept to memory
    h.finger[j].phalange[m] = cc;
   }
  }
 }

 private    void drawHand(Graphics g)
 {
drawArm(g);
 drawPalm(g);
 drawFingers(g);
 }


 private    void dynamics(Graphics g)
 {
 //Loop for the scenes
  for(int k = 0; k<nScenes; k++)
  {
   //System.out.println("scene " + k);

drawHand(g);
    //delay  + erase drawing
```

```
    if (k < nScenes-1)
     {
      delay(delay);
      //erase cube: we repaint everything in
      //the background color.
       g.setColor(Color.pink);
       g.fillRect(0,0,xWin,yWin);
     }//end of if
  }//end of for nScenes
 }

 //This is the main method
 public void paint (Graphics g)
 {
  //Number of scenes of the presentation.
  nScenes = 20;
  //Delay between two scenes
  delay =  800;

  System.out.println("A proto-hand");
  //An instance of Hand is initialized


  //background color
  g.setColor(Color.pink);
  g.fillRect(0,0,xWin,yWin);
  dynamics(g);
  //To reverse the movement, we invert the angles
  //of rotation
  for(int j=0; j<3;j++)
   {
h.arm.deltaPhi[j] = -h.arm.deltaPhi[j];
h.palm.deltaPhi[j] = -h.palm.deltaPhi[j];
   }

  h.arm.deltaTheta = -h.arm.deltaTheta;
  h.palm.deltaTheta = -h.palm.deltaTheta;
  for(int i=0; i<5;i++)
  {
   for(int j=0; j<3;j++)
```

```
{
     h.finger[i].phalange[j].deltaPhi[j]  =
            -h.finger[i].phalange[j].deltaPhi[j];
     h.finger[i].phalange[j].deltaTheta =
             -h.finger[i].phalange[j].deltaTheta;
}
  }
  delay(delay);
  //erase cube: we repaint everything in
  //the background color.
   g.setColor(Color.pink);
   g.fillRect(0,0,xWin,yWin);
  dynamics(g);

 }//end of paint



 //=============Beginning of Inner class Hand============

 public class Hand
 {
 private Cube arm = new Cube(1);
 private Cube palm = new Cube(1);
 private Finger finger[] = new Finger[5];




 //When memory is alloted for a hand,
 //all variables must be initialized.
 //Lower classes are initialized by default,
 //but some  redefinitions could be necessary.
 Hand()
 {
  //The arm is initialized
   final Cube c1 =    new Cube(1);
   arm = c1;
       //Dimensions and location of the arm.
   arm.scale[0] = 8;
```

```
   arm.scale[1] = 30;
   arm.scale[2] = 100;
   for(int i=0; i<8;i++)
   for(int j=0; j<3;j++)
       arm.points3d[i][j] =  -arm.scale[j]*oPoints3d[i][j];


   //The  palm is initialized
   final Cube c2 =    new Cube(1);
   palm = c2;
   //Dimensions and location of the palm.
   palm.scale[0] = 8;
   palm.scale[1] = 40;
   palm.scale[2] = 60;
   for(int i=0; i<8;i++)
for(int j=0; j<3;j++)
  palm.points3d[i][j] =  -palm.scale[j]*oPoints3d[i][j];


   //Fingers are initialized
   for(int i=0; i<5;i++)
   {
   final  Finger f =   new Finger(nPhalanges);
   finger[i] = f;
   }




 }
 }

 //=============End of Inner class Hand============

 //=============Beginning of Inner class Finger===========
 /*Variables and methods of the class Finger are
 encapsulated in the same unit.
 All definitions are non static, i.e., they are
 used as erasable workbenches and resettable tools.
```

```
 A finger is a coordinated set of 3 cubes.
 The class cube is defined below.

 All methods that are of exclusive use for fingers are
 encapsulated here.
 */

 public class Finger
 {

 //Our finger is a composition of 3 cubes or phalanges
 private   Cube[] phalange    = new Cube[3] ;


 //When memory is allotted for a finger,
 //all variables must be initialized
 Finger(int nPhalanges)
 {
 for(int m =0; m < nPhalanges; m++)
   {
    final Cube cc =    new Cube(m);
    phalange[m] = cc;
   }
 }
// The only allowed movement of a phalange is
 //inclination, forth and back, in within bounds.
 //Inclination of given phalange governs the
 //rest of the finger
 private   Cube inclination(Cube cc, int m)
 {
  //Inclination is equal to a rotation
  //Axis of rotation: the y axis.
  cc.axisRot[0] = 0;
  cc.axisRot[1]=1;
  cc.axisRot[2]=0;
  switch(m)
   {
case 0: cc.phi[0]  = cc.phi[0]  + cc.deltaPhi[0] ;
           break;
case 1: cc.phi[1]  = cc.phi[0] + cc.phi[1]
```

```
                                    + cc.deltaPhi[1] ;
           break;
case 2: cc.phi[2]  = cc.phi[1] + cc.phi[2]
                                    + cc.deltaPhi[2] ;
           break;
  }
cc.cube = cc.rotCube(cc.points3d, cc.axisRot,
                              cc.phi[m] );
  return cc;
 }

    //For visualization, the finger is rotated
//around the z axis.
private   Cube rotation(Cube cc, int m)
{
 //Rotation about the z axis.
 cc.axisRot[0] = 0;
 cc.axisRot[1]=0;
 cc.axisRot[2]=1;
     cc.theta = cc.theta + cc.deltaTheta;
 cc.cube = cc.rotCube(cc.cube, cc.axisRot, cc.theta);
 return cc;
}

 //Each phalange must have its initial position
 //that depends on the position of the phalanges below it
 private   Cube rePosit(Cube cc, int m)
 {
   //System.out.println("rePosit(m)" + m);
 switch(m)
  {
   case 0:  {
      } break;
   case 1: {
cc.oPosition[0]  =   phalange[0].cube[3][0];
cc.oPosition[1]  =   phalange[0].cube[3][1];
    cc.oPosition[2]  =   phalange[0].cube[3][2];
   break;
       }
   case 2: {
```

```
cc.oPosition[0]  =   phalange[1].cube[3][0];
cc.oPosition[1]  =   phalange[1].cube[3][1];
cc.oPosition[2]  =   phalange[1].cube[3][2];
   break;
       }
  }
  cc.cube = cc.translateCube(cc.cube, cc.oPosition);
  return cc;
 }

 //The cube is inclined, rotated, translated, projected
 private    Cube modifyCube(Cube cc, int m)
 {
  cc = inclination(cc,m);
  cc = rotation(cc,m);

  //cc.printCube(cc.cube, " after rotation");
  rePosit(cc,m);
  //cc.printCube(cc.cube , " after relocation");
  for(int i = 0; i<3;i++)
  cc.vTrans[i] = cc.vTrans[i] + cc.deltaTrans[i];
  cc.cube = cc.translateCube(cc.cube,  cc.vTrans);
  //cc.printCube(cc.cube, " after translation");
  cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
  //cc.printCube(cc.cubeProj,  " after projection");
  return cc;
 }

 }

 //=============End of Inner class Finger============

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

 //=============Beginning of Inner class Cube============



 public   class Cube
 {
```

```
  private double points3d[][] = new double[8][3];
  private   int shade2d[][] = new int[8][3];

//  All cubes are derived from the basic cube by rotations
//  and translations
  private double cube[][]  = new double[8][3];
  private   double  cubeProj[][] = new double[8][3];

  private double oPosition[] = new double[3];
  private   double axisRot[] = new double[3];
  private   double axisProj[]  = new double[3];
  private   double deltaTrans[]  = new double[3];
  private   double vTrans[]  = new double[3];
  private   double theta  = 0;
  private   double deltaTheta;

  private   double basisRot[][], basisProj[][]
                = new double[3][3];
  private   double matrixRotz[][],
  matrixRot[][],
  matrixProj[][] = new double[3][3];
//  The matrix of a projection along the z-axis
  private     double J[][]=
        { {1,0,0},
          {0,1,0},
          {0,0,0}};

  private    double scale[] = new double[3];

  int colors[][] = new int[6][3];
  private   int pointsX[]  = new int[6];
  private   int pointsY[]  = new int[6];
//  How to color faces
  private    double maxX;
  private   int nExtreme;
  private   boolean maxPoints[]  = new boolean[8];
  private   boolean coloredFace[]  = new boolean[6];
  //Angle of inclination of phalange
  private   double phi[] = new double[3];;
  private   double deltaPhi[] = new double[3];;
```

```
//Every implementation of Cube is initialized here.
//Parameter m is used for the initial position
//of cube number m
Cube(int m)
{
 //Original  form
 scale[0] = 8;
 scale[1] = 8;
 scale[2] = 30;
 for(int i=0; i<8;i++)
 for(int j=0; j<3;j++)
 points3d[i][j] =  -scale[j]*oPoints3d[i][j];
 defineColors();
 //Original position
 for(int i=0; i<3;i++)
  oPosition[i] = 0;
 //Each phalange has its own angle of inclination
 //with respect to its support
 for(int i=0; i<3;i++)
 {
 phi[i] = 0;
 deltaPhi[i]  = -0.1*(m+1);
 }
 //  The 3d cube flies along vectTrans[]
 //Each cube has its own direction of translation
 for(int i=0; i<3;i++)
 vTrans[i] = 0;
 deltaTrans[0] =  0;
 deltaTrans[1] =  0;
 deltaTrans[2] =  0;
 //  printCube();
 //  The 3d-graphic is rotated along the axisRot
 //  an angle theta that is incremented by deltaTheta.
 //  Each cube has its own velocity and direction of
 //  rotation.
 theta = 0;
 deltaTheta = -0.1;
 axisRot[0] = 0;
 axisRot[1]=0;
```

```
  axisRot[2]=1;
  //  The 3d-graphic is projected over the screen
  //  which is perpendicular to the x-axis.
  axisProj[0] = 1;
  axisProj[1]=0;
  axisProj[2]=0;


  }


 Random r  = new Random();

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

//The multiplication of two matrixes
private  double[][] matrixMult(double p[][], double q[][])
{
 double[][] helpM = new double[3][3];

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

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

//The multiplication of a matrix times a vector
```

```
private double[] matrixVector(double m[][], double v[])
{
 double[]  helpV = new double[3];
 for(int i=0; i<3; i++)
 helpV[i] =0;
 for(int i=0; i<3; i++)
 for(int j=0; j<3; j++)
 helpV[i]  =helpV[i]  + m[i][j]*v[j];
 return helpV;
}



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

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

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

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

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


//An axis (a,b,c) generates a basis B
```

```
private    double[][] basisB(double v[])
{
 double[][]  helpM = new double[3][3];
 double[]   helpT, helpS, helpF = new double[3];

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


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

//A rotation is defined by an axis v that crosses
//through 0,
//and a counterclockwise angle theta of rotation
private double[][] mRotation(double v[], double theta)
{
 matrixRotz= mRot(theta);
```

```
 basisRot = basisB(v);
 //printMatrix(matrixRotz, "Rotz");
 //printMatrix(basisRot, "BasisRot");
 double[][]  helpM = new double[3][3];
 helpM = matrixMult(matrixRotz,
 matrixTranspose(basisRot));
 helpM = matrixMult(basisRot,helpM);

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

/*A 3d cube exists only in the imagination:
* it is not seen.
* One sees a shade or projection of it over the screen.
* The shade is defined by 2d points. The edges of
* the 2d shade are inherited from the edges of the
* 3d cube.
* The 3d cube is projected into a 2d plane.
* The resultant figure must be rendered to the screen.
* A projection is done along  an axis that crosses
* through 0.
* Our projections are observed from infinite:
* we use no perspective corrections. */
private     double[][] mProj(double w[] )
{
 basisProj = basisB(w);
 double[][]  helpM = new double[3][3];
 helpM = matrixMult(J,matrixTranspose(basisProj));
 helpM = matrixMult(basisProj,helpM);
 //printMatrix(helpM, "Proj");
 //Test: Proj*Proj must be equal to Proj
 /*helpM = matrixMult(helpM,helpM);
 printMatrix(helpM, "Proj* Proj");*/
 return helpM;
}
```

```
//The cube c is rotated around axis v an angle theta
private double[][] rotCube(double c[][],
                           double v[], double theta)
{
 double[]  helpV = new double[3];
 double cc[][] = new double[8][3];
 matrixRot = mRotation( v,theta);
 //printMatrix(matrixRot, "matrixRot");
 //The 8 points of the cube must be rotated
 for(int i=0; i<8; i++)
 {
  for(int j=0; j<3; j++)
  helpV[j] = c[i][j];
  helpV = matrixVector(matrixRot, helpV);
  for(int j=0; j<3; j++)
  cc[i][j] = helpV[j];
 }
 //printCube(cc, " after rot");
 return cc;
}



//The cube c is translated   deltaV
private double[][] translateCube(double c[][],
                                         double deltaV[])
{
 double cc[][] = new double[8][3];
 for(int i=0; i<8; i++)
 for(int j=0; j<3; j++)
 cc[i][j] = c[i][j] + deltaV[j];
 return cc;
}


//The cube c is projected along axis w
private double[][] projCube(double c[][],  double w[])
{
 double[]  helpV = new double[3];
 double cc[][] = new double[8][3];
```

```
 matrixProj = mProj(w);
 for(int i=0; i<8; i++)
  {
   for(int j=0; j<3; j++)
   helpV[j] = c[i][j];
   helpV = matrixVector(matrixProj, helpV);
   for(int j=0; j<3; j++)
   cc[i][j] = helpV[j];
  }
 return cc;
}

//The rendering procedure accepts integer values
private     int[][] round(double c[][])
{
 int[][] helpM = new int [8][3];
 for(int i=0;i<8;i++)
 for(int j=0; j<3; j++)
 helpM[i][j] = (int) Math.round(c[i][j]);
 return helpM;
}

private   void printCube(double c[][], String a)
{
 System.out.println("This is the cube" +a);
 for(int i=0;i<8;i++)
 System.out.println(c[i][0] + " " + c[i][1]+ " "
                                      + c[i][2]);
}

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

private void printShade()
{
```

```
 System.out.println("This is the shade");
 for(int i=0;i<8;i++)
 System.out.println(shade2d[i][1] + " "
                                + shade2d[i][2]);
}

//Colors to paint the faces are defined
private int[][] defineColors()
{
 for(int l = 0; l<6;l++)
  {
   //We use random colors.
   //A color can be defined by three numbers,
   //each one from 0 to 255.
   //We has 6 faces, so we need
   //a matrix to keep colors.

   int c1,c2,c3;
   c1 = r.nextInt(226);
   c2 = r.nextInt(226);
   c3 = r.nextInt(226);
   //Colors are memorized
   colors[l][0]= c1;
   colors[l][1]= c2;
   colors[l][2]= c3;
  }
 return colors;
}

//what faces must be colored
private    void marked()
{
 for(int f = 0; f<6; f++)
 coloredFace[f] = false;
 for(int f = 0; f<6;f++)
 {
  for(int p = 0; p<4;p++)
  {
   for(int c = 0; c<8; c++)
    {
```

```
     if (maxPoints[c] == true)
     {
       if (faces[f][p] == c )
       coloredFace[f] = true;
       //System.out.println("coloring");
     }
   }
  }
 }
}


//The x coordinate of all points are measured
//and the extreme ones are recorded.
private    void calculateColor()
{
 //The maximum value of coordinate x
 maxX =-1000;
 for(int l = 0; l<8; l++)
  {
   if (cube[l][0] > maxX)
    {
     maxX = cube[l][0];
    }
   }
 nExtreme = 0;
 //Number of extreme points
 for(int l = 0; l<8;l++)
  {
   maxPoints[l] = false;
   if (Math.abs(cube[l][0] - maxX )< 2)
   {
    nExtreme = nExtreme +1 ;
    maxPoints[l] = true;
   }
  }
  marked();
 }
 }//=============End of inner class Cube==============
```

```
}//end of main class
```

35 ***Exercise and challenge*** *Play with the code. Discover some bugs in relation with perspective, which we will tolerate. Try to solve them. Hint: the perspective of a single cube is solved in the method calculateColor() of the class Cube but perspective is not a microscopic problem but a macroscopic one and must be solved at the top level by considering the hand as a whole. A cheap solution has been adopted here: to paint the diverse subfigures in a given order.*

36 ***Exercise*** *Make a minor modification to the previous code to model a left hand whose thumb has two phalanges.*

37 ***Challenge*** *Modify the extant code to achieve a fairer simulation of the thumb: The thumb of our proto-hand is a delusion because it is really no more than a truncated normal finger. A real thumb closes not along with the other fingers but against them to form a compact fit.*

3.3 Conclusion

We have learned how to draw a 3d hand, an exercise that is a not a trivial one. Although OOP is not absolutely necessary, it is desirable because this technology allows us to fairly represent the systemic and hierarchical structure of data and of related methods. We are now more experienced with nested classes and the form to encapsulate related information and procedures into the same unit. If we compare our exercise with living beings, we immediately conclude that we have produced just very simple caricatures and that nevertheless we needed a refine technology and a good dose of work and imagination. In other words, we are plainly authorized to claim that living beings are wonderful because of their extreme complexity and perfection.

Chapter 4

Rationalizing creationism

`Creationism can be easily re-invented.`

38 ***Introduction***

A creationist is a person that believes that living beings were created by the Creator, a spiritual intelligent person with amazing powers. One can find at present creationist people all around the world, amidst the most diverse religions and cultures, and at any economic or cultural level. This is unexpected given the permanent advance of science in explaining every natural phenomenon with the materialistic laws of physics, chemistry and biology. More astonishing, creationist beliefs are firm in spite of the great efforts of the scientific community to alphabetize the world with the teaching of the evolutionary theory.

Why do creationist beliefs are so firmly establish in our modern global society?

39 ***Purpose*** *Our thesis is that there are very simple and powerful reasons that compel people to accept the idea of a creator. We do not pretend to understand exactly how this happens in real persons but we can show that there are very simple and compelling reasons to re-invent creationism. In fact, we show in this chapter that, by considering the basic mechanism for the movement of the hand, creationism is preferable to chance as explanation to its existence. To execute our task, we will be guided by the path that leads to the re-invention of* ***gravity****. In a chapter below, we will contrast creationism and evolution.*

4.1 Let us reinvent gravity

No one has ever seen gravity. Gravity is a mathematical construction to explain the movement of falling bodies that is the only thing that we see. Nevertheless,

gravity is so eagerly accepted by everybody that it could be strange to read that this is a mere mental invention. Of course, every one of us caresses the idea that there is something physical that corresponds to our invention. By the way, it is not clear what could it be. The Theory of general relativity of Einstein (see Rodríguez, 2008) is anymore the ultimate word about gravity and what it should be is not clear.

40 ***Reinventing gravity***

One can see the movements of leaves when they are blown by the wind. That movement is erratic, random. So, we can formulate a null hypothesis saying that every movement happens at random. But if one takes an orange and gently posits it at rest in the air, it will fall down to earth. One can suppose that the earth attracts free bodies but nothing can prevent us from claiming that the falling movement of the orange was also by randomness.

Nevertheless, what does happen when we repeat our basic experiment? We see that the orange irremediably falls toward our feet. Of course, we immediately can claim that this is a verification of our belief that the earth attracts the bodies and that is why they fall down to it. Let us call that attraction gravity that means heaviness. Anyway, nothing can prevent us from believing that this repeatability also happens by randomness. Now, no one has ever reported in reality or in fiction science that somebody believes that falling behavior of bodies is due to randomness and not to gravity.

Since gravity is immersed in every mind, let us say it again clearly: no one can prove that movements that are explained by gravity did not happen by mere randomness. Nevertheless, each one of us believes in gravity. Why? It is because there is a simple and powerful legitimate reason that compels us to do so. Let us take some lines to explain it. To begin with, let us imagine that we live in a world in which an orange that is gently posited in the air can fall with probability 1/2 and that it can rise with equal probability. For sake of simplicity, let us accept these values.

If we gently posit an orange in the air, and if it falls, we have seen en event that happens with probability 1/2. If we do the experiment one time and repeat the experiment a second time and we see that the two orange falls, then, given that the movement happens at random and that the two events do not interact, we observe a compound event that happens with probability (1/2) (1/2) = 1/4: if we throw two oranges and if their direction of movement is chosen at random, the two oranges will fall one fourth of the times.

If we repeat the experiment n times, and we see that the oranges falls down to earth all the n times, then we have seen an event that can happen with probability $(1/2)^n$. As n grows, this fraction rapidly diminishes to zero. But it never is zero.

That is why a scientific argument cannot exclude that the oranges fall by randomness and not by gravity. Nevertheless, we, you and me, believe that gravity exists. Why? The reason is that we together think that while randomness is unpredictable for a concrete experiment, it is highly predictable for many repetitions.

To fix ideas, let us consider now that we have repeated the experiment 10 times. In this case, $(1/2)^{10} = 0.0009765$. Since this probability is not zero, we are allowed to believe that randomness is all to falling oranges. But the belief in gravity is also allowed. In effect, the complementary probability is $1 - 0.0009765 = 0.999023437$ and this is the probability of the event that at least one orange does not fall down to earth. Now, we contrast predictions and experimental results: were the movement due to randomness, the more we repeat the experiment the higher is the probability for at least one orange to go upward. But such result never has been reported.

So, we must choose between accepting that randomness is all to falling oranges else to postulate that falling oranges are pulled by gravity. Which option will be choose? To decide, we follow the same directive everywhere in science: we will reject that randomness is the explanation of a group of events when the probability of that group of events diminishes when the number of repetitions is increased. In that case, we will postulate the existence of something else besides randomness that creates a systematic effect: in our case, that is gravity.

Now, gravity has been extended from falling oranges to planets, to stars to the whole cosmos in spite of the fact that we cannot experiment with planets or stars. In this case, we replace direct experimentation by observation of actual conditions. Both methods, experimentation and observation, are accepted by everybody in science and are also accepted by ordinary people.

Let us remark our position: when we have various theories to explain a group of events, we will choose as acceptable those options that are robust against experimentation: the probability of each chosen theory must augment the more we repeat the experiment or the more we collect observations while we will reject those theories that are weak against experimentation and observation and whose probability of getting true diminishes the more we study nature.

Our criterion is simple and we contend that it is part of the surviving battery of any being with a nervous system.

41 *Code for a simulation*

The next program shows a number of oranges that are released to random movement, which has equal probability to go upward than to go downwards. Sometimes every member of a group of oranges falls by mere randomness, but that probability diminishes as the number of oranges grows. The same happens in the upward direction. The normal result is a mix of oranges, some going down and other going up.

```
//Program D41 oranges
//Oranges move with random direction
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;
import java.util.Random;


public class oranges extends Applet
{
 private static final long serialVersionUID = 1L;

 private static  int   xWin = 600;
 private static  int   yWin = 500;
 private static int nScenes ;
 private static int delay ;
 private static  int numberOfOranges;
 private static int dir[] = new int[100];
 private static int posY[] = new int[100];
 private static  int deltaY;
 static Random r  = new Random();

// The window is enlarged.
// This method is fired automatically
// at the beginning
 public void init()
 {
  setSize(xWin,yWin);
 }


// Regulation of the velocity of movement
 private    void delay(int delay)
  {
try
 {
Thread.sleep(delay );
 }
catch (InterruptedException e)
```

```
 {
  e.printStackTrace();
  }
  }


 //Oranges fly up else down
private static void directions()
 {
  double p = 0.5;
  for(int j = 0; j< numberOfOranges; j++)
 {
  //We generate a random number
  // in within 0 and 1
  double  c = r.nextDouble();
  // We verify gender
  if (c<p)
 dir[j] = 1;
  else
 dir[j] = -1;
 }
  }

//All oranges are drawn
 private static void scene(Graphics g)
  {
 for(int j = 0; j< numberOfOranges; j++)
   {
 g.setColor(Color.ORANGE);
 g.fillOval(5 + j*50,posY[j]   ,30,30);
 posY[j] = posY[j] + deltaY *dir[j] ;
   }
  }

 public void  paint (Graphics g)
 {
  delay = 100;
  nScenes = 22;
  numberOfOranges= 15;
  directions();
```

```
  for(int j = 0; j< numberOfOranges; j++)
   posY[j] = 200;
  deltaY = 10;
  g.setColor(Color.pink);
  g.fillRect(0,0,xWin,yWin);
  System.out.println("Movement with random direction");
  // This loop control the   number of scenes
  for(int i=0; i<nScenes; i++)
   {
// A delay must be introduced to
// make movement visible for a human being.
scene(g);
if (i < nScenes-1)
 {
  delay(delay);
      //erase scene: we repaint everything in
  //the background color.
  g.setColor(Color.pink);
  g.fillRect(0,0,xWin,yWin);
 }
   }
  }
}
```

42 ***Exercise*** *Play with the code in company of your friends and family: you can modify the variable* numberOfOranges *that has been set to 15 in the last method,* paint (Graphics g)*, and observe how frequently all oranges fall down. To modify the size of the window, change variables* xWin *and* yWin *at the very beginning of the Applet. Make sure that your guests can repeat and understand the language of modern science: in a world in which the direction of movement is chosen by randomness, it is possible that all bodies fall down by mere chance but that probability diminishes as the number of bodies grows. Since the probability that all bodies fall is not zero, one always can claim that bodies fall by randomness but not by gravity. But, since that probability diminishes as the number of bodies grows, one is allowed to postulate that there is a force that compels bodies to fall. That force is called gravity that means heaviness.*

43 ***Chance is tamed by experimental repetition.***

Let us highlight again the fact that nobody has seen gravity. The only thing we see is falling bodies. Nevertheless, no one has ever reported a person that does not

believe in gravity i.e. that thinks that the falling behavior of bodies is due to chance and not to a force of attraction of massive bodies. Such a behavior of people must be based on very simple but compelling facts. In the light of the modern scientific method, that reason is the following:

Randomness is crazy and indomitable in the world of few events. It is like a crocodile. But in the world of a large number of events, randomness tends to accommodate itself along very clear predictions, like a cow. So, repeating over and over converts chance into a law. The next code illustrates this in the manner of modern science.

44 ***The code***

The next code illustrates a robust prediction about randomness, which is the more precise the larger is the number of experiments. We take a number of oranges and posit them in the air in a world that is animated by random movement with equal probability to go upward than to go downward. The prediction is that half of the oranges will fall down. This prediction can fail for 1, 2 or 3 oranges but the more oranges are incorporated in the experiment, the more certain it is. In this sense, randomness acquires personality thanks to experimentation. The exact meaning of this phrase is shown by the next code.

```
/*Program D44 robust
When a number of oranges is released to
movement  with random direction,
which has equal probability to go up than
to go down,
the half of them will fall down by mere chance.
This is the more certain,
the larger is the number of oranges.

The program produces a graphic that shows a bar
centered around   the mean of the fraction of
oranges that fall, approximately 0.5, and with
a wing of two standard deviations, forming an
interval that encloses 95\% of the trials.
As the number of oranges grows, the mean of
the interval   gets closer and closer to 0.5
and the interval itself tends to shrink around
the mean. In that way, the fraction of events
```

```
that deviates from 1/2 tends to zero as the
number of trials tends to infinite.
*/

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


public class robust extends Applet
{
 private static final long serialVersionUID = 1L;

 private static  int   xWin = 600;
 private static  int   yWin = 500;

 private static  int numberOfOranges;
 private static  int numberOfExperiments;
 private static  int numberOfRepetitions;
 private static double fraction[] = new double[1000001];
 private static int hist[] = new int[10];
 private static double mean;
 private static double dev;

 private static  int deltaX;
 private static  int scale;
 private static  int step;
 private static boolean print;


 static Random r  = new Random();

// The window is enlarged.
// This method is fired automatically
// at the beginning
 public void init()
 {
  setSize(xWin,yWin);
 }
```

```
 //Calculation of the mean and standard dev of DataVect[]
 private static void meanVarVector(double DataVect[])
 {
  int n = numberOfRepetitions;
  /* Report of input data
   System.out.println( "Data are" );

   for ( int i = 0; i < n; i++ )
    {
     System.out.println( DataVect[ i ]  );
    }*/
  //mean
  double sum = 0;
  for(int i = 0; i < n; i++ )
     sum = sum + DataVect[i];
  mean = sum / n;
  //System.out.println( "Sum \t "+ sum);
  System.out.print(   mean + " " );

  // Let us calculate the variance
  double sum2 = 0;
  for(int i = 0; i < n; i++ )
    sum2 = sum2 + (DataVect[i]-mean)*(DataVect[i]-mean);
  double var = sum2 / n;
  dev = Math.sqrt(var);
  System.out.println(  dev );
  dev = Math.sqrt(var);
 }


 //Calculates the table of relative frequencies
 //of the data in fraction[i]
 private static void relFreq( )
 {
  for(int k = 0; k<10; k++)
hist[k] = 0;
  for(int k = 0; k<10; k++)
   {
double a = k * 0.1;
```

```
double b = (k+1) * 0.1;

for(int i=0; i<numberOfRepetitions; i++)
 {
  if ((k==0) && (fraction[i]==0)) hist[0] = hist[0] +1;
  if ((fraction[i] > a ) && (fraction[i] <= b))
     hist[k] = hist[k] +1;
 }
   }
  if (print)
{
     for(int k = 0; k<10; k++)
  System.out.println("Between 0." + k + " and 0."
                   + (k+1) + ": " + hist[k]);
}
 }



 //Oranges fly up else down
private static double oranges(int numberOfOranges )
 {
  double p = 0.5;
  int counter = 0;
  for(int j = 0; j< numberOfOranges; j++)
 {
  //We generate a random number
  // in within 0 and 1
  double  c = r.nextDouble();
  // We verify gender
  if (c<p)
counter = counter +1;
 }
  double k = counter;
  double fraction = k /numberOfOranges;
  if (print)
System.out.println(fraction );
  return fraction;
  }
```

```
//All oranges are drawn
 private static void draw(Graphics g, int j)
  {
   int k = (int) Math.round(scale * ( -mean + 2*dev+ 0.5 ))
                                                    +200;
   int l = (int) Math.round(scale * ( -mean-2*dev +0.5))
                                                   +200 ;
   int m = (int) Math.round(scale * ( -mean+0.5))+200  ;
   g.drawLine(50 + deltaX* j,   k, 50+ deltaX* j,  l);
   g.fillOval(50 + deltaX* j,    m, 2,2 );
  }


 private static void titles(Graphics g)
 {
  g.drawString("Mean and 2 standard deviations ", 300,50);
  g.drawString("of the fraction of balls that fall ",
                                         300,65);
  Integer n = numberOfRepetitions;
  g.drawString("calculated over " + n  + " repetitions",
                                         300,80);
  g.drawLine(50, 400, 600, 400);
  //This is how an integer variable is drawn.
  Integer c = step;
  g.drawString(c.toString(),   60,420);

  g.drawString("Number of oranges",   180,420);
  Integer d = numberOfExperiments;
  Integer e = c*d;
  g.drawString(  e.toString(),  d*deltaX +50  ,420);
  g.drawString(  "0.5",   10    ,205 );
  g.drawLine(30,200,40,200);
  g.drawString(  "1",   10    ,205- scale/2);
  g.drawLine(30,200- scale/2,40,200- scale/2);
  g.drawString(  "0",   10    ,205+ scale/2);
  g.drawLine(30,200+scale/2,40,200+ scale/2);
  g.drawString(  "Fraction",   10    ,20);
  g.drawLine(28, 100,28,300);
  g.setColor(Color.red);
```

```
  System.out.println("Movement with random direction");
  System.out.println("Number of oranges, " +
   "                              mean and deviation");
  System.out.println("of the fraction " +
   "                           of oranges that fall");
  System.out.println("over " + numberOfRepetitions +
                                  " repetitions" );
 }


 public void  paint (Graphics g)
 {
  /*numberOfExperiments = 50;
  numberOfRepetitions = 10;
  //vertical scale, must be even
  scale = 2*100;
  //distance between 2 successive vertical lines
  deltaX = 10;
  step = 1;*/
  numberOfExperiments = 50;
  numberOfRepetitions = 10;
  scale = 200;
  deltaX = 10;
  step = 100;
  /*numberOfExperiments = 50;
  numberOfRepetitions = 1000;
  scale = 200;
  deltaX = 10;
  step = 1000; */
  print = false; //else true
  titles(g);

  // main loop
  for(int j=1; j<=numberOfExperiments; j++)
   {
if (print)
 {
  System.out.println();
  System.out.println("***************");
  System.out.println();
```

```
  System.out.println("Number of oranges  = " + step*j);
  System.out.println("Fraction of oranges that fall");
  System.out.println("over " + numberOfRepetitions
                                  + " repetitions" );
 }
numberOfOranges = step*j;
System.out.print(numberOfOranges + " " );
for(int i=0; i<numberOfRepetitions; i++)
 {
  if (print) System.out.print("Experiment " + i + ":");
  fraction[i] = oranges(numberOfOranges);

     }
if (print)
 {
  System.out.println();
  System.out.println("Table of relative frecuencies");
 }
meanVarVector(fraction);
relFreq();
draw(g, j);
}
  }
}
```

45 ***Exercise*** *Play with the code. Make a suitable sample of experiments and claim that randomness is crazy in a world of few trials but that some folds of its behavior are the more predictable the larger is the number of trials. Concretely, if we posit a group of oranges in the air in a world in which the direction of movement is chosen at random and with probability 0.5 in the upward direction and 0.5 in the downward direction, half of the oranges will fall down. In actual experiments, exactly half of the oranges will fall only very rarely. Most of times a deviation will be observed. Nevertheless, that deviation tends to diminishes as the number of oranges grows.*

Concretely, the graphic shows a bar centered around the mean of the fraction of oranges that fall, approximately 0.5, and with a wing of two standard deviations, forming an interval that encloses 95% of the trials. As the number of oranges grows, the mean of the interval gets closer and closer to 0.5 and the interval itself

tends to shrink around the mean. In that way, the fraction of events that deviates from 1/2 tends to zero as the number of trials tends to infinite.

In terms of our discussion about gravity, no one believes that falling bodies are explained by a movement whose direction is chosen at random between two possibilities, up and down with equal probability, because in that case we should see that bodies fall down and fly up in approximately the same proportions, a prediction that is not fulfilled in our world.

The robustness of chance thanks to experimentation is the reason that enables the design of lotteries, casinos and of insurances, activities that in every country move incredibly large sums of money. The theory of probabilities specializes in making robust predictions about random processes while inferential statistics teaches one how to take decisions in a world where randomness is unavoidable.

46 ***Challenge*** *Observe that in the previous code the method relFrec() is silenced. Please, activate it and verify that it produces a table of relative frequencies of the fraction of oranges that fall. Tailor the form to draw the family of these histograms in a 3d graphic. Hint: you can report the table in one line for each number of oranges in the experiment and use Excel or Calc to draw the whole family of distributions. Next, you can try to make a Java program to do everything automatically.*

4.2 Creationism vs chance

Creationist ideas have been expressed in all cultures and since ancient times. Nevertheless, atheist attacks always have been common: the most powerful witness against gods is that every kid by age 4 already has plenty of consciousness that he or she has done very bad things against fathers and grandfathers, to say the least. Nevertheless, there is no punishment from above. And we can be bad enough that a very reasonable deduction might be that there is no God. And we also have the case of believers that turn into atheism when they feel no answer to their prayers in a very crucial moment for their lives. **Charles Darwin** seems to be one of such cases when its beloved daughter died at age 10 or so. Darwin learned in that way that death is more close to us than God and that therefore the causes of death have more explicative power than theology.

That atheism always has been a real option for ordinary people is a conclusion that one can arrive at if one reads a witness of atheism consigned in Psalm 14:1, written some 3000 years ago. Although, other people discovered that gods answer to their prayers when these are accompanied with sacrifices. These could be as demanding as to be of own sons and daughters.

While God the Creator is the mechanism used by a believer to explain the perfection and complexity of nature, the only mechanism available for atheist people

during many centuries was chance. And which one does prevail? We find that during the middle ages, everyone seemed to believe in God the Creator. Why? Of course, social pressures were very strong. But were these pressures all to it?

Let us claim that the very machinery that force people to invent gravity also operates to propel them to invent creationism. All we need is to give a glance to the anatomy of the hand. Actually, the proposed creationism is just a belief that a creative factor must exist apart from chance. Before 1850, that creative factor was identified with revealed gods but today we have the opportunity to also consider evolution (and its natural extensions to cover all nature). We consider chance alone in this chapter while evolution will be discussed in next chapters. We will use our left hand as laboratory.

47 ***Muscles come by do-undo pairs***

Most people know as a result of direct observation and experimentation over themselves that the movement of our body is due to the work of muscles that contract to exercise force over bones. Muscular fatigue caused by hard repetitive work help us to understand that our muscles can actively contract but that they cannot actively get stretched.

So, if muscles only contract, how can we undo a movement?

Every movement can be undone thanks to the fact that muscles go by do-undo pairs. This is easily illustrated with our arm, as the next caricature shows:

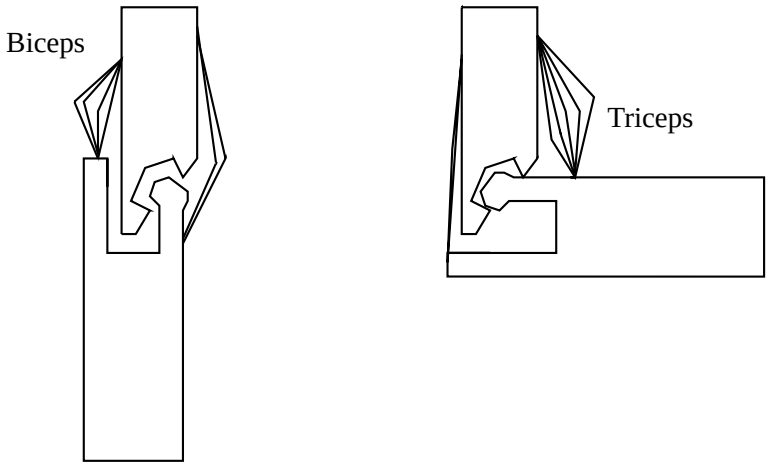


Figure 4.1. Toy model of an elbow articulation. The arm is bent or straighten by a coordinated contraction work of the biceps and triceps.

When the arm is relaxed and falls down, all of its muscles rest. But if one rises the arm, then one can actively keep it straight or actively can bend it. This is done by a coordinate work of biceps and triceps.

48 ***Exercise*** *Play with your arm and verify else refute that the arm is bent or straighten by a coordinated contraction work of the biceps and triceps. To that aim, touch your left arm with your right hand in the appropriate place while bending or stretching the arm. Share this exercise with your students, family and friends.*

49 ***Exercise*** *Play with your left hand trying to move each articulation. Discover some pair of do-undo muscles. To that aim, you can observe how the muscles of the hand and forearm move below the skin else you can touch at them.*

50 ***A code for do-undo pairs of muscles***

Let us display the situation in which the movement of fingers is powered by muscles that are organized forming do-undo pairs.

```
/*Program D50 doUndo

We display a proto-hand in which
all fingers are powered by a do-undo pair of muscles.

Our protohand has 5  fingers, a palm and an arm.
Fingers have 3  phalanges except the thumb, which has 2.

Since we use finite precision, all our moves are
approximations. Therefore, a multiplication of errors
is expected if we use modifications in tandem.
That is why  every scene is built ab initio by
modification of a unique unmodifiable cube at the origin.
The hand is located and moves at the origin
and points downwards over the screen.

The inner class Hand is encoded beginning from line 435
The inner class Finger is presented in line 497.
A finger is composed of 3 cubes. The class Cube
appears in line 616.

The main program follows.
```

```
*/

import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Polygon;
import java.util.Random;


public class doUndo extends Applet
{

 public static final long serialVersionUID = 1L;

 /*This constructor is necessary when one have various
  nested classes. In our case, we have the class Cube,
   which is used by the class Finger,
   which is used by the class Hand,
   which is used by the main class (protoHand) in here.*/
 public doUndo()
 {

 }


/*
Our basic unit is  the unitary cube at the origin.
It is composed of 8 points or vertexes = oPoints3d[][],
12 edges = edges[][] and 6 faces = faces[][].
Points are numbered from 0 to 7.
These numbers are used in
edges, which  records the pairs of
points that are joined by a line.
A face is determined by its 4 corners.

All other cubes are transformations of this
fundamental cube. The form as this is done
appears in the class Cube.
```

```
*/

//The points of the fundamental cube
private static   double oPoints3d[][]=
{
 {0,0,0},
 {1,0,0},
 {0,1,0},
 {0,0,1},
 {1,1,0},
 {1,0,1},
 {0,1,1},
 {1,1,1}
};

/*
//The edges of the fundamental cube
private static int edges[][] =
   {{0,1},
    {0,2},
    {0,3},
    {1,4},
    {1,5},
    {2,6},
    {2,4},
    {3,5},
    {3,6},
    {4,7},
    {5,7},
    {6,7}};

*/

//The faces of the fundamental cube
private static int faces[][] =
   {{0,1,4,2},
    {0,2,6,3},
    {0,3,5,1},
    {3,5,7,6},
```

```
    {1,4,7,5},
    {2,6,7,4}};



 //Measures of the window
 private    int   xWin = 600;
 private    int   yWin = 500;
 private   int nScenes ;
 private   int delay ;
 //Number of cubes in the finger
 private  static  int nPhalanges = 3;
 private  static  Boolean move = true;


 static Random r  = new Random();

 //The constructor of hand is invoked here.
 //A is now the instance of the main class hand
 private static doUndo A = new doUndo();
 //The inner class Hand is instantiated
 private static Hand  h = A.new Hand();
 //An auxiliary  instance of Finger
 private static Finger f1=  A.new Finger(nPhalanges);



//The window is enlarged.
//This method is fired automatically
//at the beginning
 public void init()
 {
  setSize(xWin,yWin);
 }

 //Regulation of the velocity of movement
 private    void delay(int delay)
 {
 try
```

```
      {
    Thread.sleep(delay );
      }
      catch (InterruptedException e)
      {
    e.printStackTrace();
      }
 }

 //face ii of the cube is drawn
 private    void drawface(Graphics g, Cube cc, int ii)
 {
 for(int l =0; l<4; l++)
  {
   cc.pointsX[l] = 200-cc.shade2d[faces[ii][l]][1];
   cc.pointsY[l] = 200-cc.shade2d[ faces[ii][l]][2];
   //System.out.println(pointsX[l] + " " + pointsY[l]);
  }
  //A polygon is defined by two vectors and an integer
   //number:
   //The first vector lists the x-coordinates of the
   //corners;
   //the second lists  the y-coordinates. The integer is
   //the number of points.
   //If the last point is different than the first,
   //the first point is closed with the last.
   Polygon face = new Polygon(cc.pointsX,
                       cc.pointsY, 4);
   g.fillPolygon(face);
 }

 private    void drawArm(Graphics g)
 {
// A working cube cc is initialized
Cube  cc = new Cube(1);
//The working cube is re-set.
   cc = h.arm ;
   //Movement begins
   if (move) f1.modifyCube(cc,0);
   cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
```

```
   //Projection is calculated
   cc.shade2d =  cc.round(cc.cubeProj );
   //printShade();
   //The cube is drawn
   cc.calculateColor();
   for(int i=0;i<6;i++)
    {
  if (cc.coloredFace[i])
   {
    g.setColor( new Color( cc.colors[i][0],
                  cc.colors[i][1],
                  cc.colors[i][2]));
    drawface(g,cc,i);
   }
     }
   //worked cube is kept to memory
   h.arm = cc;
 }

 private Cube repositPalm(Cube cc)
 {
  cc.oPosition[0]  =  h.arm.cube[3][0];
  cc.oPosition[1]  =  h.arm.cube[3][1]+5;
  cc.oPosition[2]  =   h.arm.cube[3][2];
  cc.cube = cc.translateCube(cc.cube, cc.oPosition);
  return cc;
}


 private    Cube updatePalm(Cube cc)
 {
 cc = f1.inclination(cc,1);
  cc = f1.rotation(cc,1);

  //cc.printCube(cc.cube, " after rotation");
   cc = repositPalm(cc);
  //cc.printCube(cc.cube , " after relocation");
  for(int i = 0; i<3;i++)
  cc.vTrans[i] = cc.vTrans[i] + cc.deltaTrans[i];
  cc.cube = cc.translateCube(cc.cube,  cc.vTrans);
```

```
  //cc.printCube(cc.cube, " after translation");
  cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
  //cc.printCube(cc.cubeProj,  " after projection");
 return cc;
 }

 private    void drawPalm(Graphics g)
 {
// A working cube cc is initialized
Cube  cc = new Cube(1);
//The working cube is re-set.
   cc = h.palm ;
   //Movement begins
   if (move) cc = updatePalm(cc);
   cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
   //Projection is calculated
   cc.shade2d =  cc.round(cc.cubeProj );
   //printShade();
   //The cube is drawn
   cc.calculateColor();
   for(int i=0;i<6;i++)
    {
  if (cc.coloredFace[i])
   {
    g.setColor( new Color( cc.colors[i][0],
                  cc.colors[i][1],
                  cc.colors[i][2]));
    drawface(g,cc,i);
   }
     }
   //worked cube is kept to memory
   h.palm = cc;
 }

 //Fingers are posited below the palm
 private    void locateFingers()
  {
   for(int i=0; i<5;i++)
for(int j=0; j<3;j++)
 h.finger[i].phalange[0].oPosition[j]
```

```
                                    =  h.palm.cube[3][j];
   for(int i=0; i<5;i++)
h.finger[i].phalange[0].oPosition[1]
        = h.finger[i].phalange[0].oPosition[1] -10*i ;

  }


 private    void drawFingers(Graphics g)
 {
  locateFingers();
  for(int j = 4; j>=0;j--)
  {
  if(j==4) nPhalanges = 2;
  else nPhalanges = 3;
   for(int m = 0; m < nPhalanges; m++)
   {
//A working cube cc is initialized
Cube  cc = new Cube(m);
//The working cube is re-set.
    cc = h.finger[j].phalange[m];
    //Movement begins
     if (!(move))
      {
      cc.deltaPhi[m] = 0;
      cc.deltaTheta = 0;
      }
     cc = h.finger[j].modifyCube(cc,m);
    //Projection is calculated
    cc.shade2d =  cc.round(cc.cubeProj );
    //printShade();
    //The cube is drawn
    cc.calculateColor();
    for(int i=0;i<6;i++)
     {
  if (cc.coloredFace[i])
   {
    g.setColor( new Color( cc.colors[i][0],
                  cc.colors[i][1],
                  cc.colors[i][2]));
```

```
   drawface(g,cc,i);
  }
    }
   //worked cube is kept to memory
   h.finger[j].phalange[m] = cc;
  }
 }
}

//A phalange goes up
private    void  showUp(Graphics g, int j, int m)
{
 h.finger[j].phalange[m].phi[m]
             = h.finger[j].phalange[m].phi[m]  -0.5;
 drawAP(g);
 drawFingers(g);
}

//A phalange goes back down
private    void  showDown(Graphics g, int j, int m)
{
h.finger[j].phalange[m].phi[m]
             = h.finger[j].phalange[m].phi[m]  +0.5;
 drawAP(g);
 drawFingers(g);

}


//All phalanges are moved one by one
private    void  movePhalanges(Graphics g)
{
 locateFingers();
 for(int j = 4; j>=0;j--)
 {
  if (j==4)  nPhalanges = 2;
  else nPhalanges = 3;

  for(int m = 0; m < nPhalanges; m++)
  {
```

```
 showUp(g,j,m);;
 delay(delay);
 pink(g);
 showDown(g,j,m);
 delay(delay);
 if (!((m==2) & (j == 0))) pink(g);
 if (j==4)  nPhalanges = 2;
   else nPhalanges = 3;
   }
  }
 }

//Forearm and palm are drawn
 private    void drawAP(Graphics g)
 {

 drawArm(g);
 drawPalm(g);
 }


 private    void drawHand(Graphics g)
 {
drawAP(g);
 drawFingers(g);
 }

 //Everything is erased
 private    void pink(Graphics g)
 {
  //background color
  g.setColor(Color.pink);
  g.fillRect(0,0,xWin,yWin);
 }


 private    void dynamics(Graphics g)
 {
 //Loop for the scenes
  for(int k = 0; k<nScenes; k++)
```

```
  {
   //System.out.println("scene " + k);

drawHand(g);
    //delay  + erase drawing
    if (k < nScenes-1)
     {
      delay(delay);
      //erase cube: we repaint everything in
      //the background color.
       pink(g);
     }//end of if
  }//end of for nScenes
 }


 //Fingers move a bit forth and back
 private    void jiggling(Graphics g)
 {
  move = false;
  //background color
  drawAP(g);
  movePhalanges(g);
 }

 private    void initialPosition(Graphics g)
 {
  pink(g);
  dynamics(g);
 }

 //This is the main method
 public void paint (Graphics g)
 {
  //Number of scenes of the presentation.
  nScenes = 4;
  //Delay between two scenes
  delay =  0;
  System.out.println("Do-undo study");
  initialPosition(g);
```

```
 delay =  500;
 delay(delay);
 //A normal hand jiggles
 pink(g);
 jiggling(g);
}//end of paint


//=============Beginning of Inner class Hand============

public class Hand
{
private Cube arm = new Cube(1);
private Cube palm = new Cube(1);
private Finger finger[] = new Finger[5];

//When memory is allotted for a hand,
//all variables must be initialized.
//Lower classes are initialized by default,
//but some  redefinitions could be necessary.
Hand()
{
 //The arm is initialized
  final Cube c1 =    new Cube(1);
  arm = c1;
      //Dimensions and location of the arm.
  arm.scale[0] = 8;
  arm.scale[1] = 30;
  arm.scale[2] = 100;
  for(int i=0; i<8;i++)
  for(int j=0; j<3;j++)
      arm.points3d[i][j] =  -arm.scale[j]*oPoints3d[i][j];


  //The  palm is initialized
  final Cube c2 =    new Cube(1);
  palm = c2;
  //Dimensions and location of the palm.
  palm.scale[0] = 8;
  palm.scale[1] = 40;
```

```
   palm.scale[2] = 60;
   for(int i=0; i<8;i++)
for(int j=0; j<3;j++)
 palm.points3d[i][j] =  -palm.scale[j]*oPoints3d[i][j];


   //Fingers are initialized
   for(int i=0; i<5;i++)
   {
   final  Finger f =   new Finger(nPhalanges);
   finger[i] = f;
   }




 }
 }
 //=============End of Inner class Hand============

 //=============Beginning of Inner class Finger===========
 /*Variables and methods of the class Finger are
 encapsulated in the same unit.
 All definitions are non static, i.e., they are
 used as erasable workbenches and resettable tools.

 A finger is a coordinated set of 3 cubes.
 The class cube is defined below.

 All methods that are of exclusive use for fingers are
 encapsulated here.
 */
 public class Finger
 {

 //Our finger is a composition of 3 cubes or phalanges
 private   Cube[] phalange    = new Cube[3] ;


 //When memory is alloted for a finger,
```

```
 //all variables must be initialized
 Finger(int nPhalanges)
 {
 for(int m =0; m < nPhalanges; m++)
   {
    final Cube cc =    new Cube(m);
    phalange[m] = cc;
   }
 }
// The only allowed movement of a phalange is
 //inclination, forth and back, in within bounds.
 //Inclination of given phalange governs the
 //rest of the finger
 private   Cube inclination(Cube cc, int m)
 {
  //Inclination is equal to a rotation.
  //Axis of rotation: the y axis.
  cc.axisRot[0] = 0;
  cc.axisRot[1]=1;
  cc.axisRot[2]=0;
  switch(m)
   {
case 0: cc.phi[0]  = cc.phi[0]  + cc.deltaPhi[0] ;
           break;
case 1: cc.phi[1]  = cc.phi[0] + cc.phi[1]
                                + cc.deltaPhi[1] ;
           break;
case 2: cc.phi[2]  = cc.phi[1] + cc.phi[2]
                                + cc.deltaPhi[2] ;
           break;
   }
 cc.cube = cc.rotCube(cc.points3d, cc.axisRot,
                             cc.phi[m] );
 return cc;
}
```

```
    //For visualization, the finger is rotated
//around the z axis.
private   Cube rotation(Cube cc, int m)
{
 //Rotation about the z axis.
 cc.axisRot[0] = 0;
 cc.axisRot[1]=0;
 cc.axisRot[2]=1;
     cc.theta = cc.theta + cc.deltaTheta;
 cc.cube = cc.rotCube(cc.cube, cc.axisRot, cc.theta);
 return cc;
}

 //Each phalange must have its initial position
 //that depends on the position of the phalanges below it
 private   Cube rePosit(Cube cc, int m)
 {
   //System.out.println("rePosit(m)" + m);
 switch(m)
  {
   case 0:  {
      } break;
   case 1: {
cc.oPosition[0]  =   phalange[0].cube[3][0];
cc.oPosition[1]  =   phalange[0].cube[3][1];
    cc.oPosition[2]  =   phalange[0].cube[3][2];
   break;
       }
   case 2: {
cc.oPosition[0]  =   phalange[1].cube[3][0];
cc.oPosition[1]  =   phalange[1].cube[3][1];
cc.oPosition[2]  =   phalange[1].cube[3][2];
   break;
       }
  }
  cc.cube = cc.translateCube(cc.cube, cc.oPosition);
  return cc;
 }

 //The cube is inclined, rotated, translated, projected
```

```
 private    Cube modifyCube(Cube cc, int m)
 {
  cc = inclination(cc,m);
   cc = rotation(cc,m);

  //cc.printCube(cc.cube, " after rotation");
  rePosit(cc,m);
  //cc.printCube(cc.cube , " after relocation");
  for(int i = 0; i<3;i++)
  cc.vTrans[i] = cc.vTrans[i] + cc.deltaTrans[i];
  cc.cube = cc.translateCube(cc.cube,  cc.vTrans);
  //cc.printCube(cc.cube, " after translation");
  cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
  //cc.printCube(cc.cubeProj,  " after projection");
  return cc;
 }




 }

 //=============End of Inner class Finger============

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

 //=============Beginning of Inner class Cube============



 public   class Cube
 {
  private double points3d[][] = new double[8][3];
  private   int shade2d[][] = new int[8][3];

//  All cubes are derived from the basic cube by rotations
//  and translations
  private double cube[][]  = new double[8][3];
  private   double  cubeProj[][] = new double[8][3];
```

```
  private double oPosition[] = new double[3];
  private   double axisRot[] = new double[3];
  private   double axisProj[]  = new double[3];
  private   double deltaTrans[]  = new double[3];
  private   double vTrans[]  = new double[3];
  private   double theta  = 0;
  private   double deltaTheta;

  private   double basisRot[][], basisProj[][]
               = new double[3][3];
  private   double matrixRotz[][],
  matrixRot[][],
  matrixProj[][] = new double[3][3];
//  The matrix of a projection along the z-axis
  private     double J[][]=
        { {1,0,0},
          {0,1,0},
          {0,0,0}};

  private    double scale[] = new double[3];

  int colors[][] = new int[6][3];
  private   int pointsX[]  = new int[6];
  private   int pointsY[]  = new int[6];
//  How to color faces
  private    double maxX;
  private   int nExtreme;
  private   boolean maxPoints[]  = new boolean[8];
  private   boolean coloredFace[]  = new boolean[6];
  //Angle of inclination of phalange
  private   double phi[] = new double[3];;
  private   double deltaPhi[] = new double[3];;

 //Every implementation of Cube is initialized here.
 //Parameter m is used for the initial position
 //of cube number m
 Cube(int m)
 {
  //Original  form
```

```
scale[0] = 8;
scale[1] = 8;
scale[2] = 30;
for(int i=0; i<8;i++)
for(int j=0; j<3;j++)
points3d[i][j] =  -scale[j]*oPoints3d[i][j];
defineColors();
//Original position
for(int i=0; i<3;i++)
 oPosition[i] = 0;
//Each phalange has its own angle of inclination
//with respect to its support
for(int i=0; i<3;i++)
{
phi[i] = 0;
deltaPhi[i]  = -0.1*(m+1);
}
//  The 3d cube flies along vectTrans[]
//Each cube has its own direction of translation
for(int i=0; i<3;i++)
vTrans[i] = 0;
deltaTrans[0] =  0;
deltaTrans[1] =  0;
deltaTrans[2] =  0;
//  printCube();
//  The 3d-graphic is rotated along the axisRot
//  an angle theta that is incremented by deltaTheta.
//  Each cube has its own velocity and direction of
//  rotation.
theta = 0;
deltaTheta = -0.1;
axisRot[0] = 0;
axisRot[1]=0;
axisRot[2]=1;
//  The 3d-graphic is projected over the screen
//  which is perpendicular to the x-axis.
axisProj[0] = 1;
axisProj[1]=0;
axisProj[2]=0;
```

```
  }


 Random r  = new Random();

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

//The multiplication of two matrixes
private  double[][] matrixMult(double p[][], double q[][])
{
 double[][] helpM = new double[3][3];

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

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

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

```
 helpV[i]  =helpV[i]  + m[i][j]*v[j];
 return helpV;
}



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

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

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

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

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


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

 helpT = thirdVector(v);
 helpS = secondVector(v);
```

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


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

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

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

/*A 3d cube exists only in the imagination:
* it is not seen.
* One sees a shade or projection of it over the screen.
* The shade is defined by 2d points. The edges of
* the 2d shade are inherited from the edges of the
* 3d cube.
* The 3d cube is projected into a 2d plane.
* The resultant figure must be rendered to the screen.
* A projection is done along  an axis that crosses
* through 0.
* Our projections are observed from infinite:
* we use no perspective corrections. */
private     double[][] mProj(double w[] )
{
 basisProj = basisB(w);
 double[][]  helpM = new double[3][3];
 helpM = matrixMult(J,matrixTranspose(basisProj));
 helpM = matrixMult(basisProj,helpM);
 //printMatrix(helpM, "Proj");
 //Test: Proj*Proj must be equal to Proj
 /*helpM = matrixMult(helpM,helpM);
 printMatrix(helpM, "Proj* Proj");*/
 return helpM;
}

//The cube c is rotated around axis v an angle theta
private double[][] rotCube(double c[][],
                           double v[], double theta)
{
 double[]  helpV = new double[3];
 double cc[][] = new double[8][3];
 matrixRot = mRotation( v,theta);
```

```
 //printMatrix(matrixRot, "matrixRot");
 //The 8 points of the cube must be rotated
 for(int i=0; i<8; i++)
 {
  for(int j=0; j<3; j++)
  helpV[j] = c[i][j];
  helpV = matrixVector(matrixRot, helpV);
  for(int j=0; j<3; j++)
  cc[i][j] = helpV[j];
 }
 //printCube(cc, " after rot");
 return cc;
}



//The cube c is translated   deltaV
private double[][] translateCube(double c[][],
                                          double deltaV[])
{
 double cc[][] = new double[8][3];
 for(int i=0; i<8; i++)
 for(int j=0; j<3; j++)
 cc[i][j] = c[i][j] + deltaV[j];
 return cc;
}


//The cube c is projected along axis w
private double[][] projCube(double c[][],  double w[])
{
 double[]  helpV = new double[3];
 double cc[][] = new double[8][3];
 matrixProj = mProj(w);
 for(int i=0; i<8; i++)
  {
   for(int j=0; j<3; j++)
   helpV[j] = c[i][j];
   helpV = matrixVector(matrixProj, helpV);
   for(int j=0; j<3; j++)
```

```
   cc[i][j] = helpV[j];
  }
 return cc;
}

//The rendering procedure accepts integer values
private     int[][] round(double c[][])
{
 int[][] helpM = new int [8][3];
 for(int i=0;i<8;i++)
 for(int j=0; j<3; j++)
 helpM[i][j] = (int) Math.round(c[i][j]);
 return helpM;
}

private   void printCube(double c[][], String a)
{
 System.out.println("This is the cube" +a);
 for(int i=0;i<8;i++)
 System.out.println(c[i][0] + " " + c[i][1]+ " "
                                      + c[i][2]);
}

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

private void printShade()
{
 System.out.println("This is the shade");
 for(int i=0;i<8;i++)
 System.out.println(shade2d[i][1] + " "
                                  + shade2d[i][2]);
}

//Colors to paint the faces are defined
```

```
private int[][] defineColors()
{
 for(int l = 0; l<6;l++)
  {
   //We use random colors.
   //A color can be defined by three numbers,
   //each one from 0 to 255.
   //We has 6 faces, so we need
   //a matrix to keep colors.

   int c1,c2,c3;
   c1 = r.nextInt(226);
   c2 = r.nextInt(226);
   c3 = r.nextInt(226);
   //Colors are memorized
   colors[l][0]= c1;
   colors[l][1]= c2;
   colors[l][2]= c3;
  }
 return colors;
}

//what faces must be colored
private    void marked()
{
 for(int f = 0; f<6; f++)
 coloredFace[f] = false;
 for(int f = 0; f<6;f++)
 {
  for(int p = 0; p<4;p++)
  {
   for(int c = 0; c<8; c++)
    {
     if (maxPoints[c] == true)
     {
       if (faces[f][p] == c )
       coloredFace[f] = true;
       //System.out.println("coloring");
     }
   }
```

```
  }
 }
}


//The x coordinate of all points are measured
//and the extreme ones are recorded.
private    void calculateColor()
{
 //The maximum value of coordinate x
 maxX =-1000;
 for(int l = 0; l<8; l++)
  {
   if (cube[l][0] > maxX)
    {
     maxX = cube[l][0];
    }
   }
 nExtreme = 0;
 //Number of extreme points
 for(int l = 0; l<8;l++)
  {
   maxPoints[l] = false;
   if (Math.abs(cube[l][0] - maxX )< 2)
   {
    nExtreme = nExtreme +1 ;
    maxPoints[l] = true;
   }
  }
  marked();
 }
 }//=============End of inner class Cube==============


}//end of main class
```

51 ***Exercise*** *Play with the code. Try to make with your own left hand all the movements shown by the program. Challenge the member of your family to make the same. Explain.*

52 ***The hand vs chance***

Every human of whatever cultural or socio-economical environment perceives that the hand is a marvel. How this perception arises is an intuitive operation to which the mind has no access. To help to understand the nature of that perception, let us produce a very simple but very compelling reason to believe that the hand is not the product of chance.

Our alleged reason will not be taken as the ultimate truth but only as a prompt to think further. Nor we pretend that creationist beliefs are based on our reasons alone.

Let us examine now the form as the disposition of the muscles of the hand can be explained by randomness. Yes, many things happen in this world by randomness and so we cannot exclude that the arrangement of the muscles of the hand also happened by chance. Let us imagine, for the sake of simplicity, that the probability of a do-undo pair is 1/2.

If a given muscle has its companion and together forms a do-undo pair, the situation is somehow similar to that in which one throws a coin which shows a given face up and one must match that face with another coin. This matching can happen with probability 1/2. This is also the probability that an orange falls down when it is released in the air in a world in which the orange can fall else rise with probability 1/2.

We have postulated that the probability of formation by chance of a do-undo pair is 1/2. But if we consider two pairs, then this is similar to the experiment of matching two coins and is also similar to the gravity experiment with 2 oranges. The probability of matching two coins is 1/4 as well as the probability that two oranges fall down. Since this probability is not zero, one is allowed to believe that chance is the reason that explains the fact that in a gravity experiment with two oranges, both fall down to earth. By the very same reason, one is allowed to believe that chance alone is the reason that explains the fact that the two pairs of do-undo muscles also appeared by mere chance.

If we consider n pairs of do-undo muscles, we can explain the situation with a probability of $(1/2)^n$. Given that in the hand there is more than 20 do-undo pairs, for $n = 20$ that probability is $9.5 \times 10^{-7} = 0.00000095$. And in general, this probability is not zero for every n. This means that one can believe that the muscles were accommodated by mere chance. By the same reason, one can believe that chance is the fundamental explanation of the fact that when we posit n oranges in the air, all of them fall down.

Now, nobody believes that falling bodies can be explained by chance. Why? It is because chance acquires robustness with experimentation: if chance is the fundamental explanation of falling bodies, then approximately the half of a number of

bodies that are released in the air must go upwards while the others go down. And this assertion is the more certain the larger is the number of considered bodies. But experience does not corroborate this prediction. That is why everybody believes in gravity, the active attraction of bodies by the earth.

By the very same reason, many people refuse to believe that the disposition of the muscles of the hand was achieved by chance. Why? It is because chance becomes a robust entity thanks to experimentation: were chance the fundamental explanation of the disposition of the muscles of the hand, then approximately the half of a number of muscles must go in do-undo pairs while the other half must go alone, singles. And this assertion is the more certain the larger is the number of considered muscles.

What does say experience? Experience says that if a movement can be made, then it can be undone. This assertion is certain for any movement of the hand as well as for most movements of the human body and for most movements of any animal. That is enough for many people for rejecting the idea that the disposition of muscles in do-undo pairs was achieved by chance and so they prefer to accept creationist and religious ideologies.

53 ***Exercise*** *Consider the muscles of the belly. Prove that they can be contracted. Find the muscles that undo that work. Explain.*

54 ***Exercise*** *Mark one muscle that moves a given part of your hand and condemn it to remain immobile: it cannot actively contract by any means. Try in this new situation to do ordinary tasks. Some forms of rheumatism can originate such a kind of trauma.*

55 ***A code for immobile***

The next program encodes a proto-hand whose phalanges have unequal quality of empowering: some muscles have no companion, others come with their pairing do-undo muscle while other pairs are absent.

```
/*Program D55 immobile

We display a proto-hand in which a muscle that
must form part of a do-undo pair of muscles of a phalange
has a given probability of existence.

Our protohand has 5  fingers, a palm and an arm.
```

```
Fingers have 3  phalanges except the thumb, which has 2.

Since we use finite precision, all our moves are
approximations. Therefore, a multiplication of errors
is expected if we use modifications in tandem.
That is why  every scene is built ab initio by
modification of a unique unmodifiable cube at the origin.
The hand is located and moves at the origin
and points downwards over the screen.

The inner class Hand is encoded beginning from line 474
The inner class Finger is presented in line 536.
A finger is composed of 3 cubes. The class Cube
appears in line 654.

The main program follows.

*/

import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Polygon;
import java.util.Random;


public class immobile extends Applet
{

 public static final long serialVersionUID = 1L;

 /*This constructor is necessary when one have various
  nested classes. In our case, we have the class Cube,
   which is used by the class Finger,
   which is used by the class Hand,
   which is used by the main class (immobile) in here.*/
 public immobile()
 {

 }
```

```
/*
Our basic unit is  the unitary cube at the origin.
It is composed of 8 points or vertexes = oPoints3d[][],
12 edges = edges[][] and 6 faces = faces[][].
Points are numbered from 0 to 7.
These numbers are used in
edges, which  records the pairs of
points that are joined by a line.
A face is determined by its 4 corners.

All other cubes are transformations of this
fundamental cube. The form as this is done
appears in the class Cube.


*/

//The points of the fundamental cube
private static   double oPoints3d[][]=
{
 {0,0,0},
 {1,0,0},
 {0,1,0},
 {0,0,1},
 {1,1,0},
 {1,0,1},
 {0,1,1},
 {1,1,1}
};

/*
//The edges of the fundamental cube
private static int edges[][] =
   {{0,1},
    {0,2},
    {0,3},
    {1,4},
    {1,5},
    {2,6},
```

```
    {2,4},
    {3,5},
    {3,6},
    {4,7},
    {5,7},
    {6,7}};

*/

//The faces of the fundamental cube
private static int faces[][] =
   {{0,1,4,2},
    {0,2,6,3},
    {0,3,5,1},
    {3,5,7,6},
    {1,4,7,5},
    {2,6,7,4}};



 //Measures of the window
 private    int   xWin = 600;
 private    int   yWin = 500;
 private   int nScenes ;
 private   int delay ;
 //Number of cubes in the finger
 private  static  int nPhalanges = 3;
 private  static  Boolean move = true;
 //Prob of existence of a muscle
 private static double p;
 //If muscles[j] =True then   muscle j exists
 private boolean muscles[] = new boolean[28];

 static Random r  = new Random();

 //The constructor of hand is invoked here.
 //A is now the instance of the main class hand
 private static immobile A = new immobile();
 //The inner class Hand is instantiated
```

```
 private static Hand  h = A.new Hand();
 //An auxiliary  instance of Finger
 private static Finger f1=  A.new Finger(nPhalanges);



//The window is enlarged.
//This method is fired automatically
//at the beginning
 public void init()
 {
  setSize(xWin,yWin);
 }

 //Regulation of the velocity of movement
 private    void delay(int delay)
 {
 try
     {
   Thread.sleep(delay );
     }
     catch (InterruptedException e)
     {
   e.printStackTrace();
     }
 }

 //face ii of the cube is drawn
 private    void drawface(Graphics g, Cube cc, int ii)
 {
 for(int l =0; l<4; l++)
  {
   cc.pointsX[l] = 200-cc.shade2d[faces[ii][l]][1];
   cc.pointsY[l] = 200-cc.shade2d[ faces[ii][l]][2];
   //System.out.println(pointsX[l] + " " + pointsY[l]);
  }
  //A polygon is defined by two vectors and an integer
   //number:
   //The first vector lists the x-coordinates of the
   //corners;
```

```
   //the second lists  the y-coordinates. The integer is
   //the number of points.
   //If the last point is different than the first,
   //the first point is closed with the last.
   Polygon face = new Polygon(cc.pointsX,
                      cc.pointsY, 4);
   g.fillPolygon(face);
 }

 private    void drawArm(Graphics g)
 {
// A working cube cc is initialized
Cube  cc = new Cube(1);
//The working cube is re-set.
   cc = h.arm ;
   //Movement begins
   if (move) f1.modifyCube(cc,0);
   cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
   //Projection is calculated
   cc.shade2d =  cc.round(cc.cubeProj );
   //printShade();
   //The cube is drawn
   cc.calculateColor();
   for(int i=0;i<6;i++)
    {
  if (cc.coloredFace[i])
   {
    g.setColor( new Color( cc.colors[i][0],
                   cc.colors[i][1],
                   cc.colors[i][2]));
    drawface(g,cc,i);
   }
     }
   //worked cube is kept to memory
   h.arm = cc;
 }

 private Cube repositPalm(Cube cc)
 {
  cc.oPosition[0]  =  h.arm.cube[3][0];
```

```
  cc.oPosition[1]  =  h.arm.cube[3][1]+5;
  cc.oPosition[2]  =   h.arm.cube[3][2];
  cc.cube = cc.translateCube(cc.cube, cc.oPosition);
  return cc;
}


 private    Cube updatePalm(Cube cc)
 {
 cc = f1.inclination(cc,1);
  cc = f1.rotation(cc,1);

  //cc.printCube(cc.cube, " after rotation");
   cc = repositPalm(cc);
  //cc.printCube(cc.cube , " after relocation");
  for(int i = 0; i<3;i++)
  cc.vTrans[i] = cc.vTrans[i] + cc.deltaTrans[i];
  cc.cube = cc.translateCube(cc.cube,  cc.vTrans);
  //cc.printCube(cc.cube, " after translation");
  cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
  //cc.printCube(cc.cubeProj,  " after projection");
 return cc;
 }

 private    void drawPalm(Graphics g)
 {
// A working cube cc is initialized
Cube  cc = new Cube(1);
//The working cube is re-set.
   cc = h.palm ;
   //Movement begins
   if (move) cc = updatePalm(cc);
   cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
   //Projection is calculated
   cc.shade2d =  cc.round(cc.cubeProj );
   //printShade();
   //The cube is drawn
   cc.calculateColor();
   for(int i=0;i<6;i++)
    {
```

```
  if (cc.coloredFace[i])
   {
    g.setColor( new Color( cc.colors[i][0],
                  cc.colors[i][1],
                  cc.colors[i][2]));
    drawface(g,cc,i);
   }
     }
   //worked cube is kept to memory
   h.palm = cc;
 }

 //Fingers are posited below the palm
 private    void locateFingers()
  {
   for(int i=0; i<5;i++)
for(int j=0; j<3;j++)
 h.finger[i].phalange[0].oPosition[j]
                                  =  h.palm.cube[3][j];
   for(int i=0; i<5;i++)
h.finger[i].phalange[0].oPosition[1]
        = h.finger[i].phalange[0].oPosition[1] -10*i ;

  }


 private    void drawFingers(Graphics g)
 {
  locateFingers();
  for(int j = 4; j>=0;j--)
  {
  if(j==4) nPhalanges = 2;
  else nPhalanges = 3;
   for(int m = 0; m < nPhalanges; m++)
   {
//A working cube cc is initialized
Cube  cc = new Cube(m);
//The working cube is re-set.
    cc = h.finger[j].phalange[m];
    //Movement begins
```

```
    if (!(move))
     {
     cc.deltaPhi[m] = 0;
     cc.deltaTheta = 0;
     }
    cc = h.finger[j].modifyCube(cc,m);
   //Projection is calculated
   cc.shade2d =  cc.round(cc.cubeProj );
   //printShade();
   //The cube is drawn
   cc.calculateColor();
   for(int i=0;i<6;i++)
    {
 if (cc.coloredFace[i])
  {
   g.setColor( new Color( cc.colors[i][0],
                cc.colors[i][1],
                cc.colors[i][2]));
   drawface(g,cc,i);
  }
    }
   //worked cube is kept to memory
   h.finger[j].phalange[m] = cc;
  }
 }
}

//A phalange goes up
private    void  showUp(Graphics g, int j, int m)
{
 h.finger[j].phalange[m].phi[m]
            = h.finger[j].phalange[m].phi[m]  -0.5;
 drawAP(g);
 drawFingers(g);
}

//A phalange goes back down
private    void  showDown(Graphics g, int j, int m)
{
h.finger[j].phalange[m].phi[m]
```

```
              = h.finger[j].phalange[m].phi[m]  +0.5;
  drawAP(g);
  drawFingers(g);

 }


 //Extant muscles are moved one by one
 private    void  movePhalanges(Graphics g)
 {
  locateFingers();
  int counter = 0;
  for(int j = 4; j>=0;j--)
  {
   if (j==4)  nPhalanges = 2;
   else nPhalanges = 3;

   for(int m = 0; m < nPhalanges; m++)
   {
if (muscles[counter]==true)
 {
  pink(g);
  showUp(g,j,m);
  delay(delay);
 }
else  delay(delay);

counter = counter +1;

if (muscles[counter]==true)
{
 if (counter <27) pink(g);
 showDown(g,j,m);
 delay(delay);
}
else  delay(delay);

if (j==4)  nPhalanges = 2;
   else nPhalanges = 3;
counter = counter +1;
```

```
   }
  }
 }

//Forearm and palm are drawn
 private    void drawAP(Graphics g)
 {

 drawArm(g);
 drawPalm(g);
 }


 private    void drawHand(Graphics g)
 {
drawAP(g);
 drawFingers(g);
 }

 //Everything is erased
 private    void pink(Graphics g)
 {
  //background color
  g.setColor(Color.pink);
  g.fillRect(0,0,xWin,yWin);
 }


 private    void dynamics(Graphics g)
 {
 //Loop for the scenes
  for(int k = 0; k<nScenes; k++)
  {
   //System.out.println("scene " + k);

drawHand(g);
    //delay  + erase drawing
    if (k < nScenes-1)
     {
```

```
      delay(delay);
      //erase cube: we repaint everything in
      //the background color.
       pink(g);
     }//end of if
  }//end of for nScenes
 }


 //Fingers move a bit forth and back
 private    void jiggling(Graphics g)
 {
  move = false;
  //background color
  //drawAP(g);
  movePhalanges(g);
 }

 private    void initialPosition(Graphics g)
 {
  pink(g);
  dynamics(g);
 }

 //There are up to 28 possible muscles
 //whose existence is decided here
 private    void muscles( )
 {
  double q;
  for(int i = 0; i<28; i++)
  {
q = r.nextDouble();
if (q<p) muscles[i] = true;
else muscles[i] = false;
  }
 }


 //This is the main method
 public void paint (Graphics g)
```

```
{
 //Repeatability is enabled
 r.setSeed(45);
 r.setSeed(324);
 r.setSeed(1954);
 //Number of scenes of the presentation.
 nScenes = 4;
 //Delay between two scenes
 delay =  0;
 //Probability of   existence of a muscle
 p = 0.7;
 //Muscles are defined
 muscles();
 System.out.println("Immobile");
 initialPosition(g);
 delay =  500;
 delay(delay);
 //A normal hand jiggles
 //pink(g);
 jiggling(g);
}//end of paint


//=============Beginning of Inner class Hand============

public class Hand
{
private Cube arm = new Cube(1);
private Cube palm = new Cube(1);
private Finger finger[] = new Finger[5];

//When memory is allotted for a hand,
//all variables must be initialized.
//Lower classes are initialized by default,
//but some  redefinitions could be necessary.
Hand()
{
 //The arm is initialized
  final Cube c1 =    new Cube(1);
  arm = c1;
```

```
       //Dimensions and location of the arm.
   arm.scale[0] = 8;
   arm.scale[1] = 30;
   arm.scale[2] = 100;
   for(int i=0; i<8;i++)
   for(int j=0; j<3;j++)
       arm.points3d[i][j] =  -arm.scale[j]*oPoints3d[i][j];


   //The  palm is initialized
   final Cube c2 =    new Cube(1);
   palm = c2;
   //Dimensions and location of the palm.
   palm.scale[0] = 8;
   palm.scale[1] = 40;
   palm.scale[2] = 60;
   for(int i=0; i<8;i++)
for(int j=0; j<3;j++)
 palm.points3d[i][j] =  -palm.scale[j]*oPoints3d[i][j];


   //Fingers are initialized
   for(int i=0; i<5;i++)
   {
   final  Finger f =   new Finger(nPhalanges);
   finger[i] = f;
   }




 }
 }
 //=============End of Inner class Hand============

 //=============Beginning of Inner class Finger===========
 /*Variables and methods of the class Finger are
 encapsulated in the same unit.
 All definitions are non static, i.e., they are
 used as erasable workbenches and resettable tools.
```

```
 A finger is a coordinated set of 3 cubes.
 The class cube is defined below.

 All methods that are of exclusive use for fingers are
 encapsulated here.
 */
 public class Finger
 {

 //Our finger is a composition of 3 cubes or phalanges
 private   Cube[] phalange    = new Cube[3] ;


 //When memory is alloted for a finger,
 //all variables must be initialized
 Finger(int nPhalanges)
 {
 for(int m =0; m < nPhalanges; m++)
   {
    final Cube cc =    new Cube(m);
    phalange[m] = cc;
   }
 }
// The only allowed movement of a phalange is
 //inclination, forth and back, in within bounds.
 //Inclination of given phalange governs the
 //rest of the finger
 private   Cube inclination(Cube cc, int m)
 {
  //Inclination is equal to a rotation.
  //Axis of rotation: the y axis.
  cc.axisRot[0] = 0;
  cc.axisRot[1]=1;
  cc.axisRot[2]=0;
  switch(m)
   {
case 0: cc.phi[0]  = cc.phi[0]  + cc.deltaPhi[0] ;
           break;
case 1: cc.phi[1]  = cc.phi[0] + cc.phi[1]
```

```
                                    + cc.deltaPhi[1] ;
            break;
case 2: cc.phi[2]  = cc.phi[1] + cc.phi[2]
                                    + cc.deltaPhi[2] ;
            break;
   }
 cc.cube = cc.rotCube(cc.points3d, cc.axisRot,
                              cc.phi[m] );
 return cc;
}
```

```
    //For visualization, the finger is rotated
//around the z axis.
private   Cube rotation(Cube cc, int m)
{
 //Rotation about the z axis.
 cc.axisRot[0] = 0;
 cc.axisRot[1]=0;
 cc.axisRot[2]=1;
     cc.theta = cc.theta + cc.deltaTheta;
 cc.cube = cc.rotCube(cc.cube, cc.axisRot, cc.theta);
 return cc;
}

 //Each phalange must have its initial position
 //that depends on the position of the phalanges below it
 private   Cube rePosit(Cube cc, int m)
 {
   //System.out.println("rePosit(m)" + m);
 switch(m)
  {
   case 0:  {
      } break;
   case 1: {
cc.oPosition[0]  =   phalange[0].cube[3][0];
cc.oPosition[1]  =   phalange[0].cube[3][1];
```

```
    cc.oPosition[2]  =   phalange[0].cube[3][2];
   break;
       }
   case 2: {
cc.oPosition[0]  =   phalange[1].cube[3][0];
cc.oPosition[1]  =   phalange[1].cube[3][1];
cc.oPosition[2]  =   phalange[1].cube[3][2];
   break;
       }
  }
  cc.cube = cc.translateCube(cc.cube, cc.oPosition);
  return cc;
 }

 //The cube is inclined, rotated, translated, projected
 private    Cube modifyCube(Cube cc, int m)
 {
  cc = inclination(cc,m);
   cc = rotation(cc,m);

  //cc.printCube(cc.cube, " after rotation");
  rePosit(cc,m);
  //cc.printCube(cc.cube , " after relocation");
  for(int i = 0; i<3;i++)
  cc.vTrans[i] = cc.vTrans[i] + cc.deltaTrans[i];
  cc.cube = cc.translateCube(cc.cube,  cc.vTrans);
  //cc.printCube(cc.cube, " after translation");
  cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
  //cc.printCube(cc.cubeProj,  " after projection");
  return cc;
 }




 }

 //==============End of Inner class Finger============
```

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

 //=============Beginning of Inner class Cube============


 public   class Cube
 {
  private double points3d[][] = new double[8][3];
  private   int shade2d[][] = new int[8][3];

//  All cubes are derived from the basic cube by rotations
//  and translations
  private double cube[][]  = new double[8][3];
  private   double  cubeProj[][] = new double[8][3];

  private double oPosition[] = new double[3];
  private   double axisRot[] = new double[3];
  private   double axisProj[]  = new double[3];
  private   double deltaTrans[]  = new double[3];
  private   double vTrans[]  = new double[3];
  private   double theta  = 0;
  private   double deltaTheta;

  private   double basisRot[][], basisProj[][]
                = new double[3][3];
  private   double matrixRotz[][],
  matrixRot[][],
  matrixProj[][] = new double[3][3];
//  The matrix of a projection along the z-axis
  private     double J[][]=
        { {1,0,0},
          {0,1,0},
          {0,0,0}};

  private    double scale[] = new double[3];

  int colors[][] = new int[6][3];
  private   int pointsX[]  = new int[6];
  private   int pointsY[]  = new int[6];
//  How to color faces
```

```
 private    double maxX;
 private   int nExtreme;
 private   boolean maxPoints[]  = new boolean[8];
 private   boolean coloredFace[]  = new boolean[6];
 //Angle of inclination of phalange
 private   double phi[] = new double[3];;
 private   double deltaPhi[] = new double[3];;

//Every implementation of Cube is initialized here.
//Parameter m is used for the initial position
//of cube number m
Cube(int m)
{
 //Original  form
 scale[0] = 8;
 scale[1] = 8;
 scale[2] = 30;
 for(int i=0; i<8;i++)
 for(int j=0; j<3;j++)
 points3d[i][j] =  -scale[j]*oPoints3d[i][j];
 defineColors();
 //Original position
 for(int i=0; i<3;i++)
  oPosition[i] = 0;
 //Each phalange has its own angle of inclination
 //with respect to its support
 for(int i=0; i<3;i++)
 {
 phi[i] = 0;
 deltaPhi[i]  = -0.1*(m+1);
 }
 //  The 3d cube flies along vectTrans[]
 //Each cube has its own direction of translation
 for(int i=0; i<3;i++)
 vTrans[i] = 0;
 deltaTrans[0] =  0;
 deltaTrans[1] =  0;
 deltaTrans[2] =  0;
 //  printCube();
 //  The 3d-graphic is rotated along the axisRot
```

```
  //  an angle theta that is incremented by deltaTheta.
  //  Each cube has its own velocity and direction of
  //  rotation.
  theta = 0;
  deltaTheta = -0.1;
  axisRot[0] = 0;
  axisRot[1]=0;
  axisRot[2]=1;
  //  The 3d-graphic is projected over the screen
  //  which is perpendicular to the x-axis.
  axisProj[0] = 1;
  axisProj[1]=0;
  axisProj[2]=0;


  }


 Random r  = new Random();

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

//The multiplication of two matrixes
private  double[][] matrixMult(double p[][], double q[][])
{
 double[][] helpM = new double[3][3];

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

 for(int i=0; i<3; i++)
```

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

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



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

//The second vector is perpendicular to the first
```

```
private    double[] secondVector(double[] v)
{
 double[] helpV = new double[3];
 double   x,y,z;
 double a = v[0];
 double b = v[1];
 double c = v[2];
 x=b;
 y = -a;
 z=0;
 if (b==0)
  {
   x=c;
   z = -a;
   y=0;
  }
if (a==0)
 {
  y=c;
  z = -b;
  x=0;
 }
 double norm = Math.sqrt(x*x + y*y + z*z);

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

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

```
 helpV[1]  = b/norm;
 helpV[2]  = c/norm;
 return helpV;
}


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

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


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

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

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

/*A 3d cube exists only in the imagination:
* it is not seen.
* One sees a shade or projection of it over the screen.
* The shade is defined by 2d points. The edges of
* the 2d shade are inherited from the edges of the
* 3d cube.
* The 3d cube is projected into a 2d plane.
* The resultant figure must be rendered to the screen.
* A projection is done along  an axis that crosses
* through 0.
* Our projections are observed from infinite:
* we use no perspective corrections. */
private     double[][] mProj(double w[] )
{
 basisProj = basisB(w);
 double[][]  helpM = new double[3][3];
 helpM = matrixMult(J,matrixTranspose(basisProj));
 helpM = matrixMult(basisProj,helpM);
```

```
 //printMatrix(helpM, "Proj");
 //Test: Proj*Proj must be equal to Proj
 /*helpM = matrixMult(helpM,helpM);
 printMatrix(helpM, "Proj* Proj");*/
 return helpM;
}

//The cube c is rotated around axis v an angle theta
private double[][] rotCube(double c[][],
                           double v[], double theta)
{
 double[]  helpV = new double[3];
 double cc[][] = new double[8][3];
 matrixRot = mRotation( v,theta);
 //printMatrix(matrixRot, "matrixRot");
 //The 8 points of the cube must be rotated
 for(int i=0; i<8; i++)
 {
  for(int j=0; j<3; j++)
  helpV[j] = c[i][j];
  helpV = matrixVector(matrixRot, helpV);
  for(int j=0; j<3; j++)
  cc[i][j] = helpV[j];
 }
 //printCube(cc, " after rot");
 return cc;
}



//The cube c is translated   deltaV
private double[][] translateCube(double c[][],
                                         double deltaV[])
{
 double cc[][] = new double[8][3];
 for(int i=0; i<8; i++)
 for(int j=0; j<3; j++)
 cc[i][j] = c[i][j] + deltaV[j];
 return cc;
}
```

```
//The cube c is projected along axis w
private double[][] projCube(double c[][],  double w[])
{
 double[]  helpV = new double[3];
 double cc[][] = new double[8][3];
 matrixProj = mProj(w);
 for(int i=0; i<8; i++)
  {
   for(int j=0; j<3; j++)
   helpV[j] = c[i][j];
   helpV = matrixVector(matrixProj, helpV);
   for(int j=0; j<3; j++)
   cc[i][j] = helpV[j];
  }
 return cc;
}

//The rendering procedure accepts integer values
private     int[][] round(double c[][])
{
 int[][] helpM = new int [8][3];
 for(int i=0;i<8;i++)
 for(int j=0; j<3; j++)
 helpM[i][j] = (int) Math.round(c[i][j]);
 return helpM;
}

private   void printCube(double c[][], String a)
{
 System.out.println("This is the cube" +a);
 for(int i=0;i<8;i++)
 System.out.println(c[i][0] + " " + c[i][1]+ " "
                                      + c[i][2]);
}

private   void printMatrix(double m[][], String a)
{
 System.out.println("This is the matrix " + a);
```

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

private void printShade()
{
 System.out.println("This is the shade");
 for(int i=0;i<8;i++)
 System.out.println(shade2d[i][1] + " "
                                   + shade2d[i][2]);
}

//Colors to paint the faces are defined
private int[][] defineColors()
{
 for(int l = 0; l<6;l++)
  {
   //We use random colors.
   //A color can be defined by three numbers,
   //each one from 0 to 255.
   //We has 6 faces, so we need
   //a matrix to keep colors.

   int c1,c2,c3;
   c1 = r.nextInt(226);
   c2 = r.nextInt(226);
   c3 = r.nextInt(226);
   //Colors are memorized
   colors[l][0]= c1;
   colors[l][1]= c2;
   colors[l][2]= c3;
  }
 return colors;
}

//what faces must be colored
private    void marked()
{
 for(int f = 0; f<6; f++)
```

```
 coloredFace[f] = false;
 for(int f = 0; f<6;f++)
 {
  for(int p = 0; p<4;p++)
  {
   for(int c = 0; c<8; c++)
    {
     if (maxPoints[c] == true)
     {
       if (faces[f][p] == c )
       coloredFace[f] = true;
       //System.out.println("coloring");
     }
   }
  }
 }
}


//The x coordinate of all points are measured
//and the extreme ones are recorded.
private    void calculateColor()
{
 //The maximum value of coordinate x
 maxX =-1000;
 for(int l = 0; l<8; l++)
  {
   if (cube[l][0] > maxX)
    {
     maxX = cube[l][0];
    }
   }
 nExtreme = 0;
 //Number of extreme points
 for(int l = 0; l<8;l++)
  {
   maxPoints[l] = false;
   if (Math.abs(cube[l][0] - maxX )< 2)
   {
    nExtreme = nExtreme +1 ;
```

```
    maxPoints[l] = true;
   }
  }
  marked();
 }
 }//=============End of inner class Cube==============


}//end of main class
```

56 ***Exercise*** *Play with the code. Try to imitate with your own left hand the movements shown by the program. Challenge the member of your family to make the same.*

57 ***Gods are reveled***

We have produced a frame that predicts that people are compelled by simple and powerful reasons to believe that chance is unable to explain their existence. To understand the prompts given by the instinct, let us pay attention to two facts. First: a human being is a fabric of beliefs, a capability that is possible because, as we have shown, solid beliefs can be invented assuming relatively simple considerations. Second: every human has an extremely developed capability of thinking, of analyzing. Summing up, we conclude that the prompts given by the instinct, by the intuition, cannot be taken as the ultimate truth but only as an invitation to think further. This directive will carry us along a very long and difficult road.

To reject the idea that chance is the explanation of life is a fact that compels us to look for a new explanation. In modern times, that explanation is evolution but for ancient cultures, the **belief** that chance is unable to explain the existence of life was equivalent to say that we were created. But such a **belief** was not a logical conclusion or a human postulate alone. In general, gods are not invented, instead they are reveled: the belief in gods is a product of dreams or visions that some people experienced and that are not taken as inventions of the mind but as a manifestation of super natural spiritual entities. Nevertheless, why do these claims are accepted by some people? It is because the human mind is bold enough to calculate the idea that nature alone is unable of explaining itself.

4.3 Conclusion

An irrational **belief** is one that is exercised by intuition, by instinct. It appears as a fact presented by the inner artificial reality to the consciousness. Many people

behave as if the belief in creation, as an alternative to have appeared by chance, were one of such facts of consciousness. Now, the sub-conscious machinery that produces and tests beliefs must consist of neural networks that make calculations. We have sketched a model that allows us to think that such calculations are at the same time simple and compelling and correspond to the most elementary characterization of chance: chance is crazy in a world of few trials but it acquires perfectly predictable folds when experiments are repeated a large number of times. Thus, chance can be ruled as an explanation of something if the predictions of chance do not conform to observed facts of that something. Our insight allows us to predict that when we must choose between chance and a creator, we will choose creation, an option that appears every time one looks at own left hand. In fact, the faith in a creator could be as strong as the belief in gravity and might be produced by exactly the same neural machinery.

Chapter 5

The hand of a tiny god

```
Power to change the world at will.
```

58 ***Introduction*** *Very small children can accept the idea of an invisible god that has power to protect them from every bad and that can listen to them. By the same token, people like to associate gods with creation and they are ready to think that gods have an immense power of control over matter and nature. In consequence, persons of very diverse cultures and religions like to ask their gods very specific favors that must done by someone that has the precision of a surgeon. Why? In part that is due to the fact that everybody knows people that has immense powers of control, say, a father for his children or a boss for his workers. But this is also possible because every person with a hand can experience on him or herself what a god is: a god is a person that can take decisions and that have precision instruments to execute them.*

59 ***Purpose.*** *We enjoy in this chapter the capability that Java offers to the User to feel like a god: by clicking with the mouse, one can do marvels. Warning: those marvels are done thanks to a previous and very hard work of programming made by the developers and by those that designed the Java language and by those that designed and made the hardware. So, we do not believe in magic, we believe in a previous hard work instead. Or, we believe that magic can and must be programmed.*

5.1 Detecting the mouse

The first task in relation with the mouse is **to detect** what the User does with it. We learn how this is done in the next program. Thanks to a marvel work of

programming at the basis of the Java Language, this is as easy and fun as a game for children.

60 ***The code***

A JPanel, a special type of window, is appropriate to work with the mouse. The mouse can be clicked, pressed, released, moved or dragged. The operated button can also be registered. These operations can be detected when the mouse is in the interior of the JPanel. Entering or exiting of the mouse can be also detected. The code follows:

```
/* Program D60  mouseDetector.
  Detecting the action of the mouse.
  The mouse can be
  clicked, pressed, released, moved or dragged.
  This operations can be detected when the mouse
  is in the interior of the JPanel.
  The operated button can also be registered.
  Entering or exiting of the mouse can  be also detected.
*/
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.awt.event.MouseEvent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;


//Main class: it activates the inner class MouseDetecting
public class mouseDetector
{

  static mouseDetector d = new mouseDetector();

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

```
  //The inner class is activated
  MouseDetecting m  = d.new MouseDetecting();
  //termination from the window is enabled
      m.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      m.setSize(250,250);
      m.setVisible(true);
   }


//Inner class MouseDetecting
public class MouseDetecting extends JFrame
{

 private static final long serialVersionUID = 1L;

 //The special window is declared
 private JPanel mPanel;
 private JLabel barMessage; // label that displays event information

 //Constructor
 public MouseDetecting()
  {
   //Title
   super( "Detecting the mouse" );
   //Instantiation of the JPanel
   mPanel = new JPanel();
   mPanel.setBackground( Color.gray);
   add( mPanel, BorderLayout.CENTER );
   barMessage = new JLabel( "Mouse outside frame" );
   add( barMessage, BorderLayout.SOUTH );

   //Declaration and instantiation of dectecting machinery
   MouseHandler handler = new MouseHandler();
   mPanel.addMouseListener( handler );
   mPanel.addMouseMotionListener( handler );
  }

 //Detecting machinery as inner class.
 //All methods enables services that work in parallel
 //All methods are Java-given, which hides from the User
```

```
 //the enabling low-level code.
 //The developer can change the content of each method,
 //which is added to the invisible one.
 private class MouseHandler implements MouseListener,
      MouseMotionListener
 {
  public void mouseEntered( MouseEvent event )
   {
//Some invisible code goes in here.
mPanel.setBackground( Color.pink );
System.out.format ("The mouse entered the frame" +
"at [%d, %d] \n", event.getX(), event.getY() );
   }

  public void mouseClicked( MouseEvent event )
  {
   barMessage.setText( String.format( "Clicked with " +
                              "button %d at [%d, %d]",
         event.getButton(), event.getX(), event.getY() ) );
  }

  public void mousePressed( MouseEvent event )
  {
   barMessage.setText( String.format( "Pressed at" +
   "  [%d, %d] with button %d",
        event.getX(), event.getY(), event.getButton() ) );
  }

  public void mouseReleased( MouseEvent event )
  {
   barMessage.setText( String.format( "Mouse released " +
     " at [%d, %d]",  event.getX(), event.getY() ) );
  }


  public void mouseMoved( MouseEvent event )
  {
    barMessage.setText( String.format( "Mouse moved at" +
      " [%d, %d]", event.getX(), event.getY() ) );
  }
```

```
  public void mouseDragged( MouseEvent event )
  {
    barMessage.setText( String.format( "Mouse dragged at" +
        " [%d, %d]", event.getX(), event.getY() ) );
    System.out.println("Active button = "
                              + event.getButton());
  }

  public void mouseExited( MouseEvent event )
  {
   barMessage.setText( "The mouse is outside frame" );
   mPanel.setBackground( Color.gray );
   System.out.format ("The mouse exited the frame at" +
         " [%d, %d] \n", event.getX(), event.getY() );
  }

 }  //end of inner class MouseHandler
}//end of inner class   MouseDetecting

}//end of main class
```

61 ***Exercise*** *Play with the code. Try some modifications to the detecting machinery.*

62 ***Paint*** *The next program can help to understand the basic idea of a painting program.*

```
/*Program D62 Paint
 *
 * We show the basic idea of
 * a program for painting
 */
import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import java.awt.Point;
import java.awt.Graphics;
```

```
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionAdapter;
import javax.swing.JPanel;


public class Paint
{
static Paint p = new Paint();
   public static void main( String args[] )
   {

      JFrame f = new JFrame( "Paint" );

      Canvas c = p.new Canvas();
      f.add( c, BorderLayout.CENTER );


      f.add( new JLabel(
     "Draw something with the left button of the mouse" ),
         BorderLayout.SOUTH );

      f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      f.setSize( 400, 200 );
      f.setVisible( true );
   }

   //This is a Java-given interface.
   //The modifications made by the developer are added
   //to the implementation that is hidden.
   public class Canvas extends JPanel
   {

private static final long serialVersionUID = 1L;


      private Point point;
      private boolean flag=false;

      //Constructor
      public Canvas()
```

```
      {
       //initializes the frame;
       repaint();
         // Prepares to capture dragging events
         addMouseMotionListener
         (
            new MouseMotionAdapter()
            {
               public void mouseDragged( MouseEvent event )
               {
                  //gets point
                  point = event.getPoint();
                  //enables drawing
                  flag = true;
                  //paints point
                  repaint();
               }
            }
         ) ;
      }

      //Draws a point when repaint() is called
      public void paintComponent( Graphics g )
      {
         if (flag) g.fillOval( point.x,  point.y, 5, 5 );
      }
   } // end class Canvas

} // end of main class
```

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

5.2 Pop-up menus

At its present state of development, the philosophy of OOP has its maximum expression in **Pop-up menus** that hang from the mouse: the whole programming machinery spins around the intention to serve the object. So, if your object is an orange, you might enable the possibility to cut it in pieces, or to translate it, or to

have it falling, and a list of these operations should be at the immediate disposition of the User if only the right button of the mouse is clicked. The User then chooses one option with the right button.

64 ***The code*** *The next program illustrates what is behind the magic of the pop up menus that pop when one clicks the right button of the mouse.*

```
//Program D64 PopupMenu
//A lane has been sensitized
//to the clicking of the right button of the mouse.
//In response, a popup menu is activated.

import java.awt.Color;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JFrame;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.JPopupMenu;
import javax.swing.ButtonGroup;


public class PopupMenu
{
 static PopupMenu p = new PopupMenu();
 private  static int width=50;

 public static void main( String args[] )
 {
   Lane f = p.new Lane();
   f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
   f.setSize(400,300);
   f.setVisible(true);
  }


 public class Lane extends JFrame
 {
```

```
 private static final long serialVersionUID = 1L;
 private JRadioButtonMenuItem items[];
 private final Color colorValues[] =
  {Color.BLACK, Color.CYAN, Color.MAGENTA, Color.ORANGE};
 private JPopupMenu popupMenu;

public Lane()
{
  //Constructor of the title
  super("Use the right button to find and color a lane");
  //handler for menu items
  ItemHandler handler = new ItemHandler();
  //Names of used colors
  String colors[] = {"BLACK","CYAN","MAGENTA","ORANGE"};
  //Colors are chosen by clicking a button
  ButtonGroup colorGroup = new ButtonGroup();
  //creates pop-up menu
  popupMenu = new JPopupMenu();
  //Buttons for selecting colors
  items = new JRadioButtonMenuItem[ 4 ];

  //The menu is constructed: it has 4 buttons
  for ( int count = 0; count < items.length; count++ )
  {
   items[ count ] =
         new JRadioButtonMenuItem( colors[ count ] );
   //add item to pop-up menu
   popupMenu.add( items[ count ] );
   //add item to button group
   colorGroup.add( items[ count ] );
   //add handler
   items[ count ].addActionListener( handler );
  }
  //Mouse actions are detected
  addMouseListener
   (
     new MouseAdapter()
     {
      //Detector of right button releasing operation
      public void mouseReleased( MouseEvent event )
```

```
       {
        if ((event.isPopupTrigger())&(event.getX()<width))
         popupMenu.show(
          event.getComponent(),event.getX(),event.getY());
        }
       }
     );
  }

//If an event has been detected, here is the reaction
private class ItemHandler implements ActionListener
{
 // process menu item selections
 public void actionPerformed( ActionEvent event )
 {
  //Determines which option was selected
  for ( int i = 0; i < items.length; i++ )
  {
   if ( event.getSource() == items[ i ] )
    {
 //A lane is delimited and colored
     getContentPane().setBounds(1, 1, width, 1000);
     getContentPane().setBackground(colorValues[i]);
     return;
    }
  }
 }
}

} // end of class Lane
} // end of class PopupMenu
```

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

66 ***PaintPlus*** *Let us try to add pop-up menus to our painting program: we must enable the possibility to change the color of the pen, or the thickness of its trace by just clicking on the right button of the mouse.*

```
/*Program D66 PaintPlus1
 * Popup menus have been added to
 * the paint program:
 * the User can change with the right button
 * of the mouse the color of the pen as well as
 * the thickness of the trace.
 * The present  has the following Bugs:
 * 1) To activate the right button, one must
 * click far away from the drawing, otherwise
 * the drawing will be erased.
 * 2) a black color appears without any calling
 * 3) It is impossible to use a width of trace = 3
 *    with a magenta color.
 * 4) Noise appears everywhere.
 *
 * It is not known how to make software without
 * committing bugs. Most possible, committing bugs
 * is the cheaper style of software design.

 */

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Point;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.MouseMotionAdapter;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.JPopupMenu;
import javax.swing.ButtonGroup;
```

```
public class PaintPlus1 extends JFrame
{
  private static final long serialVersionUID = 1L;

  //The application is instantiated
  static PaintPlus1 p = new PaintPlus1();

 //The painting class is declared and instantiated
 private static Canvas c = p.new Canvas();

 //the option chosen by the User
 private static int option;

 public static void main( String args[] )
 {
  //The JFrame is constructed with a title
  JFrame f = new JFrame( "PaintPlus1" );

  //specifications of f
  f.add( c, BorderLayout.CENTER );
  f.add( new JLabel("Draw something"),BorderLayout.SOUTH );
  f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  f.setSize( 400, 200 );
  f.setVisible( true );
  //Popup menu is enabled
  final pppMenu pp = p.new pppMenu();
  pp.addNotify();//useless but avoids a warning
 }


 //This class enables the use of right clicking
 public class pppMenu extends JFrame
 {
  private static final long serialVersionUID = 1L;
  private JRadioButtonMenuItem items[];

  private JPopupMenu popupMenu;

 public pppMenu()
 {
```

```
  //handler for menu items
  ItemHandler handler = new ItemHandler();
  //Names of used colors
  String options[] = {"BLUE", "MAGENTA","Pen = 3","Pen = 5"};
  //Colors are chosen by clicking a button
  ButtonGroup colorGroup = new ButtonGroup();
  //creates pop-up menu
  popupMenu = new JPopupMenu();
  //Buttons for selecting colors
  items = new JRadioButtonMenuItem[ 4 ];

  //The menu is constructed: it has 4 buttons
  for ( int count = 0; count < items.length; count++ )
  {
   items[ count ] =
         new JRadioButtonMenuItem( options[ count ] );
   //add item to pop-up menu
   popupMenu.add( items[ count ] );
   //add item to button group
   colorGroup.add( items[ count ] );
   //add handler
   items[ count ].addActionListener( handler );
  }
  //Mouse actions are detected
  //Attention: the mouse listener is added to the canvas c!
  c.addMouseListener
   (
     new MouseAdapter()
     {
      //Detector of right button releasing operation
      public void mouseReleased( MouseEvent event )
      {
       if  ( event.isPopupTrigger() )
        popupMenu.show(
         event.getComponent(),event.getX(),event.getY());
       }
      }
    );
 }
```

```
//If an event has been detected, here is the reaction
private class ItemHandler implements ActionListener
{
 // process menu item selections
 public void actionPerformed( ActionEvent event )
 {
  //Determines which option was selected
  for ( int i = 0; i < items.length; i++ )
   {
    if ( event.getSource() == items[ i ] )
    {
     option = i;
     return;
    }
  }
 }
}

} // end of class  pppMenu

 //This is a Java-given interface.
 //The modifications made by the developer are added
 //to the implementation that is hidden.
 public class Canvas extends JPanel
 {

private static final long serialVersionUID = 1L;


    private Point point;
    private boolean flag=false;

    //Constructor
    public Canvas()
    {
     //initializes the frame;
     repaint();
       // Prepares to capture dragging events
       addMouseMotionListener
```

```
      (
         new MouseMotionAdapter()
         {
            public void mouseDragged( MouseEvent event )
            {
               //gets point
               point = event.getPoint();
               //enables drawing
               flag = true;
               //paints point
               repaint();
            }
         }
      ) ;
   }

   //Draws a point when repaint() is called
   public void paintComponent( Graphics g )
   {
    int a=4;
    int b = 4;
    if (option == 0) g.setColor(Color.BLUE);
    if (option == 1) g.setColor(Color.MAGENTA);
    if (option ==2)
    {
        a=3;
        b = 3;
    }
    if (option ==3)
    {
        a=5;
        b = 5;
    }
      if (flag) g.fillOval( point.x,  point.y, a, b );
   }
 } // end of class Canvas

} // end of main class
```

67 ***Exercise*** *Run the program and play with the code. Witness the existence of some bugs. Every piece of software exists because somebody have battled with the bugs that always appear. This fact is for us a law of software design: every developing path is paved with every kind of half-cooked, bugged products.*

68 ***Partial correction of bugs***

The next version fixes the bug with colors: one can paint in the color and with the thickness one chooses. Furthermore, the drawing is not erased when the menu pops up and then disappears.

```
/*Program D68 PaintPlus2
 * We are trying to ad Popup menus   to
 * the paint program.
 * We are battling with bugs.
 * Here is another version.
 *
 * In general, when one tries to correct a bug,
 * other bugs appear while there are
 * bugs that are very difficult to correct.
 * That is why the industry releases beta versions
 * of their products to identify the more
 * frequent bugs to next work hard in their correction.
 *
 * Because bugs are immanent to design and
 * correcting bugs generate more bugs,
 * any developing path is always plagued
 * with half-cooked, dysfunctional products.
 * Nevertheless, every developer likes to do his
 * or her best to offer clean, terminated products.
 * So, a developer possibly advances everyday but
 * releases products rather from time to time.
 *
 * The whole phenomenon includes two perspectives
 * of the developing process: one by the user,
 * the other by the developer.
 * The user sees an evolution from
 * a success to a greater success.
 * The developer sees a developing path that is
```

```
 * filled in every kind of pitfalls but with
 * an increasing envelope.
 *
 * This declaration forms or basic laws of
 * software development and we declare them
 * to be valid for rational or irrational
 * developers.
 */

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Point;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.MouseMotionAdapter;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.JPopupMenu;
import javax.swing.ButtonGroup;



public class PaintPlus2 extends JFrame
{
  private static final long serialVersionUID = 1L;

  //The application is instantiated
  static PaintPlus2 p = new PaintPlus2();

 //The painting class is declared and instantiated
 private static Canvas c = p.new Canvas();

 //the option chosen by the User
 private static int option;
```

```
//thickness of the trace
private static int a;
//points must be memorized to be repainted
//when the menu appears and disappears
private static Point points [] = new Point[10000];
private static Color colors[] = new Color[10000];
private static int thickness[] = new int[10000];
//counter of drawn points.
private static int counter=0;
//inital color
private static Color color = Color.BLACK;


public static void main( String args[] )
{
 //The JFrame is constructed with a title
 JFrame f = new JFrame( "PaintPlus1" );

 //specifications of f
 f.add( c, BorderLayout.CENTER );
 f.add( new JLabel("Draw something"),BorderLayout.SOUTH );
 f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
 f.setSize( 400, 200 );
 f.setVisible( true );
 //Popup menu is enabled
 a=4;
 final pppMenu pp = p.new pppMenu();
 pp.addNotify();//useless but avoids a warning
}


//This class enables the use of right clicking
public class pppMenu extends JFrame
{
 private static final long serialVersionUID = 1L;
 private JRadioButtonMenuItem items[];

 private JPopupMenu popupMenu;

public pppMenu()
```

```
{

  //handler for menu items
  ItemHandler handler = new ItemHandler();
  //Names of used colors
  String options[] = {"BLUE", "MAGENTA","Pen = 3","Pen = 5"};
  //Colors are chosen by clicking a button
  ButtonGroup colorGroup = new ButtonGroup();
  //creates pop-up menu
  popupMenu = new JPopupMenu();
  //Buttons for selecting colors
  items = new JRadioButtonMenuItem[ 4 ];

  //The menu is constructed: it has 4 buttons
  for ( int i = 0; i < items.length; i++ )
  {
   items[ i ] =
         new JRadioButtonMenuItem( options[ i ] );
   //add item to pop-up menu
   popupMenu.add( items[ i ] );
   //add item to button group
   colorGroup.add( items[ i ] );
   //add handler
   items[ i ].addActionListener( handler );
  }

  //ATTENTION: the mouse listener is added to the canvas c!
  //Mouse actions are detected
  c.addMouseListener
   (
     new MouseAdapter()
     {
      //Detector of right button releasing operation
      public void mouseReleased( MouseEvent event )
      {
       if  ( event.isPopupTrigger() )
        popupMenu.show(
         event.getComponent(),event.getX(),event.getY());
       }
      }
```

```
     );
  }

//If an event has been detected, here is the reaction
private class ItemHandler implements ActionListener
{
 // process menu item selections
 public void actionPerformed( ActionEvent event )
 {
  //Determines which option was selected
  for ( int i = 0; i < items.length; i++ )
   {
    if ( event.getSource() == items[ i ] )
    {
     option = i;
     return;
    }
  }
 }
}

} // end of class  pppMenu

 //This is a Java-given interface.
 //The modifications made by the developer are added
 //to the implementation that is hidden.
 public class Canvas extends JPanel
 {

private static final long serialVersionUID = 1L;


    //Constructor
    public Canvas()
    {

       // Prepares to capture dragging events
       addMouseMotionListener
       (
          new MouseMotionAdapter()
```

```
          {
             public void mouseDragged( MouseEvent event )
             {
                //gets point
                points[counter] = event.getPoint();
                //data is kept to memory
                colors[counter] = color;
                thickness[counter] = a;
                counter = counter +1;
                //repaints everything
                repaint();
             }
          }
       ) ;
    }

  //Draws a point when repaint() is called
  public void paintComponent( Graphics g )
   {
//Everything is repainted
    for(int i=0; i<counter && points[i] !=null; i++)
     {
      g.setColor(colors[i]);
      a= thickness[i];
      g.fillOval( points[i].x,  points[i].y, a, a );
     }
     switch (option  )
     {
      case 0: color = Color.BLUE; break;
      case 1: color = Color.MAGENTA; break;
      case 2: a=3; break;
      case 3: a=5; break;
     }
    }
 } // end of class Canvas

} // end of main class
```

69 ***Exercise*** *Run the program and play with the code. Add one more possible color and a new thickness. Our solution can be found in the program below.*

70 ***Bugfree programs***

Every developer would like to always release only clean, bugfree terminated products. This is possible but a price must be paid: the releasing of the product must be delayed. The more complex a project is, the longer must be the delaying in its releasing. The accepted solution to this dilemma is to release beta versions and to ask for help to the Users to report the bugs they find to next attack the most frequent.

Our painting application is basically the union of two applications that worked separately: one that paints and the other that offers a popup menu. So, our task is to join them and the problems that we found might be named interconnection bugs. In this regard, there are two strategies that help a lot:

- To simply the interconnection as much as possible, so every piece of information that can be processed locally must be processed locally.
- The interconnection items must be reunited into a class, an object that is stable against automatic memory disposal. In the program below, that class is named *Data*, it plays the role of cement when one glues two tiles.

71 ***Bugfree painting program*** *In the next version all bugs of our* ***painting application*** *have been corrected. It is clean thanks to the fact that this project is not very complex.*

```
/*Program D70 PaintPlus3
 * Popup menus + sensing of the mouse
 */

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Point;
import java.awt.event.MouseEvent;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
```

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.JPopupMenu;
import javax.swing.ButtonGroup;


public class PaintPlus3 extends JFrame
{
  private static final long serialVersionUID = 1L;

 //The application is instantiated
 static PaintPlus3 p = new PaintPlus3();
 //A class with important data is instantiated
 private static Data d = p.new Data();
 //The painting class is declared and instantiated
 private static Canvas c = p.new Canvas();
 //Popup menu is  declared and instantiated
 private static pppMenu pp = p.new pppMenu();



 public static void main( String args[] )
 {
  //The JFrame is constructed with a title
  JFrame f = new JFrame( "PaintPlus3" );
  f.add( pp, BorderLayout.CENTER );
  f.add( c, BorderLayout.CENTER );
  f.add( new JLabel("Draw something"),BorderLayout.SOUTH );
  f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  f.setSize( 400, 800 );
  f.setVisible( true );
 }


//================================================
//=========Data CLASS============================
//================================================
```

```
//Data that serve communication
//among different classes is reunited into an object.
private class Data
{
 private static final long serialVersionUID = 1L;


 //Points must be memorized to be repainted
 //when the menu appears and disappears
 private  Point points [] = new Point[10000];
 private Color colors[] = new Color[10000];
 private  int thickness[] = new int[10000];

 //counter of drawn points.
 private  int counter;


public Data()
{
 Point p = new Point(0,0);
 for(int i = 0; i<10000; i++)
 {
  points[i] = (Point) p.clone();

 }
 for(int i = 0; i<10000; i++)
 colors[i] = Color.black;
 for(int i = 0; i<10000; i++)
 thickness[i] = 3;
 counter = -1;
}
}

//=========================================
//==============pppmenu class===============
//=========================================

 //This class enables the use of right clicking
 //to activate a pop up menu.
```

```
//Sensing of the dragging of the mouse is also enabled.
public class pppMenu extends JPanel
{
 private static final long serialVersionUID = 1L;
 private JRadioButtonMenuItem itemsColor[];
 private JRadioButtonMenuItem itemsThickness[];
 private JPopupMenu popupMenu;
 //Thickness of the trace
 private int a = 3;
 private  Color color = Color.BLUE;

public pppMenu()
{
  //Handler for color menu items
  ColorItemHandler colorHandler = new ColorItemHandler();
  //Names of used colors
  String optionsColor[] = {"BLUE", "MAGENTA",
                  "GREEN", "GRAY"};
  //Colors are chosen by clicking a button
  ButtonGroup colorGroup = new ButtonGroup();
  //Creates pop-up menu
  popupMenu = new JPopupMenu();
  //Buttons for selecting colors
  itemsColor = new JRadioButtonMenuItem[ 4 ];

  //The menu is constructed: it has 4 buttons
  for ( int i = 0; i < itemsColor.length; i++ )
  {
   itemsColor[ i ] =
         new JRadioButtonMenuItem( optionsColor[ i ] );
   //add item to pop-up menu
   popupMenu.add( itemsColor[ i ] );
   //add itemsColor to button group
   colorGroup.add( itemsColor[ i ] );
   //add colorHandler
   itemsColor[ i ].addActionListener( colorHandler );
   itemsColor[ 0 ].setSelected(true);
  }

  popupMenu.addSeparator();
```

```
   //handler for thickness-of-the-pen-menu items
   ThicknessItemHandler handlerThickness =
                         new ThicknessItemHandler();
   //Names of used thicknesses
   String optionsThickness[] = {"Pen = 3","Pen = 5",
                        "Pen = 7"};

   //Thickness is chosen by clicking a button
   ButtonGroup ThicknessGroup = new ButtonGroup();

   //Buttons for selecting thickness
   itemsThickness = new JRadioButtonMenuItem[ 3 ];

   //The menu is constructed: it has 3 buttons
   for ( int i = 0; i < itemsThickness.length; i++ )
   {
    itemsThickness[ i ] =
      new JRadioButtonMenuItem( optionsThickness[ i ] );
    //add item to pop-up menu
    popupMenu.add( itemsThickness[ i ] );
    //add item to button group
    ThicknessGroup.add( itemsThickness[ i ] );
    //add handler
    itemsThickness[i].addActionListener(handlerThickness);
    itemsThickness[ 0 ].setSelected( true );
   }
   // Detection of the mouse
   MouseHandler mouseHandler = new MouseHandler();
   //Sensors are added to the canvas c
   c.addMouseListener( mouseHandler );
   c.addMouseMotionListener( mouseHandler);
  }

//If an event has been detected, here is the reaction
private class ColorItemHandler implements ActionListener
{
 // processing of menu item selections
 public void actionPerformed( ActionEvent event )
 {
```

```
  //Determines which option was selected
  for ( int i = 0; i < itemsColor.length; i++ )
   {
    if ( event.getSource() == itemsColor[ i ] )
    {
     switch (i  )
      {
       case 0: color = Color.BLUE; break;
       case 1: color = Color.MAGENTA; break;
       case 2: color = Color.GREEN; break;
       case 3: color = Color.GRAY; break;
      }
    }
  }
  c.repaint();
 }
}


//If an event has been detected, here is the reaction
private class ThicknessItemHandler implements
                                          ActionListener
{
 // processing of menu item selections
 public void actionPerformed( ActionEvent event )
 {
  //Determines which option was selected
  for ( int i = 0; i < itemsThickness.length; i++ )
   {
    if ( event.getSource() == itemsThickness[ i ] )
    {
     switch (i  )
     {
      case 0: a = 3; break;
      case 1: a = 5; break;
      case 2: a = 7; break;
     }
    }
  }
  c.repaint();
```

```
 }
}

//The mouse is detected
private class MouseHandler implements MouseListener,
MouseMotionListener
{
public void mouseEntered( MouseEvent event )
{

}

public void mouseClicked( MouseEvent event )
{

}

public void mousePressed( MouseEvent event )
{

}

//When the right button is released, the
//menu pops up.
public void mouseReleased( MouseEvent event )
{
 if  ( event.isPopupTrigger() )
         popupMenu.show(
          event.getComponent(),event.getX(),event.getY());

}


public void mouseMoved( MouseEvent event )
{

}

//When the mouse is dragged with one button down
//the movement is registered and used to draw.
```

```
public void mouseDragged( MouseEvent event )
{
  d.counter = d.counter +1;
  //gets point
  d.points[d.counter] = event.getPoint();
  //data is kept to memory
  d.colors[d.counter] = color;
  d.thickness[d.counter] = a;
  c.repaint();
}

public void mouseExited( MouseEvent event )
{

}

}  //end of inner class MouseHandler

} // end of class  pppMenu

//===========================================
//==============Canvas class=================
//===========================================

 //This is the drawing class
 public class Canvas extends JPanel
 {

  private static final long serialVersionUID = 1L;

  //Draws the points when repaint() is called
  public void paintComponent( Graphics g )
   {
int b=0;
for(int i=0; i<=d.counter; i++)
      {
      g.setColor(d.colors[i ]);
         b= d.thickness[i ];
         g.fillOval( d.points[i].x, d.points[i].y, b, b );
      }
```

```
   }
 } // end of class Canvas

} // end of main class
```

72 ***Exercise and challenge*** *Run the program and play with the code. Verify that it is bug free. Some designers would consider that one must surpass the barrier of bugfree programs and to produce software with style. In this regard, notice that one can draw with all three buttons of the mouse. A good style would be to restrict drawing to the left button while the popup menu is assigned to the releasing of the right one. To implement this stylistic improvement is a good challenge for everybody.*

73 ***Which is the best style?*** *It is an experimental truth that every developer has his or her own style that distinguishes him or her from the rest. So, one can ask: which is the best style? Our answer is: the best style is the evolvable style, i.e., an style that eases changes, amendments, improvements, addition of new functions, evolution toward higher complexity. The reason is that software is before everything else a tool for people that in general do not know what they want, so they are always demanding changes and better quality. How to design evolvable software is not easy: it is nowadays a theme of intense research: see volume three and the references therein. The epitome of evolvable software is the genome.*

74 ***Our laws of software design*** *Let us summarize our experience with bugs in the next descriptive laws, which express the most direct experience of every designer:*

1. *We do not know how to produce software without bugs. We postulate that committing bugs is the cheapest way of producing software.*
2. *Because bugs are unavoidable, they must be corrected. But in general, when one tries to correct a bug, other bugs appear while there are bugs that are very difficult to correct. That is why the industry releases beta versions of their products to identify the more frequent bugs to next work hard in their correction.*
3. *Because bugs are immanent to design and correcting bugs generate more bugs, any developing path is very long and is always plagued with half-cooked, dysfunctional products. Nevertheless, every developer likes to do his or her best to offer clean, terminated products. So, a developer possibly advances everyday but releases products rather from time to time.*

4. *The whole phenomenon includes two perspectives of the developing process: one by the user, the other by the developer. The user sees an evolution from a success to a greater success. The developer sees a developing path that is filled in every kind of pitfalls but with an increasing envelope. We reported only 3 intermediate states in the developing process of our painting application, but a complete report may include more than 100 viable programs.*

5. *This declaration forms our basic laws of software design and we postulate them to be valid for both rational and irrational developers.*

75 ***Exercise*** *Discuss our laws of software design and propose an answer to the next intrigue: Are our laws, which are correct for human designers, valid for evolution? Recall that evolution has no purpose at all and therefore it cannot commit bugs, errors against design. Our answer was discussed at length in the third volume and we will review it somewhere below.*

5.3 A hand with reflexes

We contend that every person might be incited to believe in a Creator by just looking at his or her left hand and that this type of prompts produced by the instinct are based on very simple calculations, say, those that contrast the anatomical organization of muscles in do-undo pairs with a possible random organization.

We also claim that the very fact that humans have hands enable them to accept that gods can have very refined capabilities of control, such as it happens in most religions. Precisely, we have acquired now the capability to generate interactive programs in which the User can modify with the mouse the behavior of the program. Thanks to the mouse, the User becomes a tiny god. In this section, we present a program that simulates a hand that can exercise certain reflex when a finger is punched by the mouse.

76 ***The code*** *The next program displays a hand that responds with a reflex to the punching of a finger by the mouse.*

```
/*Program D76 Reflex
 * A hand responds with a reflex to
 * the punching of a finger with  the mouse.
 *
 * To identify the clicked place, a mouse listener
 * is instantiated. Next, to identify which finger
 * and which phalange were punched, one calculates
```

```
 * the distance from the point of clicking
 * to each corner of face one of each phalange.
 * The four distances are summed and the minimum value
 * over all fingers and phalanges reveals the
 * punched finger and phalange.
 * This method is 80% exact.
 */

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

import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.MouseListener;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.JPopupMenu;
import javax.swing.ButtonGroup;

import java.awt.Polygon;
import java.util.Random;


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

 //The application is instantiated
 static Reflex p = new Reflex();
 //A class with important data is instantiated
  private static Data d = p.new Data();
 //The painting class is declared and instantiated
 private static Canvas c = p.new Canvas();
 //Popup menu is  declared and instantiated
```

```
static pppMenu pp = p.new pppMenu();

public static void main( String args[] )
{
 //The JFrame is constructed with a title
 JFrame f = new JFrame( "Reflex" );

 //The popUp Menu is added to the frame
 f.add( pp, BorderLayout.CENTER );
 f.add( new JLabel("Punch a finger with the mouse"),
                          BorderLayout.SOUTH );
 f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
 c.setBackground( Color.pink );
 f.setSize( 600, 500 );
 f.setVisible( true );
 //The canvas is added to f
 f.add( c, BorderLayout.CENTER );
}

//Data that serve communication
//among different classes is reunited into an object.
private class Data
{
 private static final long serialVersionUID = 1L;
 //Flag for clicked button
 private  boolean clickedLeft;
 private  boolean clickedRight;
 private int optionColor ;
 private int optionMove ;
 private double ampliFactor;
 private int ex;
 private int ey;
 private Color color;
 private  boolean first;


public Data()
{
 //Which button of the mouse has been clicked
 clickedLeft = false;
```

```
 clickedRight = false;
 //Option of the popUp menu
 optionColor = 0;
 //Option for movement: up vs down
 optionMove =0;
 //Graphic enlarging factor
 ampliFactor = 3;
 //Coordinates of clicking event
 ex = 0;
 ey = 0;
 //Initial color for background
 color = Color.pink;
 //Flag for
 first = true;

}

}

//=================================================
//=========pppMenu CLASS============================
//=================================================

//This inner class enables the use of right clicking
//to activate a pop up menu.
//Sensing of the mouse is also enabled.
public class pppMenu extends JPanel
{
 private static final long serialVersionUID = 1L;
 private JRadioButtonMenuItem ItemsColor[];
 private JRadioButtonMenuItem ItemsMove[];
 private JPopupMenu popupMenu;


public pppMenu()
{
  //**********Color section
  //Handler for color menu items
  ColorHandler colorHandler = new ColorHandler();
  //Names of used colors
```

```
String optionColor[] = {"Pink", "Black", "Blue" };
//Colors are chosen by clicking a button
ButtonGroup colorGroup = new ButtonGroup();
//Creates pop-up menu
popupMenu = new JPopupMenu();
//Buttons for selecting colors
ItemsColor = new JRadioButtonMenuItem[ 3 ];
//The menu is constructed: it has 6 buttons
//Three buttons are for color and other 3 for the move
for ( int i = 0; i < ItemsColor.length; i++ )
{
 ItemsColor[ i ] =
       new JRadioButtonMenuItem( optionColor[ i ] );
 //add item to pop-up menu
 popupMenu.add( ItemsColor[ i ] );
 //add item to button group
 colorGroup.add( ItemsColor[ i ] );
 //add handler
 ItemsColor[ i ].addActionListener( colorHandler );
}
ItemsColor[ 0 ].setSelected(true);

popupMenu.addSeparator();

// **********Move section

// Handler for color menu items
MoveHandler moveHandler = new MoveHandler();
//Names of used moves
String optionMove[] = { "Up", "Down"};
//Moves are chosen by clicking a button
ButtonGroup moveGroup = new ButtonGroup();

//Buttons for selecting moves
ItemsMove = new JRadioButtonMenuItem[ 2 ];


//The menu is constructed: it has 2 buttons
for ( int i = 0; i < ItemsMove.length; i++ )
{
```

```
    ItemsMove[ i ] =
          new JRadioButtonMenuItem( optionMove[ i ] );
    //add item to pop-up menu
    popupMenu.add( ItemsMove[ i ] );
    //add item to button group
    moveGroup.add( ItemsMove[ i ] );
    //add handler
    ItemsMove[ i ].addActionListener( moveHandler );
   }
   ItemsMove[ 0 ].setSelected(true);


   //The popUp menu is accessed by  releasing
   //of the right button.
   //Attention: the mouse listener is added to the canvas c
   c.addMouseListener
    (
      new MouseAdapter()
      {
       //Detector of right button releasing operation
       public void mouseReleased( MouseEvent event )
       {
        if  ( event.isPopupTrigger() )
         popupMenu.show(
         event.getComponent(),event.getX(),event.getY() );
        }
       }
     );
   // Detection of the mouse
   MouseHandler handlerr = new MouseHandler();
   //Sensors are added to the canvas c
   c.addMouseListener( handlerr );

  }

//If acolor  event has been detected, here is the reaction
private class ColorHandler implements ActionListener
{
 public void actionPerformed( ActionEvent event )
 {
```

```
  //Determines which option was selected
  for ( int i = 0; i < ItemsColor.length; i++ )
   {
    if ( event.getSource() == ItemsColor[ i ] )
         d.optionColor = i;
   }
  d.clickedRight = true;
  c.repaint();
 }
}

//If a move event has been detected, here is the reaction
private class MoveHandler implements ActionListener
{
 public void actionPerformed( ActionEvent event )
 {
  //Determines which option was selected
  for ( int i = 0; i < ItemsMove.length; i++ )
   {
    if ( event.getSource() == ItemsMove[ i ] )
     d.optionMove = i;
   }
  d.clickedRight = true;
  c.repaint();
 }
}

//The mouse is detected
private class MouseHandler implements MouseListener
{
public void mouseEntered( MouseEvent event )
{
//Some invisible code goes in here.
}

public void mouseClicked( MouseEvent event )
{
 d.ex = event.getX();
 d.ey = event.getY();
 //The clicked finger and phalange are identified
```

```
 c.identify();
 c.repaint();
 d.clickedLeft = true;
 c.repaint();
}

public void mousePressed( MouseEvent event )
{
 /*System.out.format ("The mouse was pressed " +
"at [%d, %d] \n", event.getX(), event.getY() );*/
}

public void mouseReleased( MouseEvent event )
{
 /*System.out.format ("The mouse was released " +
"at [%d, %d] \n", event.getX(), event.getY() );*/
}

public void mouseExited( MouseEvent event )
{
 /*System.out.format ("The mouse exited the frame at" +
" [%d, %d] \n", event.getX(), event.getY() );*/
}

}  //end of inner class MouseHandler


} // end of class  pppMenu


 //===========================================
 //============== CLASS CANVAS ===============
 //===========================================

 /*Inner class Canvas

 We display a proto-hand which is sensible to
 punching by the mouse.

 Our protohand has 5  fingers, a palm and an arm.
```

```
Fingers have 3  phalanges except the thumb, which has 2.

Since we use finite precision, all our moves are
approximations. Therefore, a multiplication of errors
is expected if we use modifications in tandem.
That is why  every scene is built ab initio by
modification of a unique unmodifiable cube at the origin.
The basic hand is located and moves at the origin
and points downwards over the screen.

The inner class Hand is encoded beginning from line 812
The inner class Finger is presented in line 877.
A finger is composed of 3 cubes. The class Cube
appears in line 987.

*/


public class Canvas extends JPanel
{

 public static final long serialVersionUID = 1L;

/*
Our basic unit is  the unitary cube at the origin.
It is composed of 8 points or vertexes = oPoints3d[][],
12 edges = edges[][] and 6 faces = faces[][].
Points are numbered from 0 to 7.
These numbers are used in
edges, which  record the pairs of
points that are joined by a line.
A face is determined by its 4 corners.

All other cubes are transformations of this
fundamental cube. The form as this is done
appears in the class Cube.
*/


//Constructor
```

```
 /*This constructor is necessary when one have various
 nested classes. In our case, we have the class Cube,
 which is used by the class Finger,
 which is used by the class Hand,
 which is used by the class Canvas,
 which is used by the main class.
 */
  public Canvas( )
  {

  }

// The points of the fundamental cube
 private     double oPoints3d[][]=
 {
  {0,0,0},
  {1,0,0},
  {0,1,0},
  {0,0,1},
  {1,1,0},
  {1,0,1},
  {0,1,1},
  {1,1,1}
 };

 /*
// The edges of the fundamental cube
 private static int edges[][] =
    {{0,1},
     {0,2},
     {0,3},
     {1,4},
     {1,5},
     {2,6},
     {2,4},
     {3,5},
     {3,6},
     {4,7},
     {5,7},
     {6,7}};
```

```
 */

 // The faces of the fundamental cube
 private   int faces[][] =
    {{0,1,4,2},
     {0,2,6,3},
     {0,3,5,1},
     {3,5,7,6},
     {1,4,7,5},
     {2,6,7,4}};


  //Measures of the window
  private    int   xWin = 600;
  private    int   yWin = 500;
  private   int nScenes;

  //Number of cubes in the finger
  private     int nPhalanges = 3;
  private     Boolean moveArm = true;

  Random r  = new Random();

  //The constructor of hand is invoked here.
  //h and hCopy are instance of the main class Hand
  private  Hand  h =  new Hand();
  private  Hand  hCopy =  new Hand();
  //An auxiliary  instance of Finger
  private   Finger f1=   new Finger(nPhalanges);
  //Finger and phalange
  private int f, ph;
  //A constant that determines how much a phalange
  //must go up or down.
  private double deltaUp;



  // The window is enlarged.
  // This method is fired automatically
```

```
  // at the beginning
  public void init()
  {
   setSize(xWin,yWin);
  }

 //Clicked finger and phalange are identified.
 //80% exact.
 private    void  identify()
 {

  int px, py;
  int min =999999;
  for(int j = 4; j>=0;j--)
   {
if(j==4) nPhalanges = 2;
    else nPhalanges = 3;
for(int m = 0; m < nPhalanges; m++)
 {
  //A working cube cc is initialized
  Cube  cc = new Cube(m);
  //The working cube is re-set.
  cc = h.finger[j].phalange[m];

  //Projection is calculated
  cc.shade2d =  cc.round(cc.cubeProj );
  int d1 = 0;
  for(int l =0; l<4; l++)
   {
  px = 100-cc.shade2d[faces[1][l]][1];
  py =-150-cc.shade2d[ faces[1][l]][2];
  d1 = d1+ (px-d.ex) * (px-d.ex)
                  + (py-d.ey)*(py-d.ey);
   }

   if (d1<min)
    {
 min = d1;
     f = j;
     ph = m;
```

```
    }
 }
    }
  // System.out.println("Finger=" + f + " phalange=" +ph);
 }

  //face ii of the cube is drawn
  private    void drawface(Graphics g, Cube cc, int ii)
  {
   for(int l =0; l<4; l++)
    {
     cc.pointsX[l] = 100-cc.shade2d[faces[ii][l]][1];
     cc.pointsY[l] =-150-cc.shade2d[ faces[ii][l]][2];
     //System.out.println(pointsX[l] + " " + pointsY[l]);
    }
    //A polygon is defined by two vectors and an integer
    //number:
    //The first vector lists the x-coordinates of the
    //corners;
    //the second lists  the y-coordinates. The integer is
    //the number of points.
    //If the last point is different than the first,
    //the first point is closed with the last.
    Polygon face = new Polygon(cc.pointsX,
                        cc.pointsY, 4);
    g.fillPolygon(face);
  }

  //Arm is drawn
  private    void drawA(Graphics g)
  {
   //System.out.println("drawA");
   //  A working cube cc is initialized
   Cube  cc = new Cube(1);
   //The working cube is re-set.
   cc = h.arm ;

   cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
   //Projection is calculated
   cc.shade2d =  cc.round(cc.cubeProj );
```

```
 //printShade();
 //The cube is drawn
 cc.calculateColor();
 for(int i=0;i<6;i++)
{
    if (cc.coloredFace[i])
   {
    g.setColor( new Color( cc.colors[i][0],
                  cc.colors[i][1],
                  cc.colors[i][2]));
    drawface(g,cc,i);
   }
 }
//worked cube is kept to memory
h.arm = cc;
}

//Arm is drawn
private    void drawArm(Graphics g)
{
 //System.out.println("drawArm");
 //A working cube cc is initialized
 Cube  cc = new Cube(1);
 //The working cube is re-set.
 cc = h.arm ;
 //Movement begins
 if (moveArm) f1.modifyCube(cc,0);
 //worked cube is kept to memory
 h.arm = cc;
 drawA(g);
}

//Basic palm is transformed
private Cube repositPalm(Cube cc)
{
 cc.oPosition[0]  =  h.arm.cube[3][0];
 cc.oPosition[1]  =  h.arm.cube[3][1]+5*d.ampliFactor;
 cc.oPosition[2]  =   h.arm.cube[3][2];
 cc.cube = cc.translateCube(cc.cube, cc.oPosition);
 return cc;
```

```
}

 //Palm is updated
 private    Cube updatePalm(Cube cc)
 {
  cc = f1.inclination(cc,1);
  cc = f1.rotation(cc,1);
  //cc.printCube(cc.cube, " after rotation");
  cc = repositPalm(cc);
  //cc.printCube(cc.cube , " after relocation");
  for(int i = 0; i<3;i++)
  cc.vTrans[i] = cc.vTrans[i] + cc.deltaTrans[i];
  cc.cube = cc.translateCube(cc.cube,  cc.vTrans);
  //cc.printCube(cc.cube, " after translation");
  cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
  //cc.printCube(cc.cubeProj,  " after projection");
  return cc;
 }

 //Draw palm
 private    void drawP(Graphics g)
 {
  //System.out.println("drawP");
  //A working cube cc is initialized
  Cube  cc = new Cube(1);
  //The working cube is re-set.
  cc = h.palm ;
  cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
  //Projection is calculated
  cc.shade2d =  cc.round(cc.cubeProj );
  //printShade();
  //The cube is drawn
  cc.calculateColor();
  for(int i=0;i<6;i++)
 {
    if (cc.coloredFace[i])
   {
    g.setColor( new Color( cc.colors[i][0],
                   cc.colors[i][1],
                   cc.colors[i][2]));
```

```
    drawface(g,cc,i);
   }
 }
 //Memory is updated
 h.palm = cc;
 }

 //Draw palm
 private    void drawPalm(Graphics g)
 {
 //A working cube cc is initialized
 Cube  cc = new Cube(1);
 //The working cube is re-set.
 cc = h.palm ;
 //Basic palm is translated
 if (moveArm) cc = updatePalm(cc);
 h.palm = cc;
 drawP(g);
 }

 //Fingers are posited below the palm
 private    void locateFingers()
  {
   for(int i=0; i<5;i++)
 for(int j=0; j<3;j++)
  h.finger[i].phalange[0].oPosition[j]
                                    =  h.palm.cube[3][j];
   for(int i=0; i<5;i++)
 h.finger[i].phalange[0].oPosition[1]
                 = h.finger[i].phalange[0].oPosition[1]
                                    -10*i*d.ampliFactor ;
  }

 //Draw fingers
 private    void drawF (Graphics g)
 {
  //System.out.println("drawF");
  locateFingers();
  for(int j = 4; j>=0;j--)
  {
```

```
    if(j==4) nPhalanges = 2;
    else nPhalanges = 3;
    for(int m = 0; m < nPhalanges; m++)
    {
  //A working cube cc is initialized
  Cube  cc = new Cube(m);
  //The working cube is re-set.
     cc = h.finger[j].phalange[m];

     //Projection is calculated
     cc.shade2d =  cc.round(cc.cubeProj );
     //printShade();
     //The cube is drawn
     cc.calculateColor();
     for(int i=0;i<6;i++)
      {
    if (cc.coloredFace[i])
     {
      g.setColor( new Color( cc.colors[i][0],
                    cc.colors[i][1],
                    cc.colors[i][2]));
      drawface(g,cc,i);
     }
      }
     //worked cube is kept to memory
     h.finger[j].phalange[m] = cc;
    }
   }
  }

  //Draw fingers
  private    void drawFingers(Graphics g)
  {
   locateFingers();
   for(int j = 4; j>=0;j--)
   {
    if(j==4) nPhalanges = 2;
    else nPhalanges = 3;
    for(int m = 0; m < nPhalanges; m++)
    {
```

```
//A working cube cc is initialized
Cube  cc = new Cube(m);
//The working cube is re-set.
   cc = h.finger[j].phalange[m];
   //Translation and rotation of the basic arm
    if (!(moveArm))
     {
     cc.deltaPhi[m] = 0;
     cc.deltaTheta = 0;
     }
    cc = h.finger[j].modifyCube(cc,m);

   //worked cube is kept to memory
   h.finger[j].phalange[m] = cc;
  }
 }
 drawF(g);
}

//Arm, palm, finger are reposited
private void moveAPF(Graphics g)
{
drawAP(g);
 drawFingers(g);
}

//A phalange goes up
private    void  showUp(Graphics g, int j, int m)
{
 h.finger[j].phalange[m].phi[m]
             = h.finger[j].phalange[m].phi[m]-deltaUp;
moveAPF(g);
}

//A phalange goes back down
private    void  showDown(Graphics g, int j, int m)
{
 h.finger[j].phalange[m].phi[m]
             = h.finger[j].phalange[m].phi[m]+deltaUp;
 moveAPF(g);
```

```
}


//Phalange is moved
private    void  movePhalange(Graphics g)
{
 //System.out.println("movePhalange");
 moveArm = false;
 locateFingers();
 coloring(g);
 if (d.optionMove == 0) showUp(g,f,ph);
 if (d.optionMove ==1) showDown(g,f,ph);
 d.clickedLeft = false;
 hCopy = h;
}

// Forearm and palm are drawn
private    void drawAP(Graphics g)
{
 drawArm(g);
 drawPalm(g);
}

//Hand
private    void drawHand(Graphics g)
{
 drawAP(g);
 drawFingers(g);
}

//The color chosen of the menu is executed
private    void coloring(Graphics g)
{
//System.out.println("coloring");
switch (d.optionColor  )
    {
     case 0:   d.color = Color.pink; break;
     case 1:  d.color = Color.black; break;
     case 2:  d.color = Color.blue; break;
    }
```

```
c.setBackground(d.color);
g.setColor(d.color);
   g.fillRect(0,0,xWin,yWin);
   //System.out.println("color = " + d.color);
  }

  //Basic hand is transformed into the actual one
  private    void presentation(Graphics g)
  {
   //Loop for the scenes
  for(int k = 0; k<nScenes; k++)
   {
  //System.out.println("scene " + k);
  drawHand(g);
  //delay  + erase drawing
  if (k < nScenes-1)
       coloring(g);
   }//end of for nScenes
  }



 //The hand that opens the session is built
  private    void initialPosition(Graphics g)
  {
   //Repeatability is enabled
   r.setSeed(45);
   //Number of scenes of the presentation.
   nScenes = 4;
   //System.out.println("First time");
   d.first = false;
   deltaUp = 0.5;
   coloring(g);
   presentation(g);
   hCopy = h;
  }


  //When paintComponent() is called,
```

```
  //everything is repainted
  //using information in memory
  private    void refresh(Graphics g )
  {
//System.out.println("refresh");
coloring(g);
 h = hCopy;
 drawA(g);
 drawP(g);
 drawF(g);
  }

  //This method has no effect apart from
  //remedying an unknown bug (maybe in Java itself).
  //Exercise: block it out to see the result
  private void beta(Graphics g)
  {
   //System.out.println("Beta");
   c.setBackground(Color.BLACK);
   c.setBackground(Color.cyan);
   d.clickedRight = false;
  }

  //This is the main method
public void paintComponent(Graphics g)
  {
   //if this method is called for the first time
   if (d.first) initialPosition(g) ;
   else //second and following calls
   {
    //System.out.println("Repaint");

    if (d.clickedLeft) movePhalange(g);

    if (d.clickedRight)
     {
      beta(g);
     }
   }
   refresh(g);
```

```
}//end of paintComponent(G)

//=======================================================
//=============Beginning of Inner class Hand============
//=======================================================

public class Hand
{
 private Cube arm = new Cube(1);
 private Cube palm = new Cube(1);
 private Finger finger[] = new Finger[5];

 //When memory is allotted for a hand,
 //all variables must be initialized.
 //Lower classes are initialized by default,
 //but some  redefinitions could be necessary.
 Hand()
 {
  //The arm is initialized
   final Cube c1 =    new Cube(1);
   arm = c1;
      //Dimensions and location of the arm.
   arm.scale[0] = 8;
   arm.scale[1] = 30;
   arm.scale[2] = 100;
   for(int i=0; i<8;i++)
   for(int j=0; j<3;j++)
      arm.points3d[i][j] = -d.ampliFactor*arm.scale[j]
                                          *oPoints3d[i][j];


   //The  palm is initialized
   final Cube c2 =    new Cube(1);
   palm = c2;
   //Dimensions and location of the palm.
   palm.scale[0] = 8;
   palm.scale[1] = 40;
   palm.scale[2] = 60;
   for(int i=0; i<8;i++)
for(int j=0; j<3;j++)
```

```
   palm.points3d[i][j] =  -d.ampliFactor
                   * palm.scale[j]*oPoints3d[i][j];


     //Fingers are initialized
     for(int i=0; i<5;i++)
     {
     final  Finger f =   new Finger(nPhalanges);
     finger[i] = f;
     }




  }
}
//=============End of Inner class Hand============

//================================================
//=============Beginning of Inner class Finger======
//================================================

/*Variables and methods of the class Finger are
encapsulated in the same unit.
All definitions are non static, i.e., they are
used as erasable workbenches and resettable tools.

A finger is a coordinated set of 3 cubes.
The class cube is defined below.

All methods that are of exclusive use for fingers are
encapsulated here.
*/
public class Finger
{

 //Our finger is a composition of 3 cubes or phalanges
 private   Cube[] phalange    = new Cube[3] ;
```

```
   //When memory is alloted for a finger,
   //all variables must be initialized
   Finger(int nPhalanges)
   {
   for(int m =0; m < nPhalanges; m++)
     {
      final Cube cc =    new Cube(m);
      phalange[m] = cc;
     }
   }
//  The only allowed movement of a phalange is
   //inclination, forth and back, in within bounds.
   //Inclination of given phalange governs the
   //rest of the finger
   private   Cube inclination(Cube cc, int m)
   {
    //Inclination is equal to a rotation.
    //Axis of rotation: the y axis.
    cc.axisRot[0] = 0;
    cc.axisRot[1]=1;
    cc.axisRot[2]=0;
    switch(m)
     {
  case 0: cc.phi[0]  = cc.phi[0]  + cc.deltaPhi[0] ;
            break;
  case 1: cc.phi[1]  = cc.phi[0] + cc.phi[1]
                                   + cc.deltaPhi[1] ;
            break;
  case 2: cc.phi[2]  = cc.phi[1] + cc.phi[2]
                                   + cc.deltaPhi[2] ;
            break;
     }
    cc.cube = cc.rotCube(cc.points3d, cc.axisRot,
                                cc.phi[m] );
    return cc;
  }

     //For visualization, the finger is rotated
  //around the z axis.
  private   Cube rotation(Cube cc, int m)
```

```
{
 //Rotation about the z axis.
 cc.axisRot[0] = 0;
 cc.axisRot[1]=0;
 cc.axisRot[2]=1;
    cc.theta = cc.theta + cc.deltaTheta;
 cc.cube = cc.rotCube(cc.cube, cc.axisRot, cc.theta);
 return cc;
}

 //Each phalange must have its initial position that
 //depends on the position of the phalanges below it
 private   Cube rePosit(Cube cc, int m)
 {
   //System.out.println("rePosit(m)" + m);
 switch(m)
  {
   case 0:  {
      } break;
   case 1: {
cc.oPosition[0]  =   phalange[0].cube[3][0];
cc.oPosition[1]  =   phalange[0].cube[3][1];
    cc.oPosition[2]  =   phalange[0].cube[3][2];
   break;
       }
   case 2: {
cc.oPosition[0]  =   phalange[1].cube[3][0];
cc.oPosition[1]  =   phalange[1].cube[3][1];
cc.oPosition[2]  =   phalange[1].cube[3][2];
   break;
       }
  }
  cc.cube = cc.translateCube(cc.cube, cc.oPosition);
  return cc;
 }

 //The cube is inclined, rotated, translated, projected
 private    Cube modifyCube(Cube cc, int m)
 {
  cc = inclination(cc,m);
```

```
     cc = rotation(cc,m);

    //cc.printCube(cc.cube, " after rotation");
    rePosit(cc,m);
    //cc.printCube(cc.cube , " after relocation");
    for(int i = 0; i<3;i++)
    cc.vTrans[i] = cc.vTrans[i] + cc.deltaTrans[i];
    cc.cube = cc.translateCube(cc.cube,  cc.vTrans);
    //cc.printCube(cc.cube, " after translation");
    cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
    //cc.printCube(cc.cubeProj,  " after projection");
    return cc;
   }
  }//End of class Finger


  //=============End of Inner class Finger============

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

  //=============Beginning of Inner class Cube============


  public   class Cube
  {
   private double points3d[][] = new double[8][3];
   private   int shade2d[][] = new int[8][3];

//   All cubes are derived from the basic cube by rotations
//   and translations
   private double cube[][]  = new double[8][3];
   private   double  cubeProj[][] = new double[8][3];

   private double oPosition[] = new double[3];
   private   double axisRot[] = new double[3];
   private   double axisProj[]  = new double[3];
   private   double deltaTrans[]  = new double[3];
   private   double vTrans[]  = new double[3];
   private   double theta  = 0;
   private   double deltaTheta;
```

```
   private   double basisRot[][], basisProj[][]
                  = new double[3][3];
   private   double matrixRotz[][],
   matrixRot[][],
   matrixProj[][] = new double[3][3];
//   The matrix of a projection along the z-axis
   private     double J[][]=
         { {1,0,0},
           {0,1,0},
           {0,0,0}};

   private    double scale[] = new double[3];

   int colors[][] = new int[6][3];
   private   int pointsX[]  = new int[6];
   private   int pointsY[]  = new int[6];
//   How to color faces
   private    double maxX;
   private   int nExtreme;
   private   boolean maxPoints[]  = new boolean[8];
   private   boolean coloredFace[]  = new boolean[6];
   //Angle of inclination of phalange
   private   double phi[] = new double[3];;
   private   double deltaPhi[] = new double[3];;

  //Every implementation of Cube is initialized here.
  //Parameter m is used for the initial position
  //of cube number m
  Cube(int m)
  {
   //Original  form
   scale[0] = 8;
   scale[1] = 8;
   scale[2] = 30;
   for(int i=0; i<8;i++)
   for(int j=0; j<3;j++)
   points3d[i][j] =  -d.ampliFactor*scale[j]*oPoints3d[i][j];
   defineColors();
   //Original position
```

```
   for(int i=0; i<3;i++)
    oPosition[i] = 0;
   //Each phalange has its own angle of inclination
   //with respect to its support
   for(int i=0; i<3;i++)
   {
    phi[i] = 0;
    deltaPhi[i]  = -0.1*(m+1);
   }
   //  The 3d cube flies along vectTrans[]
   //Each cube has its own direction of translation
   for(int i=0; i<3;i++)
    vTrans[i] = 0;
   deltaTrans[0] =  0;
   deltaTrans[1] =  0;
   deltaTrans[2] =  0;
   //  printCube();
   //  The 3d-graphic is rotated along the axisRot
   //  an angle theta that is incremented by deltaTheta.
   //  Each cube has its own velocity and direction of
   //  rotation.
   theta = 0;
   deltaTheta = -0.1;
   axisRot[0] = 0;
   axisRot[1]=0;
   axisRot[2]=1;
   //  The 3d-graphic is projected over the screen
   //  which is perpendicular to the x-axis.
   axisProj[0] = 1;
   axisProj[1]=0;
   axisProj[2]=0;


   }


  Random r  = new Random();

// The transpose of a matrix
 private    double[][] matrixTranspose(double m[][])
```

```
 {
 double[][] helpM = new double[3][3];
 for(int i=0; i<3; i++)
 for(int j=0; j<3; j++)
 helpM[i][j] =m[j][i];
 return helpM;
 }

// The multiplication of two matrixes
 private  double[][] matrixMult(double p[][], double q[][])
 {
  double[][] helpM = new double[3][3];

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

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

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



// To find a matrix of a rotation, one must find first
// an orthogonal bases with 3 vectors
```

```
// The third vector is the axis of rotation
 private    double[] thirdVector(double[] v)
 {
  double a = v[0];
  double b = v[1];
  double c = v[2];
  double norm = Math.sqrt(a*a + b*b + c*c);
  //System.out.println("norm = " + norm);
  double[] helpV = new double[3];
  helpV[0] = a/norm;
  helpV[1]= b/norm;
  helpV[2] = c/norm;
  /*System.out.println("Third vector " + helpV[0]+
           helpV[1]+helpV[2]);*/
  return helpV;
 }

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

```
  double norm = Math.sqrt(x*x + y*y + z*z);

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

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


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

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

```
 }


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

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

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

```
 /*A 3d cube exists only in the imagination:
 * it is not seen.
 * One sees a shade or projection of it over the screen.
 * The shade is defined by 2d points. The edges of
 * the 2d shade are inherited from the edges of the
 * 3d cube.
 * The 3d cube is projected into a 2d plane.
 * The resultant figure must be rendered to the screen.
 * A projection is done along  an axis that crosses
 * through 0.
 * Our projections are observed from infinite:
 * we use no perspective corrections. */
 private     double[][] mProj(double w[] )
 {
  basisProj = basisB(w);
  double[][]  helpM = new double[3][3];
  helpM = matrixMult(J,matrixTranspose(basisProj));
  helpM = matrixMult(basisProj,helpM);
  //printMatrix(helpM, "Proj");
  //Test: Proj*Proj must be equal to Proj
  /*helpM = matrixMult(helpM,helpM);
  printMatrix(helpM, "Proj* Proj");*/
  return helpM;
 }

// The cube c is rotated around axis v an angle theta
 private double[][] rotCube(double c[][],
                            double v[], double theta)
 {
  double[]  helpV = new double[3];
  double cc[][] = new double[8][3];
  matrixRot = mRotation( v,theta);
  //printMatrix(matrixRot, "matrixRot");
  //The 8 points of the cube must be rotated
  for(int i=0; i<8; i++)
  {
   for(int j=0; j<3; j++)
   helpV[j] = c[i][j];
   helpV = matrixVector(matrixRot, helpV);
   for(int j=0; j<3; j++)
```

```
   cc[i][j] = helpV[j];
  }
  //printCube(cc, " after rot");
  return cc;
 }



// The cube c is translated   deltaV
 private double[][] translateCube(double c[][],
                                          double deltaV[])
 {
  double cc[][] = new double[8][3];
  for(int i=0; i<8; i++)
  for(int j=0; j<3; j++)
  cc[i][j] = c[i][j] + deltaV[j];
  return cc;
 }


// The cube c is projected along axis w
 private double[][] projCube(double c[][],  double w[])
 {
  double[]  helpV = new double[3];
  double cc[][] = new double[8][3];
  matrixProj = mProj(w);
  for(int i=0; i<8; i++)
   {
    for(int j=0; j<3; j++)
    helpV[j] = c[i][j];
    helpV = matrixVector(matrixProj, helpV);
    for(int j=0; j<3; j++)
    cc[i][j] = helpV[j];
   }
  return cc;
 }

// The rendering procedure accepts integer values
 private     int[][] round(double c[][])
 {
```

```
 int[][] helpM = new int [8][3];
 for(int i=0;i<8;i++)
 for(int j=0; j<3; j++)
 helpM[i][j] = (int) Math.round(c[i][j]);
 return helpM;
}

private   void printCube(double c[][], String a)
{
 System.out.println("This is the cube" +a);
 for(int i=0;i<8;i++)
 System.out.println(c[i][0] + " " + c[i][1]+ " "
                                     + c[i][2]);
}

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

private void printShade()
{
 System.out.println("This is the shade");
 for(int i=0;i<8;i++)
 System.out.println(shade2d[i][1] + " "
                                  + shade2d[i][2]);
}

// Colors to paint the faces are defined
 private int[][] defineColors()
 {
  for(int l = 0; l<6;l++)
   {
    //We use random colors.
    //A color can be defined by three numbers,
    //each one from 0 to 255.
    //We has 6 faces, so we need
```

```
    //a matrix to keep colors.

    int c1,c2,c3;
    c1 = r.nextInt(226);
    c2 = r.nextInt(226);
    c3 = r.nextInt(226);
    //Colors are memorized
    colors[l][0]= c1;
    colors[l][1]= c2;
    colors[l][2]= c3;
   }
  return colors;
 }

// what faces must be colored
 private    void marked()
 {
  for(int f = 0; f<6; f++)
  coloredFace[f] = false;
  for(int f = 0; f<6;f++)
  {
   for(int p = 0; p<4;p++)
   {
    for(int c = 0; c<8; c++)
     {
      if (maxPoints[c] == true)
      {
        if (faces[f][p] == c )
        coloredFace[f] = true;
        //System.out.println("coloring");
      }
    }
   }
  }
 }


 // The x coordinate of all points are measured
 // and the extreme ones are recorded.
 private    void calculateColor()
```

```
{
 //The maximum value of coordinate x
 maxX =-1000;
 for(int l = 0; l<8; l++)
  {
   if (cube[l][0] > maxX)
    {
     maxX = cube[l][0];
    }
   }
 nExtreme = 0;
 //Number of extreme points
 for(int l = 0; l<8;l++)
  {
   maxPoints[l] = false;
   if (Math.abs(cube[l][0] - maxX )< 2)
   {
    nExtreme = nExtreme +1 ;
    maxPoints[l] = true;
   }
  }
  marked();
 }
 }//=============End of inner class Cube==============
}
}//end of main class
```

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

78 ***Challenge*** *Verify that the program is 80% exact in the identification of punched finger and phalange. Explain why. Would you imagine how to improve the exactness?*

79 ***Challenge*** *The program uses the method beta(g) to force Java to color the screen. This forcing is an ad hoc measure: would you find the way of having Java functioning in a natural way without that method?*

80 ***Challenge*** *Add a third possibility to the reflex: to bounce up and down. If you fail, consider the possibility to hang the execution of the program from a thread (this will be explained in a next chapter).*

81 ***Challenge*** *Our thumb is not really a thumb: it is not opposable. What can you do to help our hand?*

82 ***Challenge*** *Fix the bugs caused by perspective.*

83 ***Reflexes, will, spirits and gods***

A reflex happens when a nervous signal is processed by the spinal cord instead of going to the brain. Now, the brain is not a conglomerate of reflex centers. Instead, the brain is a the central processor of inputs and the organ of decision making. This is what science says about the brain.

On the other hand, anthropology witnesses another type of belief about the entity that thinks: it is the soul or the spirit, which leaves the body when the person dies. Moreover, the human body can react to the actions of foreign spiritual entities apart form the soul of the person. So, Greeks used the work demon to indicate a supernatural being that was able of controlling a person in the same way as an actor can control the marionettes in the theater. All these facts were organized into a religious system that varying from region to region has some few invariable items across the world, say:

- A human being does not think with the head but with the soul.
- Apart from souls, there are other supernatural entities that have a personality with free will and desires and can modify the material world. People can cause them to be of help if they worship them converting them into gods. As a very generalized rule, gods love bloody sacrifices, say, of animals, of warriors that die in a tournament, of defeated enemies, even of own babies. For instances, the Muiscas, indwelling a central part of Colombia, used to sacrifice a teen ager once every fifteen years. From time to time, the press of our days relates histories about human sacrifices to Satan and other gods.

Modern religions try out special solutions to the problem of human sacrifices and generalized violence:

- Christian religion invaded the whole world by preaching that the only bloody sacrifice was that of Jesus, the Son of God, dying for the sin of the world. Nevertheless, Christendom has been blamed for having filled in blood and oppression the history of the world.
- Muslims celebrate the Eid al-Adha or Feast of Sacrifice to commemorate the willingness of Ibraham (Abraham) to obey God by sacrificing his OWN son.

This fest includes a familiar ceremony in which a cow or ram is sacrificed. The family eats about a third of the meal and donates the rest to the poor. On the other side, Muslims are famous by their propensity to war: if they do not battle you, they will fight one another. Nevertheless some of them are distinguished by their kindness and holiness even if they wear a sword or a knife. If you have enough wisdom to understand them, peace will abound.

- And, what about Buddism? From his childhood, Buddha was saddened by every type of oppression. Hence he declared: "Ahimsa Paramodharma" that means that non-hurting is the Supreme Dharma, i.e. the contact with the eternal essence happens if one recognizes that everything is vanity and that to cause pain to others is more vain than everything else. Anyway, the followers of Buddha are not better than the rest of mortals: keeping in mind that the people of Sri Lanka are among the more gentle of the world, let us notice that the recent civil war in that country between the mostly Buddhist Sinhalese majority and the Hindu Tamil minority has cost some 50,000 lives.

Which is the lesson we must learn?

It seems that mankind uses modern religions not to become purer, holier, more human, closer to God but to achieve earthy goals, revenge, security, richness, power and glory. If we elaborate this point further, we might ask: if the man uses religion for his own purposes, for augmenting his surviving and vainglory, so, what importance do gods have? They look more like parasites than as something to be worth of respect. We conclude that atheism looks theologically well founded: if gods exist, they deserve no honor, no glory. This is the form as religious people handicap others to follow their religious.

5.4 Almost human gods

We have explored the magic of mouse actions, that allows us to give orders with a click. All one needs to do is to compose a correct program that enables the actions that one wants to appear as by magic.

The very same idea appears in some religious in which gods are in the image of man: a god can reign only because he had made a previous very hard work. So, we read in the Popul Vuh, a sacred book of the Mayan literature, that gods make some tries using diverse materials to make the man before they found that the best material is maize.

By the same token, we find in Judaism that G-d created the heavens and the earth and the things that are in them using the formula:

”And G-d said, Let there be (this and that).. .., and it was so. And G-d made (this and that)”.

If we read this text in the way of fairy tales, this formula looks like magic. But the Bible is not a book about fairy tales. It is a sacred book and produces its own interpretation, which is based in the fact that G-d made the man in his image and after his likeness. This demands from us to interpret creation in the very way as an artisan could interpret it. To begin with, let us notice now that the formula for creation has two parts. The first part is related with the properties of the *word*: ”And G-d said, Let there be (this and that).. .., and it was so.” The word conveys precise information about the forms as things must be done. And precise information must be expressed with precision to distinguish right form left, up from down, light form darkness. The second part of the formula deals with implementation of the plans described by the word: ”And G-d made (this and that)”.

This looks as mechanistic as the very description of our Java programs: they are in first place composed as verbal expressions and then executed. G-d composed the program of the creation, which was next executed by G-d himself.

The identification of the Word of G-d with precise instructions to make thinks acquires its maximum justification in the light of DNA. It happens that this molecule conveys verbal instructions to make proteins and enzymes thanks to which life can be continued. These verbal instructions must be very precise and so they must precisely specified. But, what is a set of verbal instructions? It is program. If these instructions are for a computer, we have computer programs. More accurately, the ribosome functions as a computer processor while the DNA includes computer programs in a literal sense. While science argues that these instructions were specified and are updated by evolution, a literal interpretation of the genesis demands the direct participation of G-d in the original DNA specification.

This insight is not restricted to biology. We have today various models that consider physics as based on words and diverse types of computing machines. This is *digital physics* (the first paragraph of the introduction to the theme in Wikipedia can be understood by everybody).

Christian religion retakes the Judaic vision of the Word but goes as far as to teach that the Word and G-d are one and the same thing and that He is with us. This is presented by Apostle John that opens his Gospel with the next words:

At the beginning was the Word, and the Word was with God, and God was the Word.. .. All things were made by him; and without him was not any thing made that was made.. .. And the Word was made flesh, and dwelt among us, and we beheld his glory, the glory as of the only begotten of the Father, full of grace and truth.

We see again the same Judaic theme of the artisan vision of the creation with

the addendum that the Word is now identified with the Son of God, Jesus, the one that can communicate verbally with God and that has the ability of executing his specific instructions.

So, both Judaism and the Christian religion are strongly creationist: it was God that personally and with specific actions originated life and species. Does it mean that these religions are anti-evolutionary? We will discuss this question a bit later.

5.5 Conclusion

We have proposed that there are very simple and powerful mechanisms that the mind uses to invent creationism and so everyone must be prepared to share his or her existence with creationist people. We also have explored the magic of mouse clicking of modern computers and this work has allowed us to correctly understand the creationist claims of Judaism and Christian religion in which God is human-like and so His creative acts were dressed in hard work in the very same way as we can program magic acts in a computer if only we can make a program that execute for us the magic we are looking for. We have saw that Java is a neutral tool that can be used in favor of non scientific ideologies, such as religions. In the next chapters we will see how Java behave with scientific proposals such as the evolutionary theory.

Chapter 6

Rationalizing the evolutionary theory

`Evolution is a crude reality.`

84 ***Introduction*** *Evolution is nowadays part of our everyday life. In fact, we all around the world experience a permanent threaten from diverse illnesses caused by bacteria or viruses. The reason is that pathogens permanently change their genetic material, they mutate, they evolve, and the diverse variants exhibit different degree of danger and no one knows what will be next, if a tolerable challenge or a pandemic disease. So, evolution, rooted on mutation, is a crude reality that no one can deny.*

85 ***Purpose*** *While we all know that the genetic material mutates, evolves, it is also a reality that no one has ever witnessed a single case in which mutation caused a change of species: if an influenza virus mutates to a viable variant, the result is still a virus of influenza. By contrast, modern evolutionary theory says that mutation + surviving of the fittest across the wide world and during million of years constitute the explanation to the diversity of living beings and of species as well as to their complexity, perfection and adaptation to the environment. Evolutionism is a philosophical doctrine that converts the evolutionary theory in a dogma that serves as contextual background to interpret data and as light to illuminate further research. Does somebody has a reason to think like that in spite of lacking direct experimental evidence? Yes, of course. To begin with, we introduce the theme with a discussion about what evolution is today and then we share two programs that illustrate how science pretends to use evolution to explain in a very simple way morphological variability.*

6.1 A vision over evolution

We offer here a vision over evolution as it is seen today from the stand point of a social scientist. The vision of evolution as reported for the lay public such as it is seen by official authorities can be found at (Ayala et al, 2008). Strikingly enough, those authorities present as a virtue of evolution its witnessed possibility of strengthening the faith in God. In our EvolJava community, we welcome the joy to be a human being with the possibility of living feelings, intuitions and irrational **convictions**. Express your **convictions** succinctly, if that pleases you. But be serious: let the force of your convictions be translated into wisdom and endurance: wisdom to formulate new insights leading to fruitful research programs and endurance to accomplish begun work.

86 ***Evolution today***

It is enough to look at the entrails of a hen, a fish and a sheep to get convinced that all these species are variations on the same theme. *Can we provide a natural mechanism to explain those variations beginning from just some few origins or from one or maybe from no origin at all?* The election of modern science is to provide a full fledged mechanism to explain variations among living beings beginning from inanimate matter which in its turn is explained by the Big Bang, i.e. by a fluctuation of the quantum vacuum.

Let us call the proposed natural mechanism **evolution** and let us work it out as a null hypotheses of science.

As a first constitutive ingredient of our project, we have **mutation** or change in the genome from parents to offspring. In particular, we enjoy a direct estimation of the per nucleotide mutation rate in the fly, *Drosophila melanogaster*. The corresponding estimated mean value is 8.4×10^{-9} per generation (Haag-Liautard et al., 2007).

Let us highlight the fact that actual life is a sophisticated compromise between evolution and anti-evolution, say, we cannot stop mutation (it hangs ultimately from quantum mechanics) and there are genes that warm/cold the mutation rate (Radman, 2000) but nevertheless the genetic code of alive beings appears to be finely tuned to minimize the effects of mistranslation and point mutation (Freeland et al, 1998). So, the genetic code has very few variants. On top of all this, mutation does not happen by randomness alone, instead evolution is promoted by a special purpose machinery that is so amazing that it can execute even topological operations and in consequence it is very complex; actually, it is composed of various tens of enzymes that work in ensemble (Darnell et al, 1986). In general, one cannot avoid to get invaded by perplexity because of the very high level of the complexity of life.

Once mutation is recognized, the next natural proposal is that *mutations along lineages add together to give rise to the origin of races, new species and phyla*. We cannot deny that this could happen spontaneously, but *random changes probably will tend to annihilate one another. So, we need a mechanism to generate a directional tendency*. The glory for unveiling and document from nature such a mechanism belongs to **Darwin** (1859), even though he knew nothing about genes: if a change in the genetic information produces a variation that fits better a challenge of the environment, that genetic change will propagate even more than the other variants. A trend toward complexity is automatically produced because any environment is a very hard challenge for any species. By the same reason, diverse environments would produce over time genetic diversity. The connection of the changes of the environment with those in the genome or in the frequencies of their representatives is called **selection**.

Spontaneous changes plus selected traits would eventually cause a reproductive separation and/or morphological and ethological divergence. But from the fact that the domestic fly is the same all over the planet, one deduces that our proposals are not to be believed at once and that they need to be experimentally documented. Reality has shown that this problem is not that easy: *after 160 years of hard work we must get content with incipient proofs of the belief that selection correlates with speciation* (Dettman et al, 2007). The problem is that an incipient proof is no proof at all but due to the difficulty of the task, incipient proofs are considered as hits of science. Nevertheless, once and for ever, let us make it clear that science does not rely on proofs. Science is a project that relies on **conviction** + hard and wise work.

So, *what does fuel the **conviction** of a modern scientist in that the microscopic mutation + time are enough to explain the origin of species, phyla and life in general?*

The first and fundamental reason is that a human heart is a fabric of **convictions**. The second is that there are various kinds of incipient proofs of speciation and **Darwin** himself provided some of them. Just think of the finches of the Galapagos Islands, whose variability of picks reveal an abundant adaptive radiation, now studied at the gene level (Patel, 2006).

A most loved evidence in favor of evolution relies on the **genome project**, which is the systematic study of all forms of genomes. According to results that enjoy a certain tradition and are so well established, the evolutionary trees of **systematics** correlates with evolutionary trees of molecular biology and so many genes, such as cytochrome, can be traced very clearly from mammals down to the bacterium. Additionally, related genomes differ more by the form of genetic regulation than by genes themselves, say, the 94% of the human genes already appears in the genome of the chimpanzee (The Chimpanzee Sequencing and Analysis Consortium, 2005). Furthermore we know for certain that microbes absorb and discard

genes according to their needs in response to a change in the environment (Goldenfeld and Woeso, 2007). This implies that mutation does not only means changes from parents to offspring but additionally includes the live changes in the genome due to insertions of new genes taking from other organisms.

We have now a *combinatorial theory of evolution*: the genes are permanently flushed as cards and a handful corresponds to the genome of a species.

Evolution also has the support of computer simulations that show that speciation maybe an unavoidable event along the normal reproductive dynamics of a population that live in a competitive complex environment. A celebrated example is given by *terra* (Ray, 1990).

In general, science has been continually offering advances in regard with the justification of evolution. In particular, *evolution is today a tool to solve many diverse types of problems in science, technology and art*: the idea is to design an evolutionary environment in which approximate solutions to a given problem are incarnated into individuals of the evolving population and to reward with more reproduction those individuals that represent better approximations. As evolution goes on, the quality solutions is improved and evolution is allowed to run until solutions achieve the desired satisfaction.

Thus, science is dominated by **evolutionism**, which is the philosophical certainty in the power of evolution (plus its natural extensions that cover all materialism) to explain everything in spite of the fact that at present there are vacuums and paradoxes. In evolutionism, evolution does not belong in the realm of science because it cannot be the object of doubt or rejection: it is a dogma.

Now, *is there a reason to question the power of evolution to explain our very existence?*

Yes, there are many. Let us consider for instance the next point: the human genome contains more than 100 microbial genes, whose expression is made in the brain, and that do not appear in the genome of the chimp (Whitfield, 2007). It is straightforward to propose that these genes were transposed directly from bacteria as well as it is also straightforward to ask: *who apart from God could have done the very improbable right selection for the appropriate functional combination*? It is here that we recognize the power of evolutionism as a philosophical regulative principle: facts with very low probability are declared to have occurred by mere chance and no god must be annoyed.

Anyway, evolutionism must battle for a honorable place in the mark of ideologies, where we also find the religious traditions that indwelt all cultures of the world: according to some of these traditions, a human being is much more than organized matter because his or her body is a recipient of a spirit, whose reality overcomes physical death. Or more explicitly, *one can find people all over the world, very especially children under five, claiming that there are ghosts at their*

houses. On the other hand, *we have the tremendous impact of the enormous volume of information that any genome consists in*. We join these two items, **spirits + complexity**, and we have God the Creator. We just have reinvented **creationism**, which claims that life was designed and synthesized by a spiritual entity.

Among creationists, Christian fundamentalists are especial because they have done their best to get hatred by scientists. On one hand, they have shot their mortal guns: they have tried to manipulate the legal power at Courts to achieve that the book of Genesis could be taught as a scientific alternative to evolution. They try to do so in spite of the fact that they know that their academic works (Dembski, 1998) have awaken fear but no applause (Melott, 2002). On the other hand, they have administrated no wine or oil to bandage the wounded bodies of their enemies, who indeed have made a very instructive defense (Padian, 2005).

Some scientists have paid special attention to the creationist movement and continuously deliver literature about the position of science. One of them is Richard Dawkins, whose work has world-wide impact. Why? It is because he is a master in dealing with those questions that worry every human mind: where are we from and where do we go? The answer given to these questions by modern science in the light of evolution are very simple: *we appeared as a byproduct of evolution and we cannot go further than natural law allows us to go*.

It happens that a clear explanation of the ideas of evolution has not been enough to illuminate the hearts of people, and some scientists have opted for a direct attack against religion. As a part of this reaction, Dawkins publishes *The God delusion* (2006), in which he claims that traditional forms or religion are irrational and harm prone while a completely atheist science is the true religion full in epics and splendor. Now, if we look around, we must recognize that we live in the most glorious moment that science has ever seen. Just think of the possibility to predict the function of an enzyme from the corresponding DNA code (Hermann et al., 2007). This is definitively a significant lap in the direction of an ab initio synthetic dragonfly (Ball, 1995) - a dream that is proper of the science of our time. But troubles begin when we notice that amidst so many splendors, we find that cancer and aging seem to play with us much as a cat plays with the little mouse it just killed. The tremendous advances of science to combat these killers (Finkel et al, 2007) are very far from being satisfactory, a fact that one is not aware of until one sees own father or daughter dying in the middle of the most severe tortures because of them.

But modern biology has prepared since long ago an answer to the suffering of mankind: *we are nothing apart from transient instruments of the egoism of genes that prompts them to survive at whatever cost*. The election of very complex organisms, such as we are, is just one of their strategies to achieve their goal: surviving. So, our death has no meaning apart from allowing the existence of other forms of gene combinations with slightly mutated genomes. Henceforth, it is not

dead that must be feared; instead it is life that must be appreciated and enjoyed.

This form of thought was popularized by Dawkins in his book *The selfish gene* (1976), an introduction to sociobiology, whose purpose is to trace down to genes the biological causes of our conduct even of those acts that seem to be the most altruistic.

To summarize our vision, we find in evolution the dominant model of materialism: *nothing in modern science makes sense if not in the light of evolution* (mutatis mutandis, the slogan of Dobshanskyi). So, evolution together with its extensions to cover the origin of life and of the universe, defines a world view with answers to all questions, including those that concern the nature and fate of man. For this reason, evolution certainly has the power to compete with any religion. And, as it happens with all religions, not all things are clear, in particular, biology is filled in events of low probability. Science uses to explain them by claiming that they happened by mere chance, although other scientists are trying new insights (Kauffman, 1995) that nevertheless are not completely satisfactory. This explains why evolution must have an active propaganda (Editor of nature, 2008) and must be defended with philosophical digressions. By contrast, electromagnetism (Rodriguez, 2008) needs neither propaganda whatsoever nor lawsuits at the court. *This contrast shows two things: first that a water-proof all encompassing theory of evolution is as yet to be made. Second: to compose and assess relevant ideas are exceedingly hard and challenging task.* In response, **Java for the study of evolution is proposed as a very effective weapon to face those challenges**.

Nevertheless, our experience along three volumes allows us to notice that *to program evolution correctly and to grasp the important points to be discussed* are two items that together are more important than Java itself, because one can program evolution as bad as desired and one can read facts in many irrelevant ways. So, Java is no more than a tool. And tools are nothing without skilled and wise artisans and architects. So, everyone get prepared for a very long and hard run, where courage could be as important as wisdom and skills.

6.2 Evolution is science

We carefully distinguish between the theory of evolution and evolutionism. The theory of evolution is part of science while evolutionism is a school of philosophy that is ascribed to materialism. The evolutionary theory as part of science can be tested and corroborated else eventually rejected. By contrast, evolutionism teaches that each rejection of the evolutionary theory is surely based on a short sight and shallow reasoning: just work a bit more and everything will be clear and the glory

of evolution will shine as the nascent sun. Thus, evolutionism is strong because it is a fuel for scientific work and it is so important that it can be likened to materialism: it seems that the rejection by science of the evolutionary theory is equivalent to rejecting materialism, something that cannot be. Additionally, a huge expanding mass of scientific results is softly accommodated along the evolutionary perspective.

It is very important to understand how these aforementioned heroic ideas are born to light. In this regard, we find very appropriate to follow a well known work that has been applauded everywhere.

87 ***The challenge of Dawkins***

Richard Dawkins published in 1986 his celebrated book *The blind watchmaker*. This book enjoys an eternal actuality because of three reasons. First: it studies the very questions that modern biology and great religions have in common, say, Is the extreme complexity and beauty of alive beings a sign of design by God? Second: far from dismissing the question, this book allows everyone to look at it with enhanced admiration and respect for the marvel of life. Third: the reader is offended with concrete and simple mechanisms to kill and explain the diversity and complexity of life forms.

That book is precisely a challenge for us because of its mechanistic approach. Thus, we have on one hand that Java is a master of mechanistic thought and, on the other, that science is a fabric in which mechanisms are deeply studied and rejected or refined. That is why the challenge found in the book of Dawkins represents the type of challenges that we must learn to formulate and discuss.

At this moment in time, the very good idea is simply to present a proposed mechanisms found in *the blind watchmaker*. So, we present a program that teaches how evolution can be used to generate morphological diversity. We show a second program in which we mix evolution with linguistics with the aim of showing that evolution has since long ago surpassed the barriers of biology and that it is now present in all relevant spheres of our modern world: our program uses evolution to generate visual art.

6.3 The morphs of Dawkins

We present now our code. It may be used to show that simple combinatorics plus mutation has a lot to say about the origin of biomorphology and, eventually, of biodiversity. The program includes selection, which is exercised by the User.

88 ***The code of biomorphs***

Our code is a modification of that elaborated in Java by Alain Gogniat (1997) and a different version than Program C5, biomorphs, of our vol III.

This program represents for us a treasure of the inheritance left by previous generations. It is a very good and simple idea that seems to be observed in nature and contending that the genomes have two sources of variability: first, they conform diverse samples taken from a pool of genes and second, equal samples of genes can be regulated by different procedures.

The code follows:

```
/* Program D88 Morphs
 * This is a pattern generator.
 * Patterns resemble insects,
 * humans, ships, trees, towers, etc.
 * Pattern space is navigated
 * by two methods:  random assembling
 * and evolution.
 *
 * The application makes use of three main parts:
 * 1) MenuMorphs (line 110),
 *     which is  a control body  that manages the manu bar,
 * 2) The inner class Biomorphs (line 818), which is
 *    the pattern generator,
 * 3) The main method of Morphs (line 89) which
 *    is the general architect.
 *
 * The class menuMorphs is instantiated as m, line 92,
 * The class Biomorphs is instantiated as b, line 82.
 * The communication among m and b, and other
 * diverse classes that also appear, follows
 * the next formalism:
 * if the variable 'style' of b needs to be assigned
 * the value 3, we write
 * style = 3, if we are in b,
 * b.style = 3, if we are outside the scope of b.
 * And so on.
 *
```

```
 * This program is a modification to another one
 * made by Alain Gogniat (1997), who granted to the User
 * a GNU license. By inheritance,  the present version
 * also grants to you the same type of license.
 */


//Tools to be used
import java.io.File;
import java.io.IOException;
import java.util.Random;
import java.util.Vector;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.event.ActionEvent;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.ActionListener;
import java.awt.geom.Rectangle2D;
import java.awt.image.BufferedImage;
import java.awt.image.RenderedImage;

import javax.imageio.ImageIO;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JColorChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
```

```
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JTextField;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.ButtonGroup;

//********** MAIN CLASS MORPHS**********

public class Morphs
{
 //The application is instantiated
 static Morphs p = new Morphs();

 //Biomorphs is instantiated
 static biomorphs b = p.new biomorphs();

 //The JFrame which shows everything
 // is constructed with a title
 static JFrame f = new JFrame("Morphological diversity");

 //Main of Morphs: this is the general architect
 public static void main( String args[] )
  {
  //The menu bar is activated
   final MenuMorphs m = p.new MenuMorphs();
   m.addNotify();//useless but avoids a warning
   //biomorphs is added to the JFrame f
   f.add( b, BorderLayout.CENTER );
   //An announcement in the footer.
   f.add( new JLabel
   ("What is this? Evolutionism or creationism?"),
                         BorderLayout.SOUTH );
   f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
   //Dimenions of the JFrame
   f.setSize( 700, 730 );
   f.setVisible( true );
  }

//**********INNER CLASS MenuMorphs*************
//Controls the menubar
```

```
//MenuBar and fundamental controls
public class MenuMorphs  extends JFrame
{
  private static final long serialVersionUID = 1L;

  //The User can sign
  public String yourName;

  //Two colors
  public  Color colorF =Color.BLUE;
  public  Color colorG =Color.CYAN;

  //Menu items
  private JRadioButtonMenuItem compItems[];
  private JRadioButtonMenuItem fonts[];
  private JRadioButtonMenuItem show[];
  private JRadioButtonMenuItem trans[];
  private JRadioButtonMenuItem mut[];

  //font style menu items
  private JCheckBoxMenuItem styleItems[];


  //Declaration of buttonGroups
  private ButtonGroup fontButtonGroup;
  private ButtonGroup compButtonGroup;
  private ButtonGroup showButtonGroup;
  private ButtonGroup transButtonGroup;
  private ButtonGroup mutButtonGroup;

  //Diverse options for the mutation menu
  private String smut[] = {"Low", "Medium", "High","Hot"};
  //Diverse options for the transcription  menu
  private  String strans[] = {"Random", "Forced"};

  //This is the constructor of the main class
  public  MenuMorphs()
   {
//Title
```

```
super("Programming morphological diversity ");
//Background color
getContentPane().setBackground( colorF );



//*:::::::::::::::::::::::::::::::::::::::::
//The first column of  the menu is worked out

//Creates file menu
JMenu fileMenu = new JMenu( "File" );
//File menu is enabled  with Alt+ F
fileMenu.setMnemonic( 'F' );

//The first menu item is added to the first column
JMenuItem aboutItem = new JMenuItem( "Purpose..." );
//If FILE IS enabled, Purpose is open with Alt + A
aboutItem.setMnemonic( 'A' );
//Adds aboutItem to first column of the menu
fileMenu.add( aboutItem );
aboutItem.addActionListener
 (
  //Structure to answer to the user
  new ActionListener()
  {
 //Displays message dialog
 public void actionPerformed( ActionEvent event )
 {
  JOptionPane.showMessageDialog
  (
  MenuMorphs.this,
  "This is a pattern generator whose mechanism \n" +
  "imitates the evolutionary process. \n" +
  "The next generation is produced by mutation \n" +
  "+ optional recombination of the genomes of \n"+
  "the present one + spontaneous generation.\n" +
  "Natural selection is simulated by the user  \n"+
  "that selects ONE genome to be cloned, \n" +
  "else TWO  individuals,  whose genomes \n" +
  "must be recombined to \n" +
```

```
   "produce the new generation.\n"+
   "There is also a purifying selection to keep \n"+
   "genomes of the same length (8 letters).\n"+
   "It is possible to set a new beginning\n" +
   "by clicking on the NEW TRY button."+
   "\n \n" ,
   "Purpose", JOptionPane.PLAIN_MESSAGE
   );
  }
   }
  );

 // The second menu item is added to the first column
 JMenuItem paramItem = new JMenuItem( "Show parameters" );
 paramItem.setMnemonic( 'w' );
 fileMenu.add( paramItem );
 paramItem.addActionListener
 (
  new ActionListener()
  {
   // Displays message dialog
   public void actionPerformed( ActionEvent event )
   {
JOptionPane.showMessageDialog
(
 MenuMorphs.this,
 " Mutation = " + smut[b.MaxSeverity] +
 "\nTranscription =  " + strans[b.TypeTranscription]+
 "\nComplexity = " + b.complexity+
 "\n \n" ,
 "Show parameters", JOptionPane.PLAIN_MESSAGE
 );
   }
  }
 );

  //The third menu item is added to the first column
  JMenuItem lessonsItem = new JMenuItem( "Lessons" );
  lessonsItem.setMnemonic( 's' );
  fileMenu.add( lessonsItem );
```

```
  lessonsItem.addActionListener
  (
new ActionListener()
 {
  // Displays message dialog
  public void actionPerformed( ActionEvent event )
   {
  JOptionPane.showMessageDialog
  (
   MenuMorphs.this,
   "The great lesson is this: " +
   "\nThe  diversity of genomes might be" +
   "\nexplained by two sources of variability:" +
   "\nfirst, they conform  diverse samples   " +
   "\ntaken from a pool of genes  and second, " +
   "\nequal samples of genes can be regulated " +
   "\nby different procedures. ",
   "Lessons", JOptionPane.PLAIN_MESSAGE
  );
   }
 }
   );

  //The fourth menu item is added to the first column
  JMenuItem authorsItem = new JMenuItem( "Authors" );
  authorsItem.setMnemonic( 'a' );
  fileMenu.add( authorsItem );
  authorsItem.addActionListener
  (
   new ActionListener()
   {
 // Displays message dialog
 public void actionPerformed( ActionEvent event )
 {
  JOptionPane.showMessageDialog
  (
   MenuMorphs.this,
   "The original idea was   of Richard Dawkins   \n" +
   "in his famous book The 'Blind Watchmaker' (1986).\n"+
   "A Java implementation was made by Alain Gogniat" +
```

```
   " in 1997  \n" +
   "(http://www.cs.bgu.ac.il/~sipper/biomorphs/evolution.html) " +
   " \n"  +
   "that was   updated here  by " +
   "Jose del Carmen Rodriguez, 2007, 2009." +
   " \n\n" ,
   "Authors", JOptionPane.PLAIN_MESSAGE
  );
 }
   }
  );


  //The fifth menu item is added to the first column
  JMenuItem licenseItem = new JMenuItem( "License" );
  licenseItem.setMnemonic( 'a' );
  fileMenu.add( licenseItem );
  licenseItem.addActionListener
  (
   new ActionListener()
   {
       // Displays message dialog
    public void actionPerformed( ActionEvent event )
    {
     JOptionPane.showMessageDialog
     (
      MenuMorphs.this,
      "Dear User:" +
      "\nEnjoy a  GNU License. " +
      "\n\n" ,
      "License", JOptionPane.PLAIN_MESSAGE
      );
     }
    }
  );

 //The Last menu item is added to the first column
 JMenuItem exitItem = new JMenuItem( "Exit" );
 exitItem.setMnemonic( 'x' );
 fileMenu.add( exitItem );
```

```
 exitItem.addActionListener
  (
new ActionListener()
  {
   // terminates application
   public void actionPerformed( ActionEvent event )
    {
    System.exit( 0 );
    }
  }
  );

 /*************************************
  * A new  column is added to the menu
  *************************************
  */


 JMenu genMenu = new JMenu( "Genetics" );
 genMenu.setMnemonic( 'D' );

 //mutHandler responds to the user when
 //he or she prompts the mutation menu.
 MutHandler mutHandler = new MutHandler();

 //Mutation control structure
 String MutNames[]
             = { "Low", "Medium", "High", "Hot" };
 JMenu mutMenu = new JMenu( "Mutation" );
 mutMenu.setMnemonic( 'M' );

 //Creates radio button menu items for mutation
 mut = new JRadioButtonMenuItem[ MutNames.length ];
 mutButtonGroup = new ButtonGroup();

 //Creates mut radio button menu items
 for ( int count = 0; count < mut.length; count++ )
 {
  mut[ count ]
  = new JRadioButtonMenuItem( MutNames[ count ] );
```

```
 mutMenu.add( mut[ count ] );
 mutButtonGroup.add( mut[ count ] );
 mut[ count ].addActionListener( mutHandler );
}
//Initial presentation
mut[ b.MaxSeverity ].setSelected( true );

//Mutation menu is added to genMenu
genMenu.add(mutMenu );

//Transcription management
TransHandler transHandler = new TransHandler();

//Transcription control structure
String TransNames[]
            = { "Random", "Forced" };
JMenu transMenu = new JMenu( "Transcription" );
transMenu.setMnemonic( 'n' );

//Creates radio button menu items for transcription
trans = new JRadioButtonMenuItem[ TransNames.length ];
transButtonGroup = new ButtonGroup();

//Creates transcription radio button menu items
for ( int count = 0; count < trans.length; count++ )
{
 trans[ count ]
 = new JRadioButtonMenuItem( TransNames[ count ] );
 transMenu.add( trans[ count ] );
 transButtonGroup.add( trans[ count ] );
 //Reaction to the user is enabled
 trans[ count ].addActionListener( transHandler );
}
//Initial presentation is controlled from b
trans[ b.TypeTranscription ].setSelected( true );
genMenu.add(transMenu );



/***********************************
```

```
  * A new  column is added to the menu
  ************************************/

 JMenu compMenu = new JMenu( "Complexity" );
 compMenu.setMnemonic( 'C' );

 // array listing complexity levels
 String values[]
            = {"3", "4", "5", "6", "7", "8" };

 // creates radio button menu items for complexity
 compItems
  = new JRadioButtonMenuItem[ values.length ];
 compButtonGroup = new ButtonGroup();


 //Instantiation of an inner class to listen
 //at the action over radiobuttons.
 CompHandler compHandler = new CompHandler();

 //Creates complexity radio button menu items
 for(int count = 0; count < values.length; count++ )
  {
compItems[ count ] =
   new JRadioButtonMenuItem( values[ count ] );
compMenu.add( compItems[ count ] );
compButtonGroup.add( compItems[ count ] );

//Reaction to the User
compItems[ count ]
                  .addActionListener( compHandler );
  }
 //Initialization
 compItems[ 0 ].setSelected( true );
 compItems[ 4 ].setSelected(true);
 compMenu.add( compMenu );


 /*****************************************
  * A new  column will be added to the menu,
```

```
 ****************************************
 */


JMenu artMenu = new JMenu( "Art" );
artMenu.setMnemonic( 't' );

//Colors are defined epistatically:
//The Frame color is the surrounding color.
//Background color is responsible of
//the color of the rects in which morphs appear.
//Morph color is the color of the morphs.
JMenu color = new JMenu( "Color" );
color.setMnemonic( 'O' );

final JMenuItem colorf
        =  new JMenuItem( "Frame" );

color.add( colorf );
getContentPane().setBackground( colorF);
colorf.addActionListener
 (
  new ActionListener()
  {
public void actionPerformed( ActionEvent event )
{
 Color c=colorF;

final Container contentPane = getContentPane();
c= JColorChooser.showDialog
    (
      colorf, getName(), c
    );
contentPane.setBackground(c);

b.frameColor=c;
//The screen must be updated every time
//one of its parameters is changed.
b.repaint();
```

```
     }
    }
   );

 final JMenuItem colorb
    =  new JMenuItem( "Background" );

 color.add( colorb );
 getContentPane().setBackground( colorF);
 colorb.addActionListener
 (
  new ActionListener()
  {
   public void actionPerformed( ActionEvent event )
   {
Color c=Color.BLUE;
final Container contentPane = getContentPane();
    c= JColorChooser.showDialog
     (
      colorf, getName(), c
     );
contentPane.setBackground(c);
    colorF=c;
b.backGround = c;
b.repaint();
   }
  }
 );

 final JMenuItem colorm
 =  new JMenuItem( "Morphs" );

 color.add( colorm );
 getContentPane().setBackground( colorF);
 colorm.addActionListener
  (
   new ActionListener()
   {
public void actionPerformed( ActionEvent event )
{
```

```
 Color c=Color.BLUE;
 final Container contentPane = getContentPane();
 c= JColorChooser.showDialog
  (
   colorf, getName(), c
  );
 contentPane.setBackground(c);
 colorF=c;
 b.foreGround = c;
 b.repaint();
}
   }
  );

 //artMenu is added to the artMenu
 artMenu.add( color );

 //     Adds separator
 artMenu.addSeparator();

 //Instantiation of an inner class to listen
 //at the action over radiobuttons.
 FontNameHandler FontNameHandler = new FontNameHandler();

 //FONT control structure
 String fontNames[]
   = { "Serif", "MonoSpaced", "SansSerif" };
 JMenu fontMenu = new JMenu( "Font" );
 fontMenu.setMnemonic( 'n' );

 //Creates radio button menu items for font names
 fonts = new JRadioButtonMenuItem[ fontNames.length ];
 fontButtonGroup = new ButtonGroup();

 //Creates Font radio button menu items
 for ( int count = 0; count < fonts.length; count++ )
 {
  fonts[ count ]
 = new JRadioButtonMenuItem( fontNames[ count ] );
  fontMenu.add( fonts[ count ] );
```

```
 fontButtonGroup.add( fonts[ count ] );
 //Reaction to the User
 fonts[ count ].addActionListener( FontNameHandler );
}

fonts[ 0 ].setSelected( true );
fontMenu.addSeparator();

//Names of styles
String styleNames[] = {  "Bold", "Italic" };
styleItems = new JCheckBoxMenuItem[ styleNames.length ];

//Instantiation of an inner class to listen
//at actions over style checkboxes.
TypeFHandler typeFHandler = new TypeFHandler();

//Creates style checkbox menu items.
//Menus are called and activated separately.
//Radio Buttons work as disjunctive ensemble:
//the activation of one deactivates the others.
//Checkboxes can be activated and deactivated
//separately.
for(int count = 0; count < styleNames.length; count++ )
{
 styleItems[ count ] =
  new JCheckBoxMenuItem( styleNames[ count ] );
 fontMenu.add( styleItems[ count ] );
 //
 styleItems[ count ].addItemListener( typeFHandler );
}
//adds Font menu to Art menu
artMenu.add( fontMenu );

//Adds separator
artMenu.addSeparator();

//********
//ShowGenome control structure
String ShowNames[]  = { "Yes", "No" };
JMenu showMenu = new JMenu( "Show Genome " );
```

```
showMenu.setMnemonic( ’n’ );

// Creates radio button menu items for show options
show  = new JRadioButtonMenuItem[ ShowNames.length ];
showButtonGroup = new ButtonGroup();

ShowHandler showHandler =  new ShowHandler();
//Creates show radio button menu items
for ( int count = 0; count < show.length; count++ )
{
 show[ count ]
= new JRadioButtonMenuItem( ShowNames[ count ] );
 showMenu.add( show[ count ] );
 showButtonGroup.add( show[ count ] );
 //Calling of inner class showHandler
 show[ count ].addActionListener( showHandler );
}

show[ 0 ].setSelected( true );

artMenu.add(showMenu);



//adds showMenu to Art menu
artMenu.add( showMenu );

//Adds separator
artMenu.addSeparator();

//New JMenuItem
JMenuItem nameAuthor = new JMenuItem( "Your name" );
nameAuthor.setMnemonic( ’e’ );
artMenu.add( nameAuthor );


nameAuthor.addActionListener
(
 new ActionListener()
 {
```

```
public void actionPerformed( ActionEvent event )
 {
  //communication with biomorphs b
  b.myName = JOptionPane.showInputDialog
                            ("Write your name here");
  if (b.myName == null) b.myName = "JR COLOMBIA";
  b.repaint();
 }
  }
 );

 //Adds separator
 artMenu.addSeparator();

 JMenuItem myMenu = new JMenuItem( "My menuItem" );
 myMenu.setMnemonic( 'e' );
 artMenu.add( myMenu );

 myMenu.addActionListener
 (
  new ActionListener()
  {
   public void actionPerformed( ActionEvent event )
   {
JOptionPane.showMessageDialog
(  null, "I will develop my own menuItem here");
}
  }
 );

 //The menu bar is declared
 final  JMenuBar bar = new JMenuBar();

 //The whole menu bar is assembled

 bar.add( fileMenu );
 bar.add( genMenu );
 bar.add( compMenu );
 bar.add( artMenu );
```

```
 //The menubar is added to the frame f
 f.setJMenuBar( bar );

 } // end MenuFrame constructor

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

//Implementation of the diverse classes
//to react to the action of the User

//Inner class to handle action events
//from mutation menu
private class MutHandler implements ActionListener
{
 public void actionPerformed( ActionEvent event )
 {
  //
  for ( int i = 0; i < mut.length; i++)

   if ( mut[ i ].isSelected() )     b.MaxSeverity = i;
  repaint();
 }
}


 //Inner class to handle action events from
 //transcription menu.
 private class TransHandler implements ActionListener
 {
  public void actionPerformed( ActionEvent event )
  {
   //
   for ( int i = 0; i < trans.length; i++)
   {
    if ( trans[ i ].isSelected() )
    {
     if (i == 0)b.TypeTranscription = 0;
         else b.TypeTranscription = 1;
    }
   }
```

```
   b.dispTranscription();
   repaint();
  }
 }

 //***********************************
 //Inner class to handle action events
 //from complexity menu
 private class CompHandler implements ActionListener
 {
  public void actionPerformed( ActionEvent event )
  {
   //
   for ( int i = 0; i < compItems.length; i++)
{
 if ( compItems[ i ].isSelected() )
 {
  b.complexity = i+3;
  b.newGeneration=true;
  b.NewGenese();
  b.repaint();
 }
}
   System.out.println("ItemHandeler");
   repaint();
  }
 }

 //  ***************************************************
 // Inner class to handle
 // name of fonts
 private class FontNameHandler implements ActionListener
 {
  public void actionPerformed( ActionEvent event )
  {
   for ( int i = 0; i < fonts.length; i++ )
{
  if ( event.getSource() == fonts[ i ] )
      {
       b.FontName =  fonts[ i ].getText();
```

```
       System.out.println("FontNameHandeler");
      }
}
   repaint();
  }
 }


 // Inner class to handle the style of fonts
 private class TypeFHandler implements ItemListener
 {
  public void itemStateChanged(ItemEvent arg0)
  {
   int style = 0; // initialize style
   // check for bold selection
   if ( styleItems[ 0 ].isSelected() )
   style += Font.BOLD;

   // check for italic selection
   if ( styleItems[ 1 ].isSelected() )
           style += Font.ITALIC;

   b.style = style;
  repaint();
  }
 }

 //Inner class to handle events from the show menu
 private class ShowHandler implements ActionListener
 {
  public void actionPerformed( ActionEvent event )
  {
   //
   for ( int i = 0; i < 1; i++)
{
 // check for "Yes"
 if ( show[ 0 ].isSelected() )
 {
  //Communication with biomorphs b
  b.bDisplayTxtGene = true;
```

```
  break;
 }

 // check for "Not"
 if ( show[ 1 ].isSelected() )
 {
      // Communication with biomorphs b
  b.bDisplayTxtGene = false;
  break;
 }
    }

b.repaint();
  }
 }//end of showHandler


}//end of MenuMorphs


   /************
     *
      *************
       *///BIOMORPHS
   /***********
  *
     *************
   */

// Declaration of the inner class biomorphs
public class biomorphs extends JPanel
 {
  //Diverse measures for the frame an inner rects
  private static final long serialVersionUID = 1L;
  static final int FRAMEHEIGHT = 650;
  static final int FRAMEWIDTH = 700;
  static final int RECTHEIGHT = 260;
  static final int RECTWIDTH = 100;
  static final int ECART = 30;
  static final int BASE  = 30;
```

```
static final int TEXTHIGH = 30;
static final int OFFSET_TOP = 70;
static final int iTop  = 0;
static final int iLeft = 10;
static final int iTop1 = OFFSET_TOP;
static final int iTop2 = iTop1 + RECTHEIGHT + ECART;
static final int iTop3 = iTop1 + 2*(RECTHEIGHT + ECART);
static final int iOnecol = RECTWIDTH + ECART;
static final int iOneline = RECTHEIGHT + ECART;
static final int iLeftParent = iLeft+(4*iOnecol) + 20;

//Morphs
static final int NBMORPHS = 8;
private morph Morph[];
private zygote Zygote;
private genome Genome[];
private String w8[] = new String[9];

//Parents
private String[] sParent = { "Mother", "Father" } ;
private morph father, mother;
static final int MOTHER = 100;
static final int FATHER = 101;
private morph Parent[];
private int nGeneration;
private String sDNA0, sDNA1;

//Diverse dialogs
private JCheckBox cb[];
private JCheckBox cbParent[];
private JTextField txtGene[];
private JTextField txtParentGene[];
private int selected[];
private int nbSelected;

//Flags
private  boolean showMother = false;
private boolean showFather = false;
private boolean buildChildren = false;
private boolean done = false;
```

```
  private boolean first = true;
  private boolean newGeneration;
  private boolean bDisplayTxtGene =  true;

  //Label
  private JLabel doLabel;
  //Control button
  private JButton btnNewTry;

  //Diverse colors
  private Color frameColor;
  private Color backGround;
  private Color foreGround;
  private String saveAs;
  //Initial colors
  private int DEF_FRAME_COLOR = 10;
  private int DEF_BACK_COLOR = 12;
  private int DEF_FORE_COLOR = 0;
  //String data
  private String myName;
  private String FontName;
  private int style;

  //Initialization of parameters
  public  int NBVAR = 8;
  private int complexity = 7;
  private int MaxSeverity = 3;
  private int TypeTranscription = 1;
  private int NbCrossover = 3;
  private Random rnd;



//  ****************************************
//  **************Constructor**************
//  ****************************************
//   b is initialized
 public  biomorphs()
 {
  rnd = new Random();
```

```
  nGeneration = 0;
  Genome = new genome[NBMORPHS];
  Morph = new morph[NBMORPHS];
  selected = new int[2];
  selected[0] = 0;
  selected[1] = 0;
  nbSelected = 0;
  Parent = new morph[2];
  myName = "JR COLOMBIA";
  FontName = "Serif";
 }

//  ****************************************
//  ************* FIRST SCREEN  *************
//  ****************************************

//   paintComponent()  is the first method to be executed.
//  It presents the   screen,
//  where the user can ask for an action.
//  Clicks over the frame are sensed
//  and the corresponding answers are invoked.
//  The various control services are autonomous
//  with  sensoring  and action system  packed
//  in the corresponding classes or methods.

 //An extension of the constructor
 public void inito()
 {
  frameColor = new Color(0);
  backGround = new Color(0);
  foreGround = new Color(0);
  frameColor = fixColor(DEF_FRAME_COLOR);
  backGround = fixColor(DEF_BACK_COLOR);
  foreGround = fixColor(DEF_FORE_COLOR);
  newGeneration = true;
  Genese();
  DrawFrames();
  labeltext();
  resetCB();
  TypeTranscription = 1;
```

```
 }

 //Ab initio constructed morphs
 private void Genese()
 {
  for (int i = 0; i < NBMORPHS; i++)
{
 Genome[i] = new genome( rnd );
 Morph[i] = new morph(Genome[i], TypeTranscription,
                             NBVAR,complexity);
}
 }

// Reinitialization of  CBs = checkBoxs
 private void resetCB()
 {
  showMother = false;
  showFather = false;
  buildChildren = false;
  cbParent[0].setSelected(false);
  cbParent[0].setVisible(false);
  cbParent[1].setSelected(false);
  cbParent[1].setVisible(false);
  txtParentGene[0].setVisible(false);
  txtParentGene[1].setVisible(false);
  selected[0] = 0;
  selected[1] = 0;
  nbSelected = 0;
  for (int i = 0; i < NBMORPHS; i++)
 cb[i].setSelected(false);
  labeltext();
 }

//  **********************************
//  ****Graphic administration*********
//  **********************************

//Actions spin around the boxes below morphs' rects.
BoxVerifier boxVerifier = new BoxVerifier();
```

```
//Boxes for parents
private void boxParentLine(int line)
 {
  cbParent[line] = new JCheckBox(sParent[line]);
  add(cbParent[line]);
  cbParent[line].setBackground(frameColor);
  cbParent[line].setBounds(iLeftParent+ECART,
   2 + iTop1 + RECTHEIGHT + line * iOneline,
   70, 20);
  cbParent[line].addItemListener(boxVerifier);
  txtParentGene[line] = new JTextField();
  add(txtParentGene[line]);
  if (line==0)
{
 cbParent[line].setVisible(showMother);
 txtParentGene[line].setVisible(showMother &
                       bDisplayTxtGene);
}
  if (line==1)
   {
 cbParent[line].setVisible(showFather);
 txtParentGene[line].setVisible(showMother &
                    bDisplayTxtGene);
}

 txtParentGene[line].setBounds(iLeftParent,
    iTop1 + RECTHEIGHT
   + line * iOneline-TEXTHIGH+10,
   RECTWIDTH, TEXTHIGH-8);
}

//Box of mother
 private void fMother()
 {
  boxParentLine(0);
 }

//Box of father
 private void fFather()
 {
```

```
 boxParentLine(1);
}

//  Planning of the   screen
private void DrawFrames()
{
 setLayout(null);

 //   ************ Label
 doLabel = new JLabel("SELECT A CHECKBOX");
 add (doLabel);
 doLabel.setVisible(true);
 doLabel.setBounds(3*iLeft+4*iOnecol-20, iTop + 45,
             RECTWIDTH+60, 20);



//  ************ button NewTry
btnNewTry = new JButton("NEW TRY");
add (btnNewTry);
NewTry newTry = new NewTry ();
btnNewTry.addActionListener(newTry);
btnNewTry.setBounds(3*iLeft+4*iOnecol, iTop + 15,
                              RECTWIDTH, 20);


//   ************ textfields to display gen
txtGene = new JTextField[NBMORPHS];
txtParentGene = new JTextField[2];

//   ************ checkboxes to select parents
cb = new JCheckBox[NBMORPHS];
//   lblGene = new Label[NBMORPHS];
cbParent = new JCheckBox[2];

//   ************ Init textfields + checkboxes
int No = 0;
for (int line = 0; line < 2; line++)
 {
```

```
   for (int col = 0; col < NBMORPHS / 2; col++)
{
 cb[No] = new JCheckBox(""+(No+1));
 cb[No].setBackground(frameColor);
 add(cb[No]);
 cb[No].setBounds( iLeft+col * iOnecol + ECART,
   2 + iTop1 + RECTHEIGHT + line * iOneline,
   40, 20);
 cb[No].addItemListener(boxVerifier);
 txtGene[No] = new JTextField();
 add(txtGene[No]);
 txtGene[No].setEditable(false);
 txtGene[No].setVisible(bDisplayTxtGene);
 txtGene[No].setBounds( iLeft+col * iOnecol ,
    iTop1 + RECTHEIGHT
   + line * iOneline- TEXTHIGH+10,
   RECTWIDTH, TEXTHIGH-8);


 No++;
}
  //Build frames of mother and father
  fMother();
  fFather();
 }
}


 //If the newTry button is clicked upon,
 //something must be done:
 class NewTry implements ActionListener
 {
   public void actionPerformed(ActionEvent event)
   {
 resetCB();
 newGeneration=true;
 NewGenese();
 repaint();
   }
 }
```

```
 RadioChoices opCh = new RadioChoices();
 String label = "";

 //Reaction to the boxes of the morphs
 private class BoxVerifier implements ItemListener
  {
   boolean showRC = true;
   int k=0;

   public   void dispose()
{
 //JFrame is banned
 opCh.dispose();
}

 //Housekeeping
 private void reset()
 {
  opCh.choiceJRadioButton.setSelected(true);
  opCh.asexualJRadioButton.setSelected(false);
  opCh.sexualJRadioButton.setSelected(false);
  opCh.saveJRadioButton.setSelected(false);
 }

 //Reacting to the boxes of the non parental morphs
 public void itemStateChanged(ItemEvent event)
 {
  done = false;
  Integer j=0;

  for(int i = 0; i < NBMORPHS; i++)
  {

   if (cb[i].isSelected())
    {
   j = i+1;
   k = i;
   label = j.toString();
```

```
    }
  }

  if (cbParent[0].isSelected()) k = NBMORPHS;
  if (cbParent[1].isSelected()) k = NBMORPHS+1;

  //if a box is selected for
  if ( event.getStateChange() == 1)
   {
  //if a second box is selected
  if (b.nbSelected == 1)
{
 b.buildChildren = true;
 b.cbParent[0].setVisible(true);
 b.txtParentGene[0].setVisible(true);
 b.cbParent[1].setVisible(true);
 b.txtParentGene[1].setVisible(true);
 b.showMother = true;
 b.showFather = true;
 b.newGeneration=true;
 b.addSelected(label);
 b.NewGeneration();
 //cb[k].setSelected(false);
 b.nbSelected = 0;
 for (int i = 0; i < NBMORPHS; i++)
 cb[i].setSelected(false);
 cbParent[0].setSelected(false);
 cbParent[1].setSelected(false);
 done = true;
 labeltext();
 b.repaint();

}
//if first parent is selected
if ((b.nbSelected == 0) & !(done) )
   {
    //activate an option chooser
    opCh.setSize(270,200);
    opCh.setVisible(true);
    opCh.setEnabled(true);
```

```
    opCh.setAlwaysOnTop(true);
    b.addSelected(label);
    done = false;
   }

   }  else  //if a box is deselected
    {
     b.delSelected(label);
         opCh.dispose();
         b.nbSelected = 0;
         b.newGeneration = false;
        if (b.nbSelected == 0) showRC = true;
    }
  reset();
 }//End of  itemStateChanged(ItemEvent event)


 }//End of BoxVerifier


 //Choice: save graphic else asexual reproduction
 //else sexual reproduction
 public class RadioChoices extends JFrame
{

   private static final long serialVersionUID = 1L;
   //Title
   private JRadioButton choiceJRadioButton;
   //Save graphic to workspace
   private JRadioButton saveJRadioButton;
   //Asexual reproduction
   private JRadioButton asexualJRadioButton;
   //Sexual reproduction
   private JRadioButton sexualJRadioButton;
   private JButton cancelJButton;
   //The buttons form a disjunctive group
   private ButtonGroup radioGroup;

   //  constructor
   public RadioChoices()
```

```
  {
     super( " Save    else reproduce morph " );
     setLayout( new FlowLayout() );

     choiceJRadioButton =
      new JRadioButton( "MAKE A CHOICE", true);
     saveJRadioButton = new JRadioButton(
      "Save graphic to Eclipse Workspace", false);
     asexualJRadioButton = new JRadioButton(
      "Asexual reproduction", false );
     sexualJRadioButton = new JRadioButton(
    "Choice parther for sexual reproduction", false );
     cancelJButton = new JButton("CANCEL");
     add(choiceJRadioButton);
     add( saveJRadioButton );
     add( asexualJRadioButton );
     add( sexualJRadioButton );
     add(cancelJButton);

     radioGroup = new ButtonGroup();
     radioGroup.add( choiceJRadioButton );
     radioGroup.add( saveJRadioButton );
     radioGroup.add( sexualJRadioButton );
     radioGroup.add( asexualJRadioButton );

     // register events for JRadioButtons
     saveJRadioButton.addItemListener(
        new RadioButtonHandler(  ) );
     asexualJRadioButton.addItemListener(
        new RadioButtonHandler(  ) );
     sexualJRadioButton.addItemListener(
        new RadioButtonHandler(   ) );
     cancelJButton.addActionListener(new Closer());

  } // end RadioButtonFrame constructor


// private inner class to handle radio button events
private class RadioButtonHandler implements ItemListener
{
```

```
 // if a click over the window  is perceived
 public void itemStateChanged( ItemEvent event )
  {
   delay(500);
   if (  saveJRadioButton.isSelected() )
    {
      b.saveAs = JOptionPane.showInputDialog(
             "Save as (Example: mutant)");

      SaveClass s = new SaveClass();
      if  (saveAs != null)
      {
       if ( saveAs.length() > 0)
     {try
         {
  s.saveDrawing();
     }   catch (InterruptedException e)
        {
          e.printStackTrace();
         }
     }
      }
     //housekeeping
     for (int i = 0; i < NBMORPHS; i++)
       cb[i].setSelected(false);
    }

//When the button for asexual repro has been clicked upon
if (  asexualJRadioButton.isSelected() )
    {
     b.resetCB();
     b.showMother = true;
     b.newGeneration=true;
     b.buildChildren = true;
     b.cbParent[0].setVisible(true);
     b.txtParentGene[0].setVisible(true);
     b.txtParentGene[1].setVisible(false);
     b.addSelected(label );
     b.NewGeneration();
     b.doLabel.setText("   SELECT A CHECKBOX");
```

```
      cbParent[0].setSelected(false);
      b.repaint();
      b.nbSelected = 0;

    }

  //When the button for sexual repro has been clicked upon
  if (  sexualJRadioButton.isSelected() )
   {
    b.resetCB();
    b.showMother = true;
    b.newGeneration=false;
    b.buildChildren = false;
    b.cbParent[0].setVisible(true);
    b.txtParentGene[0].setVisible(true);
    b.txtParentGene[1].setVisible(false);
    b.addSelected(label );
    b.getMother();
    b.doLabel.setText("   SELECT A CHECKBOX");
    b.repaint();
    JOptionPane.showMessageDialog(null,
        "We have the mother, please, select father!");
    b.nbSelected = 1;
   }


  opCh.dispose();

 } // end method itemStateChanged

} // end private inner class RadioButtonHandler


//Termination of the option chooser opCh
private class Closer implements ActionListener
 {
public void actionPerformed(ActionEvent arg0)
{
  //Clean boxes and repaint
  opCh.dispose();
```

```
  for (int i = 0; i < NBMORPHS; i++)
 b.cb[i].setSelected(false);
  b.cbParent[0].setSelected(false);
  b.cbParent[1].setSelected(false);
  b.repaint();
}
  }

} // end of  class RadioChoices


 //Pause
 private    void delay(int delay)
 {
 try
     {
   Thread.sleep(delay );
     }
     catch (InterruptedException e)
     {
   e.printStackTrace();
     }
 }


 //Drawing the morphs of parents
 private void drawParent(Graphics g, int line)
 {
  //Draw frame
  g.setColor(frameColor);
  g.draw3DRect(iLeftParent,
     iTop1 + line * iOneline,
     RECTWIDTH, RECTHEIGHT, false);
  g.setColor(backGround);
  g.fillRect(iLeftParent,
   iTop1 + line * iOneline,
   RECTWIDTH,
   RECTHEIGHT);
  g.setColor(foreGround);
  //Draw morph
```

```
  Parent[line].phenotype( g,
 iLeftParent + RECTWIDTH / 2 ,
 iTop1 + RECTHEIGHT - BASE + line * iOneline,
 NBVAR,complexity);

  txtParentGene[0].setFont(new Font(FontName, style, 12));
  txtParentGene[1].setFont(new Font(FontName, style, 12));

  txtParentGene[0].setVisible(showMother&bDisplayTxtGene);
  txtParentGene[1].setVisible(showFather&bDisplayTxtGene);
 }

//Drawing the morphs of children
 private void drawChildren(Graphics g)
 {
  g.setColor(frameColor);
  g.fillRect(0,0,FRAMEWIDTH, FRAMEHEIGHT);

  //Rect for Generation label
  g.draw3DRect(iLeft,iTop+10,2*RECTWIDTH + ECART, 40,true);
  //Rect for myName label
  g.draw3DRect(iLeft+2*iOnecol, iTop + 10,
       2*RECTWIDTH + ECART, 40, true);
  Font f = getFont();
  //Labels are drawn
  g.setColor(Color.black);
  g.setFont(new Font("Helvetica", Font.BOLD, 24));
  g.drawString("GENERATION  "+nGeneration,iLeft+5,iTop+42);
  g.drawString(myName , iLeft+2*iOnecol +5,iTop+42);
  g.setFont(f);

  g.setColor(Color.WHITE);
  g.fill3DRect( 3*iLeft+4*iOnecol-30, iTop + 45,
             RECTWIDTH+60, 20,true);

  // Draw  frames of the eight children
  for (int line = 0; line < 2; line++)
  {
   for (int col = 0; col < NBMORPHS / 2; col++)
{
```

```
 g.setColor(frameColor);
 g.draw3DRect(iLeft+ col * iOnecol,
    iTop1 + line * iOneline,
    RECTWIDTH,
    RECTHEIGHT, false);
 g.setColor(backGround);
 g.fillRect( iLeft + col * iOnecol,
   iTop1 + line * iOneline,
   RECTWIDTH,
   RECTHEIGHT);
}
  }
 }



 //  The screen is painted or repainted
 public void paintComponent(Graphics g)
 {
  //The first screen
  if (first)
  {
   inito();
   first = false;
  } else  hideShowGenome();

  drawChildren(g);
  if ( b.showMother) drawParent(g,0);
  if ( b.showFather) drawParent(g,1);

  // Draw morphs and texts of offspring

 int j = 0;
 g.setColor(foreGround);
 // Draws two raws of children.
 // There are four children in each row.
 for (int line = 0;  line < 2; line++)
  {
   for (int col = 0; col < NBMORPHS / 2; col++)
{
```

```
  Morph[j].phenotype( g,
  iLeft + col * iOnecol + RECTWIDTH / 2,
  iTop1 + line * iOneline + RECTHEIGHT - BASE,
  NBVAR,complexity );
 txtGene[j].setFont(new Font(FontName, style, 12));

 if (newGeneration)
  {
  txtGene[j].setText(Morph[j].toString());
  w8[j]= txtGene[j].getText();
  }  else
      {
        txtGene[j].setText(w8[j]);
       }
 j++;
}
   }
  newGeneration = false;
  if (done) opCh.dispose();
 done = false;
 }


 //Reinitialization
 private void NewGenese()
 {
  resetCB();
  Genese();
  repaint();
 }

 //Genomes and morphs (phenotypes) of Children
 private void offspring()
 {
  for (int i = 0; i < NBMORPHS; i++)
   {
    Zygote = new zygote(Parent[0].Genome,Parent[1].Genome);
    Zygote.crossing_over(rnd, NbCrossover);
    Morph[i] = new morph(Zygote.getChild(rnd),
              TypeTranscription,NBVAR,complexity);
```

```
   Morph[i].mutate(MaxSeverity, rnd);
  }
}

//Mother is built
private void getMother()
{
 // Genome  of the mother
 switch (selected[0])
 {
  //If father was selected,
 case FATHER:mother = Parent[1];
 sDNA0=txtParentGene[1].getText();
 //System.out.println(sDNA0);
 //System.out.println(" FF NewGen");
 break;
 //if mother was selected
 case MOTHER:mother = Parent[0];
 sDNA0=txtParentGene[0].getText();
 break;
 //if a child was selected
 default: mother = Morph[selected[0] - 1];
 sDNA0=txtGene[selected[0] - 1].getText();
 //System.out.println(" GG sDNA0" +sDNA0);
 break;
}


//Morph  (phenotype) of mother

Parent[0] = new morph (mother.Genome, TypeTranscription,
        NBVAR,complexity);
if  (Parent[0].Isvalid() == false)
{
 Parent[0] = new morph (sDNA0, TypeTranscription,
         NBVAR,complexity);
 }

txtParentGene[0].setText(sDNA0);
//In asexual repro, the father is a clon of the mother
```

```
 Parent[1] = Parent[0] ;
}

 //Father is built
 private void getFather()
 {
  switch (selected[1])
  {
   case FATHER:father = Parent[1];
               sDNA1=txtParentGene[1].getText();
       break;
   case MOTHER:father = Parent[0];
       sDNA1=txtParentGene[0].getText();
       break;
   default:    father = Morph[selected[1] - 1];
       sDNA1=txtGene[selected[1] - 1].getText();
               break;
  }

  // Morph of father
  Parent[1] = new morph (father.Genome, TypeTranscription,
          NBVAR,complexity);

  if (Parent[1].Isvalid() == false)
   {
    Parent[1] = new morph (sDNA1, TypeTranscription,
                           NBVAR,complexity);
   }
  txtParentGene[1].setText(sDNA1);
 }

 //New generation: fathers and children
 private void NewGeneration()
 {
  nGeneration++;
  sDNA0 = "sDNA0 NewGen";
  sDNA1 = "tDNA1 NewGen";
  getMother();

  //  The   new father -if it is required
```

```
  if (nbSelected > 1)
{
  getFather();}
  if (buildChildren) offspring();
}



//  ***************************************
//  **************Subordinated methods************
//  ***************************************


//  Transforms numbers to colors

 private Color fixColor ( int selection)
  {
  Color c = new Color(0);
  switch (selection)
  {
   case  0: c = Color.black ; break;
   case  1: c = Color.blue ; break;
   case  2: c = Color.cyan ; break;
   case  3: c = Color.darkGray ;break;
   case  4: c = Color.gray ; break;
   case  5: c = Color.green ;break;
   case  6: c = Color.lightGray ; break;
   case  7: c = Color.magenta ; break;
   case  8: c = Color.orange ; break;
   case  9: c = Color.pink ; break;
   case 10: c = Color.red ; break;
   case 11: c = Color.white ; break;
   case 12: c = Color.yellow ; break;
   }
  return c;
  }


 //  'Show genome' vs 'hide genome'
 private void hideShowGenome() //OK
```

```
  {
   for (int i = 0; i < NBMORPHS; i++ )
 {
  txtGene[i].setVisible(bDisplayTxtGene);
 }
   if (nGeneration > 0 )
 {
   txtParentGene[0].setVisible(bDisplayTxtGene &
                               showMother);
   txtParentGene[1].setVisible(bDisplayTxtGene &
                               showFather);
  }
 }



 //Transcription order
 private void dispTranscription()
 {
  if (nGeneration != 0 )
  {
for (int i = 0; i < NBMORPHS; i++)
 {
  Morph[i].reTranscript(TypeTranscription,NBVAR,
                               complexity);
 }
Parent[0].reTranscript(TypeTranscription,NBVAR,
                                    complexity);
Parent[1].reTranscript(TypeTranscription,NBVAR,
                                    complexity);
  }
 }

 //  The type of action of the select button is decided
 private void labeltext()
   {
switch (nbSelected)
 {
  case 0: doLabel.setFont( new Font("SanSerif",
                              Font.PLAIN, 11));
```

```
          doLabel.setText("   SELECT A CHECKBOX");
          break;
  case 1: doLabel.setText("            MAKE A CHOICE");
          break;
  case 2: doLabel.setText("   SEXUAL REPRODUCTION");
          break;
 }
}


 //When a box is selected
 private void addSelected(String s)
  {

  Integer i;
  if (s == sParent[0])
  {
   i = new Integer(MOTHER);
   }
  else if (s == sParent[1])
          {
    i = new Integer(FATHER);
    }
  else
    {
     i = new Integer(s);
  }
  if (nbSelected == 0)
  {
   selected[0] = i.intValue();
   //increment the nbSelected variable by 1
   nbSelected++;
  } else if (nbSelected == 1)
  {
   selected[1] = i.intValue();
   nbSelected++;
  } else if (nbSelected > 1)
           {
     if (selected[0] == MOTHER)
       {
```

```
   cbParent[0].setSelected(false);
   }
    else if (selected[0] == FATHER)
     {
  cbParent[1].setSelected(false);
 }
     else
      {
   cb[selected[0]-1].setSelected(false);
  }
    selected[0] = selected[1];
    selected[1] = i.intValue();
 }
  labeltext();
 }


//  Double clicking corresponds to deleting
private void delSelected(String s)
 {
 Integer i;
 if (s == sParent[0])
  {
   i = new Integer(MOTHER);
  }
 else if (s == sParent[1])
        {
  i = new Integer(FATHER);
 } else
     {
        i = new Integer(s);
     }

 if (selected[0] == i.intValue())
  {
   selected[0] = selected[1];
   //decrement the nbSelected variable by 1
   nbSelected --;
  } else if (selected[1] == i.intValue())
    {
```

```
   if (i.intValue() == MOTHER)
    {
     cbParent[0].setSelected(false);
    } else if (i.intValue() == FATHER)
      {
   cbParent[0].setSelected(false);
  }
      else
      {
   cb[selected[0]].setSelected(false);
  }
   nbSelected -- ;
  }
  selected[1] = 0;
  labeltext();
  if (nbSelected == 0) opCh.dispose();
   }




//   ************************************************
//
//    CLASS morph:from genome to phenotype
//
//   ************************************************

class morph
 {
  private int nDirection = 0;
  private int nOrder;
  private int MaxY;
  public int dx[], dy[];
  private Vector<Dimension> FromPt;
  private Vector<Dimension> ToPt;
  public genome Genome;
  private  int gene[];
  private boolean computed;
  private int transcription;
```

```
  //Builds local variables.
  //Overloaded constructor
  //(various forms of definition).
  public morph(genome Gen, int Transcript,
        int NBVAR, int complexity)
  {
Genome = Gen;
gene = Genome.chromosome;
dx = new int[NBVAR];
dy = new int[NBVAR];

FromPt = new Vector<Dimension>();
ToPt = new Vector<Dimension>();
MaxY = 0;
transcription = Transcript;
computed = false;
  }

 //  builds local variables
 public morph(String S, int Transcript,int NBVAR,
        int complexity)
 {
  Genome = new genome(S);
  gene = Genome.chromosome;
  dx = new int[NBVAR];
  dy = new int[NBVAR];

  FromPt = new Vector<Dimension>();
  ToPt = new Vector<Dimension>();
  MaxY = 0;
  transcription = Transcript;
  computed = false;
  //System.out.println(S + " morph");
  }

 //Defines when a genome is valid
 public boolean Isvalid()
 {
  return Genome.Isvalid();
```

```
 }

 //Calls mutation
 public void mutate(int severity, Random r)
 {
  Genome.mutate(severity,r);
  computed = false;
 }

 //Conditions for transcription
 public void reTranscript(int Transcript,
                  int NBVAR, int complexity)
 {
  computed = false;
  transcription = Transcript;
  transcript(NBVAR,complexity);
 }

 //  Converts numbers  to string
 public String toString()
 {
  return Genome.toString();
 }

//From genes to geometry
 private void transcript(int NBVAR, int complexity)
 {
  dx[0] = -gene[1] / 4;
  dx[1] = -gene[0] / 4;
  dx[2] = gene[3] / 2;
  dx[3] = gene[0] / 2;
  dx[4] = gene[1] / 2;
  dx[5] = gene[2] / 2;
  dx[6] = -dx[2] / 2;
  dx[7] = -dx[5] / 2;

  dy[0] = (gene[5])/2;
  dy[1] = (gene[4]);
  dy[2] = (gene[3]);
  dy[3] = -(gene[4]);
```

```
  dy[4] = -(gene[5]);
  dy[5] = (gene[6]);
  dy[6] = (gene[0]);
  dy[7] = -dy[5];

  if (transcription == 1)
   {
   gene[8] = complexity;
    } else
      {
    //takes the absolute value of gene[8]:
    //If gene[8] < 0 is true, gene[8]= -gene[8]
    //else gene[8]=gene[8]
    gene[8] = (gene[8] < 0) ? - gene[8] : gene[8];
    //gene[8] = max(3,gene[8]):
       //If gene[8] < 3 is true, gene[8]= 3]
    //else gene[8]=gene[8]
    gene[8] = (gene[8] < 3) ?  3 : gene[8];
   }

   nOrder = gene[8];
   //builds a tree
   tree (0, 0, nOrder, nDirection,NBVAR);
  }


 //Draws   branches at (x,y)
 //  A morph is a tree, whose branches are bent.
 public void phenotype(Graphics g, int x, int y,
                 int NBVAR, int complexity)
 {
  if (computed == false)
   {
transcript(NBVAR,complexity);
   }
  //  Dimension = couple of numbers to denote
  //  the dimensions of a rectangle, width and height,
  //  or the coordinates of a vector in the plane.
  Dimension Pt1, Pt2;
  //  Draws two branches at (x,y)
```

```
  for (int i = 0; i < FromPt.size(); i ++)
   {
  Pt1 = (Dimension)FromPt.elementAt(i);
  Pt2 = (Dimension)ToPt.elementAt(i);
  //draw  two segments of a symmetric morph
  g.drawLine (x + Pt1.width, y - MaxY + Pt1.height,
              x + Pt2.width,
              y - MaxY + Pt2.height);
  g.drawLine (x - Pt1.width, y - MaxY + Pt1.height,
                       x  - Pt2.width,
                       y - MaxY + Pt2.height);
}
 }


 private void PushGene (int x1, int y1, int x2, int y2)
  {
   //calculates the maximum value of all y's
   if (y1 > MaxY )
{
 MaxY = y1;
} else if (y2 > MaxY)
            {
  MaxY = y2;
     }


//Adds (x1,y1) to vector FromPt
Dimension   dimension1 = new Dimension(x1,y1);
dimension1.height =x1;
dimension1.width =y1;
    FromPt.addElement(new Dimension(x1,y1));

//Adds (x2,y2) to vector toPt
dimension1.height =x2;
dimension1.width =y2;
ToPt.addElement(new Dimension(x2, y2));
  }

 //  Construction of a symmetric tree
```

```
 //  branch by branch
 private void tree (int x, int y, int order,
                    int dir,int NBVAR)
 {
  int Xnew, Ynew;
  if (dir < 0)
   {
   //increment of dir = NBVAR
   //dir = dir+NBVAR;
   dir += NBVAR;
   }
  else if (dir >= NBVAR)
{
   //decrement of dir = NBVAR
   //dir = dir-NBVAR;
   dir -= NBVAR;
}
  Xnew = x + order * dx[dir];
  Ynew = y + order * dy[dir];
  if ( Ynew > 100 ) Ynew = 100;
  PushGene(x, y, Xnew, Ynew);
  if (order > 0)
   {
    tree (Xnew, Ynew, order - 1, dir - 1,  NBVAR);
    tree (Xnew, Ynew, order - 1, dir + 1, NBVAR);
   }
  computed = true;
  }
 }

}//End of class Morph

 //   ************************************************
 //
 //  CLASS genome: fills the chromosome
 //
 //   ************************************************

 class genome
 {
```

```
  int NBGENE = 9;
  int MAXLEN = 7;
  public    int chromosome[];
  private boolean valid;


  public genome(Random r)
   {
 chromosome = new int[NBGENE];
 GetVal(r);
 valid = true;
}

  public genome(int c[])
  {
   chromosome = new int[NBGENE];
   for (int i = 0; i < NBGENE; i++)
 {
   chromosome[i] = c[i];
  }
   valid = true;
  }

 public genome(String S)
  {
Random rr = new Random();
chromosome = new int[NBGENE];

valid = S.trim().length() == NBGENE;
if (valid)
{
  char c[] = new char[NBGENE];
  c = S.toUpperCase().trim().toCharArray();
  for (int i = 0;  i < NBGENE-1 ; i++)
   {
   chromosome[i] = ((int)c[i] - 71) % MAXLEN;
   }
  chromosome[NBGENE-1] =
        ((int)c[NBGENE-1] - 71) % (MAXLEN+1);
  if (chromosome[NBGENE-1] < 3)
```

```
 chromosome[NBGENE-1] = 3;
   } else
      {
     GetVal(rr);
      }
}

public boolean Isvalid()
{
  return valid;
}

//  Mutations are done at vector chromosome
public void mutate(int maxSeverity, Random r)
 {
 if (maxSeverity > 0 )
 {
  //p is a random positive integer less than NBGENE
  int p = r.nextInt() % (NBGENE);
  p = (p < 0) ? -p : p;

  //severity is another random integer
  int severity = r.nextInt();

  //mute is an integer, whose absolute value
  // is less or equal than maxSeverity
  int mute = severity % maxSeverity + 1;
  //mutation is done
  //chromosome[p] =chromosome[p] + mute;
  chromosome[p] += mute;
  if (p<(NBGENE-1))
   {
  chromosome[p] %= (MAXLEN );
   } else
  {
   chromosome[p] %= (MAXLEN + 1);
  }
 }
}
```

```
//  vector chromosome is filled in
//  random numbers less than MAXLEN
private void GetVal(Random r)
 {
  for (int i = 0; i < NBGENE-1; i++)
 {
  chromosome[i] = r.nextInt() % MAXLEN;
 }
 chromosome[NBGENE-1] = r.nextInt() % (MAXLEN+1);
 }



public String toString()
{
 String S0 ;
 char c[];
 c = new char[NBGENE];
 S0 = new String(" ");
 for (int i=0; i < NBGENE; i++)
{
  c[i] = ((char)(chromosome[i]+71));
  S0 = S0 + c[i];
}
return S0;
}

}//End of class genome

//   ************************************************
//
//    CLASS zygote: two parents produce  a child
//
//   ************************************************

class zygote
{
 private genome male, female;

 public zygote (genome mG, genome fG)
```

```
  {
male =  new genome(mG.chromosome);
female = new genome(fG.chromosome);
  }

public genome getMale()
{
 return male;
}

 public genome getFemale()
 {
  return female;
 }

public genome getChild(Random r)
 {
  int p;
  p = r.nextInt();
  if ( (p) < 0)
{
   return female;
 } else
      {
    return male;
   }
}


 public   void crossing_over(Random r, int nCross)
{
 genome genomeMale = male;
 int  lenGenome = genomeMale.NBGENE;
 int i,   p, k;
 int temp;
 for (k = 0; k < nCross; k++)
   {
    p = r.nextInt() % lenGenome;
    p = p < 0 ? -p : p ;
    for ( i = p; i < lenGenome; i++)
```

```
   {
     temp =  male.chromosome[i];
     male.chromosome[i] = female.chromosome[i];
     female.chromosome[i] = temp;
   }
   }
 }

}//end of zygote


 //************************************************
 //
 // CLASS SaveClass: saves a morph to Eclipse Workspace
 //
 //************************************************


private class SaveClass
 {


  //A morph is saved to Eclipse Workspace
  private  void  saveDrawing() throws InterruptedException
  {
   //memoryDrawing draws a morph to memory
   RenderedImage  image = memoryDrawing();
   //The graphic in memory is saved to Workspace
   try
   {
//Save image
ImageIO.write((RenderedImage) image,
    "jpg", new File(b.saveAs + ".jpg"));
//Feedback
JOptionPane.showMessageDialog
 (
  null,
  " Graphic("+ image.getWidth() +
  " x " + image.getHeight()+ " pixels)" +
  "\n was saved  to file \" " +
```

```
  b.saveAs + ".jpg\" "+ "in   Eclipse workspace."+
  "\nOpen the file with Paint or Draw." +
  "\n \n" ,
  "saved As", JOptionPane.PLAIN_MESSAGE
 );
   } catch (IOException e)
           {
         e.printStackTrace();
           }
 }


 //The requested morph is painted to memory
 private void paintMorph(Graphics2D g)
 {
  int j =  b.boxVerifier.k;
  int line=0;
  int xo =  50; int yo = 230;

  g.setColor(b.backGround);
  g.drawRect ( xo,yo ,
  biomorphs.RECTWIDTH, biomorphs.TEXTHIGH-8);

  g.setColor(b.foreGround);
  //If the morph is a child
  if ( j<8)
   {
b.Morph[j].phenotype( g,xo,yo, b.NBVAR,b.complexity );
if (b.bDisplayTxtGene)
{
 g.setColor(Color.WHITE);
 g.fillRect(0, 240, biomorphs.RECTWIDTH,
            biomorphs.TEXTHIGH-8);
 g.setColor(Color.BLACK);
 g.drawString( b.Morph[j].toString(),10,255);
}
   }

  if ( j>7)//if the morph is a parent
   {
```

```
if (j==8) line = 0; if (j==9) line = 1;
b.Parent[line].phenotype(g,xo,yo,b.NBVAR,b.complexity);
if (b.bDisplayTxtGene)
 {
   g.setColor(Color.WHITE);
   g.fillRect(0, 240, biomorphs.RECTWIDTH,
              biomorphs.TEXTHIGH-8);
   g.setColor(Color.BLACK);
   g.drawString( b.Parent[line].toString(),10,255);
   b.cbParent[0].setSelected(false);
   b.cbParent[1].setSelected(false);
 }
   }
 }

  //Creates in memory the image to be saved
  private   RenderedImage  memoryDrawing()
             throws InterruptedException
  {
int width=biomorphs.RECTWIDTH;
int height = biomorphs.RECTHEIGHT ;
//Creates a buffered image in which to draw
BufferedImage image = new BufferedImage
    (width, height, BufferedImage.TYPE_INT_ARGB);
//Creates a graphics service for the buffered image
Graphics2D g2d = image.createGraphics();
//Background
g2d.setColor(b.backGround);
g2d.fill(new Rectangle2D.Double(0, 0, width, height));
//morph
paintMorph(g2d);
return image;
  }

 }//End of SaveClass

}//End of main class, Morphs.
```

89 ***Exercise and challenge*** *Run the code, play with it and try to understand it.*

If you find a bug, try to correct it but if you fail, remember that the correction of bugs is not an easy task and that is why the design of software is among the hardest problem of modern technology. The program can display up to 10 morphs at the same time if reproduction is enabled. Note for newcomers: to run the code follow the next instructions: 1) Open a new ***project****:* $File \rightarrow new \rightarrow Project$*. 2) A new window will pop up with name New Project. Select* ***Java Project****. 3) A new window will pop up. Give the name to your project, say, evolJavaIV. 4) In the same window, clik on* ***Configure JREs****. 5) In the new window, clik on* ***compiler****, in a tree at the left side of the window. 6) Configure the level of the JDK compliance: it must be 5 or higher. By december 2009, the higher possible level was 6. 7) In this project, open a new class with name* ***Morphs*** *and paste the code to it. Although you are legally granted with a GNU license, in reality you enjoy an unconditional license to use and misuse all our software: this specially so because we do know that any complex project is loaded with bugs, so their correction is always welcome.*

90 ***What is this? Evolutionism or creationism?***

As we saw in the first chapter, things have no meaning in themselves, instead, they use to acquire meaning from the background, from the surrounding knowledge. So, when the idea of the morphs was presented by Dawkins, everyone claimed: 'evolution!, evolution!'. Why? It was because the evolutionary contextualization provided by the complete book *the blind watchmaker* in a given social framework favored that appreciation.

Nevertheless, there are other backgrounds that convert each evolutionary result in a weapon of creationist religions. How this happens is very simple to understand:

- Evolution is a crude reality that no one can scape off. So, you must accept it and live with it.
- Evolution is nowadays a tool to solve every kind of problems in science, technology and art.
- If simple humans can use evolution for their profit, how much the Creator could have used it for his glory!

To better enjoy this discussion, let us present an example that shows how evolution might be used by artists. Other aspects of the discussion will be consider in coming chapter.

6.4 Evolution produces art

Nowadays, evolution is a tool to solve the most varied type of problems in science, industry, engineering and art, so much that one can even say that evolution is a way of life. To illustrate that trend, lets us see how evolution is used to generate **visual art**, when it is accompanied with grammar rules:

L-systems (the shorthand for Lindenmayer systems) were originally proposed by the biologist Aristid Lindenmayer in 1968 as a theory for explaining the form as the morphology of plants might be encoded in the genetic code. So, it proposes how to encode the arms of a tree in a string. Central to L-systems, is the notion of recursiveness: complex objects are built by iterative replacement of parts of a simple object using a set of rewriting rules or productions.

Our version below is offered as the smartest possible realization of these systems. It has a peculiarity: the system admits mutations and every mutation renders a viable system. You can read how this is achieved directly from the program below.

91 ***The turtle***

At the root of the encoding of L-systems resides a **turtle**, which is a pen to draw curves. The turtle has various variables and may be implemented in many diverse forms. We choose to elect the next description:

1. A point where the head of the turtle sits.

2. A boolean variable to say whether or not the pen is down (ready to write) or up (writing mode off).

3. A direction in which the turtle will move the next time. The length of movement is usually a constant.

4. A given symbol, say, F, says that the pen is down and encodes an order to write a segment in the direction in which the head of the turtle points to. A new symbol, say f, says that the pen is up and advances without writing in the direction in which the head of the turtle points to.

5. A set of instructions to change directions. Our turtle lives in a two dimensional world and has instructions to rotate the direction of movement by ± 30 degrees or by 180 degrees. The instruction to rotate 30 degrees is the plus symbol '+', the contrary rotation is '-', a giro of 180 degrees is 'g'.

Example: we can encode a square just as ”F+++F+++F+++F”.

There could be modifications to the magnitude of the segment that is written, to the rotation of the direction of movement, to the certainty as the operations are done, to the color of drawing, to the dimensions of the base space and so on. Compare (Drabek and Bräunl, 2007).

92 ***The code for our Lindenmayer system follows:***

```
/*Program D92, EvolArt

 We show that evolution can be used to generate art.

 In this opportunity we use an L-grammar,
 a system of rules to generate drawings consisting
 of linear traces over the screen.

 The program is divided in various parts.
 The main method, is the director, line 98.
 The second part, evoArtControls, line 119,
 contains the menu bar and fundamental commands.
 The third part, pppMenu, line 756,
 enables a popup menu.
 The fourth part, LGrammar, line 1052, takes care of
 the Lindenmayer system, which deals with the use
 of a graphicating turtle.
*/
import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.BorderLayout;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;
import java.awt.event.MouseAdapter;
```

```
import java.awt.event.MouseEvent;
import java.awt.geom.Rectangle2D;
import java.awt.image.BufferedImage;
import java.awt.image.RenderedImage;
import java.io.File;
import java.io.IOException;
import java.util.Random;
import javax.imageio.ImageIO;
import javax.swing.JColorChooser;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JPopupMenu;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JLabel;
import javax.swing.ButtonGroup;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JMenuBar;
import javax.swing.JSlider;
import javax.swing.SwingConstants;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;



public class EvolArt
{

 //frame characteristics
 private static Color colorF =Color.BLUE;
 private static  Color colorG =Color.CYAN;
 private static int Width = 470;
 private static int Height = 700;
 private boolean saved = false;

 //Developer
 private  static  String Name =
```

```
    "Jose Rodriguez, 2009 Colombia";

//The JFrame which shows everything
// is constructed with a title
static JFrame f = new JFrame("Evolution creates art");
static EvolArt e = new EvolArt();
//declaration of   variables with symbolic initialization
private  static  Dimension d = new Dimension(0,0);
private  static  DrawingData t =    e.new
         DrawingData("F", "F", d, 2, false, "?", "?", Name ) ;


//The menu bar is activated
static  evoArtControls m = e.new evoArtControls();
//The popUp menu
static pppMenu pp = e.new pppMenu();
//The drawing class
static LGrammar x = e.new LGrammar();
//The saved image can be seen with Paint or Draw
SaveClass s = new SaveClass();




/**********************Main of application **/


public static void main( String args[] )
{
 f.add( x, BorderLayout.CENTER );
 //An announcement in the footer.
 f.add( new JLabel
  ("Rigth clicking is enabled"),
                        BorderLayout.SOUTH );
 f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
 //Dimensions of the JFrame
 f.setSize( Width, Height );
 f.setVisible( true );
}
```

```
 /*************************************/
 /***********class evoArtControls ******/
 /*******Management of the menu bar*****/
 /************************************/


//MenuBar and fundamental controls
public class evoArtControls  extends JFrame
{


  private static final long serialVersionUID = 1L;

  //General controls
  //Menu items
  private JMenuItem demo[];
  private JRadioButtonMenuItem fonts[];
  //font style menu items
  private JCheckBoxMenuItem styleItems[];

  private JSlider complxSlider;
  private ButtonGroup fontButtonGroup;
  private  String   demoNames[]= {"Cells", "Bacteria",
          "Flower",  "Tree", "Bush",  "Planctum",
          "PlusMinus", "Together",  "Nice", "Escher",
          "Cubes", "Esthetics"};
  //Style of font
  private int style;
  private boolean showFlag = false;
  JMenuItem saveItem;
  private  String font;
  private JMenuItem optionsMenu[];

  //This is the constructor of the main class
  public       evoArtControls()
  {
   //Title
   super("Lindenmayer grammars, order,   " +
    "                   complexity and art.");
   //Background color
```

```
   getContentPane().setBackground( colorF );


   //*:::::::::::::::::::::::::::::::::::::::::
   //The first column of  the menu is worked out

   //Creates file menu
   JMenu fileMenu = new JMenu( "File" );
   //File menu is enabled  with Alt+ F
   fileMenu.setMnemonic( 'F' );


   //The first menu item is added to the first column
   saveItem = new JMenuItem( "Save graphic" );
   //If FILE IS enabled, Save is open with Alt + s
   saveItem.setMnemonic( 's' );
   saveItem.setEnabled(false);
   //Adds saveItem to first column of the menu
   fileMenu.add( saveItem );
   saveItem.addActionListener
   (
//Structure to answer to the User
new ActionListener()
{
 //Displays message dialog
 public void actionPerformed( ActionEvent event )
 {
  x.saveAs = JOptionPane.showInputDialog
  (
   "Save to local folder (or type a path) as: ",
   "Lsystem1"
  );
 if  (x.saveAs != null)
{
 if ( x.saveAs.length() > 0) x.save = true;
 saved = true;
}
 x.repaint();
 pp.feedback();
    }
```

```
    }
  );



JMenuItem exitItem = new JMenuItem( "Exit" );
exitItem.setMnemonic( 'x' );
fileMenu.add( exitItem );
exitItem.addActionListener
 (
   new ActionListener()
   {
  // terminates application
  public void actionPerformed( ActionEvent event )
   {
    System.exit( 0 );
   }
   }
  );

     /*************************************
      * A new  column is added to the menu
      *************************************
      */


     JMenu demoMenu = new JMenu( "Demo" );
     demoMenu.setMnemonic( 'D' );

     //The first menu item is added to the Demo menu
     demo= new JMenuItem[demoNames.length];

     DemoHandler demoHandler = new DemoHandler();
     for(int i = 0; i< demoNames.length; i++)
     {
      demo[i] = new JMenuItem(demoNames[i]);
      demoMenu.add( demo[i] );
      demo[i]. addActionListener(demoHandler);
     }
```

```
     /***********************************
      * A new  column is added to the menu
      ***********************************/

    JMenu newMenu = new JMenu( "New" );
    demoMenu.setMnemonic( 'D' );
    //The random menu item is added
    JMenu RandomItem = new JMenu( "Random" );
    newMenu.add( RandomItem );

    //The second menu item is added
    JMenuItem complexItem = new JMenuItem( "Complexity" );
    complexItem.setMnemonic( 'h' );
    complexItem.setEnabled(false);
    RandomItem.add( complexItem );
    complxSlider = new JSlider(SwingConstants.HORIZONTAL, 0, 50,30);
    complxSlider.setMajorTickSpacing(10);
    complxSlider.setPaintTicks( true );
    cplxSlider  cslider = new cplxSlider();
    complxSlider.addChangeListener(cslider);
    RandomItem.add(complxSlider);

    //The second menu item is added
    JMenuItem goItem = new JMenuItem( "Go" );
    complexItem.setMnemonic( 'o' );
    RandomItem.add( goItem );
    goItem.addActionListener
(
   new ActionListener()
   {
  //Fires a random composition
  public void actionPerformed( ActionEvent event )
   {
    x.newBeginning = true;
    x.repaint();
   }
   }
```

```
  );

    //A new menu item is added
    JMenuItem dItem = new JMenuItem( "Decoder" );
    dItem.setMnemonic( 'h' );
    newMenu.add( dItem );
    dItem.addActionListener
    (
     new ActionListener()
     {
      public void actionPerformed( ActionEvent event )
      {
       x.newBeginning = false;
       x.decode = true;
       t.rule = "F+++F+++F+++F";
       //t.rule = "-+--FF--Fg";
       t.rule = JOptionPane.showInputDialog(
                     "Type a string with f,F,+,-,g" ,
                     t.rule);
       x.repaint();
      }
     }
    );


    /***********************************
     * A new  column is added to the menu
     ***********************************/

   JMenu changesMenu = new JMenu( "Changes" );
   demoMenu.setMnemonic( 'D' );

// handler for menu items
   OptionHandler opHandler = new OptionHandler();
   //Options to be chosen
   String options[] = {"ITER--", "ITER++",
               "MUTATION"};


   optionsMenu = new JMenuItem[options.length];
```

```
 //The options menu   has 3 buttons
 for ( int i = 0; i < options.length; i++ )
 {
optionsMenu[ i ] =
        new JMenuItem( options[ i ] );
  //add item to changesMenu
changesMenu.add( optionsMenu[ i ] );
  //add handler listener
  optionsMenu[ i ].addActionListener( opHandler );
 }


/****************************************
 * A new  column will be added to the menu,
 ****************************************
 */

 JMenu artMenu = new JMenu( "Art" );
 artMenu.setMnemonic( 't' );

 //Creates color menu
 JMenu colorMenu = new JMenu( "Color" );
 colorMenu.setMnemonic( 'O' );
 artMenu.add( colorMenu );

 JMenuItem colorB =  new JMenuItem( "Background" );
 getContentPane().setBackground( colorF);
 ColorBack colorBack = new ColorBack();
 colorB.addActionListener(colorBack);
 colorMenu.add( colorB );

 JMenuItem  lineMenu = new JMenuItem( "Line" );
 lineMenu.setMnemonic( 'l' );
 ColorLine colorLine = new ColorLine();
 lineMenu.addActionListener(colorLine);
 colorMenu.add( lineMenu );

 JMenu comMenu = new JMenu( "Comments" );
 comMenu.setMnemonic( 'c' );
 artMenu.add( comMenu );
```

```
JMenuItem nameAuthor = new JMenuItem( "Author" );
nameAuthor.setMnemonic( 'a' );
comMenu.add( nameAuthor );
nameAuthor.addActionListener
 (
   new ActionListener()
   {
    public void actionPerformed( ActionEvent event )
    {
  t.author = JOptionPane.showInputDialog
                      ("Write your name here", Name);
  x.rep = true;
  x.repaint();
    }
   }
 );

JMenuItem titleMenu = new JMenuItem( "Title" );
titleMenu.setMnemonic( 'i' );
comMenu.add( titleMenu );
titleMenu.addActionListener
 (
   new ActionListener()
   {
    public void actionPerformed( ActionEvent event )
    {
     t.title = JOptionPane.showInputDialog
                      ("Type  title","L-System Artwork");
  x.rep = true;
  x.repaint();
    }
   }
 );


  //Adds separator
  comMenu.addSeparator();
```

```
//Instantiation of an inner class to listen
//at the action over radiobuttons.
ItemHandler itemHandler = new ItemHandler();

//FONT control structure
String fontNames[]
       = { "Serif", "MonoSpaced", "SansSerif" };
JMenu fontMenu = new JMenu( "Font" );
fontMenu.setMnemonic( 'n' );

//Creates radio button menu items for font names
fonts = new JRadioButtonMenuItem[ fontNames.length ];
fontButtonGroup = new ButtonGroup();

//Creates Font radio button menu items
for ( int count = 0; count < fonts.length; count++ )
{
   fonts[ count ]
     = new JRadioButtonMenuItem( fontNames[ count ] );
   fontMenu.add( fonts[ count ] );
   fontButtonGroup.add( fonts[ count ] );
   //Calling of inner class itemhandler
   fonts[ count ].addActionListener( itemHandler );
}

fonts[ 0 ].setSelected( true );
fontMenu.addSeparator();
//Names of styles
String styleNames[] = { "Bold", "Italic" };
styleItems = new JCheckBoxMenuItem[ styleNames.length];

//Instantiation of an inner class to listen
//at actions over checkboxes.
StyleHandler styleHandler = new StyleHandler();

//Creates style checkbox menu items
for(int count = 0; count < styleNames.length; count++ )
{
   styleItems[ count ] =
      new JCheckBoxMenuItem( styleNames[ count ] );
```

```
        fontMenu.add( styleItems[ count ] );
        //Calling of inner class
        styleItems[ count ].addItemListener( styleHandler );
     }
     //adds Font menu to comments menu
     comMenu.add( fontMenu );

     comMenu.addSeparator();

     final JMenuItem showItem = new JMenuItem( "Show" );
  showItem.setMnemonic( 'x' );
  comMenu.add( showItem );
  showItem.addActionListener
   (
     new ActionListener()
     {
  //Show-hide flip-flop
    public void actionPerformed( ActionEvent event )
     {
      String text;
      showFlag = !(showFlag);
      if (showFlag == true )
      {
       text = "Hide";
       showItem.setText(text);
         x.rep = true;
         t.show = true;
         x.repaint();

      }
      else
      {text = "show";
           showItem.setText(text);
           x.rep = true;
           t.show = false;
           x.repaint();
      }
    }
     }
    );
```

```
 /*****************************************
  * A new  column will be added to the menu,
  *****************************************
  */



 JMenu aboutMenu = new JMenu( "About" );
 aboutMenu.setMnemonic( 'a' );


 JMenuItem purpItem = new JMenuItem( "Purpose..." );
 purpItem.setMnemonic( 'A' );
 aboutMenu.add( purpItem );
 purpItem.addActionListener
  (
new ActionListener()
{
 //Displays message dialog
 public void actionPerformed( ActionEvent event )
 {
  JOptionPane.showMessageDialog
  (
   evoArtControls.this,
   "We use a turtle to implement " +
   "\nLindenmayer systems."+
   "\nA turtle is a pen that moves in a "+
   "\nlinear direction with or"+
   "\nwithout drawing."+
   "\nThe internal commands of the turtle are:"+
   "\nF of forward and draw."+
   "\nf of forward but without drawing."+
   "\n+ for a positive change of direction"+
   "\nof 30 degrees. "+
   "\n- is the contrary."+
   "\ng is a giro of 180 degrees."+
   "\nA Lindenmayer system generates a  " +
   "\ncorrespondence among  strings and " +
   "\nmovements of the turtle. "+
```

```
    "\nIn our encoding "+
    "\nevery random string is viable, " +
    "\nas well as every mutation." +
    "\nOur encoding is original.",
    "Purpose", JOptionPane.PLAIN_MESSAGE
   );
  }
 }
 );

 JMenuItem lessonsItem = new JMenuItem( "Lessons" );
 lessonsItem.setMnemonic( 's' );
 aboutMenu.add( lessonsItem );
 lessonsItem.addActionListener
 (
  new ActionListener()
  {
    // Displays message dialog
   public void actionPerformed( ActionEvent event )
   {
    JOptionPane.showMessageDialog
    (
     evoArtControls.this,
     "One great lesson is this: " +
     "\nsome apparently very disordered systems" +
     "\nare indeed very simple  because  "+
     "\nthey are supported by a complex machinery."+
     "\nOther lesson is:"+
     "\nrandomness and evolution  can produce order, "+
     "\nesthetics and art.",
     "Lessons", JOptionPane.PLAIN_MESSAGE
     );
    }
   }
     );


  JMenuItem authorsItem = new JMenuItem( "Authors" );
  authorsItem.setMnemonic( 'a' );
  aboutMenu.add( authorsItem );
```

```
 authorsItem.addActionListener
 (
  new ActionListener()
  {
public void actionPerformed( ActionEvent event )
{
 JOptionPane.showMessageDialog
 (
  evoArtControls.this,
  "L-systems were introduced and developed " +
  "\nin 1968 by the Hungarian theoretical biologist" +
       "\nAristid Lindenmayer (1925-1989)." +
  "\nMany realizations exist. The present one"+
  "\nwas developed by "+
  "\nJose del Carmen Rodriguez, 2007, 2009," +
  "\nas the most simple implementation that renders" +
  "\nevery string and mutation viable." +
  " \n\n" ,
  "Authors", JOptionPane.PLAIN_MESSAGE
 );
}
  }
 );



 JMenuItem licenseItem = new JMenuItem( "License" );
 licenseItem.setMnemonic( 'a' );
 aboutMenu.add( licenseItem );
 licenseItem.addActionListener
 (
  new ActionListener()
  {
   public void actionPerformed( ActionEvent event )
   {
    JOptionPane.showMessageDialog
    (
     evoArtControls.this,
     "Dear User:" +
     "\nEnjoy a  Colombian License: " +
```

```
          "\nyou can use  our software," +
          "\nmodify its code," +
          "\nand re-distribute it"+
          "\nin whatever form you want  " +
          "\nand with whatever purpose, even to get rich." +
          "\nNo charge at all, economical or otherwise." +
          "\nCourtesy of the EvolJava Community: " +
          "\nwww.evoljava.com" +
          "\n\n" ,
          "License", JOptionPane.PLAIN_MESSAGE
          );
        }
      }
    );


    //The whole menu bar is assembled
    JMenuBar bar = new JMenuBar();
    setJMenuBar( bar );
    bar.add( fileMenu );
    bar.add( demoMenu );
    bar.add( newMenu );
    bar.add( changesMenu );
    bar.add( artMenu );
    bar.add( aboutMenu );

//   The menubar is added to the frame f
    f.setJMenuBar( bar );

  } // end MenuFrame constructor


  //*************Listeners*****************

  private class DemoHandler implements ActionListener
  {
public void actionPerformed(ActionEvent event)
 {
  for ( int i = 0; i < demoNames.length; i++)
       if (  event.getSource() == demo[ i ] )  x.demoi = i;
```

```
  x.demoF = true;
  pp.xSlider.setValue(0);
  pp.ySlider.setValue(0);
  x.repaint();
 }
  }


  private class cplxSlider implements ChangeListener
  {
  public void stateChanged(ChangeEvent move)
   {
    x.complexity = complxSlider.getValue();
    x.complexity = x.complexity +5;
    x.repaint();
   }
  }


  private class ColorBack implements ActionListener
  {
public void actionPerformed(ActionEvent arg0)
{
 Color c=Color.BLUE;
         final Container contentPane = getContentPane();
         c= JColorChooser.showDialog
          (
           contentPane, getName(), c
          );
         if ((c != null) & (getName() != ""))
      {
       contentPane.setBackground(c);
          colorF=c;
      }
         x.rep = true;
         x.repaint();
}
  }

  private class ColorLine implements ActionListener
```

```
  {
public void actionPerformed(ActionEvent arg0)
{
 Color c=Color.CYAN;
     final Container contentPane = getContentPane();
     c= JColorChooser.showDialog
          (
           contentPane, getName(), c
          );
     if ((c != null) & (getName() != ""))
      {
       contentPane.setBackground(c);
          colorG=c;
      }
          x.rep = true;
          x.repaint();

}
  }


  private class ItemHandler implements ActionListener
  {
     public void actionPerformed( ActionEvent event )
     {
        // Font style selection
        for ( int i = 0; i < fonts.length; i++ )
        {
           if ( event.getSource() == fonts[ i ] )
             font = fonts[ i ].getText();
        }
        x.repaint();
     }
  }


  //*************************************************
  // Inner class to handle
  // item events from check box menu items
  private class StyleHandler implements ItemListener
```

```
  {
     // process font style selections
     public void itemStateChanged( ItemEvent e )
     {
        style = 0; // initialize style

        // check for bold selection
        if ( styleItems[ 0 ].isSelected() )
           style += Font.BOLD;

        // check for italic selection
        if ( styleItems[ 1 ].isSelected() )
           style += Font.ITALIC;

        x.repaint();
     }
  }


//If an event has been detected, here is the reaction
  private class OptionHandler implements ActionListener
  {
  // process menu item selections
  public void actionPerformed( ActionEvent event )
  {
   if ( event.getSource() == optionsMenu[ 0 ] )
      x.previous = true;
   if ( event.getSource() == optionsMenu[ 1 ] )
      x.next = true;
   if ( event.getSource() == optionsMenu[ 2 ] )
      x.mutation = true;
   x.repaint();
  }
 }//End of optionHandler
}//End of evoArtControl

/*************************************/
/***********class pppMenu ************/
/****Managment of the pop up menu*****/
/*************************************/
```

```
//This class enables the use of right clicking
//to activate a pop up menu.
//sensing of the mouse is also enabled.
public class pppMenu extends JFrame
{
private static final long serialVersionUID = 1L;
private JPopupMenu popupMenu;
private JMenuItem optionsMenu[];
//Slider for horizontal position
private JSlider xSlider;
//Slider for vertical position
private JSlider ySlider;




public pppMenu()
{
//handler for menu items
OptionHandler opHandler = new OptionHandler();
//Options to be chosen
String options[] = {"ITER--", "ITER++", "NEW",
             "MUTATION","SAVE"};
//creates pop-up menu
popupMenu = new JPopupMenu();

optionsMenu = new JMenuItem[options.length];
//The popup menu is constructed: it has 5 buttons
for ( int i = 0; i < options.length; i++ )
{
optionsMenu[ i ] =
       new JMenuItem( options[ i ] );
 //add item to pop-up menu
 popupMenu.add( optionsMenu[ i ] );
 //add listener
 optionsMenu[ i ].addActionListener( opHandler );
}
```

```
optionsMenu[ 4 ].setEnabled(false);
//Mouse actions are detected
//Attention: the mouse listener is added to the frame f
f.addMouseListener
 (
   new MouseAdapter()
   {
    //Detector of right button releasing operation
    public void mouseReleased( MouseEvent event )
    {
     if  ( event.isPopupTrigger() )
      popupMenu.show(
       event.getComponent(),event.getX(),event.getY());

     }
    }
  );

//X-position slider
JMenuItem xPosItem = new JMenuItem( "X-position" );
xPosItem.setMnemonic( 'x' );
xPosItem.setEnabled(false);
popupMenu.add( xPosItem );

xSlider = new JSlider(SwingConstants.HORIZONTAL,-250,250,0);
xSlider.setMajorTickSpacing(100);
xSlider.setPaintTicks( true );
XSlider x = new XSlider();
xSlider.addChangeListener(x);
xSlider.setVisible(true);
popupMenu.add(xSlider);

//Y-position slider
JMenuItem yPosItem = new JMenuItem( "Y-position" );
yPosItem.setMnemonic( 'y' );
yPosItem.setEnabled(false);
popupMenu.add( yPosItem );

ySlider = new JSlider(SwingConstants.HORIZONTAL,-250,250,0);
```

```
ySlider.setMajorTickSpacing(10);
ySlider.setPaintTicks( true );
YSlider y = new YSlider();
ySlider.addChangeListener(y);
ySlider.setVisible(true);
popupMenu.add(ySlider);

}

//If an event has been detected, here is the reaction
private class OptionHandler implements ActionListener
{
//process menu item selections
public void actionPerformed( ActionEvent event )
{
if ( event.getSource() == optionsMenu[ 0 ] )
   x.previous = true;
if ( event.getSource() == optionsMenu[ 1 ] )
   x.next = true;
if ( event.getSource() == optionsMenu[ 2 ] )
    {
   x.newBeginning = true;
   pp.xSlider.setValue(0);
   pp.ySlider.setValue(0);
    }
if ( event.getSource() == optionsMenu[ 3 ] )
   x.mutation = true;

if ( event.getSource() == optionsMenu[ 4 ] )
 {
x.saveAs = JOptionPane.showInputDialog(
    "Save to local folder (or type a path) as:",
"Lsystem1");
   if  (x.saveAs != null)
    {
     if ( x.saveAs.length() > 0) x.save = true;
     saved = true;
    }
 }
x.repaint();
```

```
feedback();
}

}//End of optionHandler


private void feedback()
{
 if (saved)
  JOptionPane.showMessageDialog
  (
null,
" Graphic was saved  to file  " +
"\n" + x.saveAs + ".jpg\" "+ "in the local folder " +
"\nor in the indicated path."+
"\nOpen the file with Paint or Draw." +
"\n \n" ,
"saved As", JOptionPane.PLAIN_MESSAGE
  );
}


//Reading instrument of X position
class XSlider implements ChangeListener
{
public void stateChanged(ChangeEvent move)
{
 x.deltaX = xSlider.getValue();
 x.rep = true;

 x.repaint();
}
}

//Reading instrument of Y position
class YSlider implements ChangeListener
{
public void stateChanged(ChangeEvent move)
{
 x.deltaY = -ySlider.getValue();
```

```
 x.rep = true;
 x.repaint();
}
}

} // end of class  pppMenu

/************************************/
/***********class DrawingData *********/
/**********Data for a drawing*********/
/************************************/


// contains the information encoding for a drawing
public  class DrawingData
 {
  //The axiom is the fundamental building block
  String axiom;
  //The rule defines the first-level construction
  String rule;
  //Root is the position of the root
  Dimension root;
  //nIter is the number of times that the
  //axiom must be replaced by the rule.
  int nIter;
  boolean show;
  String title = "Untitled";
  String gen = "No Gen";
  String author = Name;



//Initializing

  DrawingData(String a, String  rul,
      Dimension r,   int n )
{
 axiom = a;
 rule = rul;
 root = r;
```

```
 nIter = n;
}


  DrawingData(String a, String  rul,
      Dimension r,   int n, boolean b,
      String t, String g, String u )
{
 axiom = a;
 rule = rul;
 root = r;
 nIter = n;
 show = b;
 title = t;
 gen = g;
 author = u;

}


  DrawingData(DrawingData t)
{
 axiom = t.axiom;
 rule = t.rule;
 root = t.root;
 nIter = t.nIter;
 show = t.show;
 title = t.title;
 gen = t.gen;
 author = t.author;
}
}

/*************************************/
/***********class saveClass ***********/
/****Saves drawing to workspace,******/
/***local folder or specified path***/
/*********************************/
```

```
private class SaveClass extends JFrame
{

 private static final long serialVersionUID = 1L;

 RenderedImage  image;

 //Saves the image to a jpg file
 private  void  saveDrawing() throws InterruptedException
 {
  image = memoryDrawing();
  try
   {
//Save image
ImageIO.write((RenderedImage) image,
      "jpg", new File(x.saveAs + ".jpg"));
    } catch (IOException e)
       {
       e.printStackTrace();
       }
  x.rep = true;
  x.save = false;
  x.repaint();
 }

 //Creates in memory the image to be saved
 private   RenderedImage    memoryDrawing()
                          throws InterruptedException
 {
  int width=Width;
  int height = Height;
  //Creates a buffered image in which to draw
  BufferedImage image = new BufferedImage
   (width, height, BufferedImage.TYPE_INT_ARGB);
  //Creates a graphics contents on the buffered image
  Graphics2D g2d = image.createGraphics();
  //Background
  g2d.setColor( colorF);
  g2d.fill(new Rectangle2D.Double(0, 0, Width, Height));
  //Paints the image
```

```
  x.synthesier(g2d);
  //paints comments
  if (t.show)
  {
   g2d.setFont( new Font( m.font, m.style, 12 ));
   g2d.drawString(t.title, 20, 540);
   g2d.drawString("Gen = " + t.rule, 20, 560);
   g2d.drawString(t.author, 20, 580);
  }
 return image;
 }
}


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

LGrammar: creates, shows and saves  the image

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


public class LGrammar  extends JPanel
{
 private static final long serialVersionUID = 1L;

 //Communication of control section with Lgrammar

 private boolean decode = false;
 //Communication of pppMenu with Lgrammar
 private boolean demoF  = false;
 private boolean previous  = false;
 private boolean next  = false;
 private boolean mutation  = false;
 private boolean newBeginning  = false;
 private boolean save = false;

 private boolean rep   = false;

 private int demoi = -1;
 private String saveAs = "";
```

```
private int tx;
private int ty;



//To draw the image, we use a turtle,
//which is a pen that has a direction of movement.
//At the beginning, the turtle looks upwards.
double theta = Math.PI/2;
//The angle of inclination of the direction
//can be change by deltaTheta.
double  deltaTheta = Math.PI/6;
//Dimension is a point in the plane.
Dimension position = new Dimension(225,50);
Dimension movement = new Dimension(0,0);
//Each trace has a scale
double scale;
//The Lindenmayer system begins with the axiom
String  axiom ="F";
//The L systems replaces recurrently the axiom
//by  the rule.
String  rule;
String ruleNew = "";
String ruleDemo;

//The number of iterations of the recurrence:

int nIter;
int complexity;
//Displacement
int deltaX;
int deltaY;
String mRule;


//A StringBuffer is a  heavy duty String: is an object.
StringBuffer workingString = new StringBuffer("");
String code = "";
```

```
 // Turn on  the random generator

 Random r  = new Random();

 //Constructor= initialization
 public LGrammar()
 {
  complexity = 30;
  scale = 20;
  deltaX= 0;
  deltaY = 0;
  t.author = Name;
 }



private void clear(Graphics g)
{
 Graphics2D g2d= (Graphics2D)g;
 g2d.setColor( colorF);
 g2d.fill(new Rectangle2D.Double(0, 0, Width, Height));
}




 //Movements of the turtle
 //Forward moves and paints
 private  void Forward( Graphics g)
 {

  Graphics2D g2d= (Graphics2D)g;

  Dimension position2 = new Dimension(0,0);
  movement.width =
  (int) Math.round(scale * Math.cos(theta));
  movement.height =
  -(int)Math.round(scale * Math.sin(theta));
```

```
 position2.width = position.width + movement.width;
 position2.height = position.height + movement.height;
 g2d.drawLine(position.width, position.height,
      position2.width, position2.height );
 position.width = position2.width;
 position.height = position2.height;
}

//forward moves but does not paint
private  void forward( )
{

 Dimension position2 = new Dimension(0,0);
 movement.width =
 (int)Math.round(scale * Math.cos(theta));
 movement.height=
 -(int)Math.round(scale * Math.sin(theta));

 position2.width = position.width + movement.width;
 position2.height = position.height + movement.height;


 position.width = position2.width;
 position.height = position2.height;
}

//deltaTheta +
private  void rotateLeft(   )
{
 theta =theta + deltaTheta;
}

//deltaTheta -
private  void rotateRight(    )
{
 theta =theta- deltaTheta;
}

//half round rotation
private  void giro(   )
```

```
 {
  theta =theta + Math.PI;
 }


 //A stringBuffer is translated to drawing actions
 private  void decode( StringBuffer workingString,
                 Graphics g)
 {
  position.width = t.root.width;
  position.height = t.root.height;
  int n = workingString.length();
  for( int i = 0; i<n; i++)
  {
char symbol =workingString.charAt(i);

switch(symbol)
 {
case 'F':Forward( g);break;
case 'f':forward( ); break;
case '+':rotateLeft(  );break;
case '-':rotateRight( );break;
case 'g':giro( );break;
     }
  }
 }

 //The axiom is replaced by the rule.
 //The overall idea is this:
 // a rule traces a path.
 //To insert a subpath, the original position
 //must be restored. Thus, the subpath  is  completed
 //with its inverse path.
 //But moreover,  upon insertion
 //the original direction must be restored.
 //Hence, some additional bookkeeping must be done.
 private StringBuffer replaceAll
             (StringBuffer here,
                String  what,
                String byWhat)
```

```
{
 int m = what.length();
 int n = here.indexOf(what);
 int l = byWhat.length();
 while (n>=0)
 {
  here = here.replace(n,n+m,byWhat);
  n = here.indexOf(what,n+l);
 }
 return here;
}



//A string is inverted
private  String invert( String  a)
{
 String  b =  "";
 int n = a.length();
 for(int i=n-1; i>=0;i--)
b = b+a.charAt(i) ;
 return b;
}

//The backward path
private  String antiString( String s)
{
 String  a= s.replaceAll("F", "h");
 a= a.replaceAll("f", "h");
 a= a.replaceAll("\\+","m");
 a= a.replaceAll("-","p");
 a= a.replaceAll("h","f");
 a= a.replaceAll("m","-");
 a= a.replaceAll("p","+");
 a=invert(a);
 return a;
}


private  void showComments(Graphics g)
```

```
 {
  Graphics2D g2d= (Graphics2D)g;
  g2d.setColor( colorG);
  g2d.setFont( new Font( m.font, m.style, 12 ));
  g2d.drawString(t.title, 20, 540);
  g2d.drawString("Gen = " + t.rule, 20, 560);
  g2d.drawString(t.author, 20, 580);
 }


 //Axiom and rule are made into a super-rule
 private  void synthesier(  Graphics g)
 {
  clear(g);
  String bigRule = t.rule + "g"+ antiString(t.rule)+"gF";
  workingString = new StringBuffer(bigRule);
  for (int i = 0; i< t.nIter; i++)
   workingString =
          replaceAll(workingString,t.axiom,bigRule);
  Graphics2D g2d= (Graphics2D)g;
  g2d.setColor( colorG );
  decode(workingString,g);
  if (t.show) showComments( g);
 }

 //a rule is synthesized ab initio
 private String  improvization()
 {
   String  impro=  "";
     int n = r.nextInt(complexity)+5;
     String  alphabet =  "Ffg+-";

   for(int i = 1; i<=n;i++)
   {
  int j = r.nextInt(5);
  char s = alphabet.charAt(j) ;
impro = impro+s;
   }
   return impro;
 }
```

```
 private void demo(Graphics g)
 {
 switch (demoi)
  {
   case 0:  cells(g); break;
   case 1:  bacteria(g); break;
   case 2:  flower(g); break;
   case 3:  tree(g); break;
   case 4:  bush(g); break;
   case 5:  planctum(g); break;
   case 6:  plusMinus(g); break;
   case 7:  together(g);   break;
   case 8:  nice(g); break;
   case 9:  Escher(g); break;
   case 10: cubes(g); break;
   case 11: esthetics(g); break;
  }
 tx = t.root.width;
 ty = t.root.height;
 demoF = false;
 m.saveItem.setEnabled(true);
 pp.optionsMenu[ 4 ].setEnabled(true);
 }

 private void previous( Graphics g)
 {

  if (t.nIter >=1)
{
 t.nIter = t.nIter - 1;
 synthesier(g);
}
  tx = t.root.width;
  ty = t.root.height;
  previous = false;
 }
```

```
private void next(Graphics g)
{

 t.nIter = t.nIter + 1;
 tx = t.root.width;
 ty = t.root.height;
 synthesier(g);
 next= false;
}

private  void newL(Graphics g)
{
 t.axiom = "F";
 t.rule  = ""+improvization();
 t.root.width = 200;
 t.root.height = 450;
 t.nIter = 2;
 t.title = "WithoutTitle";
 tx = t.root.width;
 ty = t.root.height;
 synthesier(g);
 newBeginning= false;
 m.saveItem.setEnabled(true);
 pp.optionsMenu[ 4 ].setEnabled(true);

}

//Insertion or deletion
private void mutate(Graphics g)
{

  int n = t.rule.length();
  int i = r.nextInt(n);
  StringBuffer brule= new StringBuffer(t.rule);
    double p = r.nextDouble();
  if (p>0.5)
  {
   String  alphabet =  "Ffg+-h";
   int j = r.nextInt(6);
```

```
   char s = alphabet.charAt(j) ;
   String insertion = ""+s;
   if (insertion.equals("h")) insertion = "F+Fgf-g";
   brule = brule.insert(i, insertion);
  }
  else brule = brule.deleteCharAt(i);
  t.rule = brule.toString();
  t.nIter = 2;
  synthesier(g);
  mutation= false;
  tx = t.root.width;
  ty = t.root.height;

}


//:::::::::::::::::: Demo

private void cells(Graphics g)
{
 t.axiom =  "F";
 t.rule  = "FF-fgg-Ff++fF";
 t.root.width = 50;
 t.root.height = 450;
 t.title = "Cells";
 t.nIter = 3;
 synthesier(g);
}

private void flower(Graphics g)
{
 t.axiom =  "F";
 t.rule = "F--g+gFgf++gFgf";
 t.root.width=225;
 t.root.height = 450;
 t.title = "Flower";
 t.nIter = 3;
 synthesier(g);
}
```

```
 private void tree(Graphics g)
 {
  t.axiom =  "F";
  t.rule=
  "F++g-gFFgf----------g+gFFgf+gFgf+gFFgf-gFgf+gFFgf-gFgf";
  t.root.width=185;
  t.root.height = 450;
  t.title = "Tree";
  t.nIter = 3;
  synthesier(g);
 }

 private void bush(Graphics g)
 {
   t.axiom =  "F";
   t.rule =  "FF-FF+++-gg-+Fgfg--Fggfg-g++-f";
   t.root.width=185;
   t.root.height = 450;
   t.title = "Bush";
   t.nIter = 3;
   synthesier(g);
}


 private void bacteria(Graphics g)
 {
  t.axiom =  "F";
  t.rule=
 "F-----g+gFgf +gFgf+gFgf+gFgf+gFgf+gFgf+gFgf+gFgf+gFgf";
  t.root.width=225;
  t.root.height = 440;
  t.title = "Bacterium";
  t.nIter = 3;
  synthesier(g);
 }

 private void planctum(Graphics g)
 {
   t.axiom =  "F";
   t.rule = "F-F+F+F-gFgf+gFFgf";
```

```
  t.root.width=195;
  t.root.height = 410;
  t.title = "Planctum";
  t.nIter = 3;
  synthesier(g);
 }



private void plusMinus(Graphics g)
{
 t.axiom =  "F";
 t.rule ="F+gfgg+FgF";
 t.root.width=200;
 t.root.height = 400;
 t.title = "PlusMinus";
 t.nIter =  3;
 synthesier(g);
}


private void together( Graphics g)
{
 t.axiom =  "F";
 t.rule =  ("gf-g+gF+-+fgggf-gFf++g-F+ffggg");
 t.root.width = 190;
 t.root.height = 340;
 t.title = "Together";
 t.nIter = 3;
 synthesier(g);

}


private void nice( Graphics g)
{
 t.axiom =  "F";
 t.rule = "---FffggFf-f-f++gF-";
 t.root.width = 175;
 t.root.height = 300;
```

```
 t.title = "Nice";
 t.nIter = 3;
 synthesier(g);
}

private void Escher( Graphics g)
{
 t.axiom =  "F";
 t.rule =  "--F++g-+gF++ggfF+++";
 t.root.width=225;
 t.root.height = 440;
 t.title = "Escher";
 t.nIter = 3;
 synthesier(g);
}

private void cubes( Graphics g)
{
 t.axiom =  "F";
 t.rule =  "g+gfgf+fggFgg-ff-FFf+g+FF+fgg-";
 t.root.width=205;
 t.root.height = 250;
 t.title = "Cubes";
 t.nIter = 3;
 synthesier(g);
}

private void esthetics( Graphics g)
{
 t.axiom =  "F";
 t.rule =  "--gFffgg-+Fg---f-Ff";
 t.root.width=230;
 t.root.height = 320;
 t.title = "Esthetics";
 t.nIter  = 3;
 synthesier(g);
}

private  void decoding(Graphics g)
{
```

```
  t.axiom =  "F";
  t.root.width=230;
  t.root.height = 320;
  t.nIter  = 3;
  synthesier(g);
  tx = t.root.width;
  ty = t.root.height;
}

private  void repeat(Graphics g)
{
 t.root.width = tx + deltaX ;
 t.root.height = ty + deltaY ;
 synthesier(g);
 rep = false;
}

//Creates the image
private  void createImage(Graphics g)
                        throws InterruptedException
{
 Graphics2D g2d= (Graphics2D)g;
 //Defines the color of background:
 g2d.setColor( colorF);
 g2d.fill(new Rectangle2D.Double(0, 0, Width, Height));
 //Exposition
 if (demoF) demo(g2d);
 // one  less iteration
 if (previous == true) previous(g2d);
 //one  more iteration
 if (next == true) next(g2d);
 //rule is mutated
 if (mutation == true) mutate(g2d);
 //new rule
 if (newBeginning == true) newL(g2d);
 //A rule or gen according to the User
 if (decode == true) decoding(g);
 //Save drawing
 if (save) s.saveDrawing();
 //Housekeeping
```

```
  if (rep) repeat(g);
 }

 //Drawing manager
 public void paintComponent( Graphics g)
  {
   rep = true;
   f.setSize(Width,Height);
   try
    {
     //  paint the screen with
 // the image created by createImage(g)
 createImage(g);
    }  catch ( InterruptedException e)
        {
     e.printStackTrace();
        }

  } //end of paintComponent()

 }// End of LGrammar

}//End of evolArt
```

93 ***Exercise*** *Run the program and play with it. Hunt for bugs and try to correct them.*

94 ***Is evolution enough?***

By looking at the two previous programs, we have seen the sort of arguments exhibited by science to demonstrate that evolution is all we need to explain life in all its majesty and complexity: with very hard work and creativity, science enhances at every moment the power of evolution to answer to every specific question about the origin of life or of species. Many are the scientists and persons that think like that and that happily go around. In fact, according to materialism, matter exists and nothing else exists, so everything must be explained in terms of evolution plus chance.

But many are also the persons that consider that materialism renders a false description of what life is. Materialism might be rejected on direct experience when the person feels or sees ghosts. But nobody has ever witnessed that a ghost or a god had made a living being. So, the belief in an spiritual realm does not imply that a Creator must exist. So, why do some people believe that we exist thanks to a Creator? It is because they accept the revealed truths of their religions which include a creation. Their faith is nourished by the inner peace that they feel when they search for their gods. This is true for a Christian as for a Budist and is promised by all religions that include contemplative practices. Now, these believers do not see contradiction between a Creator and evolution, because evolution is considered by them as a divine tool and so it is useless to ask where the power of evolution ends and where the power of God begins. Kauffman ends his book *At Home in the Universe* suggesting that the Creator could have used evolution as a tool for design before Creation.

There are at last a group of persons that see a contradiction between their religion and evolution. It is curious that all these persons are Christian believers: their religion preaches a personal God that is close to everyone much as a father is close to his sons and daughters. Evolution contradicts the Creator because evolution poses a gap between him and us. In fact, every creationist must then conclude that evolution is not enough to explain life otherwise the Creator would be similar to a deprecated mannequin. These people also propose logical, scientific arguments denying the sufficiency of evolution and its natural extensions to explain the origin of species or of life but all their arguments have awaken scorn amidst the scientific community.

Dear Reader, let me to express my personal position:

- The anthropological witness about gods, demons and spirits is overwhelming and now includes many videos taken by security cameras and cellular phones. It is not intelligent to deny that reality. Can we include the spiritual realm into a new extended materialism?

- The existence of a spiritual realm by no means imply the existence of a Creator or of a God that will judge every one. The belief in a Creator is a matter of faith and nothing else. How this faith is born, maintained and nourished is a matter of current scientific study.

- To the best of our scientific development, we cannot explain the origin of life or of species without including events of exceedingly low probability. Those that reject the existence of a Creator would argue: everything is possible for chance! Those that already believe in a Creator would respond: Will chance make you free from the power of Satan and sin? As we see, there is no

possible dialog among believers and scientists. Our only choice is mutual respect together with a ruthless search for clearer and fairer concepts.

- The problem of the sufficiency of materialism + evolution + chance to explain life must be solved by personal **conviction** but a correct formulation of that problem is a matter that includes quantitative accounts and in that respect Java can help a lot. This will be illustrated below. Warning: Java is not the truth nor the way to it, it is only a tool that can be used to shed some most possibly insufficient light on many types of problems.

95 ***Exercise*** *Clearly and succinctly formulate your opinion about the problem of the fairness and sufficiency of materialism + evolution (and its natural extensions) + chance to explain the origin of life and of species. Suppose that a great majority of persons in the world accept your opinion: would that help peacemaking among the diverse religions and science or would it favor one more war?*

6.5 Conclusion

Evolution is a crude reality that no one can deny. Whether or not that reality is enough to explain life in all its forms and marvels is an intriguing matter. Science is dominated by the idea that a positive answer to that question is the correct one. In this regard, we have seen two programs, one that illustrates the power of evolution to generate morphological diversity and another one that shows how artists might use evolution in their creative job. Nevertheless, evolution as a reality is naturally incorporated in the world views defined by creationists religions: evolution is a tool that the Creator uses for his glory! Nevertheless, for a minority formed by fundamentalist Christian believers, evolution cannot be enough to explain the origin of life and of species otherwise the Creator would be similar to a deprecated mannequin. The problem of the sufficiency of materialism + evolution + chance to explain life must be solved by a personal decision but a correct formulation of that problem is a matter that includes quantitative accounts and in that respect Java can help a lot, a fact that will be shown below.

Chapter 7

Evolution is quantitative

Consistency upon repetition

96 ***Introduction***

Evolution contains two ingredients, natural selection and randomness. Natural selection is a term that refers to a systematic effect. For instance, individuals that have a better breathing machinery (better in terms of design and engineering), have more possibilities for surviving and reproduction than those that not. But those beings with the best breathing machineries are prone to take greater risks and so some of them could die before reproduction. So, we expect that the genes that are passed to the next generation belong to those with good breathing machineries but intelligent enough to be prudent. By contrast, randomness accounts for all exceptions to the rule given by natural selection. Instance: those individuals with the worst breathing machinery got relegated and separated from the flock while the rest fell chased by predators. Consequence: a new evolutionary branch was to begun from the worst individuals and that happened thanks to randomness.

One arrives to the conclusion that when one is allowed to use natural selection and randomness to explain something in biology one finds no mysteries at all because everything is possible. The problem is not to show the existence of an explanation but to choose what explanation is the best among myriads. In this way, chance is made into a philosophy: if something must be explained and if its not explained by natural selection, then it is explained by chance. This is the actual philosophy of modern biology. In this philosophy, it does not make sense to doubt that we arouse by evolution because a philosophy is an operative principle that by definition is uncontestable. The first use of a philosophy is to define what must be taken as an unquestionable truth and what as lies, illusion or noise. So, any

claim or witness about an spiritual reality is taken by materialism as a lie or as an unfounded illusion or as a fraud.

We see that if one adopts the official philosophy of modern biology, all further discussions about the sufficiency of materialism and evolution to explain life are senseless. Nevertheless, it makes sense in that philosophy to ask: What characteristics does evolution have that incite others to invent and adopt alien philosophies apart from that consigned in the international journals of modern biology?

97 ***Purpose*** *The force of evolution resides in the fact that it is crude reality. And as such, it can be studied, and more to the point, it can be simulated. Our purpose in this chapter is to show that chance and evolution have a way of being, a personality, that exhibits and reinforces itself with repetition. This will prepare us for a thematic that will be studied in the next chapter: the personality of evolution is precisely the very fact that makes the evolutionary theory very weak in the market of ideologies because it makes sense to ask: Does nature violate the personality of evolution?*

7.1 Chance has a personality

Ordinary experimentation shows us that chance has a way of being, a personality, that exhibits and reinforces itself with repetition.

98 ***Dies and lotteries***

Let us consider a die. We are used to expressions of the type *the probability of 5 is 1/6* and we know that this means that if we throw the die 6 million times, the die will show up a 5 approximately one million times. But if we throw the die 3 times, only 3 times, no one can predict what will happen.

In short: chance is almighty in a world of few events but it acquires a personality in a world of a large number of repetitions. So, if you win the lottery once, you can claim that it is by chance. But if you win the lottery 3 times in a run, then you must expect to be investigated because of illicit enrichment: no one is prepared to believe your story without exhaustive verification. Why? Because chance acquires a personality that shows itself in repetitions (for lotteries and the same person, 3 is already a very large number). According to the personality of chance, lotteries are expected to be won by different persons with a probability near to 1, to certainty.

All this thematic is studied by the theory of probabilities and is at the heart of the scientific method. The idea is very simple and very natural: if you have a theory to explain events and if a given event violates the personality of chance, you are invited to reject the theory and to formulate and study a new one.

Example (continued): if you have the theory that says that you won the lottery 3 times in a run by mere chance, then, please, realize that your theory violates the personality of chance and be ready to be investigated as suspect of trying to cover illicit cash with a fictitious lottery. In general, to violate the personality of chance implies the prompt to invent a new theory. We see that in the scientific method there is no final theory but only a method to decide when it is justified to reject a theory to invent a new one. In general, scientific tests of theories cannot prove that they are correct but only that they are false (Popper, 1935). If you need to know about this in more detail and if you find the ordinary literature very confuse, please, give a look at (Rodríguez, 2010).

That is as we function in ordinary life and that is as we function in technology, in ordinary science and in everything that is related with chance, as chance games and actuary.

What do we need to apply this vision to the study of evolution? We need to play with evolution long enough to unveil and characterize its personalty to then find a descriptor according to which actual life and species violate that personality. Else, we must prove than no descriptor exists that discriminates actual life from ordinary evolutionary events. Along this trend of ideas, we continue with our task: the characterization of the personality of chance and of evolution to discuss next in which way actual life might violate those personalities.

7.2 Chance vs Retro-engineering

We have said that ordinary experimentation shows us that chance has a way of being, a personality that exhibits and reinforces itself with repetition. But exactly, what does this mean?

The official answer is given by the theory of probabilities and the diverse distribution functions. Our answer will come in the form of a simulation targeted to support the next claim: *everything is possible for chance but it burns too much resources, so much that the whole Universe will be insufficient for its needs.*

99 ***The retro-engineering problem***

A look at the retro-engineering problem and the way as it is solved by chance and next by evolution can help us to grasp our previous claim.

The *retro-engineering problem* consists in reproducing an already known solution to a problem. For all effects, a solution can be imagined as a string of words in a given language that might explain, for instance, how to build a craft. As a general rule, we are not interested in receipts 10 words long but in procedures that take years of study and experimentation to be unveiled. So, we are discussing

the re-discovery of complexity, i.e., we try to explain how we can reproduce the complexity that already exists.

Our retro-engineering problem is related to the problem of explaining complexity of living beings as follows: we are not interesting in explaining the creation of complexity ab initio. Instead, we want to understand the origin of the complexity of the living beings we know. So, we want to reproduce a known solution. Henceforth, we are dealing with a retro-engineering problem.

The biological complexity that already exists was explained during the XX century by science as the result either of randomness or of accumulation of small change. Randomness is invoked in *The Blind Watchmaker* (Dawkins, 1986) as follows: "I don't know who it was first pointed out that, given enough time, a monkey bashing away at random on a typewriter could produce all the works of Shakespeare. The operative phrase is, of course, given enough time." (pg 46). The author next continues with a simulation in which he synthesizes a phrase of Hamlet: "Methinks it is like a weasel" (in reference to the similitude of a cloud to a weasel, a carnivorous mammal.) He finds that the phrase can indeed be synthesized by randomness but only after a long time. It is very instructive to have access to direct experimentation with this problem.

The next simulation shows how one can use chance to guess a string when the only feedback that one can receive is whether or not one has guessed that number. But we have a problem: when one uses a single processor, the most limiting resource is time and when one tries to make a job using chance and nothing else running time could be very long even diverging to infinite. So, we must enable the halting at will of a running program.

100 ***How to stop a program?***

Java is a developing platform for experts. But it must attract everybody. How does this is possible? The solution is to enable programming at low level, for beginners, and also enable programming for experts at a very high level.

Beginners must do their job and go home happily. So, they are not intended to begin tasks that might consume a lot of time. Therefore Eclipse has provided for them the possibility to abort the program by clicking on the red button of the console. This in general functions very well. But experts consider this option as deprived of elegance and is insufficient for the study of complex problems. Halting a program from inside is in Java a task for experts, as well as the design of GUI and 3D graphics. The same happens with evolution: it can be simulated by everyone but, as we will be very soon, one can finely program evolution as bad as desired.

Now, there are some few principles that one must take into account to design a program with the possibility to be halted and resumed at will:

1. Every program defines automatically one or more execution threads about which the User is not aware of. So, the user cannot stop them. To be able to stop a program, the User must define his or her own thread and hang his task from it. Because the thread belongs to the User, he or she can stop it. In the program below, our thread is named *auxThread*.

2. Java is OOP (Oriented Object Programming) and so it deals very poorly with separate variables. Please, unite your in- and out-variables in an object which is defined by a class. In the program below, our object is defined in the class *Job*.

3. We use a GUI (Graphic User Interface), so to start the execution, the User must press the *Go* button and to stop execution, the User must click on the *Stop* button. These actions trigger respectively the starting of the thread and its halting. Please, look at the code that implements the corresponding protocol.

4. It happens that the *stop* button interrupts the execution. This interruption must be used by the Developer to nullify the thread and to halt the task under execution. For this reason, the task must be in between a loop that contains a boolean flag that enables the task to be continued. To that aim, we use in the program randomSearch, below, the flag *go*.

5. A nullified threads does not disappear. Instead, it is converted into a ghost that operates as effectively as when it was alive. The reason is that a thread vanishes only when it is swept by Java but Java sweeps useless things only at its own will. At the meantime, ghost threads can cause a lot of problems. One cannot stop the functioning of ghost threads but one must interdict its manifestation in our world. That is why one of the conditions for execution is to verify that the current thread is the thread that one works with. That is why we verify the equality of the *currentThread* with our *auxThread*.

6. Anyway, ghost threads could cause diverse problems that are attacked with a verification that we are working with our thread and with suitable exception handling. For instance, in the program below, ghost threads use to try to update text fields. This is not allowed by Java monitors and an exception (an infrequent problem) might be generated. That is why some clauses that update text fields are surrounding with try-catch expressions.

7. It is plainly possible that the program does not stop even if one implements all aforementioned ideas but it produces no exception either. In that case, the User must detect the most important terminal variables: when a given value

is reached, the job ends and the program stops. The User must include in the STOP method some clauses in which those variables are given the terminator values. To enable further work, a reinitialization of variables might be necessary. Please, look at the program Shakespeare, below, to see how all this done.

101 ***The code for chance***

The next program takes a string and reports the number of random trials to guess it. We have three options for the alphabet:

- binary numbers, like 100110;
- digits, like 2367;
- chars, like dfgy3d.

We offer two modalities to guess a given string: all at once else step by step along a ladder. In the last case, one takes the first char of the given string and guesses it by chance. Once it has been guessed, one tries to guess the substring with the first 2 chars of the target string. The number of random trials to guess it is reported. And so on for string *k* chars long until the User gets tired or satisfied. In the last case, he or she gets convinced that chance can guess any string no matter how long it could be albeit wasting a lot of time. Next a graphic of the needed number of trials to guess the goal string against *k* is drawn.

```
/*Program D101  randomSearch
 A string of length k is declared as goal
 and it is guessed by random searching.
 The number of trials done to
 achieve the goal is reported.

 Moral: everything is possible for chance
 but it wastes too much resources.

 The program has three main parts:
 First: the main program, line 76, which
 is a regulative unit.
 Second: the program that executes the searching
 by random exploration, line 122
 Third: the GUI (graphic User Interface) controller,
```

```
 line 368.

 This is the first time that the possibility to halt
 at will a running execution  is included in our programs.
*/

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;

import java.awt.TextArea;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Random;

import javax.swing.Box;
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.JRootPane;
import javax.swing.JSplitPane;

import javax.swing.JTextField;

import javax.swing.SwingUtilities;



//Main class
public class randomSearch
{
 //Instantiation of various classes
 private static randomSearch t = new randomSearch();
 //Some relevant information is packed into a class.
 //this allows stability of memories against thread threats.
```

```
 private static Job h = t.new Job();
 //The JFrame which shows everything
 // is constructed with a title
 private static JFrame f = new JFrame("Chance: " +
  "To modify parameters, end with enter.");

 private static GUI w =  t.new GUI();
 private static  Search s = t.new  Search();

 //A particular string
 private  static String individual = "";
 private static String typeAlphabet;
 private static String alphabet =
    "abcdefghijklmnopqrstuvwxyzABCDEF" +
    "GHIJKLMNOPQRSTUVWXYZ 0123456789,.?!'";
 //length of the alphabet
 private static int   lengthAlpha;


 // This thread is used to enable termination of lengthy tasks.
 private volatile Thread auxThread;

 public static void main(String[] args)
  {

   //WGUI is added to the JFrame f
   f.add( w, BorderLayout.CENTER );
   f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
   //Dimensions of the JFrame
   f.setSize( 600, 620 );
   w.setSize(30,20);
   f.setVisible( true );
   f.add( s, BorderLayout.CENTER );

  }

//*****************************************
//****************JOB CLASS ******************
//*****************************************
```

```
 //This auxiliary class unites some fundamental
 //variables into an object
 private  class Job
 {
  String goal;
  Integer l;
  long nTrials;
  boolean show;
  boolean makeGraphic;
  int nPoints;

  //Mandatory initialization
  Job( )
{
 goal = "1010001";
 l = goal.length();
 nTrials = 1;
 show = false;
 makeGraphic = false;
 nPoints = 0;
}

  }//end of class Job

//*****************************************
//****************SEARCH CLASS ******************
//*****************************************

 //This class is responsible for guessing the goal
 private class  Search  extends JPanel
 {
  private static final long serialVersionUID = 1L;
  //Boolean condition to continue else halt
  private    boolean go = true;
  //Points for a graphic
  double PointsSim[][] = new double[100][2];
  double PointsExp[][] = new double[100][2];
  int Points[][] = new int[100][2];

  double xmax, ymax;
```

```
  Integer base = 1;
  int deltaY;

  //Constructor: mandatory initialization;
  Search()
  {
   for(int i = 0; i < 100; i++)
    for(int j = 0; j < 2; j++)
  {
   Points[i][j] = 0;
   PointsSim[i][j] = 0;
   PointsExp[i][j] = 0;
  }
  }

  // Turn on of the random generator
  Random r  = new Random();

  // This method generates random chars
  private     char  randomChar()
  {
   lengthAlpha = alphabet.length();
   int i = r.nextInt(lengthAlpha);
   return   alphabet.charAt(i);
  }

  // A string of n chars is generated
  private   String generateString(int n)
  {
   individual = "";
   char c = '0';
   //An individual is assembled char by char
   for(int j = 0; j < n; j++)
{
 if (typeAlphabet.equals("Binary"))
{
 c =   (char) (r.nextInt(2)+48);
 base = 2;
    }
 if (typeAlphabet.equals("Digits"))
```

```
    {
 c =   (char) (r.nextInt(10)+48);
 base = 10;
    }
 if (typeAlphabet.equals("Chars"))
    {
 c =  randomChar();
 base = 69;
    }
 // s is concatenated to Individual[i]
 individual  = individual + c;
}
return individual;
  }

 // The goal is defined
 private   String   defineProblem(int n)
  {
   String g  = generateString(n);
   return g;
  }

 //   The goal is achieved by guessing at random
 private    void randomSolution(String goal)
                          throws InterruptedException
 {
  long trial = 0;
  String a;
  h.l = goal.length();
  try
   {
if ((Thread.currentThread() == auxThread)
            & (auxThread != null))
 {
  w.goalTF.setText(h.goal);
  w.lengthTF.setText(((Integer) h.l).toString());
 }
   }
  catch(NullPointerException e)
  {
```

```
   e.printStackTrace();
  }
  while (go)
   {
trial = trial + 1;
a = generateString(h.l);
if (h.show)
 {
if (Thread.currentThread() == auxThread)
w.news.append("Goal = " + h.goal +
      " Trial number " + trial + " " +
      " Generated =  " + a + "\f");
 }
if (a.equals(goal))
 {
  h.nTrials = trial;
  w.news.append("Goal = " + h.goal + " of lenght "
                 + h.l + " guessed by trial "
                 + h.nTrials + "\n" );
  go =false;
 }
    }
 }

 //One chooses to match a given string
 private  void  oneProblem() throws InterruptedException
  {
while  ((go) &   (Thread.currentThread() == auxThread))
 {
  h.goal = defineProblem(h.l);
  randomSolution(h.goal);
  w.btnNewTry.setEnabled(true);
 }
  }

 //The ladder job is defined and solved
 private    void  work() throws InterruptedException
  {

   while  ((go) &   (Thread.currentThread() == auxThread))
```

```
{
 h.goal = defineProblem(h.l);
 randomSolution(h.goal);
 PointsSim[h.l][0] = h.l;
 PointsSim[h.l][1] = h.nTrials;
 h.nPoints = h.l;
 go = true;
 h.l =  h.l+1;
}
  }

 private void findMax()
 {
  xmax = 0;
  ymax = 0;
  //Max values are detected
  for(int i = 1; i < h.nPoints; i++)
   {
if (PointsSim[i][0] > xmax) xmax =  PointsSim[i][0];
if (PointsSim[i][1] > ymax) ymax =  PointsSim[i][1];
   }
  for(int i = 1; i < h.nPoints; i++)
  {
if (PointsExp[i][0] > xmax) xmax =  PointsExp[i][0];
if (PointsExp[i][1] > ymax) ymax =  PointsExp[i][1];
  }
 }

 //Data of a graphic is normalized to be placed
 //in a given rectangle
 private void rescale( double[][] PointsD)
 {

  //scale factor are defined
  double  fx;
  double  fy;
  fx = 240 / xmax;
  fy = -360 / ymax;

  int tx = 60;
```

```
  int ty = 560;
  //Scaling formulae
  for(int i = 1; i < h.nPoints; i++)
  {
    Points[i][0] =  (int) (PointsD[i][0]*fx +tx);
   Points[i][1] =  (int)  (PointsD[i][1]*fy +ty);
  }
 }

 //The graphic is drawn
 private void draw(Graphics g )
 {
  for(int count = 1; count < h.nPoints-1; count++)
   g.drawLine(Points[count][0], Points[count][1]+deltaY,
   Points[count+1][0],Points[count+1][1]+deltaY);
 }

 private void makeGraphic(Graphics g )
 {



  //An exponential function serves as reference
  for(int i = 1; i < h.nPoints; i++)
   {
PointsExp[i][0] =  i;
PointsExp[i][1] =  Math.pow(base, i);
   }
  findMax();
  deltaY=0;
  g.drawRect( 50,180+deltaY, 295,390);
  g.setColor(Color.BLACK);
  g.fillRect(50,180+deltaY, 295,390);
  rescale(PointsSim);
  g.setColor(Color.RED);
  g.drawString("Simulation: Number of trials vs " +
            " Length of the goal", 60, 200+deltaY);
  //Simulation data are drawn
  draw(g);
  rescale(PointsExp);
```

```
  g.setColor(Color.CYAN);
  //Exponential is drawn
  g.drawString("Exponential = " + base.toString()
                 + "^ n", 60, 220+deltaY);
  draw(g);
  h.makeGraphic = false;
 }


//This is the main (but not mandatory)  method,
 //used only for painting
 @Override public void paintComponent( Graphics g)
 {
  super.paintComponent(g);
  if (h.makeGraphic)
  {
   makeGraphic(g);
   //Delete the next line and see what happens.
   //Challenge:
   //Can you find an elegant solution to the
   //ensuing problems (if any)?
   w.buttons();
  }

 }

 }//End of class Search

//*****************************************
//****************GUI CLASS ******************
//*****************************************

 //This class operates the graphic interface GUI
 private class GUI extends JPanel
 {
  private static final long serialVersionUID = 1L;
  private JTextField lengthTF;
  private JTextField  goalTF;
  String TypeNames[]  = {"Binary", "Digits", "Chars" };
```

```
//private String goalPhrase = "METH";
private Integer nLength = h.l;
private int caseNumb;

//Control button

private JButton btnPurpose;
private JButton btnNewTry;
private JButton btnGo;
private JButton btnStop;
private TextArea news;

private JRadioButtonMenuItem Show[];
private ButtonGroup ShowButtonGroup;
private JRadioButtonMenuItem Type[];
private ButtonGroup TypeButtonGroup;

// Planning of the   screen
private   GUI( )
{
 //Organization of buttons
 Box box1 = Box.createHorizontalBox();
 Box box2 = Box.createHorizontalBox();
 Box box3 = Box.createHorizontalBox();
 Box box4 = Box.createHorizontalBox();
 Box box5 = Box.createVerticalBox();



 //  ************ button Purpose
 btnPurpose = new JButton("Purpose");
 add (btnPurpose);
 Purpose purpose = new Purpose ();
 btnPurpose.addActionListener(purpose);

 // btnPurpose.setBounds(50, 50, 50, 20);

 //     ************   Type of String

 JLabel typeL = new JLabel("Type of alphabet " );
```

```
   box1.add(typeL);
   // Creates radio button menu items for type options
   Type  = new JRadioButtonMenuItem[ TypeNames.length ];
   TypeButtonGroup = new ButtonGroup();

   TypeHandler typeHandler =  new TypeHandler();
   //Creates type radio button menu items
   for ( int count = 0; count < Type.length; count++ )
    {
 Type[ count ]
   = new JRadioButtonMenuItem( TypeNames[ count ] );
 box1.add( Type[ count ] );
 TypeButtonGroup.add( Type[ count ] );
 //Calling of inner class typeHandler
 Type[ count ].addActionListener( typeHandler );
}
   //Initial setting : binary numbers;
   Type[ 0 ].setSelected( true );
   typeAlphabet =  "Binary";
   add(box1);

   //  ************   Show work

   JLabel showL = new JLabel("Show work " );
   box2.add(showL);
   String ShowNames[]  = { "Yes", "No" };
   // Creates radio button menu items for Yes-No options
   Show  = new JRadioButtonMenuItem[ ShowNames.length ];
   ShowButtonGroup = new ButtonGroup();

   ShowHandler showHandler =  new ShowHandler();
   //Creates show radio button menu items
   for ( int count = 0; count < Show.length; count++ )
{
 Show[ count ]
   = new JRadioButtonMenuItem( ShowNames[ count ] );
 box2.add( Show[ count ] );
 ShowButtonGroup.add( Show[ count ] );
 //Calling of inner class showHandler
 Show[ count ].addActionListener( showHandler );
```

```
}
   //Initial setting : work is not shown;
   Show[ 1 ].setSelected( true );
   add(box2);


   //  ************ button Go
   btnGo = new JButton("GO for the ladder job");
   box3.add (btnGo);

   btnGo.addActionListener
(
 new ActionListener()
  {
   public void actionPerformed(ActionEvent e)
    {
     go();
    }
  }
);


  //  ************ button NewTry
  btnNewTry = new JButton("NEW RANDOM TRY");
  box3.add (btnNewTry);
  NewTry newTry = new NewTry ();
  btnNewTry.addActionListener(newTry);

  //  ************ txtField length
  JLabel lengthLabel= new JLabel(
   "  Length of  goal  to be matched ");
  lengthTF = new JTextField(nLength.toString()+
                            "        " );
  lengthTF.setLocation(100, 20);
  box3.add(lengthLabel);
  box3.add(lengthTF);
  Length length = new Length();
  lengthTF.addActionListener(length);
  add(box3);
```

```
 // ************ txtField goalPhrase
 JLabel phraseLabel= new JLabel("  Goal to be matched ");
 goalTF = new JTextField( h.goal +
   "                                      ");
 box4.add(phraseLabel);
 box4.add(goalTF);
 Goal goal = new Goal();
 goalTF.addActionListener(goal);



 //  ************ button Stop
 btnStop = new JButton("STOP");
 box4.add (btnStop);
 btnStop.setEnabled(false);
 btnStop.addActionListener
  (
new ActionListener()
 {
  public void actionPerformed(ActionEvent e)
   {
    stop();
   }
   }
  );

 add(box4);




//************* newspaper
box5.add(news = new TextArea("Subgoals and  matching " +
                    "generations \n" , 4, 78));



news.setFont(new Font("Courier", Font.PLAIN, 12));
news.setEditable(false);
```

```
add(box5);
}


//Inner class to deal with the purpose  button
class Purpose implements ActionListener
{
 public void actionPerformed(ActionEvent event)
 {
  JOptionPane.showMessageDialog
  (null,
   "This simulation illustrates what we think " +
   "\nis the fundamental description of chance:  " +
   "\nEverything is possible for chance but" +
   "\nit wastes too much resources, so much  " +
   "\nthat our whole universe will not be enough" +
   "\nto cope with its needs."
  );
  repaint();
 }
}


//If the newTry button is clicked upon,
//something must be done:
class NewTry implements ActionListener
{
 public void actionPerformed(ActionEvent event)
 {
  caseNumb = 1;
  task();
 }
}

// Inner class to handle events from the Type menu
private class TypeHandler implements ActionListener
{
 public void actionPerformed( ActionEvent event )
```

```
  {
   for ( int i = 0; i < 3; i++)
{
 // check for "Binary"
 if ( Type[ 0 ].isSelected() )
 {
  typeAlphabet = "Binary";
  break;
     }

 // check for "Digits"
 if ( Type[ 1 ].isSelected() )
  {
   typeAlphabet = "Digits";
   break;
  }

 // check for "Chars"
 if ( Type[ 2 ].isSelected() )
  {
   typeAlphabet = "Chars";
   break;
  }
 }
}
  }//end of TypeHandler


 //Inner class to handle events from the show menu
 private class ShowHandler implements ActionListener
 {
  public void actionPerformed( ActionEvent event )
  {
   //
   for ( int i = 0; i < 2; i++)
{
 // check for "Yes"
 if ( Show[ 0 ].isSelected() )
  {
   h.show = true;
```

```
   break;
 }

// check for "Not"
if ( Show[ 1 ].isSelected() )
 {
  h.show = false;
  break;
 }
}
  }
 }//end of ShowHandler

 // Ending blanks are deleted
 private String purify0(String s)
 {
  String p = "";
  char d = ' ';
  char c = 'c';
  boolean flag = true;
  int n= s.length();
  int counter = 0;
  while (flag)
  {
   c = s.charAt(n-1-counter);
   if ((c == d)) counter = counter +1;
   else flag = false;
  }
  p = s.substring(0,n-counter);
  return p;
 }

 // Only numbers are accepted
 private String purify1(String s)
 {
  String p = "";
  char c = 'c';
  for (int i = 0; i < s.length(); i++)
  {
   c = s.charAt(i);
```

```
  for(int j = 48; j< 58; j++)
 if (c == (char) (j)) p = p+c;
 }
 return p;
}

// Only binary numbers are accepted
private String purify2(String s)
{
 String p = "";
 char c = 'c';
 for (int i = 0; i < s.length(); i++)
 {
  c = s.charAt(i);
  for(int j = 48; j< 50; j++)
 if (c == (char) (j)) p = p+c;
 }
 return p;
}


//The User can decide the length of the random goal
class Length implements ActionListener
{
 public void actionPerformed(ActionEvent event)
{
 String st = lengthTF.getText();
 st = purify1(st);
 if (st.equals(""))
 {
  JOptionPane.showMessageDialog(null,
  "Please, type a number");
 }
 else
 {
  nLength = new Integer(st).intValue();
  h.l = nLength;
  caseNumb = 1;
  task();
 }
```

```
}
}



//Reader of the goal
class Goal implements ActionListener
 {
  public void actionPerformed(ActionEvent event)
   {
String st1 = goalTF.getText();
//ending blanks are deleted

st1 = purify0(st1);
//Is st1 a number?

String st2 = purify1(st1);
String st3 = purify2(st2);
//Is st1 a binary number?
if (st1.equals(st3))
   {

    typeAlphabet = "Binary";
    Type[ 0 ].setSelected( true );
    Type[ 1 ].setSelected( false );
    Type[ 2 ].setSelected( false );
   }
//Is st1 a number 0123..9?
if ((st1.equals(st2)) & (!(st1.equals(st3))))
   {

    typeAlphabet = "Digits";
    Type[ 0 ].setSelected( false );
    Type[ 1 ].setSelected( true );
    Type[ 2 ].setSelected( false );
   }
//Is st1 a string with chars?
if (!(st1.equals(st2)))
  {
    typeAlphabet = "Chars";
```

```
     Type[ 0 ].setSelected( false );
 Type[ 1 ].setSelected( false );
 Type[ 2 ].setSelected( true );

    }


 h.goal =  st1;
 h.l = h.goal.length();
 lengthTF.setText(h.l.toString());
 goalTF.setText(h.goal);
 caseNumb = 2;
 task();
}
   }//end of class Goal

  //Trigger of action
  public void go( )
  {
   ensureEventThread();
   caseNumb = 0;
   task();
  }


  //When the action goes on,
  //Everything must be withdrawn from the USER
  //Except the stop button
  private void editionFalse()
  {
   btnGo.setEnabled(false);
   lengthTF.setEditable(false);
   goalTF.setEditable(false);
   btnNewTry.setEnabled(false);
   btnStop.setEnabled(true);
   Type[0].setEnabled(false);
   Type[1].setEnabled(false);
  }

  //The USER acquires edition power
```

```
  private void editionTrue()
  {
   auxThread = null;
   btnGo.setEnabled(true);
   btnNewTry.setEnabled(true);
   lengthTF.setEditable(true);
   goalTF.setEditable(true);
   Type[0].setEnabled(true);
   Type[1].setEnabled(true);
   h.show = false;
   Show[ 1 ].setSelected( true );
   btnStop.setEnabled(false);
   }

  //The action is hung from an execution thread.
  //In that way, it can be stopped
  private void task()
  {
   //Edition is forbidden
   editionFalse();
   repaint();
   //The action is encapsulated
   Runnable watchRun = new Runnable()
   {
public void run()
{
 w.news.setText("");
 try
  {
   s.go = true;
   switch (caseNumb)
   {
 case 0 : //Ladder job
 {
  h.l = 1;
  s.work();
  h.makeGraphic = true;
  s.repaint();
  break;
  }
```

```
          // a given phrase
 case 1 : s.oneProblem(); break;
         // length
 case 2  : s.randomSolution(h.goal);break;
}
editionTrue();
   } catch (InterruptedException e)
  {
e.printStackTrace();
  }
getReady( );

   }
 };

  //Encapsulated action  is connected to the thread
  auxThread = new Thread(watchRun, "auxThread");
  auxThread.start();
 }


  //If the execution is interrupted,
  //some housekeeping must be done.
  private void getReady( )
  {
   Runnable r = new Runnable()
    {
 public void run()
  {
   try {
    ensureEventThread();
    btnStop.setEnabled(false);
    btnGo.setEnabled(true);
   } catch (Exception x)
             {
              x.printStackTrace();
             }
   }
};
  SwingUtilities.invokeLater(r);
```

```
 }

 //Am I working with a ghost or with my thread?
 private void ensureEventThread()
 {
  if (SwingUtilities.isEventDispatchThread())
 return;
  throw new RuntimeException("Undue calling");
 }

 //The task is halted
 private void stop()
 {
  ensureEventThread();
  if (auxThread != null)
{
 auxThread.interrupt();
 editionTrue();
 s.go = false;
 w.news.append("Job cancelled  \n");


}

 }

 //An unknown bug causes troubles
 //that are remedied with some housekeeping.
 private void buttons()
 {
     btnPurpose.setVisible(false);
 btnPurpose.setVisible(true);
 lengthTF.setVisible(false);
 lengthTF.setVisible(true);
 Show[0].setVisible(false);
 Show[1].setVisible(false);
 Show[0].setVisible(true);
 Show[1].setVisible(true);

  goalTF.setVisible(true);
```

```
  btnNewTry.setVisible(true);
   btnGo.setVisible(true);
 btnStop.setVisible(true);
 }

}//End of class GUI


}//End of main application
```

102 ***Exercise*** *Run the program and play with the code. Pay attention to the form as the program enables the possibility to stop a running execution. Would you unveil some lines that try to make the GUI children-proof?*

103 ***Exercise*** *Using the previous program, design and run suitable experiments to get convinced else to refute that*

- *Chance can solve everyone task but there are some simple tasks that chance can execute rather fast while others might demand more time and others huge amounts of time.*
- *The number of trials needed by chance to guess a string n letters long is approximately equal to a^n where a is the numbers of letters in the used alphabet.*
- *Relative errors in the former prediction about the number of trials needed by chance to guess a goal are large when the goal to be find is short and the number of letters of the alphabet is small. But those errors tend to decrease in relative proportion when the numbers of letters in the alphabet or the length of the string to be guessed get large.*
- *These points are summarized as follows: complexity forces chance to behave in the exponential regime. This assertion will define for us the personality of chance.*
- *Make a suitable definition of complexity appropriate for this case.*

104 ***Chance is very expensive***

Chance can solve every task if only it is furnished with the appropriate alphabet. And it is not excluded that by mere casualty a very large job could be resolved by chance at its very first trial. For these reasons, chance always will have a position in the realm of science as a possible explanation of our complexity. We say officially that *chance is the null hypothesis of science*.

Nevertheless, complexity condemns chance to behave in the exponential regime. To understand the corresponding implications, let us try to explain the complexity of DNA using chance alone: the DNA alphabet has 4 letters and DNA strings can be very large, with 1000 millions bases. In this case, chance is condemned to make around $4^{1000000000}$ trials to explain its existence. But what is the problem?

The problem with chance is that in the world of exponentials all the resources of the Universe are extremely tiny: we have $10^{80} < 4^{160}$ atoms in the world and the age of the universe is calculated in 10^{15} years that are less than 4^{100} quanta of time (a quantum of time is the needed time by the fastest reaction to get accomplished. We took it as 10^{-50} seconds). The consequence is that to explain a normal DNA string by mere chance, we are assuming that chance guessed the string in the very first trials.

Now, we cannot refute the idea that single very complex events could appear by mere chance at once or in the very first trials. Nevertheless, the same explanation cannot be used to get rid of a huge number of very complex events as those that are involved in the existence of life and of species: the variability and volume of the involved complexity would condemn chance to behave in the exponential regime, a fact that excludes out our Universe as the possible realm where chance worked out its solutions.

7.3 The evolutionary theory

Chance is strong in science to explain our complexity in the measure that we cannot concoct more powerful scientific theories to explain that complexity. But after the publishing of *the Origin of Species* by **Charles Darwin** in 1857, we have the evolutionary theory which is currently interpreted as a theory about the origin of complexity supposing that some primordial life already exists. Let us review some interesting points about that theory.

105 ***Accumulating small change***

The evolutionary theory supposes that the complexity of life arouse in the very same way as a writer or an engineer or an artist creates complexity by accumulation

of small change. To support this claim, we need, in first instance, to replace the human intelligence of the writer or engineer by an automatic mechanism that shall be represented in nature. The basic evolutionary mechanism that amounts to an artificial creative intelligence is the following:

1. We define a goal to be matched.
2. We generate random strings and assess their performance to match the goal.
3. We choose those strings that best fit the goal, subject them to mutation and recombination to form a new generation of strings, which in their turn must be assessed and so on until we match the target or get satisfactorily close to it.

Let us notice that evolution is fed by randomness. We get pray at once of two preoccupations:

1. What does evolution make to lower the cost of chance, which by its very nature is very expensive?
2. If it happens that evolution can lower the cost of using chance as the fundamental fuel, does this happens in a repeatable form?

The next simulation will help us to understand what we mean when we say that: *evolution is proficient, in a perfectly repeatable form, to solve the retro-engineering problem*. This describes the personality of evolution such as it is perceived in the first impression. A more serious description of that personality will appear in the next chapter.

106 ***The repeatable proficiency of evolution***

After having witnessed the excessive price required by chance to explain extant complexity, we are commended now to study the evolutionary proposal. Let us read the introduction to the theme given by Dawkins in his *The Blind Watchmaker* in regard with a pertinent simulation in which evolution is given the task of guessing a phrase of 28 letters:

”[The program] again begins by choosing a random sequence of 28 letters, just as before:

WDLMNLT DTJBKWIRZREZLMQCO P

It now ’breeds from’ this random phrase. It duplicates it repeatedly, but with a certain chance of random error -’mutation’ - in the copying. The computer examines the mutant nonsense phrases, the ’progeny’ of the original phrase, and chooses

the one which, *however slightly*, most resembles the target phrase, METHINKS IT IS LIKE A WEASEL".

The author next explains what happens if the same process is iterated again and again: in some few generations the target phrase is reached.

Dawkins wrote his simulations in Basic and Pascal. Very soon, various programs were written by different professional developers to implement his ideas (see vol II). We provide here our version that everybody could learn by direct experimentation whether or not evolution can lower the price demanded by chance and if that is done in a systematic, repeatable form.

107 ***Our code that matches a given phrase***

```
/*
 * Program D107 Shakespeare
 * Our problem is  to match a given phrase,
 * which is traditionally taken from Shakespeare.
 * The idea is that his grandiosity
 * is not outside of the scope and power of
 * the evolutionary methods.
 *
 * We divide the goal in subgoals and each one of them
 * is guessed by evolution.
 *
 * The program has three principal parts:
 * 1) a main procedure that is a regulative unit,
 *    line 67.
 * 2) an inner class WGUI that controls
 *    the graphic interface, line 77.
 * 3) an inner class WeaselPro that is in charge
 *    of evolution, line 525.
 *
 */


import java.awt.BorderLayout;
import java.awt.Font;

import java.awt.TextArea;
```

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;


import javax.swing.Box;
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.JSlider;
import javax.swing.JTextField;
import javax.swing.SwingConstants;
import javax.swing.SwingUtilities;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import java.util.Random;


public class Shakespeare
{
 private static final long serialVersionUID = 1L;

 // The application is instantiated
 static Shakespeare t = new Shakespeare();
 static WGUI w = t.new WGUI();
 static WeaselPro p = t.new WeaselPro();

 // The JFrame which shows everything
 // is constructed with a title
 static JFrame f =
    new JFrame("Evolution matchs Shakespeare");

 //This thread is used to enable termination of
 //Lengthy tasks.
 private volatile Thread auxThread;
```

```
public static void main( String args[] )
 {
  //WGUI is added to the JFrame f
  f.add( w, BorderLayout.CENTER );
  f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  //Dimensions of the JFrame
  f.setSize( 600, 600 );
  f.setVisible( true );
 }

private class WGUI extends JPanel
{
 private static final long serialVersionUID = 1L;
 private JTextField lengthTF;
 private JTextField  phraseTF;
 private JTextField  popSizeTF;

 private String goalPhrase =
                  "METHINKS IT IS LIKE A WEASEL";
 private Integer nLength = goalPhrase.length();
 private Integer  popSize = 200;
 private int mutRateInt = 10;
 //mutRate = mutRateInt/100;
 private double mutRate  = 0.1;
 private boolean recomb = false;



 //Control button

 private JButton btnPurpose;
 private JButton btnNewTry;
 private JButton btnGo;
 private JButton btnStop;
 private TextArea news;
 private JRadioButtonMenuItem Rec[];
 private ButtonGroup RecButtonGroup;
```

```
 private JSlider mutSlider;
// Planning of the   screen
private   WGUI()
{
 //Organization of buttons
 Box box1 = Box.createHorizontalBox();
 Box box2 = Box.createHorizontalBox();
 Box box3 = Box.createHorizontalBox();
 Box box4 = Box.createHorizontalBox();
 Box box5 = Box.createHorizontalBox();
 Box box6 = Box.createHorizontalBox();

 //  ************ button Purpose
 btnPurpose = new JButton("Purpose");
 add (btnPurpose);
 Purpose purpose = new Purpose ();
 btnPurpose.addActionListener(purpose);


 //  ************ button NewTry
 btnNewTry = new JButton("NEW RANDOM TRY");
 box1.add (btnNewTry);
 NewTry newTry = new NewTry ();
 btnNewTry.addActionListener(newTry);


 //  ************ txtField length
 JLabel lengthLabel= new JLabel(
  "  Length of the goal phrase to be matched ");
 lengthTF = new JTextField( "  " + nLength.toString() );
 lengthTF.setLocation(100, 20);
 box1.add(lengthLabel);
 box1.add(lengthTF);
 Length length = new Length();
 lengthTF.addActionListener(length);
 add(box1);

 // ************ txtField goalPhrase
 JLabel phraseLabel=new JLabel("  Phrase To be matched ");
 phraseTF = new JTextField( goalPhrase + "             ");
```

```
box2.add(phraseLabel);
box2.add(phraseTF);
Phrase phrase = new Phrase();
phraseTF.addActionListener(phrase);
add(box2);

//  ************ txtField popSize
JLabel popSizeLabel= new JLabel("Population size");
popSizeTF = new JTextField("     " + popSize.toString());

box3.add(popSizeLabel);
box3.add(popSizeTF );
PopSize popSize = new PopSize();
popSizeTF.addActionListener(popSize);
add(box3);

//***************** Mutation

JLabel mutLabel=
         new JLabel("Mutation rate/char/generation");
box4.add(mutLabel);
mutSlider =
   new JSlider(SwingConstants.HORIZONTAL, 0, 100,10);
mutSlider.setMajorTickSpacing(10);
mutSlider.setValue( mutRateInt );
mutSlider.setPaintTicks( true );
Slider slider = new Slider();
mutSlider.addChangeListener(slider);
box4.add(mutSlider);
add(box4);

//  ************   Recombination

JLabel recL = new JLabel("Recombination " );
box5.add(recL);
String RecNames[]  = { "Yes", "No" };
// Creates radio button menu items for rec options
Rec  = new JRadioButtonMenuItem[ RecNames.length ];
RecButtonGroup = new ButtonGroup();
```

```
RecHandler recHandler =  new RecHandler();
//Creates rec radio button menu items
for ( int count = 0; count < Rec.length; count++ )
 {
  Rec[ count ]
 = new JRadioButtonMenuItem( RecNames[ count ] );
  box5.add( Rec[ count ] );
  RecButtonGroup.add( Rec[ count ] );
  //Calling of inner class recHandler
  Rec[ count ].addActionListener( recHandler );
 }
//Initial setting: no recombination;
Rec[ 1 ].setSelected( true );
add(box5);

//  ************ button Go
btnGo = new JButton("GO");
box6.add (btnGo);
btnGo.addActionListener
(
 new ActionListener()
 {
    public void actionPerformed(ActionEvent e)
    {
        go();
      }
  }
);


//  ************ button Stop
btnStop = new JButton("STOP");
box6.add (btnStop);
btnStop.setEnabled(false);
btnStop.addActionListener
(
  new ActionListener()
  {
    public void actionPerformed(ActionEvent e)
```

```
      {
          stop();
      }
    }
   );
  add(box6);


  //************* newspaper
  add(news = new TextArea("Subgoals and  matching " +
                      "generations \n" , 25, 80));
  news.setFont(new Font("Courier", Font.PLAIN, 12));
  news.setEditable(false);

  //  ************ End with enter
  JLabel warning = new JLabel("To modify parameters," +
                     "\n end with enter." );
  add(warning);
}


//Inner class to deal with the purpose  button
class Purpose implements ActionListener
{
  public void actionPerformed(ActionEvent event)
  {
JOptionPane.showMessageDialog(null,
    "Evolution is very good at solving"+
"\na type of problems that can be divided" +
"\nin tiny sub-problems. " +
"\nTo illustrate this, we solve " +
"\nthe problem  consisting in" +
"\nguessing  a well known information, say," +
"\na work of Shakespeare, for instance, a phrase" +
"\nof Hamlet: 'Methinks it is like a weasel'" +
"\n ( in reference to the similitude of a  " +
"\ncloud to a  weasel, a carnivorous mammal)." +
"\nAdapted from " +
"The blind watchmaker, by Richard Dawkins (1986)." +
"\nThe great goal is divided in subgoals and" +
```

```
 "\neach one of them is guessed by evolution.");
 repaint();
   }
 }


 //If the newTry button is clicked upon,
 //something must be done:
 class NewTry implements ActionListener
 {
   public void actionPerformed(ActionEvent event)
   {
 goalPhrase = p.generateString(w.nLength);
 p.bigGoal = goalPhrase;
 p.bigGoalLength = goalPhrase.length();
 repaint();
     phraseTF.setText(w.goalPhrase + "     ");
     task();
   }
 }


 //Reading instrument of mutation rate
 class Slider implements ChangeListener
 {
public void stateChanged(ChangeEvent move)
 {
  mutRateInt = mutSlider.getValue();
  mutRate = mutRateInt;
  mutRate = mutRate/100;
  p.mutRate = mutRate;
 }

 }

//Inner class to handle events from the rec menu
 private class RecHandler implements ActionListener
 {
  public void actionPerformed( ActionEvent event )
  {
```

```
   //
   for ( int i = 0; i < 2; i++)
{
 // check for "Yes"
 if ( Rec[ 0 ].isSelected() )
 {
  //Communication with p
  recomb = true;
  break;
 }

 // check for "Not"
 if ( Rec[ 1 ].isSelected() )
 {
      // Communication with p
  recomb = false;
  break;
 }
    }
  p.recomb = recomb;

  }
 }//end of recHandler

 //Only numbers are accepted
  private String purify(String s)
 {
  String p = "";
  char c = 'c';
  for (int i = 0; i < s.length(); i++)
   {
c = s.charAt(i);
for(int j = 48; j< 58; j++)
      if (c == (char) (j)) p = p+c;
   }
return p;
 }

 //The User can decide the length of the random phrase
 class Length implements ActionListener
```

```
 {
   public void actionPerformed(ActionEvent event)
   {
String s = lengthTF.getText();
s = purify(s);
if (s == "")
{
 JOptionPane.showMessageDialog(null,
     "Please, type a number");
}
else
{
 nLength = new Integer(s).intValue();
 p.bigGoalLength = nLength;
 goalPhrase = p.generateString(nLength);
 phraseTF.setText(goalPhrase + "     ");
 p.bigGoal = goalPhrase;
 p.bigGoalLength = goalPhrase.length();
 p.mutRate = mutRate;
}
   }
 }

 //Sensor of population size
 class PopSize implements ActionListener
 {
   public void actionPerformed(ActionEvent event)
   {
String s =  popSizeTF.getText();
s = purify(s);
if (s == "")
{
 JOptionPane.showMessageDialog(null,
     "Please, type a number");
}
else
{
 popSize =  new Integer(s).intValue();
 p.nIndiv = popSize;
}
```

```
  }
}

//Reader of the phrase
class Phrase implements ActionListener
{
  public void actionPerformed(ActionEvent event)
  {
goalPhrase =  phraseTF.getText() ;
p.bigGoal  = goalPhrase;
p.bigGoalLength = nLength;
p.mutRate = mutRate;
phraseTF.setText(goalPhrase);
nLength = goalPhrase.length();
p.bigGoalLength = goalPhrase.length();
lengthTF.setText(nLength.toString());
  }
}

public void go( )
 {
  ensureEventThread();

  p.mutRate = mutRate;
  p.bigGoal  = goalPhrase;
  p.bigGoalLength = goalPhrase.length();
  p.recomb = recomb;
  p.nIndiv = popSize;
  w.nLength = goalPhrase.length();

  p.problem = true;
  task();
 }

//When the action goes on,
//everything must be withdrawn from the USER
//Except the stop button
private void editionFalse()
{
 btnGo.setEnabled(false);
```

```
  btnStop.setEnabled(true);
  btnPurpose.setEnabled(false);
  btnNewTry.setEnabled(false);
  for (int i = 0; i<2; i++) Rec[i].setEnabled(false);
  mutSlider.setEnabled(false);
  lengthTF.setEditable(false);
  phraseTF.setEditable(false);
  popSizeTF.setEditable(false);
 }

//The USER acquires  edition power
 private void editionTrue()
 {
  btnGo.setEnabled(true);
  btnStop.setEnabled(false);
  btnPurpose.setEnabled(true);
  btnNewTry.setEnabled(true);
  for (int i = 0; i<2; i++) Rec[i].setEnabled(true);
  mutSlider.setEnabled(true);
  lengthTF.setEditable(true);
  phraseTF.setEditable(true);
  popSizeTF.setEditable(true);
 }

 private void task()
  {
   //Edition is forbidden
   editionFalse();


   Runnable watchRun = new Runnable()
   {
 public void run()
  {
    work();
    getReady( );
  }
   };

   auxThread = new Thread(watchRun, "auxThread");
```

```
   auxThread.start();
  }


 private void work()
 {
  p.solveTask();
  editionTrue();
 }

 private void getReady( )
 {
   Runnable r = new Runnable()
    {
  public void run()
   {
     try {
         ensureEventThread();
            btnStop.setEnabled(false);
            btnGo.setEnabled(true);

         } catch (Exception x)
             {
              x.printStackTrace();
             }
   }
};
 SwingUtilities.invokeLater(r);
 }




 private void ensureEventThread()
 {
  if (SwingUtilities.isEventDispatchThread())
 return;
  throw new RuntimeException("Error on fly");

 }
```

```
 private void stop()
 {
  ensureEventThread();
  if (auxThread != null)
   {
auxThread.interrupt();
editionTrue();
    //Terminator variables are given their final values
p.problem = false;
w.nLength = 0;
p.bigGoalLength = 0;
w.news.append("\n JOB CANCELLED JOB CANCELLED   " +
        "JOB CANCELLED JOB CANCELLED   \n \n");


   }
 }

}//End of WGUI

 class WeaselPro
 {
  /* Evolution of a population
     of strings.

     Strings are at the same time
     genotype and phenotype.
     They are like ribozymes.
     Selection is organismic, in the sense that
     the genome is assessed as a whole,
     but the big goal is divided in subgoals.

     Individuals are kept in the array
     Individual[]. It is an array of strings.


     The population has nIndiv members
```

```
   We use a  divide, conquer and defend algorithm:
   to match the goalPhrase of length i
   the program   defines and uses evolution to solve
   a succession of i subgoals taking from bigGoal
   the first k chars for 1<= k <= i.
   This  speeds the algorithm but
   lacks semantic fairness:
   a string acquires sense in a human language
   not char by char but instead by great units,
   say, roots, words and small subphrases.
*/


int N;
int nIndiv = w.popSize;
String Individual[ ];//Predeclaration
String Individual2[ ];
String  individual ="";
int stringLength;
int bigGoalLength;
// Order  is an array of integers.
int Order[];//pre-declaration
int gen;
boolean problem = true;
boolean goOn = true;
boolean recomb = false;
String bigGoal = "";
String subGoal = "";
String champString = "";
int   Champ=0;
int Fitness[] = new int[20000];
int FitnessC[] = new int[20000];
double mutRate = w.mutRate;
int subGoalNumber;

private   final String alphabet =
  "abcdefghijklmnopqrstuvwxyzABCDEF" +
  "GHIJKLMNOPQRSTUVWXYZ 0123456789,.?!'";
private int   lengthAlpha;
```

```
 // Turn on of the random generator
 Random r  = new Random();

 // This method generates random chars
 private     char  randomChar()
  {
lengthAlpha = alphabet.length();
int i = r.nextInt(lengthAlpha);
return   alphabet.charAt(i);
  }

 // A random string of length stringLength is generated
 private   String generateString(int stringLength)
 {
   individual = "";
   //An individual is assembled char by char.
   //This does not happen with human languages.
   for(int j = 0; j < stringLength; j++)
    {
     // We generate a random char
     char c =  randomChar();
     String s = ""+c;
     // s is concatenated to Individual[i]
     individual  = individual +s;
 }
 return individual;
}

// An initial population of strings is set up.
private   void createPop(int stringhLength )
{
 //Formal declaration of our arrays.
 Individual= new String[20000];
 Individual2= new String[20000];
 for(int i = 0; i< nIndiv; i++)
  {
   Individual[i]=generateString(stringLength);
   Individual2[i]=Individual[i];
  }
```

```
   // Declarations and default initializations
   // of other arrays.
   Order= new int[20000];
   for(int i = 0; i< nIndiv; i++)
 Order[i]=0;
 }

 public void report1()
 {
  w.news.append("Gen = " +gen +
     " Achieved goal = " + Individual[Champ ] + "\n");
  w.news.append( "Sub-task ended by generation = "
                        + gen + "\n");
  w.news.append( "Number of trials = "
                  + gen*nIndiv + "\n");
 }


 //The more similar to the goal, the fitter.
 //Exception for the sake of ghost threads
 public int fitness(String subGoal, String indiv)
 throws NullPointerException
  {
   int errors = 0;
   int len = subGoal.length();
   for (int i = 0; i < len; ++i)
{
   //In case of ghost threads
   if  ( (i<0) || (i > indiv.length()-1))
   {
    problem = false;
    return errors = 0;
   }
   if ((Thread.currentThread() == auxThread)
          & (auxThread != null))
 {
   if (!(indiv.charAt(i) == subGoal.charAt(i)))
   errors = errors +1;
 }
   if  ( (i<0) || (i > indiv.length()-1))
```

```
   {
    problem = false;
    w.nLength = 0;
    return errors = 0;
   }
}
   return errors;
  }



 private void fitnessPop()
 {
  for(int i = 0; i < nIndiv; i++)
Fitness[i] =nIndiv;
  for(int i = 0; i < nIndiv; i++)
  {
Fitness[i] = fitness(subGoal, Individual[i]);
FitnessC[i] =Fitness[i];
   //Fitness 0 means no errors
   if (Fitness[i] == 0)
 {
  Champ = i;
  champString = Individual[i];
  problem =false;
 }
  }
 }

 //We sort individuals by fitness
 private void sort()
 {
  if (problem ==true)
  {
   for(int i = 0; i< nIndiv;i++)
   {
int champ = 0;
for(int j = 0; j< nIndiv;j++)
if (FitnessC[j] < FitnessC[champ])   champ = j;
    //The array Order classifies individuals by fitness by
```

```
//equal or decreasing order.
Order[i] = champ;
FitnessC[champ] = stringLength;
}
   }
 }

 //Individuals are sorted by fitness
 private   boolean sorting(String subGoal)
  {
   fitnessPop();
   sort();
   int m = 1;
   if (gen > 10) m = 10;
   if (gen > 100) m = 100;
   if (gen > 100) m = 100;
   if (gen > 1000) m = 1000;
   if (gen > 10000) m = 10000;
   if (gen % m == 0)
{
 w.news.append("Gen = " +gen +
     " The best  =  " + Individual[Order[0] ] + "\n");
}
   if (problem == false)  report1();
   return problem;
  }

  // The new generation is made
  private    void newGeneration()
   {
    int   i=0;
    //The fittest individuals are reproduced
    // numbCopies times
    int numbCopies = nIndiv;
    int   k=0;  //rank, 0 is the highest

    while (i<nIndiv)
     {
      int counter = numbCopies;
      while ((counter>0) & (i<nIndiv))
```

```
      {
       Individual2[i] = Individual[Order[k]];
       i++;
       counter= counter-1;
      }
      k++;
     }
    for(int j = 0; j<nIndiv; j++)
     {
      Individual[j] = Individual2[j];
     }
  }


  //mutRate is defined per char per generation
  //Exception for the sake of ghost threads
  private   String change(String a)
  throws NullPointerException
  {
   for(int l = 0; l < stringLength; l++)
   {
double s = r.nextDouble();
if (s < mutRate)
{
  if (a.length()==1) return ""+randomChar();
 else
 {
  //In case of a ghost
 if ( (l<0) || (l > a.length()-1))
 {
  problem = false;
  return a;
 }
      a = a.substring(0, l) + randomChar()
                     +  a.substring(l + 1);

 }
}
   }
   return a;
```

```
  }

  // Individuals are mutated except the champ
  private   void Mutation()
  {
   for(int i=0; i<nIndiv; i++)
if (i != Champ) Individual[i] = change(Individual[i] );
  }

  //The champ is cloned
  //One random char is added to each individual
  private   void  cloning()
  {
   for (int i = 0; i<nIndiv ; i++)
{
 Individual[i] = champString ;
 Individual[i] = Individual[i] + randomChar();
}
  }

  private   void title()
  {
   w.news.setText("");
   w.news.append("Goal Phrase = "  + bigGoal + "\n");
   w.news.append("Length of goal = " + bigGoalLength+ "\n");
   w.news.append("Population size = " +  nIndiv + "\n");
   w.news.append("Mutation rate/char/gen = "
                            + w.mutRate + "\n");
   w.news.append("Mutation + selection");
   if (recomb) w.news.append(" + recombination");
   w.news.append( "\n");
  }

  // This method  looks for a partner
  // different than l.
  private   int  chooseInd(int l)
  {
   // done is a flag that says when
   // we succeed:
   int done = 0;
```

```
   int number=l;
   {
while (done == 0)
 {
   number = r.nextInt(nIndiv);
   if (number ==   l)
 done = 0;
   else done = 1;
  }
}
   return number;
}

  //Individuals l and k recombine
  private   void rec(int l, int k)
  {
String s = Individual[l];
String t = Individual[k];
String  t1= t;
String  t2 = "";
    String  s1= s;
    String  s2 = "";
int m= s.length();
    if (m>2)
     {
      int m1 = r.nextInt(m-1)+2 ;
  s1= s.substring(0,m1);
  s2= s.substring( m1);
  t1= t.substring(0,m1);
  t2= t.substring( m1);
    }
    //Recombination
    Individual[l]=s1.concat(t2);
    Individual[k]=t1.concat(s2);
  }

  //Recombination is pairwaise
  //and produce pairs.
  private   void recombination()
  {
```

```
for(int i = 0; i< nIndiv; i++)
 {
  int l = i;
  int k = chooseInd(l);
  rec(l,k);
 }
  }


  //Given a bigGoal, it is guessed by evolution
  private   void  evolSolution()
  {

   //A initial population is created
   stringLength = 1;
   createPop(stringLength);
   //Big problem is divided in a ladder of subproblems
   for(int i = 1; i<= bigGoalLength; i++)
   {
stringLength = i;
problem = true;
subGoal = bigGoal.substring(0, i);
subGoalNumber = i;
w.news.append("subGoal " + i+ " = " +subGoal+ "\n");
//Individuals are sorted by length
problem = sorting(subGoal);
while  ( (problem)
& ((Thread.currentThread() == auxThread)
            & (auxThread != null))  )
     {
  //The best individuals reproduce
  newGeneration();
  //The new population is subjected to mutation
  Mutation();
  //Recombination is chosen by the User
  if (recomb) recombination();
  gen++;
  problem = sorting(subGoal);
     }
if (i < w.nLength)
```

```
 {
  cloning();
  gen++;
 }
}
   w.news.append("TASK COMPLETE  \n");
   }

 public   void solveTask(  )
 {
  w.news.setText("");
  title();
  gen = 1;
  while ((problem) & ((Thread.currentThread() == auxThread)
         & (auxThread != null))  )
   evolSolution(  );
 }

}//End of WeaselPro

}//End of main class
```

108 ***Exercise*** *Play with the program. Observe the code and verbalize its design and functioning.*

109 ***Exercise*** *Design a suitable set of experiments to decide whether or not the next proposition are true or at least plausible:*

1. *Guessing a phrase is a problem that is solved by evolution almost immediately.*
2. *A suitable choice of parameters is necessary for evolution to guess a phrase in linear time in the length of the phrase.*
3. *There are assignations of parameters that convert evolution in a nightmare possibly not better than chance.*
4. *In spite of all problems, evolution explains biological complexity in a reliable way for science.*

110 ***Evolution in the XX century***

One usually has not enough patience to wait for chance to guess a phrase with more than 7 letters. Nevertheless, there are suitable combination of parameters for which a phrase with 100 chars is guessed by evolution almost immediately if one uses the previous program.

The awaken feeling of triumphalism is overwhelming. And the science of the XX century was dominated by such a feeling: the international literature considered that to question the power of evolution to explain biological complexity was not a sane idea.

Everything seems to indicate by 2009 that the confidence of science in the belief that evolution is all to living beings can only be reinforced during the XXI century.

111 ***Evolution in the EvolJava Community***

The fundamental point for us is that evolution is a reality. And as such, it can be studied by everybody regardless of his or her religious or philosophical affiliation. Two aims are particularly important:

- To push evolution toward unthinkable limits of sophistication with astonishing applications.
- To clearly understand why evolution is weak in the market of human ideologies. In particular, to clarify the relation of evolution with complexity, a matter that is called for everywhere at every moment. We must begin, of course, with assessing our program Shakespeare as a model of complexity.

In regard with the first aim, our former program Shakespeare is presented here as a challenge: would you design a program that uses evolution to guess a phrase faster and with less memory resources than our program once its parameters have been optimally tuned?

To illustrate how one deals with the second aim, please, get a look at the next chapter.

7.4 Conclusion

Chance can explain everything but a price must be paid: one must assume that to explain the complexity of life, chance guessed the correct information in the very first trials and event that might had happened but with an exceedingly low probability. By contrast, one can attune evolution to guess the information contained in a phrase or in a DNA segment with a price to be paid that seems to be more than

reasonable. Facts like this encourages science to teach that evolution (and its natural extensions to cover the origin of life and of evolution) is the one and only cause of our existence. Our work in the EvolJava Community is polarized by the fact that evolution is a reality: we consider that it is a duty to use personal **convictions** as light to design enlightening simulations of evolution together with experiments and applications and as force to pursue your projects until the end. Very important for us is to clarify the exact relation of evolution with complexity, a matter that will given another round in the next chapter.

Chapter 8

Evolution vs complexity

`Evolution is limited, complexity is not.`

112 *Introduction*

Evolution is a reality and as such it must be limited at least in one aspect. But, What is strong enough to limit evolution? What are the limits of evolution? Does actual life resides in within those bounds or is it alien to the possibilities of evolution? If we choose to explain actual life by evolution, can we decide if evolution was used as a tool by an intelligent being or if instead it ran autonomously?

We find that these questions are very interesting for many persons and that answers are not given if not in the presence of a personal **conviction** that cannot be plainly defended using solely experimental findings. Dear Reader, most possibly you are used to read in the international literature that evolution is a clean scientific result. Not indeed: the theory of evolution is a project that is sustained by the **conviction** of very smart and laborious people but not by science itself because no **belief** can be sustained by science. Quite to the contrary, the only **purpose of science** is to study and elaborate beliefs so that they could be tested. This is what we do here.

We can understand *The Origin of Species* as a support to the **conviction** of **Darwin** that God created some few and incipient forms of life with the possibility to evolve and that since then evolution ran autonomously. Let us observe that he expresses the **conviction** that he had at the moment when he published his work very succinctly and very clearly and at the very end of his book:

> *There is grandeur in this view of life, with its several powers, having been originally breathed into a few forms or into one; and that, whilst*

> *this planet has gone cycling on according to the fixed law of gravity, from so simple a beginning endless forms most beautiful and most wonderful have been, and are being, evolved.*

We recognize here a modus operandi that is good for everybody: **convictions** shed light to formulate appealing insights and give force to pursue them until the end even if one must battle against illness, as was his case. Our conclusion is simple: **convictions** create targets and give fuel to work and that is why we welcome clear and succinct formulations of convictions in our Community (we dedicated the first chapter to argue that the possibility that humans have of falling pray of convictions is biology given: it is a wonderful biological possibility, it is a miracle!).

How does **Darwin** support his **conviction**? The work of Darwin contains two main points:

- Data showing that all around the world one can find motives for thinking of evolution.
- Some incomplete theorization trying to imagine how evolution can be made into a powerful theory that explains the forms of life in all its complexity and grace beginning with an initial inception.

One of the puzzles that his theory was to solved was the problem of the extreme complexity of living beings, which is trans-culturally perceived by just looking at an eye, either of a fly or of a mammal. Darwin is plainly aware of the problems created by complexity and so he works out gradualism: big problems are not resolved in toto, rather complexity is gathered by small change.

To see with our own eyes how this goes one, we programmed chance and evolution in the previous chapter to match a given phrase: chance was not given the possibility to divide the goal phrase in suitable subgoals and, of course, it fell short of time. By contrast, our program Shakespeare precisely divides the goal in a succession of goals that were orderly conquered and defended by evolution: no string seems to be long enough to be consider that it must be outside of the the span of evolution.

Of course, the problem of matching a given phrase was not made to kill time. Instead, the problem of guessing a given phrase was for us a model of the problem of explaining extant complexity, concretely, the complexity of the DNA that encodes for our bodies. So, it could seem that we have the option to claim that evolution is the correct explanation of our existence. Nevertheless, we have as yet no reason to make such a claim.

Why?

The answer is this: to the best of our good will we have made just the work of the apprentice clown that tries to imitate a given personage. While we appreciate the works of clowns, we all know that the true personage is the aim of our interest because he or she is able to create and recreate himself or herself at each moment. So, we would dream of a program to simulate Shakespeare with the possibility to compose new artworks that would wonder every one. Nevertheless, such a program is too heavy for us and we choose to study here a simple and natural problem that relates languages and reality.

In our program Shakespeare we consider the language us a succession of symbols. Such a description of language is deceptive because of its extreme simplicity: the chosen phrase from Shakespeare can be replaced by a succession of symbols that can be composed with a method less than 10 lines long. What else must be included in our characterization of human languages that could grasp in a simple but definitive way the difficulty of creating an artwork?

We need a caricature of complexity that must be both simple and effective. Our good news is that mathematicians have worked thoroughly this point and have produced many possible candidates. We choose one that is already at the center of all their studies. Our caricature of complexity is *SAT* that is the *Satisfaction problem of mathematical logic*. SAT is an elementary test to know whether or not you say the truth given that we have no time nor money to go behind facts to unveil the truth. All we do is to help you to express yourself and to relate one thing with another. We annotate all what you say and then we come back to look for possible contradictions. As we see, our approach originates a problem that maybe solved automatically.

Let us recall that one of our aims is to rationalize the human soul with its capability of adopting **convictions**. Given that the human heart is a fabric of convictions, we think that most possibly the neural machinery to fabricate then is indeed complex but is instantiated with very simple programs. Let us contend that SAT is one of those facilities. In fact, SAT has more than 3000 interpretations that cover different situations of ordinary life, science and technology. Even small children already know and enjoy the art of speaking without falling in contradictions and, by age 5, all children pretend to be professional cheaters. So, we hope that a deep study of SAT will be of great sociological value. Our work here is directed to arm a given conviction:

113 ***Purpose*** *We show some results that are aligned along the following* ***conviction****: complexity is bold, strong and ubiquitous enough to challenge evolution. To support our* ***conviction****, we include a study about how good is evolution to solve a classical problem of mathematical logic known as SAT (Satisfaction problem). This problem models an ordinary characteristic of human languages: they include*

the possibility to convey information in regard with the situation in which many prerequisites that work in ensemble are necessary to achieve a desired output or function, for example, to express oneself without falling into contradictions.

8.1 What is complexity?

Complexity has many flavors. Our choice is defined by the need we have of explaining life using a very reduced amount of resources, time and matter. Therefore, we orient complexity to somehow represent a measure of the needed resources to achieve a task. Our approach can be grasped at once if we consider in hindsight the problem of matching a succession of symbols that was consider in the last chapter.

114 ***Model problem Shakespeare***

Our **model problem Shakespeare** is to guess a given phrase. For our model problem Shakespeare, the larger is the phrase to be matched or the number of letters in the used alphabet, the longer one must work to guess it. So, complexity must be defined as a function of phrase length and the cardinality (number of letters) of the alphabet. Concretely:

If the alphabet has a letters and a phrase has n chars taken at random, the expected number of trials needed by chance to guess it is a^n: chance works in the exponential regime. But, with appropriate election of parameters, and dividing the goal in subgoals, evolution needs around n generations. So, evolution works in the linear regime to achieve the same goal.

We see that needed resources strongly depends on the used method to solve the task. So, evolution can turn into an possible explanation of biological complexity if the following three points are successfully fulfilled:

1. To show that the essential model problems of biology can be solved by human beings in the linear regime.

2. To argue that natural evolution is not less intelligent than us: if we can work in the linear regime, natural evolution also can and even since its inception or appearing.

3. To guarantee that the linear regime is ordinary in natural evolution or at least that it can happen with a significantly high probability, say greater than 0.05.

Let us consider the Shakespeare model problem. Does the previous chapter fulfills the program above for this problem? Let us see:

1. We found that with appropriate choice of parameters, our simulation of evolution successfully works in the linear regime.

2. The project of devising DNA computers is well advanced and we have no trouble in believing that these computers also can run our algorithm and solve the problem in a linear regime.

3. By playing sometime with our program D107, one might conclude that the linear regime is found when mutation is kept around 0.1 and the population size must be around 500. Both parameters can be modified although the system is more sensible to a change in the mutation rate. For instance, if mutation rates increases pass 0.5 the linear regime is lost. Recombination seems to be unimportant for the chosen values of mutation rate and population size. In summary, we feel free to consider that the liner regime is highly probable.

We consider therefore that evolution provides a reliable solution to the model problem Shakespeare while a better optimization is not excluded. For instance, one must investigate the role of recombination for all values of mutation and of population size. In particular, what does happen when the pop size is lowered to 2?

Actually, simple and interesting questions about anyone problem seem to be naturally formulated but the demanded quantity of work to study them surpasses the possibilities of one person working with a single desktop. The solution to this problem is provided by the eJGrid, which will come next. Let us imagine how this institution would work to explore our Shakespeare model problem: interested persons in the study of this problem are joined into a team. For each member of the team, we throw a set of random numbers that assign values to the parameters of mutation, pop size and recombination. Each person studies the behavior of the system for the assigned to him or her values of parameters and reports results to the manager and some other witnesses chosen among the members of the team. The manager elaborates a global inform that is presented to the team for discussion. All this can be done through the e-mail service but automatic computer to computer communication might also be implemented.

The importance of the model problem Shakespeare resides in the fact that we get acquainted by direct contact with the sort of things that were brought forth by experts to sustain the theory of evolution by the end of the XX century.

In this model, the language is seen as a succession of symbols and nothing else. The very fact that the phrase of Shakespeare can be changed by a string of symbols that can be synthesized by a method less than 10 lines long shows that the creativity of Shakespeare is indeed not used in this model. Henceforth, we cannot

pretend that our model problem Shakespeare could sustain the evolutionary theory against skeptics.

This does not mean that the theory of evolution is false. This means that if we want a robust defense or refutation of the evolutionary theory then we must work it out by ourselves. This seems to be very hard. At the meantime, let us pay attention to the fact that to dress **convictions** in logical, rational argumentation is harder than we can imagine. Nevertheless, one always must realize that dismantling logical reasons does not cause harm to a **conviction** instead its followers are challenged to produce more fine argumentation else to drawback into resentment and loneliness.

So, we ask: what must we include in a model of language to capture the difficulty, the complexity of creating a good literary document? We know of course that languages deal with semantics, with the fact that strings of symbols are used to denote things and actions of the world. It seems that the study of semantics is a complex affair. Fortunately, mathematicians have furnished us with a model problem, whose study is indeed at the level of our present possibilities, it is SAT or satisfaction problem.

8.2 SAT as a model of complexity

Human languages are very rich, very complex but include a part that lends itself to easy systematization and that is directly related to complexity.

115 ***From language to SAT***

The proposition '3 is even' is false, while '4 is even' is true. In general, any expression that can be classified as true or false given a reference set is called a **proposition**.

Let us consider the expression over integer numbers 'p(x) = x is even'. This expression is true for $x = 8$, i.e., $p(8)$ is true, for $x = 7$, $p(7)$ is false, and for each integer number it is true else false. So, 'p(x) = x is even' is not a proposition but becomes a proposition if we instantiate x, i.e., if we replace 'x' by a given integer number. An expression of this type over a given universal set is called a **predicate**.

A language composed of predicates is appropriate for describing events that come classified in one of two classes. Thus, it is the more simple setting under which one can study the world and make science.

The use of predicates is usual for humans, intelligent beings. To study the possibilities of natural evolution, we prefer to restrict ourselves to propositions, which are appropriate to describe concrete events of nature.

One can generate **formulas** or compound propositions beginning from elementary propositions or variables or from other formulas by the use of logic operators:

- The NOT or negation operator: which takes one proposition p and produces the proposition denoted $NOT(p)$ or $\sim p$ that is false whenever p is true and is true whenever p is false.
- The AND operator takes two propositions p and q and produces a third one, noted $AND(p,q)$ or $p \wedge q$ that is true if and only both of p **and** q are true.
- The OR operator takes two formulas p and q and produces a third one, denoted $OR(p,q)$ or $p \vee q$ that is true if one of p **or** q is true and false if both are false.

To go further, let us clarify that the NOT operator is mono-ary, i.e., it takes a proposition and produces another, while the AND and OR operators are binary, i.e., take two propositions and produce one. Nevertheless, AND and OR can be extended very easily to more than two propositions, say $p \vee q \vee r$ can be taken as $p \vee (q \vee r)$ or $(p \vee q) \vee r$ and in both cases they compute to the same final output for every truth assignment of the variables.

One can see that with the set of operators $\{NOT, AND, OR\}$, one has more than enough to generate all possible operators that take one, two or any number of inputs to produce one compound proposition. For that reason, we restrict ourselves to these operators.

One can always find an assignment of truth values for p such that $NOT(p)$ could be true: just assign FALSE to p. Also, we can assign TRUE to both p and q to make $AND(p,q) = p \wedge q$ true. If we assign TRUE to p and FALSE to q, $OR(p,q) = p \vee q$ becomes true. But one cannot find an assignment of truth values for $p \wedge \sim p = AND(p, NOT(p))$ such that this compound formula could become true, because we deny the possibility that a proposition could be true and false at the same time.

We say that a proposition is **satisfiable** when we can show a truth assignment to its variables that makes the proposition true. We also say that the truth assignment satisfies the proposition or the formula. Example: $p \wedge (q \vee (\sim r)) = AND(p, OR(q, NOT(r)))$ is true if p is true and q is true, while r could be true or false. So, this formula is satisfiable. The proposition $OR(r, NOT(r))$ is true for any truth value of r, so it is satisfiable. $AND(r, NOT(r))$ is not.

SAT or the satisfiability problem of mathematical logic consists in studying the assignment of truth values to the variables of a proposition with the purpose of making it true. SAT proposes a question in regard with a given proposition: Does this proposition is satisfiable? To answer YES you must show a satisfying assignment while to answer NOT you must prove that all assignments evaluate to false.

To be satisfiable is a restriction imposed over propositions. We feel that this restriction is directly related to complexity because it amounts to the requirement to speak without falling into inner contradictions and we all know by direct experience that this is indeed difficult.

In mathematics, to decide whether a formula is satisfiable or not, one must try out every possible truth assignment. But to consider the possibilities of evolution, we will consider just a given number of random trials, because evolution does not have a systematic mean to execute every possible truth assignment.

Now, let us ask ourselves a fundamental question: In which way is SAT fundamentally related to evolution? In other words: Is evolution fundamentally restricted? The answer is by no means obvious, for suppose that one goes to a trial at the court. If we are ready to say the truth and only the truth, we never will commit contradictions. So, to pretend that evolution could commit contradictions seems to be equivalent to pretend that evolution could lie. That is unthinkable.

Nevertheless, there is another insight that looks at evolution as fundamentally restricted by a type of constrains that is well captured by SAT. To fix ideas, let us consider a poisonous snake: it must produce an inhibitor of, say, the transmission of chemical signals among synapses in a potential enemy **but not** in its own body. So, the arising of a poisonous snake is an inherently constrained problem. Now, is this situation fundamentally general and inherent to evolution and life? Yes, it is. To see that, it is enough to consider the problem of digestion: one must digest food **but not** oneself. To solve this constraint, the spleen synthesizes digestive enzymes that are inactive in the spleen but after deletion of some amino acids in the intestine, it becomes active and can digest food. Or let us consider the cell, in which everything must be recycled **but not** at the same time, so defective enzymes must be recognized and digested.

Let us see now why SAT is a very good representation of the constraints that life must face at the most fundamental level:

In general, everything that happens in the cell can also happen in the inanimate matter only that there is no coherent trend among diverse properties and the veloc-

ity of the process is not enough to counterbalance self-poisoning and dissipation. Poisoning comes from thermodynamics: the very products of a chemical reaction inhibits the reaction if these are accumulated en great enough quantities. Therefore, life is impossible without excretory units. Dissipation arises because the chemical subunits of life are complex substances that can be destroyed by water, radiation and heat.

Now, coherence and velocity are the natural characteristic of catalytic ensembles. From the stand point of SAT this reads: do this and that and this other and....this but not that. The negative constraints arise from allosteric regulation as follows: many enzymes are sensible to some other molecules that can stimulate then to proceed faster or slower. This capability is used in the cell to form highly coordinated ensembles that keep products at precisely those high concentrations that allow reactions to flow **but without** causing harm to other paths. In fact, as the number of enzymes grows, the probability of undesired allosteric interference through products augments.

We see that the brand of life is **complexity**. This easily explains why all dreams of created in the lab life end in unfulfilled promises. The only estimation of the minimum complexity that enables life comes from the smallest parasite, the *Mycoplasma genitalium*, whose genome contains 470 genes and 580,070 base pairs (Fraser et al, 1995). Laboratory estimation of the minimum number of the essential to life genes rendered 256 (Mushegian and Koonin, 1996), other 265 to 350 (Hutchison et al, 1999), and other 256 (Pereyre et al, 2009). This discussion is difficult and involves such simple a question as what a gene is. Anyway, it seems that nobody will discuss that the **minimum genome** has more than 200 genes and more than 200000 base pairs.

We conclude that life, such as we know it, is fundamentally restricted by self-constrains that can be expressed verbally by expressions of the sort *do this but not that*. That is why SAT is good to study the possibilities of evolution ab initio: from the stand point of SAT, evolution is a diffusive process in logical space restricted to satisfiable propositions. A lethal mutation could be seen as a contradiction of evolution.

116 ***How expensive is SAT***

Our preoccupation is to explain our existence with available resources but without falling into the exponential regime. The model problem Shakespeare showed us that if this problem were all to life, evolution would be a correct and nice explanation of it: evolution breaks the exponential regime replacing it with a linear one. But we just discovered that a more fair representation of of biological processes and of the world in general is given by SAT.

SAT is related with the problem of expressing oneself without falling into contradictions while trying to say something interesting. Everyone knows by personal experience that this is not at all an easy task, so we could expect that SAT demands huge amounts of computing resources to be solved.

Let us see how intriguing is the problem of accurately calculating the cost of SAT.

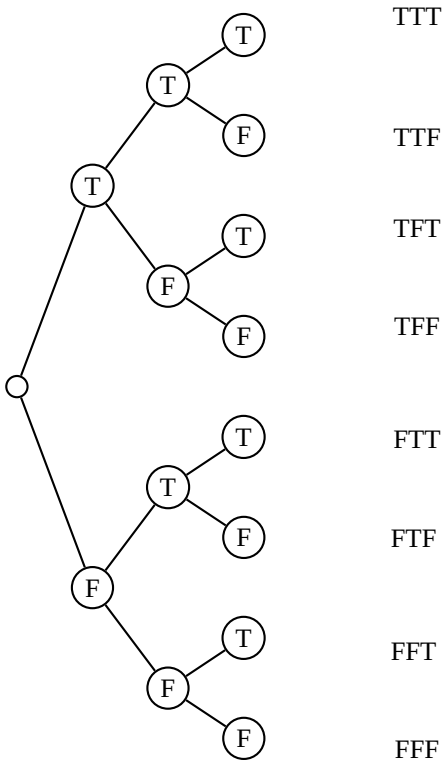


Figure 8.0. All 2^3 valuations of a set of 3 boolean variables are associated with a tree that have 2^3 leaves. The valuation TFT associates True to the first variable, False to the second and True to the third one. Each valuation can be built in 3 steps.

To fix ideas,let us consider that we deal with a formula with 3 variables, for instance $(\sim p \vee (p \vee r) \wedge (\sim q \wedge (p \wedge r))$. To decide SAT over this proposition, we must evaluate all possible truth assignments. The first step is to generate all possible truth assignments as in the tree of the accompanying graphic.

That tree is built just by posing at the root the initialization symbol, from which two branches leave, one for False, one for True, the two possible values of the first

variable. Next, from each new node, two branches leave for the two possible values of truth of the second variable. And so on.

Let us observe that this tree produces both a mean of listing all possible assignments (printed at the right side) and a systematic form of using parallelism to generate all possibilities. Let us mention one implication and one intrigue. The conclusion is that the very work of listing all truth assignments already falls in the exponential regime, 2^n, where n is the number of boolean variables. This means that if we work with a serial processor, we fall immediately in the exponential regime.

The intrigue arises because the horizontal direction in the graphic represents serial computations, one after another, while the vertical direction represents parallelism, the possibility to carry computations using different processors at different places. This fact is very interesting because evolution works at many fronts at the same time, so, parallelism seems at first sight to be a close relative of evolution. Now, each branch in the tree has n nodes, so the tree seems to say that it must be possible to break the exponential regime of serial computers to a linear one if only we use a parallel computer with many processors.

But, how many?

If we give to the parallel computer the architecture given by the three, we need 2^n processors, one for each terminal leaf. So, this architecture interchanges an exponential number of computations that must be done serially by an exponential number of processors that must work in the temporal linear regime. But what does happen if we consider all possible architectures? And what if we include dynamic architectures?

As we see, all this is very intriguing, but for now, we must recognize that the problem of generating a complete set of truth assignments is solved by us in the exponential regime, be it in serial operations or in parallel processors.

If we have a mean of generating all truth assignments, the next step to decide SAT is to replace the truth assignments into the chosen formula. How much does this cost?

The good news is that any formula can also be represented by a tree. Let us illustrate it with a very simple example:

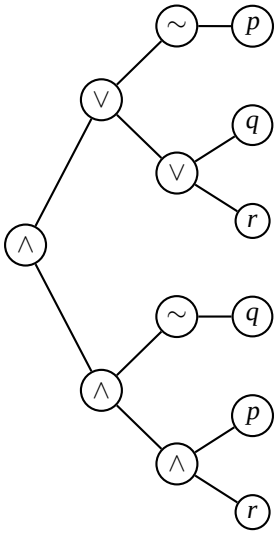


Figure 8.1. This tree represents the formula $(\sim p \vee (p \vee r) \wedge (\sim q \wedge (p \wedge r))$.

One gets the appreciation that this tree is not as complex as that the previous one, so we will dispense for it.

We are left with the load of the exponential regime be it in the number of serial computations or in the number of processors to be used. Everything else is a nice speculation that needs a lot of work and of expertise to be converted in an academic study. In this regard, let us see now a fact that is really surprising.

117 ***What do you say?***

Our logical propositions represent possible sentences of human languages. Say, the proposition $(p \wedge r) \wedge (\sim q \vee p)$ corresponds to the sentence *John wins and smiles and (he is not loved by his girlfriend or he wins)*. As we see, one can get confused by the meaning of a given sentence. Fortunately, a clarifying algorithm exists that reformulates a sentence in clear terms. Let us learn how that algorithm functions over a concrete example, precisely over the aforementioned sentence.

The semantic value of a proposition is given by the form as the proposition gets true else false as a reaction to a given evaluation with suitable truth assignments.

So, we pose in a table every possible form of evaluation and we get a truth table. If the proposition is always false for each truth assignment, it is equivalent to the proposition $p \wedge \sim p$ that is a contradiction. The problem becomes more interesting if the proposition or formula is satisfiable, i.e., if there exists at least a truth assignment that evaluates the proposition to true. What do we have in our case? The truth table follows:

Truth table			
p	q	r	$(p \wedge r) \wedge (\sim q \vee p)$
T	T	T	T
T	T	F	F
T	F	T	T
T	F	F	F
F	T	T	F
F	T	F	F
F	F	T	F
F	F	F	F

Next, we take those lines of the truth table that output true, the first and third ones. The truth values for p, q, r are in the first line TTT, and TFT in the third. The value TTT generates the proposition $(p \wedge q \wedge r)$ while TFT generates $(p \wedge \sim q \wedge r)$. Because we have two channels that produce true as output, we join them with and OR operator: $(p \wedge q \wedge r) \vee (p \wedge \sim q \wedge r)$. This means that the proposition or formula $(p \wedge r) \wedge (\sim q \vee p)$ has exactly the same truth table as $(p \wedge q \wedge r) \vee (p \wedge \sim q \wedge r)$, a fact that one can verify over the corresponding truth table:

Truth table			
p	q	r	$(p \wedge q \wedge r) \vee (p \wedge \sim q \wedge r)$
T	T	T	T
T	T	F	F
T	F	T	T
T	F	F	F
F	T	T	F
F	T	F	F
F	F	T	F
F	F	F	F

We say that $(p \wedge r) \wedge (\sim q \vee p)$ and $(p \wedge q \wedge r) \vee (p \wedge \sim q \wedge r)$ are equivalent or semantically equivalent because their truth tables coincide.

Let us verbalize the proposition $(p \wedge q \wedge r) \vee (p \wedge \sim q \wedge r)$ to see how much we have gained in clarity: John wins and his girlfriend loves him and he smiles, or, he wins and his girl friend does not love him and he smiles. The new sentences is so clear that we can see at once that the girlfriend of John causes no effect over the truth table and that, therefore, we can simplify the original sentence to the final expression $p \wedge r$: John wins and smiles. Please, over the truth table for the three variables, p, q, r, with eight rows, calculate the truth values of $p \wedge r$ to verify that we have a new table that is equivalent to the very original one.

A proposition of the form $(p \wedge q \wedge r) \vee (p \wedge \sim q \wedge r)$ is, by definition, in **normal disjunctive form**: it is an OR package of AND propositions, which contain the variables, negated or not, joined by the AND operator. Each AND string is called a clause. The normal disjunctive form is full when every clause contains all variables of the discourse. The formula $(p \wedge q \wedge r) \vee (p \wedge \sim q \wedge r)$ is in normal disjunctive full form, has two clauses over three variables.

118 ***Oracles***

We have shown that thanks to the truth table every satisfiable proposition is equivalent to a given disjunctive normal full form. The converse is also true: any disjunctive normal full form is always satisfiable: just take the first clause, and compose a truth assignment by instinct: true for a variable when it is not negated and false when it is negated. In that way the first clause gets true and with it the whole proposition. For instance, if the first clause is $p \wedge \sim q \wedge r \sim s$ then we can give the assignment $TFTF$ and the proposition becomes automatically true. Since this procedure can be repeated over any formula that has at least one satisfying assignment, we can conclude: the set of satisfiable propositions and the set of propositions in disjunctive normal full form coincide.

We know now how to associate a disjunctive form to any satisfiable proposition. The disjunctive form can be given a satisfying assignment all at once by just looking at it. In other words, we have converted SAT in a super-simple problem.

But, have we kill the exponential regime that we associated to SAT in the previous paragraphs?

No, by no means. The reason is that we was able to reformulate SAT is an obvious form only because we made a truth table for the original proposition. But a truth table has 2^n rows, where n is the number of variables. This means that the construction of a truth table is a task that to the best of our understanding belongs in the exponential regime.

In short, SAT can be reduced to an obvious problem if one already solves a helping problem that belongs in the exponential regime. If we had a computer that could immediately make for us the truth table we would say that we have an **oracle**. This allows us to reformulate what we have done: we have shown that an oracle has the power to convert SAT into a problem with an obvious immediate solution. Our only experiment with oracles has shown us that oracles that solve SAT could demand and exponential cost in the number of variables.

As a challenge for us, we see that the truth table transforms a maybe unreadable proposition into a very clear one, so clear that it can be solved by mere inspection killing the complexity of SAT. But it seems that the cost of solving SAT does not disappear but it is absorbed by the process that makes the truth table, by the oracle, which must deal with a problem that generates a work that is possibly in the exponential regime. This is situation seems to be very general: if you want to clarify something or to express yourself very clearly, you must do a lot of work, a tremendous effort that must be dressed in a lot of expertise. That is why we like to emphasize the importance for **convictions**: they give fuel to go that far and more.

Strong warning: the scientific literature of the XX century used to assume that evolution is an oracle that solves all problems. Anymore: evolution (in the hands of an Expert) can solve anyone problem but a price in computation must be paid, and it could amount to an exponential regime.

8.3 Evolution vs the disjunctive subclass

How does evolution deal with problems that have an obvious solution? To explore this question, let us challenge evolution with the disjunctive family.

119 ***Our simulation***

The next code uses evolution to find a truth assignment that evaluates a given disjunctive form to true. We use full forms, i.e., for a given number of variables, all variables appear in every clause. Our notation is defined by the next correspondence:

- Clauses are in within parentheses.
- Clauses are separated by OR.
- OR is surrounded by one space from each side.

- Variables are denoted by a small w.
- Variables are known by their position.
- All clauses have the same number of variables.
- Write OR in place of or.
- Write A instead or AND.
- Write N instead of Not.

Example:

(wAw) OR (NwAw) OR (wANw)
corresponds to
$(p \wedge q) \vee (\sim p \wedge q) \vee (p \wedge \sim q)$
The truth assignment 10 assigns true to the first variable and false to the second.

The program outputs the number of necessary trials to find the first truth assignment that satisfies the form. One can see that randomness solves this same problem in the exponential regime (see volume 3 of this series). How does evolution behaves?

Warning: Restricting SAT to the family of disjunctive forms originates a problem that can be solved in the linear regime by using the very same idea that was employed in our model problem Shakespeare: divide, conquer and defend. We do not use that idea here because it cannot be applied to the generality of formulaes and because we want to see what evolution gives from itself when it is not helped by the Developer.

```
/* Program D119, Disjunctive.
 *
 * We solve a subclass of SAT by means of evolution.
 * SAT is a model  that captures the idea
 * that complexity arises when one must do things
 * subject to  incompatible restrictions:
 * do this but not that.
 *
 * SAT or Satisfaction is the problem of
 * knowing whether or not a given logical proposition
 * is satisfiable, i.e. if there exists a truth
```

```
 * assignment that evaluates the proposition to true.
 *
 * Our subclass consists in the
 * disjunctive normal full forms with n variables and
 * k clauses. All clauses contain the n variables.
 * Variables might be negated or not.
 * The prob of negation of a variable is 0.5.
 * Variables are joined by and And operator.
 * Clauses are joined by an OR operator.
 *
 * We take disjunctive normal forms,
 * which are always satisfiable, and inquire
 * over the number of   individuals or trials
 * that are necessary
 * to get the first satisfying assignment.
 *
 * Random   equiprobable truth values are assigned
 * to the  variables and
 * the prop is evaluated to true else false.
 *
 */

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.TextArea;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.BitSet;
import java.util.Random;

import javax.swing.Box;
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JRadioButtonMenuItem;
```

```
import javax.swing.JSlider;
import javax.swing.JTextField;
import javax.swing.SwingConstants;
import javax.swing.SwingUtilities;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;



public class Disjunctive
{

 private static final long serialVersionUID = 1L;

 private static Disjunctive t = new Disjunctive();
 //Some relevant information is packed into a class. This
 //allows stability of memories against thread threats.
 static Job h = t.new Job();

 private static GUI w =  t.new GUI();
 private static  Search s = t.new  Search();
 //The JFrame which shows everything
 // is constructed with a title
 private static JFrame f =
             new JFrame("SAT: disjunctive family. " +
              "To modify parameters," +
                    " end with enter.");


 // This thread is used to enable the User to
 //terminate  lengthy tasks.
 private  Thread auxThread;

 public static void main(String[] args)
 {
  //WGUI is added to the JFrame f
  f.add( w, BorderLayout.CENTER );
  f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  //Dimensions of the JFrame
  f.setSize( 600, 670 );
```

```
  f.setVisible( true );
  f.add( s, BorderLayout.CENTER );
 }

//*****************************************
//****************JOB CLASS ****************
//*****************************************

//This auxiliary class unites some fundamental
//variables into an object
private  class Job
{
 Integer nClauses;
 Integer nVar;
 Integer popSize;
 Double mutRate;
 String prop;
 boolean show;
 boolean recomb;
 Long cost;
 Boolean makeGraphic;
 //Points for a graphic
 double PointsSim[][] = new double[100][2];
 double PointsExp[][] = new double[100][2];
 int Points[][] = new int[100][2];
 int nPoints;
 double xmax, ymax;
 Integer base = 2;
 int deltaY;

 //Mandatory initialization
 Job( )
{
 nClauses = 3;
 nVar = 2;
 popSize = 4;
 mutRate = 0.2;
 prop = "(wAw) OR (NwAw) OR (wANw)";
 show = false;
 recomb = false;
```

```
 makeGraphic = false;
 for(int i = 0; i < 100; i++)
  for(int j = 0; j < 2; j++)
   {
 PointsSim[i][j] = 0;
 PointsExp[i][j] = 0;
 Points[i][j] = 0;
}
}

 }//end of class Job

 //This class is responsible for finding
 //a satisfying truth assignment
 private class  Search  extends JPanel
 {

private static final long serialVersionUID = 1L;

// Boolean condition to continue else halt
  private    boolean go = true;


  //Constructor: mandatory initialization;
  Search()
  {

  }

 // Turn on  the random generator
    Random r  = new Random();

 //Number of clauses
 private   Integer nClauses;
 //Number of variables, could be as large as maxV.
 //Variables will be represented by numbers.
 private   Integer nVar;

 //Proposition in formation
 private   String a,prop;
```

```
//flags
private   boolean print = false;
private   boolean problem = true;



//A population of binary numbers is considered.
//The number of individuals is constant
private   int numbInd;
private   int maxNumbInd = 10000;
private   BitSet Ind1[]   = new BitSet[maxNumbInd];
private   BitSet Ind2[]   = new BitSet[maxNumbInd];
private   int FitnessInd[]
                              = new int[maxNumbInd];
private   int OrderInd[]  = new int[maxNumbInd];
private   int gen;
private   boolean recombination;

private   double mutationRate;
private long cost;




private   void setParameters( )
{
 //Number of clauses
 nClauses = 1;
 //Number of variables
 nVar = 1;
 //Number of propositions

 numbInd = h.popSize;
 recombination = h.recomb; //true = recombination is enabled
 mutationRate = h.mutRate;

}

//Not operators are inserted with prob 0.5
private    String negations( )
```

```
 {
 for(int j=0; j<a.length();j++)
   {
if (a.charAt(j)=='w') //
 {
  int d = r.nextInt(2);
  if (d == 0)
   a = a.substring(0, j)+ "f" + a.substring(j+1);
 }
   }
a = a.replace("f", "Nw");
  return a;
 }


 //Composes a random normal disjunctive full w-proposition
 //with nClauses clauses. The NOT operator is included
 //for each w with probability 0.5.
 private    String wProposition()
 {
  a = "(w";
  for(int i= 0; i < nClauses-1; i++)
  {
   for(int j = 0; j < nVar-1; j++)
     a = a.concat("Aw");
   a= a.concat(") OR (w");
  }
  for(int j = 0; j < nVar-1; j++)
     a = a.concat("Aw");
  a= a.concat(")");
  negations();
  return a;
 }

//Converts a BitSet value into a String
 private String decode(BitSet t)
{
String s= "";
for(int i =0; i < nVar; i++)
 {
```

```
if (t.get(i)==true) s = s + "T";
 else s = s + "F";
 }
return s;
}

//The proposition is evaluated
 private     String evaluation(BitSet s)
 {
  BitSet ss = s;
  prop = h.prop;
  if (print)
  {
   w.news.append("\n");
   w.news.append("Truth evaluation begins" + "\n");
   w.news.append("prop = " + h.prop + "\n");
   w.news.append("Truth assignment = " + decode(ss) + "\n");
  }
  int u = 0;
  for(int j=0; j<prop.length();j++)
   {
    if (prop.charAt(j)==’w’)
     {
      if (ss.get(u) == true)
   prop = prop.substring(0, j)+ "T" +
                                prop.substring(j+1);
      if (ss.get(u) == false)
   prop = prop.substring(0, j)+ "F" +
                                prop.substring(j+1);
  u = u +1;
  if (u == nVar) u=0;
 }
   }
  if (print)
  w.news.append("Evaluation   = " + prop + "\n");
   return prop;
 }

 //The proposition is evaluated until the end
 //Not operators are evaluated at the
```

```
 //very beginning for they apply over the next
 //to the right variable.
 private     String simplification(String b)
 {
  int counter = 0;
  int n =  b.length();
  b = b.trim();
  n =  b.length();
  while ( (b.length()>1) & (go) )
   {
 b=  b.replace("NT","F");
 b=  b.replace("NF","T");
 if (print)
 w.news.append("Truth value1 = " + b + "\n");
 b=  b.replace("TAT","T");
 b=  b.replace("TAF","F");
 b=  b.replace("FAT","F");
 b=  b.replace("FAF","F");

 if (print)
 w.news.append("Truth value2 = " + b + "\n");

 b=  b.replace("(T)","T");
 b=  b.replace("(F)","F");

 if (print)
 w.news.append("Truth value3 = " + b + "\n");

 b=  b.replace("T OR T","T");
 b=  b.replace("T OR F","T");
 b=  b.replace("F OR T","T");
 b=  b.replace("F OR F","F");
 if (print)
 w.news.append("Truth value4 = " + b + "\n");
 counter = counter +1;
if (counter == n)
{
w.news.append(
  "SOMETHING IS WRONG WITH THE PROPOSITION \n");
go = false;
```

```
break;
}
   }

  return b;
 }



 //Ind1 is a population of binary strings.
 //The BitSet class is used to instantiate
 //binary strings as individuals
 private   void initializePop()
 {
  //s is false everywhere
  BitSet s = new BitSet();

  for(int i = 0; i < numbInd; i++)
Ind1[i] = (BitSet) s.clone();
  //Reinitialization with random truth values.
  for(int i = 0; i < numbInd; i++)
  {
   for(int j = 0; j < nVar; j++)
{
   int l = r.nextInt(2);
   if (l == 1)
    Ind1[i].set(j); //the j-th bit is true
}
  }
 }




// Does an individual   satisfy  the target?
   private int assessInd(BitSet s)
 {
  String b = evaluation(s);
  b=simplification(b);
  int  fitness=0;
```

```
  if (b.equals("T")) fitness = 1;
  return fitness;

 }


// Individuals are sorted by fitness.
// the fittest is the first, with minimum error
private   boolean sortingInd()
 {
  int   Champ;

  //FitnessIndc[] is a copy of FitnessInd[]
  //used as workbench
  int FitnessIndc[];
  FitnessInd= new int[maxNumbInd];
  FitnessIndc= new int[maxNumbInd];
  for(int i = 0; i< numbInd; i++)
   {
    FitnessInd[i] =0;
    FitnessIndc[i] =0;
   }
  for(int i = 0; i< numbInd; i++)
  {
   FitnessInd[i] = assessInd(Ind1[i]);
   FitnessIndc[i] = FitnessInd[i];
  }
  //We sort individuals by fitness
  //0 is the minimum fitness
  for(int i = 0; i< numbInd;i++)
  {
   Champ = 0;
   for(int j = 0; j< numbInd;j++)
   if (FitnessIndc[j] > FitnessIndc[Champ]) Champ = j;
   //The array OrderInd orders individuals by  fitness:
   //The fittest is  zero
   OrderInd[i] = Champ;
   FitnessIndc[Champ] = 0;
  }
 // if Pattern has been guessed
```

```
 if (FitnessInd[OrderInd[0]] == 1)
 {
  problem = false;
  w.news.append(" The Proposition is satisfiable \n");
  w.news.append(" by truth assignment "
             + decode(Ind1[OrderInd[0]]) + "\n");
  prop = h.prop;
 }
 return problem;
 }

// The new generation is made
 private    void newGenerationInd()
 {
  int   i=0;
  //The fittest individuals are reproduced
  // numbCopies times
  int numbCopies = 10;
  int   k=0;  //rank, 0 is the highest
  while (i<numbInd)
  {
   int counter = numbCopies;
   while ((counter>0) & (i<numbInd))
   {
Ind2[i] = (BitSet) Ind1[OrderInd[k]].clone();
i++;
    counter= counter-1;
   }
  k++;
 }
 for(int j = 0; j<numbInd; j++)
  {
   Ind1[j] = (BitSet) Ind2[j].clone();
  }
 }


// Recombination is enabled
private   BitSet recInd(int l)
{
```

```
 //We choose a partner at random
 int k = r.nextInt(numbInd);
 //b is false everywhere
 BitSet b = new BitSet();
 //b is the recombinant of l and k
 for(int j = 0; j < nVar; j++)
{
   int m = r.nextInt(2);
   if (m == 1) //take the bit from l
    {
   if (Ind1[l].get(j)== true ) b.set(j);
    }
   if (m == 0)//take the bit from k
    {
   if (Ind1[k].get(j)== true ) b.set(j);
    }
}
 return b;
}


// The definitive population  is a recombinant one.
private   void recombinationInd()
{
 //Ind2 is a copy of Ind1
 for(int i=0; i<numbInd; i++)
 Ind2[i] =  (BitSet) Ind1[i].clone();
 //Ind2 is a recombinant version of Ind1
 for(int l=0; l<numbInd; l++)
 Ind2[l] = recInd(l );
 //Ind2 is transferred to Ind1
 for(int i=0; i<numbInd; i++)
 Ind1[i] = (BitSet) Ind2[i].clone() ;
}



// Individual  a is mutated at a random place.
  private   BitSet pointMutationInd(BitSet a)
  {
```

```
   int l = r.nextInt(nVar);
   a.flip(l);
   return a;
  }

 // Individuals are mutated
 private   void mutationInd()
 {
  for(int i=0; i<numbInd; i++)
   {
double p= r.nextDouble();
if (p < mutationRate)
      Ind1[i] = pointMutationInd(Ind1[i]) ;
   }
 }

 // Given a goal, it is guessed by evolution
 private   void  evolution()
 {
  //The new generation is produced
  //as copies of the fittest
  newGenerationInd();
  if (recombination) recombinationInd();
  //The new population is subjected to mutation
  mutationInd();
  gen++;
  problem = sortingInd();
 }

 public   void report1()
 {
  w.news.append(" Prop = " + prop + "\n");
  w.news.append(" nClauses = " + nClauses + "\n");
  w.news.append(" nVar = " + nVar + "\n");
  w.news.append(" Population size = " + numbInd + "\n");
  w.news.append(" Mutation rate per bit per gen = "
                          + h.mutRate + "\n" );
  w.news.append(" Recombination = "
          + h.recomb + "\n" );
 }
```

```
public   void report2()
{
 w.news.append(" Number of generations = " + gen + "\n");
 w.news.append(" Cost = " + h.cost + " trials. \n");
 w.news.append("\n");
 w.news.append("*****************" + "\n");
 w.news.append("*****************" + "\n");
 w.news.append("*****************" + "\n");

}


private void solveProblem(String prop)
{
 h.prop = prop;
 report1();
 problem = true;
 initializePop();
 while   ((problem) & (go))
  {
   print = h.show;
   evolution();
  }
 cost = numbInd*gen;
 h.cost = cost;
 report2();
 h.PointsSim[h.nVar][0] = h.nVar;
 h.PointsSim[h.nVar][1] = h.cost;
 h.nPoints = h.nVar;
}

private   void oneProblem()
{
 gen = 0;

 prop = wProposition();

 w.propTF.setText(prop);
 solveProblem(prop);
```

```
 }

//The ladder job is defined and solved
 private    void  work() throws InterruptedException
  {
   setParameters();
   nClauses = 1;
   h.nClauses = nClauses;
   nVar = 1;
   while  ((go) &   (Thread.currentThread() == auxThread))
{
 h.nVar = nVar;
     w.nClausesTF.setText(h.nClauses.toString());
     w.nVarTF.setText(h.nVar.toString());
     problem = true;
     oneProblem();
     problem = true;
     nVar = nVar +1;
   }
 }

 private void findMax()
 {
  h.xmax = 0;
  h.ymax = 0;
  //Max values are detected
  for(int i = 1; i < h.nPoints; i++)
   {
if (h.PointsSim[i][0] > h.xmax) h.xmax =  h.PointsSim[i][0];
if (h.PointsSim[i][1] > h.ymax) h.ymax =  h.PointsSim[i][1];
   }
  for(int i = 1; i < h.nPoints; i++)
  {
if (h.PointsExp[i][0] > h.xmax) h.xmax =  h.PointsExp[i][0];
if (h.PointsExp[i][1] > h.ymax) h.ymax =  h.PointsExp[i][1];
  }

 }

 //Data of a graphic is normalized to be placed
```

```
 //in a given rectangle
 private void rescale( double[][] PointsD)
 {

  //scale factor are defined
  double  fx;
  double  fy;
  fx = 240 / h.xmax;
  fy = -360 / h.ymax;

  int tx = 60;
  int ty = 565;
  //Scale formulae
  for(int i = 1; i < h.nPoints; i++)
  {
    h.Points[i][0] =  (int) (PointsD[i][0]*fx +tx);
    h.Points[i][1] =  (int)  (PointsD[i][1]*fy +ty);
  }
 }

 //The graphic is drawn
 private void draw(Graphics g )
 {
  for(int count = 1; count < h.nPoints-1; count++)
   g.drawLine(h.Points[count][0], h.Points[count][1]+h.deltaY,
   h.Points[count+1][0],h.Points[count+1][1]+h.deltaY);
 }

 private void drawer(Graphics g )
 {
//An exponential function serves as reference
  for(int i = 1; i < h.nPoints; i++)
   {
h.PointsExp[i][0] =  i;
h.PointsExp[i][1] =  Math.pow(h.base, i);
   }
  findMax();
  h.deltaY=30;
  g.drawRect( 50,185+h.deltaY, 295,390);
  g.setColor(Color.BLACK);
```

```
  g.fillRect(50,185+h.deltaY, 295,390);
  rescale(h.PointsSim);
  g.setColor(Color.RED);
  g.drawString("Simulation: Number of trials vs " +
            " Number of variables", 60, 205+h.deltaY);
  //Simulation data are drawn
  draw(g);
  rescale(h.PointsExp);
  g.setColor(Color.CYAN);
  //Exponential is drawn
  g.drawString("Exponential = " + h.base.toString()
                 + "^ n", 60, 225+h.deltaY);
  draw(g);
  h.makeGraphic = false;

 }

 //This is the main (but not mandatory)  method,
 //used only for making a graphic
 @Override public void paintComponent( Graphics g)
 {
  super.paintComponent(g);
  if (h.makeGraphic)
{
  drawer(g);
      //Delete the next line and see what happens.
     //Challenge:
     //Can you find an elegant solution to the
     //ensuing problems (if any)?
     w.buttons();
 }
 }

 }//End of Search class

//*****************************************
//****************GUI CLASS ****************
//*****************************************

//This class operates the graphic interface GUI
```

```
private class GUI extends JPanel
{
 private static final long serialVersionUID = 1L;
 private JTextField nClausesTF;
 private JTextField nVarTF;
 private JTextField popSizeTF;
 private JTextField  propTF;


 private int mutRateInt = (int) (  100*h.mutRate);

 private boolean recomb = false;

 private Integer nClauses = h.nClauses;
 private Integer nVar = h.nVar;
 private Integer popSize = h.popSize;


 //Control button
 private JRadioButtonMenuItem Rec[];
 private ButtonGroup RecButtonGroup;

 private JSlider mutSlider;
 private JButton btnPurpose;
 private JButton btnNewTry;
 private JButton btnGo;
 private JButton btnStop;
 private TextArea news;
 private JRadioButtonMenuItem Show[];
 private ButtonGroup ShowButtonGroup;
 private int  caseNumb;

 JLabel showL;
 JLabel mutLabel;
 JLabel nClausesLabel;
 JLabel nVarLabel;
 JLabel popSizeLabel;
 JLabel recL;
 JLabel propLabel;
```

```
 // Planning of the   screen
 private   GUI( )
 {
  //Organization of buttons
  Box box1 = Box.createHorizontalBox();
  Box box3 = Box.createHorizontalBox();
  Box box4 = Box.createHorizontalBox();
  Box box5 = Box.createHorizontalBox();
  Box box6 = Box.createHorizontalBox();
  Box box7 = Box.createHorizontalBox();

  //  ************ button Purpose
  btnPurpose = new JButton("Purpose");
  box1.add (btnPurpose);
  Purpose purpose = new Purpose ();
  btnPurpose.addActionListener(purpose);

  //  ************ button Go
  btnGo = new JButton("GO for the ladder job");
  box1.add (btnGo);

  btnGo.addActionListener
(
 new ActionListener()
  {
   public void actionPerformed(ActionEvent e)
    {
     go();
    }
  }
);



  //  ************ button NewTry
  btnNewTry = new JButton("NEW RANDOM TRY");
  box1.add (btnNewTry);
  NewTry newTry = new NewTry ();
  btnNewTry.addActionListener(newTry);
```

```
//************   Show work

   showL = new JLabel("Show work " );
  box1.add(showL);
  String ShowNames[]  = { "Yes", "No" };
  // Creates radio button menu items for Yes-No options
  Show  = new JRadioButtonMenuItem[ ShowNames.length ];
  ShowButtonGroup = new ButtonGroup();

  ShowHandler showHandler =  new ShowHandler();
  //Creates show radio button menu items
  for ( int count = 0; count < Show.length; count++ )
{
  Show[ count ]
   = new JRadioButtonMenuItem( ShowNames[ count ] );
 box1.add( Show[ count ] );
 ShowButtonGroup.add( Show[ count ] );
 //Calling of inner class showHandler
 Show[ count ].addActionListener( showHandler );
}
  //Initial setting : work is not shown;
  Show[ 1 ].setSelected( true );
  add(box1);



 //  ************ txtField nClauses
 nClausesLabel= new JLabel(
   "  Number of clauses ");
 nClausesTF = new JTextField(nClauses.toString()+
                             "        " );
 box3.add(nClausesLabel);
 box3.add(nClausesTF);
 NClauses nClauses = new NClauses();
 nClausesTF.addActionListener(nClauses);


// ************ txtField nVar
 nVarLabel= new JLabel(
```

```
   "Number of variables ");
 nVarTF = new JTextField(nVar.toString()+
                              "        " );
 box3.add(nVarLabel);
 box3.add(nVarTF);
 NVar nVar = new NVar();
 nVarTF.addActionListener(nVar);


 //  ************ txtField popSize
 popSizeLabel= new JLabel("Population size");
 popSizeTF = new JTextField("     " + popSize.toString());
 box3.add(popSizeLabel);
 box3.add(popSizeTF );
 PopSize popSize = new PopSize();
 popSizeTF.addActionListener(popSize);
 add(box3);

//***************** Mutation

 mutLabel= new JLabel(
                  "    Mutation rate/bit/generation");
 mutSlider = new JSlider(SwingConstants.HORIZONTAL,
                                        0, 100,10);
 box4.add(mutLabel);
 box4.add(mutSlider);
 mutSlider.setMajorTickSpacing(10);
 mutSlider.setValue( mutRateInt );
 mutSlider.setPaintTicks( true );
 Slider slider = new Slider();
 mutSlider.addChangeListener(slider);

 add(box4);




 //  ************   Recombination
```

```
 recL = new JLabel("Recombination " );
 box5.add(recL);
 String RecNames[]  = { "Yes", "No" };
 // Creates radio button menu items for rec options
 Rec  = new JRadioButtonMenuItem[ RecNames.length ];
 RecButtonGroup = new ButtonGroup();

 RecHandler recHandler =  new RecHandler();
 //Creates rec radio button menu items
 for ( int count = 0; count < Rec.length; count++ )
  {
   Rec[ count ]
 = new JRadioButtonMenuItem( RecNames[ count ] );
   box5.add( Rec[ count ] );
   RecButtonGroup.add( Rec[ count ] );
   //Calling of inner class recHandler
   Rec[ count ].addActionListener( recHandler );
  }
 //Initial setting: no recombination;
 Rec[ 1 ].setSelected( true );
 add(box5);


 // ************ txtField Proposition
 propLabel= new JLabel(" Proposition ");
 propTF = new JTextField( h.prop +
    "                                          "+
    "                                          "+
    "       ");
 box6.add(propLabel);
 box6.add(propTF);
 Prop prop= new Prop();
 propTF.addActionListener(prop);




// ************ button Stop
 btnStop = new JButton("STOP");
```

```
 box6.add (btnStop);
 btnStop.setEnabled(false);
 btnStop.addActionListener
  (
 new ActionListener()
  {
   public void actionPerformed(ActionEvent e)
    {
     stop();
    }
    }
   );
add(box6);


//************* newspaper
box7.add(news = new TextArea("Propositions and  matching " +
                    "generations \n" , 4, 80));
news.setFont(new Font("Courier", Font.PLAIN, 12));
news.setEditable(false);

add(box7);
}


//Inner class to deal with the purpose  button
class Purpose implements ActionListener
{
 public void actionPerformed(ActionEvent event)
 {
  JOptionPane.showMessageDialog
  (null,
   "The program composes logical propositions  " +
   "\nand uses evolution to find  satisfying assignments."
   + "\nChosen propositions are always satisfiable."+
   "\nThe needed  number of generations and of individuals"
   + "\nto solve the task are reported." +
   "\nThe ladder job consist in solving a succession  " +
   "\nof propositions with only one clause and with   " +
   "\nk variables, k ranging from one to a number decided"
```

```
   + "\nby the User." +
   "\nA graphic helps one to answer  to the question:" +
   "\ndoes evolution work in the exponential regime? "
  );
 }
}


//If the newTry button is clicked upon,
//something must be done:
private class NewTry implements ActionListener
{
 public void actionPerformed(ActionEvent event)
 {
  caseNumb = 1;
  task();
 }
}


//Reading instrument of mutation rate
private class Slider implements ChangeListener
{
public void stateChanged(ChangeEvent move)
 {
  mutRateInt = mutSlider.getValue();
  double mutR = mutRateInt;
   h.mutRate = mutR/100;
 }

}

//Inner class to handle events from the rec menu
private class RecHandler implements ActionListener
{
 public void actionPerformed( ActionEvent event )
 {
  //
  for ( int i = 0; i < 2; i++)
{
```

```
 // check for "Yes"
 if ( Rec[ 0 ].isSelected() )
 {
  recomb = true;
  break;
 }
 // check for "Not"
 if ( Rec[ 1 ].isSelected() )
 {
  recomb = false;
  break;
 }
   }
 h.recomb = recomb;

 }
}//end of recHandler

//Sensor of population size
private class PopSize implements ActionListener
{
  public void actionPerformed(ActionEvent event)
  {
String s =  popSizeTF.getText();
s = purify(s);
if (s == "")
{
 JOptionPane.showMessageDialog(null,
    "Please, type a number");
}
else
{
 popSize =  new Integer(s).intValue();
 h.popSize = popSize;
}
  }
}
```

```
//Inner class to handle events from the show menu
private class ShowHandler implements ActionListener
{
 public void actionPerformed( ActionEvent event )
 {
  //
  for ( int i = 0; i < 2; i++)
{
 // check for "Yes"
 if ( Show[ 0 ].isSelected() )
  {
   h.show = true;
   break;
 }

// check for "Not"
if ( Show[ 1 ].isSelected() )
 {
  h.show = false;
  break;
 }
}
 }
}//end of ShowHandler

// Only numbers are accepted
private String purify(String s)
{
 String p = "";
 char c = 'c';
 for (int i = 0; i < s.length(); i++)
 {
  c = s.charAt(i);
  for(int j = 48; j< 58; j++)
  if (c == (char) (j)) p = p+c;
 }
 return p;
}
```

```
//The User can decide the number of clauses
private class NClauses implements ActionListener
{
 public void actionPerformed(ActionEvent event)
{
 String st = nClausesTF.getText();
 st = purify(st);
 if (st.equals(""))
 {
  JOptionPane.showMessageDialog(null,
  "Please, type a number");
 }
 else
 {
  nClauses = new Integer(st).intValue();
  h.nClauses = nClauses;
 }
}
}

//The User can decide the number of variables
private class NVar implements ActionListener
{
 public void actionPerformed(ActionEvent event)
{
 String st = nVarTF.getText();
 st = purify(st);
 if (st.equals(""))
 {
  JOptionPane.showMessageDialog(null,
  "Please, type a number");
 }
 else
 {
  nVar = new Integer(st).intValue();
  h.nVar = nVar;
 }
}
}
```

```
private boolean check(String st1)
{
 boolean correct = true;
 st1 = st1.trim();
 int n = st1.length();
 char d = ' ' ;
 int  countPar = 0;
 int countO = 0;
 int countR = 0;
 int countAnd = 0;
 for (int i = 0; i < n; i++)
 {
  d = st1.charAt(i);
  if (d == '(') countPar = countPar +1;
  if (d == ')') countPar = countPar -1;

  if (d == 'A') countAnd = countAnd +1;
  if (d == 'O') countO = countO +1;
  if (d == 'R') countR = countR +1;
 }
 if ( (countPar == 0) & (countO == countR))
  {
h.nClauses = countO +1;
h.nVar = (countAnd / h.nClauses) + 1;
  }
 else
 {
  JOptionPane.showMessageDialog(null,

  "Please, type a full disjunctive form." +
  "\nUse the next conventions:" +
  "\n1)Clauses are in within parentheses." +
  "\n2)Clauses are separated by OR."+
  "\n3)OR is surrounded by one space from each side."+
  "\n4)Variables are denoted by a small w. " +
  "\n5)Variables are known by their position." +
  "\n6)All clauses have the same number of variables."+
  "\n7)Write OR in place of or." +
  "\n8)Write A instead or AND." +
  "\n9)Write N instead of Not" +
```

```
  "\nExample: (wAw) OR (NwAw) OR (wANw)" );
  correct = false;
 }
 return correct;
}

//Reader of the proposition
class Prop implements ActionListener
 {
  public void actionPerformed(ActionEvent event)
   {
 String st1 = propTF.getText();

 if (check(st1))
 {
 h.prop =  st1;
 caseNumb = 2;

 task();
 }

}
  }

 //Trigger of action
 public void go( )
 {
  ensureEventThread();
  caseNumb = 0;
  task();
 }


 //When the action goes on,
 //Everything must be withdrawn from the USER
 //Except the stop button
 private void editionFalse()
 {
  btnGo.setEnabled(false);
  nClausesTF.setEditable(false);
```

```
 nVarTF.setEditable(false);
 propTF.setEditable(false);
 popSizeTF.setEditable(false);
 btnNewTry.setEnabled(false);
 btnStop.setEnabled(true);

 Rec[0].setEnabled(false);
 Rec[1].setEnabled(false);
 mutSlider.setEnabled(false);
}

//The USER acquires edition power
private void editionTrue()
{
 auxThread = null;
 btnGo.setEnabled(true);
 btnNewTry.setEnabled(true);
 nClausesTF.setEditable(true);
 nVarTF.setEditable(true);
 popSizeTF.setEditable(true);
 propTF.setEditable(true);
 btnStop.setEnabled(false);
 Rec[0].setEnabled(true);
 Rec[1].setEnabled(true);
 mutSlider.setEnabled(true);
 }

private void communicate()
{
 s.nClauses = h.nClauses;
 s.nVar = h.nVar;
 s.numbInd = h.popSize;
 s.mutationRate = h.mutRate;
 s.recombination = h.recomb;
 s.prop = h.prop;


}

//The action is hung from an execution thread.
```

```
 //In that way, it can be stopped
 private void task()
 {
  //Vales are passed to s
  communicate();
  //Edition is forbidden
  editionFalse();
  //The action is encapsulated
  Runnable watchRun = new Runnable()
  {
   public void run()
{
 w.news.setText("");
 try
  {
   s.go = true;
   switch (caseNumb)
   {
case 0 : //Ladder job
 {
  s.work();
  h.makeGraphic = true;
  s.repaint();
  break;
 }
        //randomTry
case 1 : s.oneProblem();break;
        //prop given by the User
case 2  : s.solveProblem(h.prop);break;
   }
   editionTrue();
  } catch (InterruptedException e)
  {
e.printStackTrace();
  }
  getReady( );
 }
};

 //Encapsulated action  is connected to the thread
```

```
 auxThread = new Thread(watchRun, "auxThread");
 auxThread.start();
}


 //If the execution is interrupted,
 //some housekeeping must be done.
 private void getReady( )
 {
  Runnable r = new Runnable()
   {
 public void run()
  {
   try {
    ensureEventThread();
    btnStop.setEnabled(false);
    btnGo.setEnabled(true);
   } catch (Exception x)
             {
              x.printStackTrace();
             }
   }
};
  SwingUtilities.invokeLater(r);
 }

//Am I working with a ghost or with my thread?
private void ensureEventThread()
{
 if (SwingUtilities.isEventDispatchThread())
 return;
 throw new RuntimeException("Undue calling");
}


//The task is halted
private void stop()
{
 ensureEventThread();
 if (auxThread != null)
```

```
{
 auxThread.interrupt();
 editionTrue();
 s.go = false;
 w.news.append("Job cancelled  \n");
}
}


//An unknown bug causes troubles
//that are partially remedied
//with some housekeeping.
private void buttons()
{
    btnPurpose.setVisible(false);
btnPurpose.setVisible(true);

 Show[0].setVisible(false);
 Show[1].setVisible(false);
 Show[0].setVisible(true);
 Show[1].setVisible(true);

 btnNewTry.setVisible(false);
 btnNewTry.setVisible(true);
  btnGo.setVisible(false);
  btnGo.setVisible(true);
  btnStop.setVisible(false);
 btnStop.setVisible(true);
 nClausesTF.setVisible(false);
 nClausesTF.setVisible(true);
 nVarTF.setVisible(false);
 nVarTF.setVisible(true);
 popSizeTF.setVisible(false);
 popSizeTF.setVisible(true);
 propTF.setVisible(false);
 propTF.setVisible(true);
 Rec[0].setVisible(false);
 Rec[0].setVisible(true);
 Rec[1].setVisible(false);
 Rec[1].setVisible(true);
```

```
 mutSlider.setVisible(false);
 mutSlider.setVisible(true);
 mutLabel.setVisible(false);
 mutLabel.setVisible(true);
 showL.setVisible(false);
 showL.setVisible(true);

 nClausesLabel.setVisible(false);
 nVarLabel.setVisible(false);
 popSizeLabel.setVisible(false);
 recL.setVisible(false);
 propLabel.setVisible(false);

 nClausesLabel.setVisible(true);
 nVarLabel.setVisible(true);
 popSizeLabel.setVisible(true);
 recL.setVisible(true);
 propLabel.setVisible(true);

}



}//End of class GUI

}//End of main class
```

120 ***Exercise*** *Run the code and verify that the program works in the exponential regime to solve the chosen subclass of SAT.*

121 ***Challenge*** *To use the previous program, one must observe a tight formalism to enter a disjunctive form. Help the User with an intelligent corrector.*

122 ***Morals*** *We gave to evolution the task of solving a problem whose solution can be found at once by inspection. Nevertheless, we found that the presented implementation of evolution solves the same task but fells pray of the exponential regime. What does this mean? We see three implications:*

1. *One can finely program evolution as bad as desired and the path to excellence must be looked for with care and tenacity. This situation appears with relative frequency in slightly complex applications but becomes endemic in complex ones. Therefore, you can choose Java for the study of Evolution as a challenging profession for a life.*

2. *One must make sure that one is not confounding* ***neodarwinism*** *with* ***lamarckism****! Yes: mutations are at random in* ***neo-darwinism*** *while they are biased in* ***Lamarckism****, directed by whatever reason. It is usual to identify Lamarckism with the slogan that says: the necessity creates the organ. Lamarckism appears very commonly in modern science when someone claims that evolution was the cause of a given change along the fossil record. Such a claim always can be done, but let us distinguish when it is done according to lamarckism and when according to neodarwinism. In lamarckism, one makes the claim and go to publish. So, modern Lamarckism reads: the necessity creates the correct mutation. In neodarwinism things are very different: the correct mutation usually corresponds to a complex change and this means that it appeared not by random mutation but as the winner mutation among many many random trials that may represent half-cooked products. So, to pass from lamarckism to neodarwinism, one must exhibit a winner fossil among many suboptimal versions.*

3. *It is naive to claim that evolution is the solution to some given problem without exhibiting the actual program that solves the problem and the regime in which it works. Additionally, if you want to claim that natural evolution solves a problem using a solution that you devised and that beats thousands of other proposals, please, show that nature can find your solution with a probability that is greater, say, than* 10^{-1000}*. In this way, you are making of evolution a part of science and not of mythology and you will save your honor because you are doing your best to keep* ***lamarckism*** *at bay. If you fail to show that your solution has a significant probability of occurrence in nature, you do not need to claim that evolution is the correct explanation because in materialism chance is preferable: it is also an explanation that never can be ruled out to complete satisfaction and it is by far simpler than evolution. And, please: Where are your predictions?*

Our program is demanding. But, in which other form will we be heard by the next generation of scientists?

Actually, our demands are elementary, inescapable. They might be met in the following form: if you discover that your program works only if some parameters are finely tuned, develop a wider world in which you parameters are also included

in what must struggle for life and evolve. If you think that it is your idea that is very smart, tailor a world in which ideas compete one with another for solving your problem. If you feel that you lack computing power, push our community to work in the eJGrid.

8.4 A nice challenge

We have seen that our first attempt of using evolution to solve the disjunctive family works in the exponential regime. Let us use the very same technology to solve another class, the conjunctive family, which is directly related to the disjunctive one.

123 ***Conjunctive normal forms***

An expression like

$$(\sim p \vee \sim q \vee \sim r) \wedge (\sim p \vee q \vee \sim r)$$

is called a **conjunctive normal form**. It is an *And* package of propositions, every one of which is string of variables, negated or not, joined by the *OR* operator. Each *OR* string is called a clause. The normal conjunctive form is full when every clause contains all variables of the discourse. The aforementioned formula is full and has two clauses over three variables.

Conjunctive normal forms reflect at the same time complexity and creativity. To see this, let us consider the mentioned form: q appears negated in the first clause while it appears as it is in the second one. This can be interpreted as complexity: if you do something, q, dual effects appear: it helps in one place but it harms in another one. Creativity is deployed in the other variables: assign False to variables p and r and you will have no contradictory effects and the overall form evaluates to true. That is why this family is wonderful to capture as the constraints to enable function as the many possibilities that eventually exist to solve a problem.

Conjunctive normal forms have satisfiable exponents, such as $p \wedge q$ which is true for the TT evaluation. And also have not satisfiable members such as $p \wedge \sim p$, which is a contradiction.

The existence of satisfiable and not satisfiable propositions in the conjunctive family causes our algorithms to decide SAT to work in the exponential regime in the number of variables. The reason is the following: to prove that a formula is satisfiable, we must show a truth assignment that evaluates to true. But to declare

that a formula is not satisfiable, we must show that all possible truth assignments evaluate to false. The problem is that the only available form to prove that all assignments of a conjunctive form evaluate to false is a direct and exhaustive verification of all 2^n assignments that are possible for a set of n variables.

Now, we do not recognize the possibility that evolution could make an exhaustive listing of all possible truth assignments. That is why we play with random assignments that must come in multitude to guarantee with a high probability that a formula is really not satisfiable. Technically, our algorithm classify therefore as **probabilistic**.

One expects that if one passes from complete certainty to almost complete certainty, the complexity of problems might significantly diminish. To see this, consider the next possible algorithm: look at the first clause and give by instinct to the first variable a truth assignment to cause the clause to evaluate to true. Look at the second clause: if with the given value of the first variable it evaluates to true, pass to the third clause. If with the given value of the first variable the second clause evaluates to false, give to the second variable the appropriate value to render the clause true. And so on. If you have scanned all clauses and some variables remain to receive an assignment, give the value true to them and then decide the proposition as satisfiable. If you have assigned values to all variables, evaluate the proposition and read what you must decide. This is a really efficient algorithm but its fairness remains to be investigated. Questions of this sort conform a very rich discipline: the theory of approximation algorithms (Hochbaum, 1995).

The aforementioned comments could mean that we do not need to make any simulation in regard with the conjunctive family: we are already in the exponential regime. Nevertheless, conjunctive normal forms define a very good setting for making genuine experiments, as follows:

If we define, α as the ratio of clauses to variables, then we have the possibility to make a very transparent experiment: one can inquire over the dependence of satisfiability on the value of α:

When α is low, with less clauses than variables, one is making a discourse of unrelated things and so one hardly can fall into a contradiction. Thus, for low values of α most propositions are satisfiable. But, when we begin to tie the ends, then we use more clauses on the same number of variables, and the probability of falling into contradiction grows. Hence, for high values of α we shall expect that most random propositions in conjunctive normal form are unsatisfiable, i.e. that almost all truth assignments end in a final false evaluation.

So, let us see how to solve the conjunctive family. We have at least two options to solve that family. The first one is to use the **laws of Morgan**, which are:

$\sim (p \wedge q) = \sim p \vee \sim q$
$\sim (p \vee q) = \sim p \wedge \sim q$

So, if we take a proposition in normal conjunctive form, such as

$(\sim p \vee \sim q \vee \sim r) \wedge (\sim p \vee q \vee \sim r)$

we can rewrite it as the negation of its negation:
$= \sim [\sim [(\sim p \vee \sim q \vee \sim r) \wedge (\sim p \vee q \vee \sim r)]]$

Using now the laws of Morgan, the original proposition is also equal to:

$= \sim [\sim (\sim p \vee \sim q \vee \sim r) \vee \sim (\sim p \vee q \vee \sim r)]$
$= \sim [(p \wedge q \wedge r) \vee (p \wedge \sim q \wedge r)]$

We see here an expression which is the negation of a disjunctive form. For this reason, we might re-use the software that we have developed for the disjunctive family to solve the conjunctive one.

A second method to solve the conjunctive family consists in designing new software but using the same approach that was used to attack the disjunctive family. This is the method we choose.

124 ***Our code***

The next code proposes a probabilistic solution to the conjunctive family.

```
/* Program D124, Conjunctive.
 *
 * We solve a subclass of SAT by means of evolution.
 * SAT is a model  that captures the idea
 * that complexity arises when one must do things
 * subject to  incompatible restrictions:
 * do this but not that.
 *
 * SAT or Satisfaction is the problem of
 * knowing whether or not a given logical proposition
 * is satisfiable, i.e. if there exists a truth
```

```
 * assignment that evaluates the proposition to true.
 *
 * Our subclass consists in the
 * conjunctive normal full forms with n variables and
 * k clauses. All clauses contain the n variables:
 * we work with full forms.
 * Variables might be negated or not.
 * The prob of negation of a variable is 0.5.
 * Variables are joined by and And operator.
 * Clauses are joined by an OR operator.
 *
 * Random   equiprobable truth values are assigned
 * to the  variables and
 * the prop is evaluated to true else false.
 *
 * We inquire over the number of   individuals or trials
 * that are necessary to get a satisfying assignment for a
 * full normal conjunctive form.
 *
 * A proposition is declared to be unsatisfiable if
 * after 2^(number of Variables) generations no
 * satisfiable assignment has  been found.
 *
 * A graphic that displays the relation between the
 * number of variables and the number of trials
 * that are necessary to find a satisfying assignment
 * is reported given that only satisfiable forms are
 * considered.
 */

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.TextArea;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.BitSet;
import java.util.Random;
```

```
import javax.swing.Box;
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.JSlider;
import javax.swing.JTextField;
import javax.swing.SwingConstants;
import javax.swing.SwingUtilities;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

public class Conjunctive
{
 private static final long serialVersionUID = 1L;
 //Some relevant information is packed into a class. This
 //allows stability of memories against thread threats.
 Job h = new Job();
 private static Conjunctive t = new Conjunctive();
 private static  Search s = t.new  Search();
 private static GUI w =  t.new GUI();

 //The JFrame which shows everything
 // is constructed with a title
 private static JFrame f = new JFrame("SAT: " +
  "conjunctive subclass: " +
  "to modify parameters end with enter.");



 // This thread is used to enable the User to
 //terminate  lengthy tasks.
 private volatile Thread auxThread;

 public static void main(String[] args)
 {
  //WGUI is added to the JFrame f
```

```
  f.add( w, BorderLayout.CENTER );
  f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  //Dimensions of the JFrame
  f.setSize( 600, 670 );
  f.setVisible( true );
  f.add( s, BorderLayout.CENTER );
 }

//*****************************************
//****************JOB CLASS ****************
//*****************************************

//This auxiliary class unites some fundamental
//variables into an object
private  class Job
{
 //number of boolean variables per clause
 Integer nVar;
 //ratio of number of clauses to that of variables
 double  alpha;
 //number of variables
 Integer nClauses;
 //population size
 Integer popSize;
 //mutation rate per bit per generation
 Double mutRate;
 //proposition to be evaluated
 String prop;
 //to show or not to show actual computations
 boolean show;
 //recombination: yes else not
 boolean recomb;
 //number of trials or total number of individuals
 //that were needed to take a decision.
 Long cost;
 //Points for a graphic
 boolean makeGraphic = false;
 double PointsSim[][] = new double[1000][2];
 double PointsExp[][] = new double[1000][2];
 int Points[][] = new int[1000][2];
```

```
 int nPoints;
 double xmax, ymax;
 Integer base = 2;
 int deltaY;


 //Mandatory initialization
 Job( )
{
 //Number of boolean variables
 nVar = 2;
 //Ratio of numb of clauses to that of variables
 alpha = 2.0;
 //Number of clauses
 nClauses = (int) alpha * nVar;
 //Population size
 popSize = 4;
 //Mutation rate
 mutRate = 0.2;
 //Proposition
 prop = "(wORw) A (wORNw) A (NwORNw)";
 //To show or not to show actual work
 show = false;
 //Recombination: yes else not.
 recomb = false;
 for(int i = 0; i < 100; i++)
    for(int j = 0; j < 2; j++)
  {
   PointsSim[i][j] = 0;
   PointsExp[i][j] = 0;
   Points[i][j] = 0;
  }
}




 //Converts a double number into a string
 private String ToString(double alpha)
 {
```

```
  Integer a = (int) (alpha*10);
  Integer b = (int) alpha;
  Integer c = a - 10*b;
  b  = ( b % 10);
  String str = b.toString() + "." + c.toString();
  return str;
 }

 }//end of class Job

 //This class is responsible for finding
 //a satisfying truth assignment
 private class  Search  extends JPanel
 {

private static final long serialVersionUID = 1L;

// Boolean condition to continue else halt
  private    boolean go = true;


  //Constructor: mandatory initialization;
  Search()
  {

  }

 // Turn on  the random generator
    Random r  = new Random();

 //Proposition in formation
 private   String a;
 //propositions
 private String prop, c;

 private   boolean print = false;
 private   boolean problem = true;

 private   int maxNumbInd = 10000;
 private   BitSet Ind1[]   = new BitSet[maxNumbInd];
```

```
 private   BitSet Ind2[]   = new BitSet[maxNumbInd];
 private   int FitnessInd[]
                                  = new int[maxNumbInd];
 private   int OrderInd[]  = new int[maxNumbInd];
 private   int gen;
 private  int maxGen;
 private   boolean recombination;

 private long cost;

 //Not operators are inserted with prob 0.5
 private    String negations( )
 {
 for(int j=0; j<a.length();j++)
   {
if (a.charAt(j)=='w') //
 {
  int d = r.nextInt(2);
  if (d == 0)
   a = a.substring(0, j)+ "f" + a.substring(j+1);
 }
   }
a = a.replace("f", "Nw");
  return a;
 }

 //Composes a random normal conjunctive full w-proposition
 // with h.nClauses clauses. The NOT operator is included
 //for each w with probability 0.5.
 private  String wProposition()
 {
  if (print)
   w.news.append("Construction of the l-proposition \n");
  //Proposition is initialized.
  a = "(w";
  for(int i= 0; i < h.nClauses-1; i++)
  {
   for(int j = 0; j < h.nVar-1; j++)
     a = a.concat("ORw");
   a= a.concat(") A (w");
```

```
  }
  for(int j = 0; j < h.nVar-1; j++)
     a = a.concat("ORw");
  a= a.concat(")");
  if (print)
  w.news.append("w-prop without NOT = " + a + "\n");
  negations();
  if (print)
  w.news.append("w-prop with NOTs   = " + a + "\n");
  return a;
 }

//Converts a BitSet value into a String
 private String decode(BitSet t)
{
String s= "";
for(int i =0; i < h.nVar; i++)
 {
if (t.get(i)==true) s = s + "T";
 else s = s + "F";
 }
return s;
}

//The proposition is evaluated
 private     String evaluation(BitSet s)
 {
  prop = c;
  if (print)
  {
   w.news.append("\n");
   w.news.append("Truth evaluation begins" + "\n");
   w.news.append("prop = " + h.prop + "\n");
   w.news.append("Truth assignment = " + decode(s) + "\n");
  }
  int u = 0;
  for(int j=0; j<prop.length();j++)
   {
    if (prop.charAt(j)==’w’)
     {
```

```
      if (s.get(u) == true)
   prop = prop.substring(0, j)+ "T" +
                                prop.substring(j+1);
  else
   prop = prop.substring(0, j)+ "F" +
                                prop.substring(j+1);
  u = u +1;
  if (u == h.nVar) u=0;
}
   }
  if (print)
  w.news.append("Truth value8 = " + prop + "\n");
   return prop;
}

//The proposition is evaluated until the end
//Not operators are evaluated at the
//very beginning for they apply over the next
//to the right variable.
private     String simplification(String b)
{
 int counter = 0;
 int n =  b.length();
 b = b.trim();
 n =  b.length();
 while ( (b.length()>1) & (go) )
  {
b=  b.replace("NT","F");
b=  b.replace("NF","T");
if (print)
w.news.append("Truth value1 = " + b + "\n");
b=  b.replace("TORT","T");
b=  b.replace("TORF","T");
b=  b.replace("FORT","T");
b=  b.replace("FORF","F");

if (print)
w.news.append("Truth value2 = " + b + "\n");

b=  b.replace("(T)","T");
```

```
 b=  b.replace("(F)","F");

 if (print)
 w.news.append("Truth value3 = " + b + "\n");

 b=  b.replace("T A T","T");
 b=  b.replace("T A F","F");
 b=  b.replace("F A T","F");
 b=  b.replace("F A F","F");
 if (print)
 w.news.append("Truth value4 = " + b + "\n");
 counter = counter +1;
if (counter == n)
{
w.news.append(
  "SOMETHING IS WRONG WITH THE PROPOSITION \n");
go = false;
break;
}
   }

  return b;
 }



 //Ind1 is a population of binary strings.
 //The BitSet class is used to instantiate
 //binary strings as individuals
 private   void initializePop()
 {
  //s is false everywhere
  BitSet s = new BitSet();

  for(int i = 0; i < h.popSize; i++)
Ind1[i] = (BitSet) s.clone();
  //Reinitialization with random truth values.
  for(int i = 0; i < h.popSize; i++)
  {
   for(int j = 0; j < h.nVar; j++)
```

```
{
   int l = r.nextInt(2);
   if (l == 1)
    Ind1[i].set(j); //the j-th bit is true
}
  }
 }



// Does an individual   satisfy  the target?
   private int assessInd(BitSet s)
 {
  String b = evaluation(s);
  b=simplification(b);
  int  fitness=0;
  if (b.equals("T")) fitness = 1;
  return fitness;

 }


// Individuals are sorted by fitness.
// the fittest is the first, with minimum error
private   boolean sortingInd()
 {
  int   Champ;

  //FitnessIndc[] is a copy of FitnessInd[]
  //used as workbench
  int FitnessIndc[];
  FitnessInd= new int[maxNumbInd];
  FitnessIndc= new int[maxNumbInd];
  for(int i = 0; i< h.popSize; i++)
   {
    FitnessInd[i] =0;
    FitnessIndc[i] =0;
   }
  for(int i = 0; i< h.popSize; i++)
```

```
  {
   FitnessInd[i] = assessInd(Ind1[i]);
   FitnessIndc[i] = FitnessInd[i];
  }
  //We sort individuals by fitness
  //0 is the minimum fitness
  for(int i = 0; i< h.popSize;i++)
  {
   Champ = 0;
   for(int j = 0; j< h.popSize;j++)
   if (FitnessIndc[j] > FitnessIndc[Champ]) Champ = j;
   //The array OrderInd orders individuals by  fitness:
   //The fittest is  zero
   OrderInd[i] = Champ;
   FitnessIndc[Champ] = 0;
  }
 // if Pattern has been guessed
 if (FitnessInd[OrderInd[0]] == 1)
 {
  problem = false;
  w.news.append(" The Proposition is satisfiable \n");
  w.news.append(" by truth assignment "
             + decode(Ind1[OrderInd[0]]) + "\n");
  prop = c;
 }
 return problem;
 }

// The new generation is made
 private    void newGenerationInd()
 {
  int   i=0;
  //The fittest individuals are reproduced
  // numbCopies times
  int numbCopies = 10;
  int   k=0;  //rank, 0 is the highest
  while (i<h.popSize)
  {
   int counter = numbCopies;
   while ((counter>0) & (i<h.popSize))
```

```
   {
Ind2[i] = (BitSet) Ind1[OrderInd[k]].clone();
i++;
    counter= counter-1;
   }
  k++;
 }
 for(int j = 0; j<h.popSize; j++)
  {
   Ind1[j] = (BitSet) Ind2[j].clone();
  }
 }


// Recombination is enabled
private   BitSet recInd(int l)
{
 //We choose a partner at random
 int k = r.nextInt(h.popSize);
 //b is false everywhere
 BitSet b = new BitSet();
 //b is the recombinant of l and k
 for(int j = 0; j < h.nVar; j++)
{
   int m = r.nextInt(2);
   if (m == 1) //take the bit from l
    {
   if (Ind1[l].get(j)== true ) b.set(j);
    }
   if (m == 0)//take the bit from k
    {
   if (Ind1[k].get(j)== true ) b.set(j);
    }
}
 return b;
}


// The definitive population  is a recombinant one.
private   void recombinationInd()
```

```
{
 //Ind2 is a copy of Ind1
 for(int i=0; i<h.popSize; i++)
 Ind2[i] =  (BitSet) Ind1[i].clone();
 //Ind2 is a recombinant version of Ind1
 for(int l=0; l<h.popSize; l++)
 Ind2[l] = recInd(l );
 //Ind2 is transferred to Ind1
 for(int i=0; i<h.popSize; i++)
 Ind1[i] = (BitSet) Ind2[i].clone() ;
}



// Individual  a is mutated at a random place.
  private   BitSet pointMutationInd(BitSet a)
  {
   int l = r.nextInt(h.nVar);
   a.flip(l);
   return a;
  }

 // Individuals are mutated
 private   void mutationInd()
 {
  for(int i=0; i<h.popSize; i++)
   {
double p= r.nextDouble();
if (p < h.mutRate)
      Ind1[i] = pointMutationInd(Ind1[i]) ;
   }
 }

 // Given a goal, it is guessed by evolution
 private   void  evolution()
 {
  //The new generation is produced
  //as copies of the fittest
  newGenerationInd();
  if (recombination) recombinationInd();
```

```
  //The new population is subjected to mutation
  mutationInd();
  gen++;
  problem = sortingInd();
  if (gen == maxGen)
  {
problem = false;
w.news.append(" No satisfying assignment was found" +
" after " + gen * h.popSize + " trials \n");
  }
 }

 public   void report1()
 {
  w.news.append(" Prop = " + prop + "\n");
  w.news.append(" h.nVar = " + h.nVar + "\n");
  double a = h.nClauses;
  double b = h.nVar;
  double alphaDouble = a / b;
  w.news.append(" Alpha = " + alphaDouble + "\n");
  w.news.append(" h.nClauses = " + h.nClauses + "\n");
  w.news.append(" Population size = " + h.popSize + "\n");
  w.news.append(" Mutation rate per bit per gen = "
                          + h.mutRate + "\n" );
  w.news.append(" Recombination = "
          + h.recomb + "\n" );
 }

 public   void report2()
 {
  w.news.append(" Number of generations = " + gen + "\n");
  w.news.append(" Cost = " + h.cost + " trials. \n");
  w.news.append("\n");
  w.news.append("******************" + "\n");
  w.news.append("******************" + "\n");
  w.news.append("******************" + "\n");

 }

 //Solves proposition prop
```

```
 private void solveProblem(String prop)
 {
  c = prop;
  h.prop = prop;
  report1();
  maxGen = (int) Math.ceil( Math.pow(2, h.nVar));
  problem = true;
  initializePop();
  while   ((problem) & (go))
   {
    print = h.show;
    evolution();
   }
  cost = h.popSize*gen;
  h.cost = cost;
  report2();
  h.PointsSim[h.nVar][0] = h.nVar;
  h.PointsSim[h.nVar][1] = h.cost;
  h.nPoints = h.nVar;
 }

 //Solves one proposition
 private   void oneProblem()
 {
  gen = 0;
  prop = wProposition();
  w.propTF.setText(prop);
  solveProblem(prop);
 }

//The ladder job is defined and solved
 private    void  work() throws InterruptedException
  {

   h.nVar = 1;
   h.nClauses =   (int)(h.alpha * h.nVar);
   while  ((go) &   (Thread.currentThread() == auxThread))
{
 h.nClauses = h.nClauses;
     w.nClausesTF.setText(h.nClauses.toString());
```

```
     w.nVarTF.setText(h.nVar.toString());
     w.popSizeTF.setText(h.popSize.toString());
     problem = true;
     oneProblem();
     problem = true;
     h.nVar = h.nVar +1;
     h.nClauses = h.nVar * ( (int) h.alpha ) ;
   }
 }

 //To draw the graphic, the maximum value of
 //data and of an exponential is found.
 private void findMax()
 {
  h.xmax = 0;
  h.ymax = 0;
  //Max values are detected
  for(int i = 1; i < h.nPoints; i++)
   {
if (h.PointsSim[i][0] > h.xmax) h.xmax =  h.PointsSim[i][0];
if (h.PointsSim[i][1] > h.ymax) h.ymax =  h.PointsSim[i][1];
   }
  for(int i = 1; i < h.nPoints; i++)
  {
if (h.PointsExp[i][0] > h.xmax) h.xmax =  h.PointsExp[i][0];
if (h.PointsExp[i][1] > h.ymax) h.ymax =  h.PointsExp[i][1];
  }

 }

 //Data of a graphic is normalized to be placed
 //in a given rectangle
 private void rescale( double[][] PointsD)
 {

  //scale factor are defined
  double  fx;
  double  fy;
  fx = 240 / h.xmax;
  fy = -360 / h.ymax;
```

```
  int tx = 60;
  int ty = 585;
  //Scale formulae
  for(int i = 1; i < h.nPoints; i++)
  {
    h.Points[i][0] =  (int) (PointsD[i][0]*fx +tx);
   h.Points[i][1] =  (int)  (PointsD[i][1]*fy +ty);
  }
 }

 //The graphic is drawn
 private void draw(Graphics g )
 {
  for(int count = 1; count < h.nPoints-1; count++)
   g.drawLine(h.Points[count][0], h.Points[count][1]+h.deltaY,
   h.Points[count+1][0],h.Points[count+1][1]+h.deltaY);
 }

 private void drawer(Graphics g )
 {
//An exponential function serves as reference
  for(int i = 1; i < h.nPoints; i++)
   {
h.PointsExp[i][0] =  i;
h.PointsExp[i][1] =  Math.pow(h.base, i);
   }
  findMax();
  h.deltaY=30;
  g.drawRect( 50,205+h.deltaY, 295,390);
  g.setColor(Color.BLACK);
  g.fillRect(50,205+h.deltaY, 295,390);
  rescale(h.PointsSim);
  g.setColor(Color.RED);
  g.drawString("Simulation: Number of trials vs " +
            " Number of variables", 60, 225+h.deltaY);
  //Simulation data are drawn
  draw(g);
  rescale(h.PointsExp);
  g.setColor(Color.CYAN);
```

```
  //Exponential is drawn
  g.drawString("Exponential = " + h.base.toString()
                 + "^ n", 60, 245+h.deltaY);
  draw(g);
  h.makeGraphic = false;

 }


 //This is the main (but not mandatory)  method,
 //used only for making a graphic
 @Override public void paint( Graphics g)
 {
  super.paintComponent(g);
  if (h.makeGraphic)
{
  drawer(g);
    //Delete the next line and see what happens.
   //Challenge:
   //Can you find an elegant solution to the
   //ensuing problems (if any)?
   w.buttons();
}
 }

 }//End of Search class


//*******************************************
//****************GUI CLASS ******************
//*******************************************

//This class operates the graphic interface GUI
private class GUI extends JPanel
{
 private static final long serialVersionUID = 1L;
 private JTextField nClausesTF;
 private JTextField nVarTF;
 private JTextField alphaTF;
 private JTextField popSizeTF;
```

```
private JTextField  propTF;

private int mutRateInt = (int) (  100*h.mutRate);
private Integer popSize = h.popSize;


//Control button
private JRadioButtonMenuItem Rec[];
private ButtonGroup RecButtonGroup;

private JSlider mutSlider;
private JButton btnPurpose;
private JButton btnNewTry;
private JButton btnGo;
private JButton btnStop;

private TextArea news;
private JRadioButtonMenuItem Show[];
private ButtonGroup ShowButtonGroup;
private int  caseNumb;

private  JLabel showL;
private  JLabel mutLabel;
private JLabel nClausesLabel;
private JLabel nVarLabel;
private JLabel popSizeLabel;
private JLabel recL;
private JLabel propLabel;
private  JLabel alphaLabel;


// Planning of the   screen
private   GUI( )
{
 //Organization of buttons
 Box box1 = Box.createHorizontalBox();
 Box box3 = Box.createHorizontalBox();
 Box box4 = Box.createHorizontalBox();
 Box box5 = Box.createHorizontalBox();
 Box box6 = Box.createHorizontalBox();
```

```
  Box box7 = Box.createHorizontalBox();

  //  ************ button Purpose
  btnPurpose = new JButton("Purpose");
  box1.add (btnPurpose);
  Purpose purpose = new Purpose ();
  btnPurpose.addActionListener(purpose);

  //  ************ button Go
  btnGo = new JButton("GO for the ladder job");
  box1.add (btnGo);

  btnGo.addActionListener
(
 new ActionListener()
  {
   public void actionPerformed(ActionEvent e)
    {
     go();
    }
  }
);



  //  ************ button NewTry
  btnNewTry = new JButton("NEW RANDOM TRY");
  box1.add (btnNewTry);
  NewTry newTry = new NewTry ();
  btnNewTry.addActionListener(newTry);


//************   Show work

  showL = new JLabel("Show work " );
  box1.add(showL);
  String ShowNames[]  = { "Yes", "No" };
  // Creates radio button menu items for Yes-No options
  Show  = new JRadioButtonMenuItem[ ShowNames.length ];
  ShowButtonGroup = new ButtonGroup();
```

```
  ShowHandler showHandler =  new ShowHandler();
  //Creates show radio button menu items
  for ( int count = 0; count < Show.length; count++ )
{
 Show[ count ]
   = new JRadioButtonMenuItem( ShowNames[ count ] );
 box1.add( Show[ count ] );
 ShowButtonGroup.add( Show[ count ] );
 //Calling of inner class showHandler
 Show[ count ].addActionListener( showHandler );
}
  //Initial setting : work is not shown;
  Show[ 1 ].setSelected( true );
  add(box1);

  //************ txtField h.nVar
  nVarLabel= new JLabel(
     "Number of variables ");
  nVarTF = new JTextField(h.nVar.toString()+
                               "        " );
  box3.add(nVarLabel);
  box3.add(nVarTF);
  NVar nVar = new NVar();
  nVarTF.addActionListener(nVar);


  //************ txtField h.alpha
  alphaLabel= new JLabel( "Alpha  ");
  alphaTF = new JTextField(h.ToString(h.alpha)+
                  "                      " );
  box3.add(alphaLabel);
  box3.add(alphaTF);
  Alpha alpha = new Alpha();
  alphaTF.addActionListener(alpha);




 //  ************ txtField h.nClauses
```

```
 nClausesLabel= new JLabel(
   "Number of clauses ");
 nClausesTF = new JTextField(h.nClauses.toString()+
                              "        " );
 box3.add(nClausesLabel);
 box3.add(nClausesTF);
 NClauses nClauses = new NClauses();
 nClausesTF.addActionListener(nClauses);




 //  ************ txtField h.popSize
 popSizeLabel= new JLabel("Population size");
 popSizeTF = new JTextField("     " + h.popSize.toString());

 box3.add(popSizeLabel);
 box3.add(popSizeTF );
 PopSize popSize = new PopSize();
 popSizeTF.addActionListener(popSize);
 add(box3);

//***************** Mutation

 mutLabel= new JLabel(
                  "Mutation rate/bit/generation");
 box4.add(mutLabel);
 mutSlider = new JSlider(SwingConstants.HORIZONTAL,
                                        0, 100,10);
 mutSlider.setMajorTickSpacing(10);
 mutSlider.setValue( mutRateInt );
 mutSlider.setPaintTicks( true );
 Slider slider = new Slider();
 mutSlider.addChangeListener(slider);
 box4.add(mutSlider);
```

```
 //  ************   Recombination

 recL = new JLabel("Recombination " );
 box4.add(recL);
 String RecNames[]  = { "Yes", "No" };
 // Creates radio button menu items for rec options
 Rec  = new JRadioButtonMenuItem[ RecNames.length ];
 RecButtonGroup = new ButtonGroup();

 RecHandler recHandler =  new RecHandler();
 //Creates rec radio button menu items
 for ( int count = 0; count < Rec.length; count++ )
  {
   Rec[ count ]
 = new JRadioButtonMenuItem( RecNames[ count ] );
   box4.add( Rec[ count ] );
   RecButtonGroup.add( Rec[ count ] );
   //Calling of inner class recHandler
   Rec[ count ].addActionListener( recHandler );
  }
 //Initial setting: no recombination;
 Rec[ 1 ].setSelected( true );
 add(box4);


// ************ button Stop
 btnStop = new JButton("STOP");
 box5.add (btnStop);
 btnStop.setEnabled(false);
 btnStop.addActionListener
  (
 new ActionListener()
  {
   public void actionPerformed(ActionEvent e)
    {
     stop();
    }
    }
   );
```

```
 add(box5);

 // ************ txtField Proposition
 propLabel= new JLabel(" Proposition ");
 propTF = new JTextField( h.prop +
    "                                          "+
    "                  ");
 box6.add(propLabel);
 box6.add(propTF);
 Prop prop= new Prop();
 propTF.addActionListener(prop);

add(box6);

//************* newspaper
box7.add(news = new TextArea("Propositions and  matching " +
                    "generations \n" , 4, 80));
news.setFont(new Font("Courier", Font.PLAIN, 12));
news.setEditable(false);
add(box7);


}


//Inner class to deal with the purpose  button
class Purpose implements ActionListener
{
 public void actionPerformed(ActionEvent event)
 {
  JOptionPane.showMessageDialog
  (null,
   "The program composes logical propositions  " +
   "\nin conjunctive normal full form" +
   "\nand uses evolution to find  satisfying assignments."+
   "\nThe needed  number of generations and of individuals"
   + "\nto solve the task are reported." +
   "\nThe ladder job consist in solving a succession  " +
   "\nof propositions with   k variables,  " +
   "\nk ranging from one to a number decided"
```

```
   + "\nby the User." +
   "\nAlpha is the ratio of the number of clauses " +
   "\nto variables. So:" +
   "\nNumber of clauses = alpha * number of variables."
  );
 }
}


//If the newTry button is clicked upon,
//something must be done:
class NewTry implements ActionListener
{
 public void actionPerformed(ActionEvent event)
 {
  caseNumb = 1;
  task();
 }
}


//Reading instrument of mutation rate
class Slider implements ChangeListener
{
 public void stateChanged(ChangeEvent move)
  {
   mutRateInt = mutSlider.getValue();
   double mutR = mutRateInt;
   h.mutRate = mutR/100;
  }
}

//Inner class to handle events from the rec menu
private class RecHandler implements ActionListener
{
 public void actionPerformed( ActionEvent event )
 {
  //
  for ( int i = 0; i < 2; i++)
{
```

```
 // check for "Yes"
 if ( Rec[ 0 ].isSelected() )
 {
  h.recomb = true;
  break;
 }
 // check for "Not"
 if ( Rec[ 1 ].isSelected() )
 {
  h.recomb = false;
  break;
 }
   }
 }
}//end of recHandler

//Sensor of population size
class PopSize implements ActionListener
{
  public void actionPerformed(ActionEvent event)
  {
String s =  popSizeTF.getText();
s = purify(s);
if (s == "")
{
 JOptionPane.showMessageDialog(null,
    "Please, type a number");
}
else
{
 popSize =  new Integer(s).intValue();
 h.popSize = popSize;
}
  }
}



//Inner class to handle events from the show menu
```

```
private class ShowHandler implements ActionListener
{
 public void actionPerformed( ActionEvent event )
 {
  //
  for ( int i = 0; i < 2; i++)
{
 // check for "Yes"
 if ( Show[ 0 ].isSelected() )
  {
   h.show = true;
   break;
 }

// check for "Not"
if ( Show[ 1 ].isSelected() )
 {
  h.show = false;
  break;
 }
}
 }
}//end of ShowHandler

// Only numbers are accepted
private String purify(String s)
{
 String p = "";
 char c = 'c';
 for (int i = 0; i < s.length(); i++)
 {
  c = s.charAt(i);
  for(int j = 48; j< 58; j++)
  if (c == (char) (j)) p = p+c;
 }
 return p;
}

//The User can decide the number of clauses
class NClauses implements ActionListener
```

```
{
 public void actionPerformed(ActionEvent event)
{
 String st = nClausesTF.getText();
 st = purify(st);
 if (st.equals(""))
 {
  JOptionPane.showMessageDialog(null,
  "Please, type a number");
 }
 else
 {
  h.nClauses = new Integer(st).intValue();
  h.nClauses = h.nClauses;
  String str = " ";
  double num = h.nClauses;
  double den = h.nVar;
  double ad = num / den;
  Integer a = (int) Math.floor(ad);
  if (h.nClauses == a * den) str = "0";
  else
  {
   Integer b = (int) (ad*10) - a*10;
   str = b.toString();
  }
  alphaTF.setText(a.toString()+"." + str);
 }
}
}

 //The User can decide the number of variables
class NVar implements ActionListener
{
 public void actionPerformed(ActionEvent event)
{
 String st = nVarTF.getText();
 st = purify(st);
 if (st.equals(""))
 {
  JOptionPane.showMessageDialog(null,
```

```
  "Please, type a number");
 }
 else
 {
  h.nVar = new Integer(st).intValue();
  h.nVar = h.nVar;
  h.nClauses =  (int) (h.alpha * h.nVar);
  nClausesTF.setText(h.nClauses.toString());
 }
}
}

//Only numbers and a point are  accepted
private double decode(String s)
{
 double number = 0;
 int place = 0;
 boolean flag = true;
 char c = 'c';
 s = s.trim();

 for (int i = 0; i < s.length(); i++)
 {
  c = s.charAt(i);
  if ((c == '.') & (flag ))
 {
  flag = false;
  place = i;
 }
  for(int j = 48; j< 58; j++)
  if (c == (char) (j)) number  = number * 10 + (j-48);
 }
 if (place >0) number = number /
                  ( Math.pow(10, s.length() - place-1));
 return number;
}


//The User can decide the value of alpha
class Alpha implements ActionListener
```

```
{
 public void actionPerformed(ActionEvent event)
{
 String stAlpha = alphaTF.getText();
 h.alpha = decode(stAlpha);
 h.nClauses = (int) (h.alpha * h.nVar);
 nClausesTF.setText(h.nClauses.toString());
}
}

//A basic checking for correctness of st
private boolean check(String st1)
{
 boolean correct = true;
 st1 = st1.trim();
 int n = st1.length();
 char d = ' ' ;
 int  countPar = 0;
 int countO = 0;
 int countR = 0;
 int countAnd = 0;
 for (int i = 0; i < n; i++)
 {
  d = st1.charAt(i);
  if (d == '(') countPar = countPar +1;
  if (d == ')') countPar = countPar -1;

  if (d == 'A') countAnd = countAnd +1;
  if (d == 'O') countO = countO +1;
  if (d == 'R') countR = countR +1;
 }
 if ( (countPar == 0) & (countO == countR))
  {
h.nClauses = countAnd +1;
h.nVar = (countO / h.nClauses) + 1;
  }
 else
 {
  JOptionPane.showMessageDialog(null,
```

```
  "Please, type a full conjunctive form." +
  "\nUse the next conventions:" +
  "\n1)Clauses are in within parentheses." +
  "\n2)Clauses are separated by A."+
  "\n3)A is surrounded by one space from each side."+
  "\n4)Variables are denoted by a small w. " +
  "\n5)Variables are known by their position." +
  "\n6)All clauses have the same number of variables."+
  "\n7)Write OR in place of or." +
  "\n8)Write A instead or AND." +
  "\n9)Write N instead of Not" +
  "\nExample: (NwORw) A (wORNw) A (wORNw)" );
  correct = false;
 }
 return correct;
}

//Reader of the proposition
class Prop implements ActionListener
 {
  public void actionPerformed(ActionEvent event)
   {
 String st1 = propTF.getText();

 if (check(st1))
 {
 h.prop =  st1;
 caseNumb = 2;

 task();
 }

}
  }

 //Trigger of action
 public void go( )
 {
  ensureEventThread();
  caseNumb = 0;
```

```
 task();
}


//When the action goes on,
//everything must be withdrawn from the USER
//except the stop button
private void editionFalse()
{
 btnGo.setEnabled(false);
 nClausesTF.setEditable(false);
 nVarTF.setEditable(false);
 alphaTF.setEditable(false);
 propTF.setEditable(false);
 popSizeTF.setEditable(false);
 btnNewTry.setEnabled(false);
 btnStop.setEnabled(true);
 Rec[0].setEnabled(false);
 Rec[1].setEnabled(false);
 mutSlider.setEnabled(false);
}

//The USER acquires edition power
private void editionTrue()
{
 auxThread = null;
 btnGo.setEnabled(true);
 btnNewTry.setEnabled(true);
 nClausesTF.setEditable(true);
 nVarTF.setEditable(true);
 alphaTF.setEditable(true);
 popSizeTF.setEditable(true);
 propTF.setEditable(true);
 btnStop.setEnabled(false);
 Rec[0].setEnabled(true);
 Rec[1].setEnabled(true);
 mutSlider.setEnabled(true);
 }
```

```
 //The action is hung from an execution thread.
 //In that way, it can be stopped
 private void task()
 {
  //Edition is forbidden
  editionFalse();
  //The action is encapsulated
  Runnable watchRun = new Runnable()
  {
   public void run()
{
 w.news.setText("");
 try
  {
   s.go = true;
   switch (caseNumb)
   {
case 0 : //Ladder job
 {
  s.work();
  h.makeGraphic = true;
  s.repaint();
  break;
 }
        //randomTry
case 1 : s.oneProblem();break;
        //prop given by the User
case 2  : s.solveProblem(h.prop);break;
   }
   editionTrue();
  } catch (InterruptedException e)
  {
e.printStackTrace();
  }
  getReady( );
 }
};

 //Encapsulated action  is connected to the thread
 auxThread = new Thread(watchRun, "auxThread");
```

```
 auxThread.start();
}


 //If the execution is interrupted,
 //some housekeeping must be done.
 private void getReady( )
 {
  Runnable r = new Runnable()
   {
 public void run()
  {
   try {
    ensureEventThread();
    btnStop.setEnabled(false);
    btnGo.setEnabled(true);
   } catch (Exception x)
             {
              x.printStackTrace();
             }
   }
};
  SwingUtilities.invokeLater(r);
 }

//Am I working with a ghost or with my thread?
private void ensureEventThread()
{
 if (SwingUtilities.isEventDispatchThread())
 return;
 throw new RuntimeException("Undue calling");
}


//The task is halted
private void stop()
{
 ensureEventThread();
 if (auxThread != null)
{
```

```
 auxThread.interrupt();
 editionTrue();
 s.go = false;
 w.news.append("Job cancelled  \n");
}
}

//An unknown bug causes troubles
//that are partially remedied
//with some housekeeping.
private void buttons()
{
  btnPurpose.setVisible(false);
btnPurpose.setVisible(true);

 Show[0].setVisible(false);
 Show[1].setVisible(false);
 Show[0].setVisible(true);
 Show[1].setVisible(true);

 btnNewTry.setVisible(false);
 btnNewTry.setVisible(true);
  btnGo.setVisible(false);
  btnGo.setVisible(true);

 nClausesTF.setVisible(false);
 nClausesTF.setVisible(true);
 nVarTF.setVisible(false);
 nVarTF.setVisible(true);
 popSizeTF.setVisible(false);
 popSizeTF.setVisible(true);

 Rec[0].setVisible(false);
 Rec[0].setVisible(true);
 Rec[1].setVisible(false);
 Rec[1].setVisible(true);

 mutSlider.setVisible(false);
 mutSlider.setVisible(true);
 mutLabel.setVisible(false);
```

```
 mutLabel.setVisible(true);
 showL.setVisible(false);
 showL.setVisible(true);

 nClausesLabel.setVisible(false);
 nVarLabel.setVisible(false);
 popSizeLabel.setVisible(false);
 recL.setVisible(false);

 alphaLabel.setVisible(false);
 alphaLabel.setVisible(true);

 nClausesLabel.setVisible(true);
 nVarLabel.setVisible(true);
 popSizeLabel.setVisible(true);
 recL.setVisible(true);
 propLabel.setVisible(false);
 propLabel.setVisible(true);

     propTF.setVisible(false);
 propTF.setVisible(true);
 btnStop.setVisible(false);
 btnStop.setVisible(true);

}
}//End of class WGUI

}//End of main class
```

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

126 ***Exercise*** *Design an experiment to verify else reject the next propositions that will be taken by the Author as correct :*

1. *For $\alpha = 1/n$, one clause and n variables, all propositions are satisfiable. In effect, you have one single clause with n variables connected by OR, so one*

can give by instinct a value to the first variable to make the whole proposition true.

2. *As α grows from $1/n$ to infinite, the proportion of satisfiable propositions declines from one to zero. For a number of variables of 2 and $\alpha = 10$, almost all propositions evaluate to false. As the number of variables increases, the α that is needed to make most propositions false also increases.*

3. *There is a significant proportion of satisfiable propositions that reveal themselves in the very first trials, specially for low α. There are many possible valuations that evaluate those propositions to true.*

4. *There are values of α for which the proportion of satisfiable propositions equals that of the not satisfiable ones and for these values, one works in the exponential regime to solve a satisfiable proposition. This happens, for instance, when the number of variables = 4, $\alpha = 10$, number of clauses = 40, population size = 4.*

5. *Warning: to exactly follow the relation of α with satisfiability we need algorithms that produce exhaustive listing of possible valuations with freedom from repetitions. Our algorithms are not like that: the registered proportion of satisfiable clauses may depend on, say, the mutation rate. So, we can not claim exactness but our probabilistic insights seem to be correct as comparison with exact results (Monasson et al, 1999) would show.*

127 ***The yes-no asymmetry***

When a proposition is not satisfiable, we need to check all possible truth assignments to verify that, but when a proposition is satisfiable, all we need is to show a satisfying assignment that eventually can be found in early trials. So, SAT is overloaded more by unsatisfiable propositions than for satisfiable ones. But, how much?

The next table shows an experimental sampling that points to the astonishing fact that easy to decide propositions live side by side with others that need more than 2^{nVar} trials to be decided by our algorithms, where *nVar* is the number of variables. Assorted combinations were chosen to make the probability of satisfaction of almost equal value to that of not satisfaction. The symbolism used to encode the information in the last column is the following: a number, say, 4, means the number of trials that were necessary to find a satisfiable truth assignment to a proposition generated at random in within the class declared in the first columns. The char x means that we were unable to find a satisfying assignment and so we declare the generated proposition as false. Parameters: population size = 4, mutation rate =

0.51, Recombination = false. Number of generations to decide that a proposition is false = 2^{nVar}. Other conventions: *nClauses* means number of clauses, α is the ratio of clauses to variables.

nVar	α	nClauses	2^{nVar}	Sample of results
2	3	6	4	4,4,x,4,x,x,x,4,4,,x,4
3	6.3	19	8	4,x,4,28,16,4,x,4,x,x,x,8
4	12.5	50	16	4,20,x,x,x,x,x,44,12,4,x
5	22	110	32	x,x,x,x,32,36,60,x,x,112,x
6	41.6	250	64	40,x,x,x,196,32,x,200,x
7	80	560	128	x,x,4,x,x,40,184,20

We are used to think that SAT is solved in the exponential regime, but this vision is too coarse for modern standards: what is clear is that there are propositions easy to decide for our algorithms while others demand lengthy operations. So, we prefer to say that SAT is decided in the exponential regime for the worst case meaning the extreme worst case. But there are a significant proportion of instances of SAT that are decided at once, almost at once or after a not too long effort.

To improve our vision of SAT from the stand point of evolution, let us study now a variant of SAT that is known as MAX-K-SAT. In this setting, we investigate the maximum number of clauses that are satisfied by an assignment.

128 *The MAX-K-SAT gradualistic attack over SAT*

We have dealt with SAT using an all-else-nothing semantics. In effect, we have defined the fitness of a valuation as one if the valuation satisfies the proposition and zero otherwise. This discontinuous semantics can be weakened for conjunctive forms to get a gradualistic version as follows: we redefine the fitness of a valuation as the number of clauses in the proposition that are satisfied by the valuation. For instance, if we have the proposition $(p \vee q) \wedge (\sim p \vee q)$, the fitness of the valuation TF, true for p and false for q, is 1 because with that valuation, the clause $(p \vee q)$ gets true while $(\sim p \vee q)$ is false. In the old semantics, the fitness should have been 0 because the whole proposition gets false with the given valuation. Our new insight give rise to the next definition:

MAX-K-SAT=Maximum K-satisfiability is the problem whose goal is to find an assignment that maximizes the number of satisfied clauses for a given conjunctive normal formula over N variables, where each clause contains at most K variables and each chosen variable can be negated. We see that a proposition is satisfiable if and only if MAX-K-SAT ends with a valuation that satisfies all clauses,

i.e. if the maximal fitness in the population is equal to the number of clauses in the proposition. So, MAX-K-SAT defines a gradualistic extension of SAT because the fitness can effectively take on intermediate values that lie in within the extreme ones associated to the old semantics.

129 *Our code*

The next code uses evolution to solve MAX-K-SAT and reports the distribution of fitness. Given a proposition, the fitness of a valuation is defined as the number of clauses satisfied by it.

```
/* Program D129, MAXKSAT.
 *
 * We use evolution to solve MAX-K-SAT and to report
 * a distribution of fitness in each generation.
 *
 * SAT is a model  that captures the idea
 * that complexity arises when one must do things
 * subject to  incompatible restrictions:
 * do this but not that.
 *
 * SAT or Satisfaction is the problem of
 * knowing whether or not a given logical proposition
 * is satisfiable, i.e. if there exists a truth
 * assignment that evaluates the proposition to true.
 *
 * We work with  conjunctive normal  forms.
 * Variables are joined by and And operator.
 * Clauses are joined by an OR operator.
 * Every clause contains k variables.
 * Variables might be negated or not.
 * The prob of negation of a variable is 0.5.
 *
 * Random   equiprobable truth values are assigned
 * to the  variables and
 * the prop is evaluated to true else false.
 *
```

```
 * A proposition is declared to be unsatisfiable if
 * after 2^(number of Variables) generations no
 * satisfiable assignment has  been found.
 *
 * MAX-K-SAT is an optimization problem, whose goal is to
 * find an assignment that maximizes the number of
 * satisfied clauses for a  given  conjunctive normal
 * formula, where each clause contains at most kVar
 * variables chosen among a set of nVar variables.
 * The fitness of a valuation is given by
 * the number of satisfied clauses.
 *
 * Since the fitness can take on intermediate values
 * in within extreme ones, this   setting defines
 * a gradualistic embedding of SAT.
 *
 * Warning:  expressions of the type
 * (NwORNw) A (wORw) A (wORNw) A (NwORw)
 * are not propositions in this program. They are  schemes:
 * In the clause (NwORNw), the first w must be replaced
 * by a variable and the second w for another variable,
 * but
 * the first w  could be replaced by the third variable
 * among a list of four and the second w could be replaced
 * by the fourth variable  taken from the same list.
 * And in other clause, replacements could be otherwise.
 */

import java.awt.BorderLayout;
import java.awt.Font;
import java.awt.TextArea;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.BitSet;
import java.util.Random;

import javax.swing.Box;
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JFrame;
```

```
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.JSlider;
import javax.swing.JTextField;
import javax.swing.SwingConstants;
import javax.swing.SwingUtilities;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

public class MAXKSAT
{
 private static final long serialVersionUID = 1L;
 //MAXKSAT is instantiated
 private static MAXKSAT t = new MAXKSAT();
 //Some relevant information is packed into a class. This
 //allows stability of memories against thread threats.
 Job h = new Job();
 // This thread is used to enable the User to
 //terminate  lenghty tasks.
 private volatile Thread auxThread;
 //Processing unit is instantiated
 private static Distribution s = t.new  Distribution();
 //Graphic User Interface is instantiated
 private static GUI w =  t.new GUI();

 //The JFrame which shows everything
 // is constructed with a title
 private static JFrame f =
           new JFrame("MaxKSAT: conjunctive subclass");


 public static void main(String[] args)
 {
  //WGUI is added to the JFrame f
  f.add( w, BorderLayout.CENTER );
  f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  //Dimensions of the JFrame
  f.setSize( 650, 600 );
```

```
  f.setVisible( true );
  f.add( s, BorderLayout.CENTER );
  //This helps Java to show the GUI
  f.setSize( 650, 670);

 }

//*******************************************
//****************JOB CLASS ******************
//*******************************************

//This auxiliary class unites some fundamental
//variables into an object
private  class Job
{
// Number of propositions
 Integer nProps;
 //Total number of boolean variables
 Integer nVar;
 //Number of boolean variables per clause
 Integer kVar;
 //ratio of number of clauses to
 //total number  of variables
 double  alpha;
 //number of clauses
 Integer nClauses;
 //population size
 Integer popSize;
 //mutation rate per bit per generation
 Double mutRate;
 //to show or not to show actual computations
 boolean show;
 //recombination: yes else not
 boolean recomb;

 //A proposition is declared unsatisfiable if after maxGen
 //generations no satisfying assignment has been found.
 long maxGen;
```

```
 //Mandatory initialization
 Job( )
{
 //Number of propositions
 nProps = 3;
 //Number of boolean variables
 nVar = 3;
     // Number of variables in a clause
 kVar = 2; //must be less or equal than nVar
 //Ratio of numb of clauses to that of variables
 alpha = 2.0;
 //Number of clauses
 nClauses = (int) alpha * nVar;
 //Population size
 popSize = 4;
 //Mutation rate
 mutRate = 0.2;
 //To show or not to show actual work
 show = false;
 //Recombination: yes else not.
 recomb = false;
 maxGen  = (int) Math.ceil( Math.pow(2, nVar));
}




 //Converts a double number into a string
 private String ToString(double numb)
 {
  Integer a = (int) (numb*10);
  Integer b = (int) numb;
  Integer c = a - 10*b;
  b  = ( b % 10);
  String str = b.toString() + "." + c.toString();
  return str;
 }

 }//end of class Job
```

```
//*******************************************
//*************** CLASS DISTRIBUTION ********
//*******************************************

//This class responds for the actual computation
//of the distribution of fitness

private class Distribution extends JPanel
{

 private static final long serialVersionUID = 1L;

 // Turn on  the random generator
 Random r  = new Random();

 // Boolean condition to continue else halt
 private    boolean go = true;

 //Max number of allowed variables

 private   int   maxV = 1000;
 private   int maxClauses  = 10000;
 //Each clause contains a subset of variables.
 private   boolean chosenVar[][] =
                        new boolean[maxClauses][maxV];
 private   int listChosen[][] = new int[maxClauses][maxV];
 //Propositions in formation
 private   String a,b,c;
 //Boolean flag
 private   boolean problem = true;
 //if the prop is satisfiable, sat = true else false.
 private   boolean sat = false;


 //A population of binary numbers is considered.
 //The number of individuals is constant
 private   int maxNumbInd = 10000;
 private   BitSet Ind1[]   = new BitSet[maxNumbInd];
```

```
 private   BitSet Ind2[]   = new BitSet[maxNumbInd];
 private   int FitnessInd[]
                                = new int[maxNumbInd];
 private   int OrderInd[]  = new int[maxNumbInd];
 private   int minFitness, maxFitness;
 private   int gen ;
 private   int distri[]   = new int[8];


 //Not operators are inserted with prob 0.5
 private    String negations( )
 {
 for(int j=0; j<a.length();j++)
   {
if (a.charAt(j)=='w')
 {
  int d = r.nextInt(2);
  if (d == 0)
   a = a.substring(0, j)+ "f" + a.substring(j+1);
 }
   }
a = a.replace("f", "Nw");
  return a;
 }

 //Composes a random normal disjunctive full w-proposition
 //with h.nClauses clauses. Each clause has h.kVar variables
 //that are sampled from a set with h.nVar variables.
 //The NOT operator is included for each w
 //with probability 0.5.
 private    String lProposition()
 {
  //w.news.append("Construction of the w-proposition \n");
  //Proposition is initialized.
  a = "(w";
  for(int i= 0; i < h.nClauses-1; i++)
  {
   for(int j = 0; j < h.kVar-1; j++)
     a = a.concat("ORw");
   a= a.concat(") A (w");
```

```
 }
 for(int j = 0; j < h.kVar-1; j++)
    a = a.concat("ORw");
 a= a.concat(")");
 if (h.show)
 w.news.append("w-scheme without NOT = " + a + "\n");
 negations();
 if (h.show)
 w.news.append("w-scheme with NOTs   = " + a + "n");
 return a;
}

//For each clause, a sample of h.kVar variables is made
//among a set of h.nVar variables
private    void chooseKVars()
{

 // w.news.append("chooseKVars \n");
 for(int k = 0; k < h.nClauses; k++)
   for(int i = 0; i < h.nVar; i++)
    chosenVar[k][i] = false;
 for(int k = 0; k < h.nClauses; k++)
 {
  int n = 0;
  //w.news.append("\n");
  //w.news.append("k = "  + k + " \n");
  while (n < h.kVar)
  {
   int j = r.nextInt(h.nVar);
   if (chosenVar[k][j] == false)
 {
  chosenVar[k][j] =true;
  //w.news.append(j + " ");
  n = n+1;
 }
  }
 }
 //Chosen variables are listed
 for(int k = 0; k < h.nClauses; k++)
 {int  n = 0;
```

```
   for(int i = 0; i < h.nVar; i++)
   {
   if    ( chosenVar[k][i] == true)
  {
listChosen[k][n] = i;
n = n+1;
// w.news.append(i + " ");
  }
   }
   //w.news.append("\n");
  }
 }


 //Each w of the w-proposition is replaced by one of
 //the variables. All variables appear in each clause.
 //Variables are not letters;
 //they are numbers from 0 to nVar-1;
 private    String proposition(String a)
 {
  //w.news.append(" \n");
  //w.news.append("l-prop --> proposition \n");
  int u = 0;
  //The sample of k vars is made for each clause;
  chooseKVars();
  int k = 0;
  for(int j=0; j<a.length();j++)
  {
   if (a.charAt(j)=='w')
{

  Integer vari = listChosen[k][u];
  String s = vari.toString();
  a = a.substring(0, j)+ s + a.substring(j+1);

  u = u +1;
  if (u==h.kVar)
  {
  u=0;
  k = k+1;
```

```
  }
}

   //w.news.append("k = " + k + "u = " + u + "\n");
  }
  if (h.show)
  w.news.append("prop356               = " + a + "\n");
  return a;
 }


 private    void printingJob(BitSet s)
 {
   w.news.append("\n");
   w.news.append("Truth evaluation begins \n");
   w.news.append("Scheme = " + b + "\n");
   w.news.append("Tested valuation = " + decode(s) + "\n");
   w.news.append("Each clause has "+h.kVar );
   w.news.append(" variables chosen among " + h.nVar+"\n");
   /*
   w.news.append("Chosen variable = 1, unchosen 0: \n");
   for(int k = 0; k < h.nClauses; k++)
{
  w.news.append("Clause " + k + ": ");
  for(int i = 0; i < h.nVar; i++)
   {
 if (chosenVar[k][i] == true)
 w.news.append("1");
  else w.news.append("0");
   }

  w.news.append( "\n");
   }
   */
   w.news.append("List of chosen variables \n");
   for(int k = 0; k < h.nClauses; k++)
{
  w.news.append("Clause " + k + ": ");
  for(int i = 0; i < h.kVar; i++)
  w.news.append(listChosen[k][i] + " ");
```

```
  w.news.append("Valuation: ");
  for(int i = 0; i < h.kVar; i++)
   {
 if (s.get(listChosen[k][i]))
w.news.append("T");
 else
 w.news.append("F");
   }
  w.news.append("\n");
  }
 }

//The proposition is evaluated
 private     String evaluation(BitSet s)
 {
  b = c;
  if (h.show) printingJob(s);
  int u = 0;
  int k = 0;
  for(int j=0; j<b.length();j++)
   {
    if (b.charAt(j)==’w’)
     {
       if (s.get(listChosen[k][u]) == true)
    b = b.substring(0, j)+ "T" + b.substring(j+1);
   else
   b = b.substring(0, j)+ "F" + b.substring(j+1);
   u = u +1;
   if (u == h.kVar)
{
   u=0;
   k = k+1;
}
     }
   }
  if (h.show)
  w.news.append("Evaluation = " + b + "\n");
   return b;
 }
```

```
//The proposition is evaluated until the end
 //Not operators are evaluated at the
 //very beginning for they apply over the next
 //to the right variable.
 private     String simplification(String b)
 {
 // while ( b.length()>1 )
 boolean progress = true;

 while (progress)
   {
     String f = ""+b;
 b=  b.replace("NT","F");
 b=  b.replace("NF","T");
 b=  b.replace("TORT","T");
 b=  b.replace("TORF","T");
 b=  b.replace("FORT","T");
 b=  b.replace("FORF","F");

 b=  b.replace("(T)","T");
 b=  b.replace("(F)","F");


 if (h.show)
 w.news.append("Truth value = " + b + "\n");

 if (b.equals(f)) progress = false;
   }
  return b;
 }

//Converts a BitSet value into a String
   private String decode(BitSet t)
 {
  String s= "";
  for(int i =0; i < h.nVar; i++)
   {
if (t.get(i)==true) s = s + "T";
 else s = s + "F";
   }
```

```
  return s;
 }


 //The distribution is set everywhere to  zero.
 private   void resetDistri( )
 {
for(int i = 0; i<=7; i++)
distri[i] = 0;
 }


 //Our pop is formed by valuations.
 //Ind1 is a population of binary strings.
 //The BitSet class is used to instantiate
 //binary strings as individuals
 private   void initializePop()
 {
  BitSet s = new BitSet();
  //Ind1 is false everywhere
  for(int i = 0; i < h.popSize; i++)
Ind1[i] = (BitSet) s.clone();
  //Reinitialization with random truth values.
  for(int i = 0; i < h.popSize; i++)
  {
   for(int j = 0; j < h.nVar; j++)
{
   int l = r.nextInt(2);
   if (l == 1)
    Ind1[i].set(j); //the j-th bit is true
}
  //w.news.appendln("Ind1 : " + decode(Ind1[i]));
  }
 }


// Does an individual   satisfy  the target?
   private int assessInd(BitSet s)
 {
  String b = evaluation(s);
```

```
 b=simplification(b);
 int  fitness=0;
 for(int i = 0; i< b.length(); i++)
 {
  if (b.charAt(i) == 'T') fitness = fitness +1;
 }
 if (h.show)
 {
  w.news.append( "fitness = " + fitness +
                      " of " + h.nClauses + "\n");
 }
 return fitness;
}

private   void updateDistri()
{
 resetDistri();
 double nC = h.nClauses;
 double den = nC / 7;
 int k = 0;
 for(int i = 0; i< h.popSize; i++)
  {
 double f = FitnessInd[i];

 k = (int) Math.floor( f/ den);
distri[k] = distri[k] + 1;

  }
 for(int i = 0; i<=7; i++)
  w.news.append( distri[i] + " ");
 double inf = minFitness;
 inf = inf / h.nClauses;
 double sup = maxFitness;
 sup = sup / h.nClauses;
 String s1  = String.valueOf(inf);
 int n = s1.length();
 if (n>4)
 s1 = s1.substring(0,4);
 String s2  = String.valueOf(sup);
  n = s2.length();
```

```
  if (n>4)
   s2 = s2.substring(0,4);
  w.news.append(
  "Min = " + minFitness + "(" + s1 + ")  Max = " +
           maxFitness + "("+  s2 + ")\n");
 }

//Individuals are sorted by fitness.
//the fittest is the first, with minimum error
private   boolean sortingInd()
 {
  int   Champ;

  //FitnessIndc[] is a copy of FitnessInd[]
  //used as workbench
  int FitnessIndc[];
  FitnessInd= new int[maxNumbInd];
  FitnessIndc= new int[maxNumbInd];
  for(int i = 0; i< h.popSize; i++)
   {
    FitnessInd[i] =0;
    FitnessIndc[i] =0;
   }
  for(int i = 0; i< h.popSize; i++)
   {
    FitnessInd[i] = assessInd(Ind1[i]);
    FitnessIndc[i] = FitnessInd[i];
    //w.news.appendln("Fitness i " + FitnessInd[i]);
   }

  //We sort individuals by fitness
  //0 is minimum fitness
  for(int i = 0; i< h.popSize;i++)
  {
   Champ = 0;
   for(int j = 0; j< h.popSize;j++)
   if (FitnessIndc[j] > FitnessIndc[Champ]) Champ = j;
   //The array OrderInd orders ind by fitness.
   //The fittest is number zero
   OrderInd[i] = Champ;
```

```
   FitnessIndc[Champ] = 0;
  }
  maxFitness = FitnessInd[OrderInd[0]];
  minFitness = FitnessInd[OrderInd[h.popSize-1]];
  updateDistri();
  /*
  for(int j = 0; j< h.popSize;j++)
  w.news.appendln("Order = " +  OrderInd[j] + " "
                + FitnessInd[OrderInd[j]]);
  */
 if (maxFitness == h.nClauses)
 {
  problem = false;
  sat = true;
  // w.news.appendln("Champ = " + Ind1[OrderInd[0]]);
  b = c;
 }

 return problem;
 }

// The new generation is made
 private    void newGenerationInd()
 {
  int   i=0;
  //The fittest individuals are reproduced
  // numbCopies times
  int numbCopies = 10;
  int   k=0;  //rank, 0 is the highest
  while (i<h.popSize)
  {
   int counter = numbCopies;
   while ((counter>0) & (i<h.popSize))
   {
//w.news.appendln(i + "" + l);
Ind2[i] = (BitSet) Ind1[OrderInd[k]].clone();
    //w.news.appendln(i + " " + counter + " " + k);
i++;
    counter= counter-1;
   }
```

```
 k++;
}
for(int j = 0; j<h.popSize; j++)
 {
  Ind1[j] = (BitSet) Ind2[j].clone();
   /*  w.news.appendln( j+ " Ind1 "
                                + (BitSet) Ind1[j]);
    */
 }
}


// Recombination is enabled
private   BitSet recInd(int l)
{
 //We choose a partner at random
 int k = r.nextInt(h.popSize);
 //b is false everywhere
 BitSet b = new BitSet();
 //b is the recombinant of l and k
 for(int j = 0; j < h.nVar; j++)
  {
   int m = r.nextInt(2);
   if (m == 1) //take the bit from l
 {
 if (Ind1[l].get(j)== true ) b.set(j);
 }
   if (m == 0)//take the bit from k
{
 if (Ind1[k].get(j)== true ) b.set(j);
    }
  }
 return b;
}


// The definitive population is a recombinant one.
private   void recombinationInd()
{
 //Ind2 is a copy of Ind1
```

```
  for(int i=0; i<h.popSize; i++)
  Ind2[i] =  (BitSet) Ind1[i].clone();
  //Ind2 is a recombinant version of Ind1
  for(int l=0; l<h.popSize; l++)
  Ind2[l] = recInd(l );
  //Ind2 is transferred to Ind1
  for(int i=0; i<h.popSize; i++)
  Ind1[i] = (BitSet) Ind2[i].clone() ;
 }



 // Individual  a is mutated at a random place.
 private   BitSet pointMutationInd(BitSet a)
 {
  int l = r.nextInt(h.nVar);
  a.flip(l);
  return a;
 }

 // Individuals are mutated
 private   void mutationInd()
 {
  for(int i=0; i<h.popSize; i++)
   {
    double p= r.nextDouble();
if (p < h.mutRate)
      Ind1[i] = pointMutationInd(Ind1[i]) ;
   }
 }

 //A single generation is calculated
 private   void  oneGeneration()throws InterruptedException
 {
  //The new generation is produced
  //as copies of the fittest
  newGenerationInd();
  if (h.recomb) recombinationInd();
  //The new population is subjected to mutation
  mutationInd();
```

```
  gen++;
  w.news.append("Gen = " + gen + ": ");
  problem = sortingInd();
  if (gen == h.maxGen)
{
 problem = false;
 sat = false;
}
 }

 public   void report1()
 {
   w.news.append("\n");
   w.news.append("PARAMETERS \n");
   w.news.append("Number of propositions = " +
                       h.nProps + " \n");
   w.news.append("Number of Variables per clause = " +
           h.kVar + "\n");
   w.news.append("Total number of variables  = " +
           h.nVar+ "\n");
   w.news.append("alpha " + h.alpha + "\n");
   w.news.append("Number of clauses = " +
                       h.nClauses + "\n");
   w.news.append("Population size " + h.popSize + "\n");

   w.news.append("Mutation rate/bit/generation " +
                       h.mutRate + "\n");
   w.news.append("Recombination = " +
           h.recomb + "\n");
   w.news.append("\n");
 }


 public   void work()
 {
  report1();
  double nSat = 0;
  w.news.append("A prop is declared unsatisfiable when \n");
  w.news.append("no satisfying assignment is found\n");
  w.news.append("after "+  h.maxGen + " generations \n \n");
```

```
  //Please, pay attention to the next strange loop:
  //it contains a condition for halting
  //that can be manipulated by the User
  for(int i = 0; ( (i < h.nProps) & (go)); i++)
  {
   w.news.append("********** \n");
   w.news.append("Prop numb = " + i+ "\n");
   w.news.append("Number of clauses = " + h.nClauses+ "\n");
   w.news.append("Fitness = number of" +
                               " satisfied clauses \n");


   problem = true;
   h.maxGen = (int) Math.ceil( Math.pow(2, h.nVar));
   problem = true;
   sat = false;
   b = lProposition();
   c = b;
   w.news.append("Prop = " + c + "\n");
   proposition(b);

   initializePop();
   w.news.append("Distribution of fitness in 7 " +
          "classes, "  + h.popSize + " individuals \n");
   w.news.append("Min and max values reported with " +
                               "their percentages \n" );
   gen = 0;
   while   ((problem) & (go) )
   {
    try
      {
oneGeneration();
  } catch (InterruptedException e)
    {
 e.printStackTrace();
    }
   }
   w.news.append("MaxFitness = " + maxFitness + "\n");
   if (sat == true)
   {
```

```
      nSat = nSat +1;
      w.news.append("Satisfiable \n ");
    }
    else w.news.append("Unsatisfiable \n");

  }
  double p = 1-nSat / h.nProps;
  w.news.append("Prob of Un-Satisfiability = " + p + "\n");


 }
}



//*****************************************
//****************GUI CLASS ******************
//*****************************************

//This class operates the graphic interface GUI

private class GUI extends JPanel
{
 private static final long serialVersionUID = 1L;
 private JTextField nPropsTF;
 private JTextField nClausesTF;
 private JTextField nVarTF;
 private JTextField kVarTF;
 private JTextField alphaTF;
 private JTextField popSizeTF;

 private int mutRateInt = (int) (  100*h.mutRate);
 private Integer popSize = h.popSize;

 private JLabel showL;
 private JLabel nPropsLabel;
 private JLabel nVarLabel;
 private JLabel kVarLabel;
 private JLabel alphaLabel;
 private JLabel nClausesLabel;
```

```
private JLabel popSizeLabel;
private JLabel mutLabel;
private JLabel recL;
private JLabel warning ;


//Control button
private JRadioButtonMenuItem Rec[];
private ButtonGroup RecButtonGroup;

private JSlider mutSlider;
private JButton btnPurpose;
private JButton btnNewTry;
//private JButton btnGo;
private JButton btnStop;
private TextArea news;
private JRadioButtonMenuItem Show[];
private ButtonGroup ShowButtonGroup;


//Organization
private Box box1 = Box.createHorizontalBox();
private Box box2 = Box.createHorizontalBox();
private Box box3 = Box.createHorizontalBox();
private Box box4 = Box.createHorizontalBox();
private Box box5 = Box.createHorizontalBox();



// Planning of the   screen
private   GUI( )
{


 //  ************ button Purpose
 btnPurpose = new JButton("Purpose");
 box1.add (btnPurpose);
 Purpose purpose = new Purpose ();
 btnPurpose.addActionListener(purpose);
```

```
//************   Show work

  showL = new JLabel(" Show work " );
  box1.add(showL);
  String ShowNames[]  = { "Yes", "No" };
  // Creates radio button menu items for Yes-No options
  Show  = new JRadioButtonMenuItem[ ShowNames.length ];
  ShowButtonGroup = new ButtonGroup();

  ShowHandler showHandler =  new ShowHandler();
  //Creates show radio button menu items
  for ( int count = 0; count < Show.length; count++ )
{
 Show[ count ]
   = new JRadioButtonMenuItem( ShowNames[ count ] );
 box1.add( Show[ count ] );
 ShowButtonGroup.add( Show[ count ] );
 //Calling of inner class showHandler
 Show[ count ].addActionListener( showHandler );
}
  //Initial setting : work is not shown;
  Show[ 1 ].setSelected( true );

//************ button NewTry
  btnNewTry = new JButton("NEW TRY");
  box1.add (btnNewTry);
  NewTry newTry = new NewTry ();
  btnNewTry.addActionListener(newTry);
  add(box1);

  // ************ button Stop
  btnStop = new JButton("STOP");
  box1.add (btnStop);
  btnStop.setEnabled(false);
  btnStop.addActionListener
   (
   new ActionListener()
    {
```

```
     public void actionPerformed(ActionEvent e)
      {
       stop();
      }
    }
   );
  add(box1);

//************ txtField h.nProps
  nPropsLabel= new JLabel(
     "Number of propositions ");
  box2.add(nPropsLabel);
  nPropsTF = new JTextField(h.nProps.toString()+
                                "        " );
  box2.add(nPropsTF);
  NProps nProps = new NProps();
  nPropsTF.addActionListener(nProps);



//************ txtField h.kVar
  kVarLabel= new JLabel(
     "Number of variables per clause");
  box2.add(kVarLabel);
  kVarTF = new JTextField(h.kVar.toString()+
                                "        " );
  box2.add(kVarTF);
  KVar kVar = new KVar();
  kVarTF.addActionListener(kVar);
  add(box2);


  //************ txtField h.nVar
  nVarLabel= new JLabel(
     "Total number of variables");
  box3.add(nVarLabel);
  nVarTF = new JTextField(h.nVar.toString()+
                                "        " );
  box3.add(nVarTF);
  NVar nVar = new NVar();
```

```
 nVarTF.addActionListener(nVar);



 //************ txtField h.alpha
 alphaLabel= new JLabel( "Alpha  ");
 box3.add(alphaLabel);
 alphaTF = new JTextField(h.ToString(h.alpha)+
                "                     " );
 box3.add(alphaTF);
 Alpha alpha = new Alpha();
 alphaTF.addActionListener(alpha);




//  ************ txtField h.nClauses
nClausesLabel= new JLabel(
  "Number of clauses ");
box3.add(nClausesLabel);
nClausesTF = new JTextField(h.nClauses.toString()+
                           "        " );
box3.add(nClausesTF);
NClauses nClauses = new NClauses();
nClausesTF.addActionListener(nClauses);



//  ************ txtField h.popSize
popSizeLabel= new JLabel("Population size");
box3.add(popSizeLabel);
popSizeTF = new JTextField("     " + h.popSize.toString());
box3.add(popSizeTF );
PopSize popSize = new PopSize();
popSizeTF.addActionListener(popSize);

add(box3);
```

```
//***************** Mutation slider

 mutLabel= new JLabel(
                  "Mutation rate/bit/generation");
 box4.add(mutLabel);
 mutSlider = new JSlider(SwingConstants.HORIZONTAL,
                                          0, 100,10);
 mutSlider.setMajorTickSpacing(10);
 mutSlider.setValue( mutRateInt );
 mutSlider.setPaintTicks( true );
 box4.add(mutSlider);
 Slider slider = new Slider();
 mutSlider.addChangeListener(slider);
 add(box4);


 //  ************   Recombination

 recL = new JLabel("Recombination " );
 box5.add(recL);
 String RecNames[]  = { "Yes", "No" };
 // Creates radio button menu items for rec options
 Rec  = new JRadioButtonMenuItem[ RecNames.length ];
 RecButtonGroup = new ButtonGroup();

 RecHandler recHandler =  new RecHandler();
 //Creates rec radio button menu items
 for ( int count = 0; count < Rec.length; count++ )
  {
   Rec[ count ]
 = new JRadioButtonMenuItem( RecNames[ count ] );
   box5.add( Rec[ count ] );
   RecButtonGroup.add( Rec[ count ] );
   //Calling of inner class recHandler
   Rec[ count ].addActionListener( recHandler );
  }
 //Initial setting: no recombination;
 Rec[ 1 ].setSelected( true );
 add(box5);
```

```
//************* newspaper

add(news = new TextArea("Distribution of fitness " +
             "in K-MAX-SAT conjunctive family." +
             "\n" , 25, 80));
news.append("Max number of Generations = "
                            + h.maxGen + "\n");
news.setFont(new Font("Courier", Font.PLAIN, 12));
news.setEditable(false);

//  ************ End with enter
warning = new JLabel("To modify parameters," +
                     " end with enter." );
add(warning);

repaint();

}


//Inner class to deal with the purpose  button
class Purpose implements ActionListener
{
 public void actionPerformed(ActionEvent event)
 {
  JOptionPane.showMessageDialog
  (null,
   "We use evolution to solve MAX-K-SAT," +
   "\nwhose goal is to find an assignment that " +
   "\nmaximizes the number of satisfied clauses " +
   "\nfor a  given conjunctive normal formula," +
   "\nwhere each clause contains at most K variables " +
   "\nchosen among a set of nVar variables." +
   "\nThe prob of negation of a variable is 0.5." +
   "\nRandom   equiprobable truth values are assigned " +
   "\nto the  variables and each clause is evaluated " +
```

```
   "\nto true else false. The fitness of a valuation" +
   "\nis given by the number of satisfied clauses."+
   "\n" +
   "\nSince the fitness can take on intermediate values"+
   "\nin within extreme ones, this   setting defines"+
   "\na gradualistic embedding of SAT." +
   "\n" +
   "\nA proposition is declared to be unsatisfiable if"+
   "\nafter 2^(number of Variables) generations no"+
   "\nsatisfiable assignment has  been found."+
   "\n"+
   "\nThe program reports the distribution of fitness " +
   "\nin seven classes in each generation."
  );
 }
}


//If the newTry button is clicked upon,
//something must be done:
class NewTry implements ActionListener
{
 public void actionPerformed(ActionEvent event)
 {
  task();
 }
}


//Reading instrument of mutation rate
class Slider implements ChangeListener
{
 public void stateChanged(ChangeEvent move)
  {
   mutRateInt = mutSlider.getValue();
   double mutR = mutRateInt;
   h.mutRate = mutR/100;
  }
}
```

```
//Inner class to handle events from the rec menu
private class RecHandler implements ActionListener
{
 public void actionPerformed( ActionEvent event )
 {
  //
  for ( int i = 0; i < 2; i++)
{
 // check for "Yes"
 if ( Rec[ 0 ].isSelected() )
 {
  h.recomb = true;
  break;
 }
 // check for "Not"
 if ( Rec[ 1 ].isSelected() )
 {
  h.recomb = false;
  break;
 }
   }
 }
}//end of recHandler

//Sensor of population size
class PopSize implements ActionListener
{
  public void actionPerformed(ActionEvent event)
  {
String s =  popSizeTF.getText();
s = purify(s);
if (s == "")
{
 JOptionPane.showMessageDialog(null,
    "Please, type a number");
}
else
{
 popSize =  new Integer(s).intValue();
 h.popSize = popSize;
```

```
}
  }
}



//Inner class to handle events from the show menu
private class ShowHandler implements ActionListener
{
 public void actionPerformed( ActionEvent event )
 {
  //
  for ( int i = 0; i < 2; i++)
{
 // check for "Yes"
 if ( Show[ 0 ].isSelected() )
  {
   h.show = true;
   break;
 }

// check for "Not"
if ( Show[ 1 ].isSelected() )
 {
  h.show = false;
  break;
 }
}
 }
}//end of ShowHandler

// Only numbers are accepted
private String purify(String s)
{
 String p = "";
 char c = 'c';
 for (int i = 0; i < s.length(); i++)
 {
  c = s.charAt(i);
```

```
  for(int j = 48; j< 58; j++)
  if (c == (char) (j)) p = p+c;
 }
 return p;
}

//The User can decide the number of propositions
class NProps implements ActionListener
{
 public void actionPerformed(ActionEvent event)
{
 String st = nPropsTF.getText();
 st = purify(st);
 if (st.equals(""))
 {
  JOptionPane.showMessageDialog(null,
  "Please, type a number");
 }
 else
 {
  h.nProps = new Integer(st).intValue();
 }
}
}


//The User can decide the number of clauses
class NClauses implements ActionListener
{
 public void actionPerformed(ActionEvent event)
{
 String st = nClausesTF.getText();
 st = purify(st);
 if (st.equals(""))
 {
  JOptionPane.showMessageDialog(null,
  "Please, type a number");
 }
 else
 {
```

```
  h.nClauses = new Integer(st).intValue();
  String str = " ";
  double num = h.nClauses;
  double den = h.nVar;
  h.alpha  = num / den;
  Integer a = (int) Math.floor(h.alpha);
  if (h.nClauses == a * den) str = "0";
  else
  {
   Integer b = (int) (h.alpha*10) - a*10;
   str = b.toString();
  }
  alphaTF.setText(a.toString()+"." + str);
 }
}
}

//The User can decide the number of variables per clause
class KVar implements ActionListener
{
 public void actionPerformed(ActionEvent event)
{
 String st = kVarTF.getText();
 st = purify(st);
 if (st.equals(""))
 {
  JOptionPane.showMessageDialog(null,
  "Please, type a number");
 }
 else
 {

  h.kVar = new Integer(st).intValue();
  if (h.kVar > h.nVar)
   {
    JOptionPane.showMessageDialog(null,
    "Number of variables per clause must be less than " +
    " or equal to the total number of variables");
    h.kVar = h.nVar;
    kVarTF.setText(h.kVar.toString());
```

```
   }
  h.maxGen = (int) Math.ceil( Math.pow(2, h.nVar));
 }
}
}

 //The User can decide the total number of variables
class NVar implements ActionListener
{
 public void actionPerformed(ActionEvent event)
{
 String st = nVarTF.getText();
 st = purify(st);
 if (st.equals(""))
 {
  JOptionPane.showMessageDialog(null,
  "Please, type a number");
 }
 else
 {
  h.nVar = new Integer(st).intValue();
  h.nVar = h.nVar;
  h.nClauses =  (int) (h.alpha * h.nVar);
  nClausesTF.setText(h.nClauses.toString());
 }
}
}

//Only numbers and a point are  accepted
private double decode(String s)
{
 double number = 0;
 int place = 0;
 boolean flag = true;
 char c = 'c';
 s = s.trim();

 for (int i = 0; i < s.length(); i++)
 {
  c = s.charAt(i);
```

```
  if ((c == '.') & (flag ))
 {
  flag = false;
  place = i;
 }
  for(int j = 48; j< 58; j++)
  if (c == (char) (j)) number  = number * 10 + (j-48);
 }
 if (place >0) number = number /
                 ( Math.pow(10, s.length() - place-1));
 return number;
}


//The User can decide the value of alpha
class Alpha implements ActionListener
{
 public void actionPerformed(ActionEvent event)
{
 String stAlpha = alphaTF.getText();
 h.alpha = decode(stAlpha);
 h.nClauses = (int) (h.alpha * h.nVar);
 nClausesTF.setText(h.nClauses.toString());
}
}


 //When the action goes on,
 //everything must be withdrawn from the User
 //except the stop button
 private void editionFalse()
 {
  nPropsTF.setEditable(false);
  nClausesTF.setEditable(false);
  nVarTF.setEditable(false);
  alphaTF.setEditable(false);
  popSizeTF.setEditable(false);
  btnNewTry.setEnabled(false);
  btnStop.setEnabled(true);
  Rec[0].setEnabled(false);
```

```
  Rec[1].setEnabled(false);
  mutSlider.setEnabled(false);
 }

 //The User acquires edition power
 private void editionTrue()
 {
  auxThread = null;
  nPropsTF.setEditable(true);
  btnNewTry.setEnabled(true);
  nClausesTF.setEditable(true);
  nVarTF.setEditable(true);
  alphaTF.setEditable(true);
  popSizeTF.setEditable(true);
  btnStop.setEnabled(false);
  Rec[0].setEnabled(true);
  Rec[1].setEnabled(true);
  mutSlider.setEnabled(true);
  }


 //The action is hung from an execution thread
 //that is property of the User.
 //In that way, it can be stopped
 private void task()
 {
  //Edition is forbidden
  editionFalse();
  //The action is encapsulated
  Runnable watchRun = new Runnable()
  {
   public void run()
{
 w.news.setText("");
 s.go = true;
 s.work();
 editionTrue();
 getReady( );
}
 };
```

```
 //Encapsulated action  is connected to the thread
 auxThread = new Thread(watchRun, "auxThread");
 auxThread.start();
}


 //If the execution is interrupted,
 //some housekeeping must be done.
 private void getReady( )
 {
  Runnable r = new Runnable()
   {
 public void run()
  {
   try {
    ensureEventThread();
    btnStop.setEnabled(false);
    //btnGo.setEnabled(true);
   } catch (Exception x)
             {
              x.printStackTrace();
             }
   }
};
  SwingUtilities.invokeLater(r);
 }

//Am I working with a ghost or with my thread?
private void ensureEventThread()
{
 if (SwingUtilities.isEventDispatchThread())
 return;
 throw new RuntimeException("Undue calling");
}


//The task is halted
private void stop()
{
```

```
 ensureEventThread();
 if (auxThread != null)
{
 auxThread.interrupt();
 editionTrue();
 s.go = false;
 w.news.append("\n+++++++++++++  \n");
 w.news.append("JOB CANCELLED  \n");
 w.news.append("+++++++++++++++\n\n");
}
}

}//End of class WGUI

}//End of main class
```

130 ***Exercise*** *Play with the code and try to understand it.*

131 ***Exercise*** *Devise and run some experiments to back or refute the next propositions that will be taken as true by the Author:*

1. Unsatisfiable propositions usually have as much as 6/7 of the number of clauses satisfied.

2. Evolution produces almost immediately valuations that satisfy as much as 6/7 of the number of clauses of anyone proposition.

132 ***The mystery of perfection***

We have seen that there are many problems that are very easy for evolution, that evolution is good at finding approximate solutions to some complex problems while it needs huge amounts of computations to perfectly solve other ones. We conclude that evolution has troubles to pass from a good approximate solution to perfection. To see what it means, let us consider the next example:

We have bladder. This is a recipient that allows us to pour out urine at will. Actually, bladder is not necessary: many persons are operated of the kidneys and must temporarily use a plastic tube that is directly connected to the kidneys to evacuate urine. So, evacuation of urine is necessary but this can be done without bladder, by using in most mammals, say, a conduct along their tail. So, bladder is

the brand of perfection. Where is the record fossil indicating that there is evolution from no bladder to a high quality bladder?

This example show that we can make very easily a point in biology from the existence of perfection whose mandatory evolution from viable but not very fine solutions is not seen in nature. As we saw in volumes II and III of this series, one can not professionally study this problem without plainly considering evolution as software developer. So, genetic programming is the field that must study the problem, a daunting problem that we cannot save for an undefined future.. ..

At the meantime, we contend that we have found a rationalization of the faith in the Creator that millions of people around the world have: the excel perfection of alive beings and of the fossil record is taken by them as a sign that they did not appeared by natural means and so their existence can be explained by God alone.

8.5 In hindsight

The **conviction** that the evolutionary theory is the correct explanation of our existence was 'proved' by the end of the XX century using the model problem Shakespeare. In fact, one can see that evolution is very good at solving this problem. We have in mind darwinian evolution without any trace of **lamarckism**, without helps from behalf of the developer. Therefore the Reader is invited to develop his or her own clean program to verify our claim because our program D107 has special aids. Else, study a program already made for the purpose, for instance, program B14, Weasel, Vol II of this series.

Model problem Shakespeare considers language as a string of symbols and nothing else. We think that this problem represents the retro-engineering problem of reproducing a known information. But we find no reason to consider that this vision enables the programming of automatic creation of artworks that might compete with those of Shakespeare. So, we deny the possibility to use the success of evolution to deal with this problem to claim that by the same token evolution is the correct explanation of the DNA strings that encodes for our anatomy and physiology.

Now, given that a human language is much more than a string of symbols, what must we include into a model to capture important elements of the complexity of creative works, without falling into intractable problems?

We claim that SAT, the satisfaction problem of mathematical logic, is perfect: as we have shown, it includes folds that are simple enough to be dealt with a single desktop but it is also rich enough to allow for important discussions as follows:

- The disjunctive family can be solved by a human being at once. Nevertheless, evolution needs to work in the exponential regime to solve it, so one

can not associate significant probabilities to events in this class. This fact push us to require from the Academic Community to stop claiming that the fossil record or compared anatomy or physiology or genomics or whatever is explained by evolution just because its looks in hindsight as evolutionary. That way of thinking was proper of the XX century when **lamarckism** was allowed. The present is more demanding: to begin with, you need to prove that the evolutionary step you are discussing can happen with a significant probability. Modern science would say that the threshold must be greater than 0.05 but 10^{-1000} will serve also at least for a beginning.

- Our study of the conjunctive family invites us to believe that evolution has indeed many possibilities to be the correct explanation of many changes along, say, the fossil record: many problems are very easy for evolution. Nevertheless, to accept or reject that invitation is the responsibility of the Reader. The problem is that SAT is a model of complexity, a caricature, whose relation with reality can be easily questioned: SAT is not complex enough to correctly reflect reality. The correct model for natural evolution is genetic programming, the developing of software by means of simulated evolution, but its study is very very heavy, a fact that we witnessed in Vol II of this series. Nevertheless, the Author adheres to the belief that many, many things in biology can be explained by natural evolution: just consider the formation of dog races.

- While evolution can solve many problems at once, it is also true that some problems could be very demanding for evolution (even with very clever aids given by the Developer). Nevertheless, it seems that evolution is very good to find approximate solutions that by far surpass mediocrity. Summing up, we question the ability of evolution to explain the perfection that abounds everywhere in alive beings. But, what does this mean? The only interesting meaning of our protest could be the following: the evolution of perfection is mandatory with abundant tracks across populations and in the fossil record. For instance, Microsoft Windows is updated on a daily schedule and so we have a fossil for each day. Is the same pretension corroborated by nature? The aim of the next chapter is to show over an example how Java can help us to discuss this question.

- SAT correctly describes the complexities of trying to say something interesting but not contradictory, a problem that every human being faces at each moment. In other words, we consider that every person is a master of SAT. For this reason, we contend that we have found a rationalization of the faith in the Creator that millions of people around the world have: the excel per-

fection of all alive beings is taken by them as a sign that they did not appeared by natural means and so their existence can be explained by God alone.

8.6 Conclusion

Our study of SAT, the satisfaction problem of mathematical logic, has strengthen on us the belief that complexity has an objective ingredient that covers all forms of computations, evolution included. So, there are problems that are easy for evolution as well as other ones that generate an exponential regime of work. Nevertheless, we witnessed how evolution is very good at finding approximate solutions that surpass mediocrity. Combining all results: to accept that evolution is the correct explanation of the excel complexity of alive beings is equivalent to predict that a whole trend of evolution from half cooked products to perfection must observed as in the fossil record as in modern populations. This mandatory prediction fails. We contend that this failure rationalizes the belief in the Creator and we predict that creationist religions will accompany us forever.

Chapter 9

Two visions over evolution

```
Just look at your hand!
```

133 ***Introduction***

I feel a great pleasure in being able of moving my left hand. I desire and it happens. This is astonishing. I also rejoice in how perfect and multifunctional my hand is: I can type words as well as caress my face or make a fit to show my strength.

Many people share with me a vivid interest in the hand. In fact, the hand is at the very core of the evolutionary theory and a lot of quantitative research has been carried to forge, depict, test and refine its evolution based scientific understanding.

Now, who could challenge the fact that there is an evolution of the hand, given that even children can fabricate good prototypes?

We can!

Yes, we can: we find no trouble in believing you that your small daughter made a good mechanical model of the hand. But our troubles began when we embarked at the beginning of this volume on the task of simulating a hand that could look a bit realistic. From the very beginning and at every moment, we have made only one thing: to suffer and not stop suffering.

We have suffered a lot but we have erased all traces of our suffering. Nevertheless an evolutionary trend can be observed from the cube to the hand that answers with a reflex to punching. So, we see that there are two visions over evolution: one by the final User, restricted to well terminated programs, and the other by the Developer, which includes every sort of pitfalls and half cooked products.

134 ***Purpose*** *Using our work in this volume in simulating a hand, we illustrate the contrast between the two versions of evolution, the fist given by a final User*

and the second by a Developer. Next we make a simulation to motivate the idea that the evolution depicted by evolutionists corresponds to a point of view given by a final User and not by a Developer.

9.1 Our experience

Let us imagine that we ask somebody to depict the evolutionary process that the Author of this volume went through to end with his most sophisticated version of a simulated hand. Answers could vary depending on the experience of the person. We present just two versions, that of a person of the lay public and one that tries to present an historic approximation.

135 ***A not professional version might look like this:***

- D10: cube with edges and nothing else
- D12: cube with faces
- D23: cube as an object
- D25: cubes and more cubes
- D29: finger as a coordinated ensemble of cubes
- D31: finger as an object
- D34: proto-hand as a coordinated ensemble of fingers
- D41: Oranges (reinventing gravitation)
- D44: robust (we can make science in spite of randomness)
- D50: do-undo movement
- D55: immobile (some movements were not possible)
- D71: Reflex = an interactive hand

136 ***A pseudo-historic version***

Historically, the developing of the programs of the present volume was approximately like this:

1. FASE I: From a cube to an isolated hand

- D10: cube with edges and nothing else
- D12: cube with faces
- D23: cube as an object
- D25: cubes and more cubes
- D29: finger as a coordinated ensemble of cubes
- D31: finger as an object
- D34: proto-hand as a coordinated ensemble of fingers
- D41: Oranges (reinventing gravitation)
- D44: robust (we can make science in spite of randomness).
- D50: do-undo movement
- D55: immobile (some movements were not possible)

2. FASE II: Mouse management and an interactive hand

   - D60: mouse detector
   - D62: Paint
   - D64: PopupMenu
   - D66: PaintPlus with a bug
   - D68: PaintFree, bugsfree
   - D70: SSuperPaint
   - d71: Early version of Reflex D71, an interactive hand

3. FASE III: Stop button and communication among diverse classes

   - D83: Morphs
   - D87: EvoArt
   - D95: randomSearch
   - D101: Shakespeare
   - D113: disjunctive
   - D118: conjunctive
   - D122: MAXKSAT

4. FASE IV: Refinement

   - D71: actual version of Reflex, an interactive hand.

5. FASE V: Updating

    - D71: Products were originally prepared using the versions of Java and Eclipse that were at hand. But for the final release, the latest versions of Java and Eclipse were downloaded. When programs were tested on these new versions, new and forgotten bugs appeared and so a large trend of corrections was begun.

This story is only approximated in the following sense: apart from bugs that kill the program, a developer aims at a good style, which eventually is not very good. Anyway, to to try to improve the style even a bit may generate exaggerated quantities of work about which the developer is the only persons that knows. So, any program pass over many, many intermediate versions.

We might enrich this story to cover thousand of sheets but all of them could be replaced with two activities. First: to challenge the Reader to make a program on his or her own that he or she could see how many unspeakable stories lie behind a terminated program. Second: to argue that all stories of this sort have a commonality: no program can be terminated at the first trial, instead every partial success is the fruit of a huge number of corrections. Perfection is never guaranteed because imperfection is almost certain.

137 ***Two extremal versions***

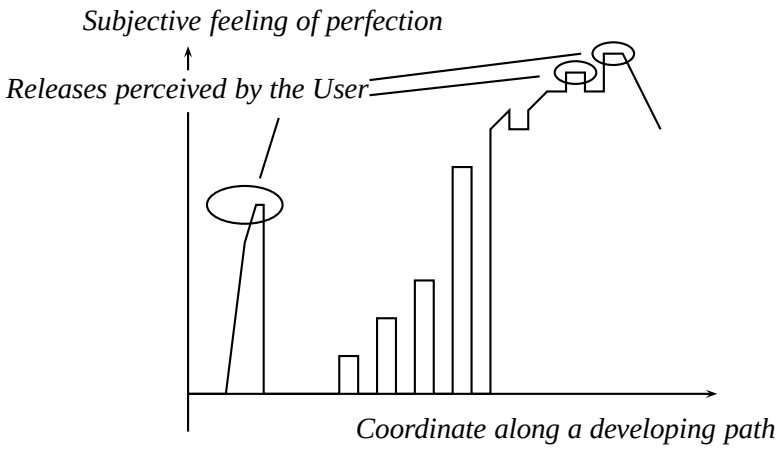


Figure 9.0. Evolution of software is mandatory but the perception of this process is different for the Developer Team (broken line) than for the User (Elliptical enclosers).

The two most important groups of persons in the trade of software development are that of final Users and that of Developers. They perceive evolution toward perfection of their products in radically two different ways. For the Developer Team, perfection is only statistical and begins to appear after a very tortuous path that includes the correction of many bugs with varying degrees of fatality. On the other side, the User sees an evolution that goes from not a bad product to a good project to a better one to excellent quality. When projects are very complex, with every kind of multifunctionality, the User also shares the feeling of dissatisfaction caused by half cooked products. The accompanying graphic has been taken from Vol III where this phenomenon was discussed at first

9.2 Our claim

We saw in Volume II that the genome is software and that the evolutionary theory reads: evolution is the software developer responsible for the existence of the genome. (A piece of software is a text, a string of symbols that are grouped according to precise syntactic rules that conveys a message with instructions to operate a processor according to a given code. In biology, the software is written in DNA or RNA chains, the processor is the ribosome and the code is the genetic code plus a lot of signaling to correctly read genetic messages. According to the evolutionary theory, modern genomes were synthesized by randomness and surviving of the fittest.)

Given that the developing of software will be the axis of our discussions, let us make the next definition:

138 ***SOFT The problem of of developing software. The developer could be a conscious being, like a human, or an unconscious process, like evolution. If the conscious being has a purpose, we speak of design. Natural evolution is presented as the development of the genome, natural software, by evolution which acquires a purpose per se: to gain more and more surviving capability. So natural evolution also has a purpose, a function, which is to increase fitness, to augment surviving reproduction.***

The most evident fact for a human designer is that SOFT it is very, very difficult no matter how smart or experienced one might be or how good the tools that one uses are. SOFT is much more complex that SAT, so much that while we know by direct experimentation many things about the relation of SAT and evolution, we know very little about SOFT apart from the very fact that its relation with evolution

burns too much computing time, so much that its study seems unaccessible when one has only a desktop.

All we know about evolution is that it is a natural process that has no magic in itself. Henceforth, natural evolution must obey natural law. Concretely: any evolutionary change corresponds to a computation that takes time and memory resources. Our experiments and those of the Academic Community allows us to firmly believe that natural evolution is a method of solving problems that is as effective as the generality of computing paradigms. Therefore, evolution is under the power of the theory of complexity that studies the needed resources, time and memory, to solve problems.

In consequence, and as we claimed in Vol III, the evolution depicted by modern science corresponds to the description of an evolutionary process as reported by the final User and that both, science and nature, lack a description corresponding to the Developer. In fact, all what we know about software developing compels us to firmly believe that every path along anyone developing history is plagued with every kind of pitfalls and with an endless number of minor corrections and updatings after the most perfect releases. Now, what does nature say?

Our perception about nature including the fossil record is consigned in the next three points:

- Only very finely terminated products are observed.
- Living beings come armed with an automatic mechanism to update themselves to the last minute needs of the client, which needs to survive in possibly changing environments: it is evolution.
- The evolutionary mechanism with which living beings come armed amounts to a computing capability that includes no magic. This means that some problems of adaptation can be solved just in front of your eyes while others that are common demand huge amounts of resources and of time. That is why no one claims that the surviving problems that menace the whole planet will be solved by evolution. Instead, only extinction is prophesied and this is serious: the fossil record shows many events of massive extinctions. Now, the limited computing power of evolution does not mean that we must reject evolution as the explanation of our existence. It means that we must test it against the tracks that it should had left: non is found, so there is no scientific reason to accept that we are the product of evolution. The reason to accept the evolutionary theory cannot be scientific, but it belongs in the real of philosophy.

It is clear that apart from extreme perfection we see no other thing in living beings. This contradicts what we expect from nature in the light of our studies

of SAT together with the experience of the Academy and of the Industry in the developing of software.

Now, **what is perfection?**

Perfection is abstracted by looking at the diverse solutions to a problem: some work quite finely while others left something to be desired in relation with the best ranked, be they real or imagined. So, when we say that we see only perfection we mean that we expect to see a lot of relative imperfection across fossils and living populations. But, actually what? To gain some insight, let us explore a detail of a very concrete problem.

9.3 Example: vertebrates

Vertebrates include animals such as snakes, salamanders, crocodiles, dolphins, whales, dinosaurs, birds, cows, horses, cats, lions, gorillas, men and women and all of them are distinguished because they have a vertebral column, a backbone that organizes all of their structure.

Many vertebrate species has disappeared. In ancient times it was allowed to imagine that disappeared species were somehow underdeveloped but this is anymore plausible: experts have gain their fame precisely because they have succeeded in showing that extinct species were as perfect as those that remain alive and that are a permanent motive of praise and glory. The TV channel Animal Planet has understood this quite well and has made a lot of propaganda about it. In short: the vertebrates we see or have existed correspond to final, well terminated products of a fabric whose trademark is extremely high quality and creativity.

Nevertheless, an evolutionary process has been superimposed over the history of vertebrates. It might be interesting to see how this is done. In this regard, it would be instructive to see over a very simple case how evolutionary trees are concocted. We will do it in the style of cladistics that nowadays replaces the ancient perspective adopted by Linnaeus, in which one organized what one sees and only what one sees.

139 ***Cladistics***

The purpose of cladistics is to organize data in a binary tree (clad means branch) whose branching is determined by relations of maximal similitude and minimal difference.

Suppose we have three families of objects that could be animals, cars, computers, houses, planets: A, B,C. Our procedure to organize them is as follows:

- We produce a table of properties that serve to identify objects and that therefore are useful to distinguish them one from another.
- We use the table of properties to define a metric that measures the discrepancy among the diverse families of objects. Many metrics can be defined by just changing the weight assigned to the diverse properties. We define a system of weights for each characteristic and consign generated distances in a distance matrix.
- We use the distance matrix to decide which is closer to A, if B else C (with many diverse descriptors the probability that both A and B lie at equal distances from A is nil). Suppose the the closer family to A is B. Then we can draw a tree in which A an B are connected as branches to a node that we will call $A \wedge B$. This node corresponds to a prediction: there must be an ancestor to both A and B. This ancestor, $A \wedge B$, and C are connected by a common ancestor $A \wedge B \wedge C$. So, the tree looks like this:

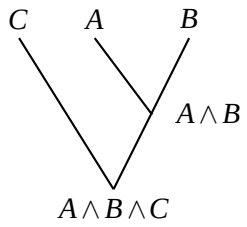


Figure 9.1. Based on observable data and the maybe subjective weights given to them, we propose to explain the existence of objects A, B, C through this evolutionary tree in which we predict that there must be a common ancestor $A \wedge B \wedge C$ from which two lineages were born C and $A \wedge B$. Next in time, from $A \wedge B$ lineages A and B diverged.

140 ***Exemple*** *Let us propose a cladistic tree for the next three artworks:*

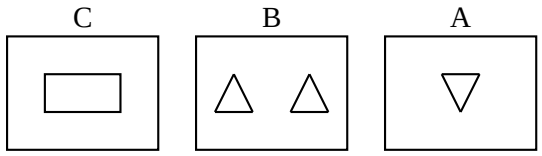


Figure 9.2. Follow your instinct to propose a cladistic evolutionary tree for these three objects.

It seems to us that to explain data, two main options would capture the attention of people. This is the first:

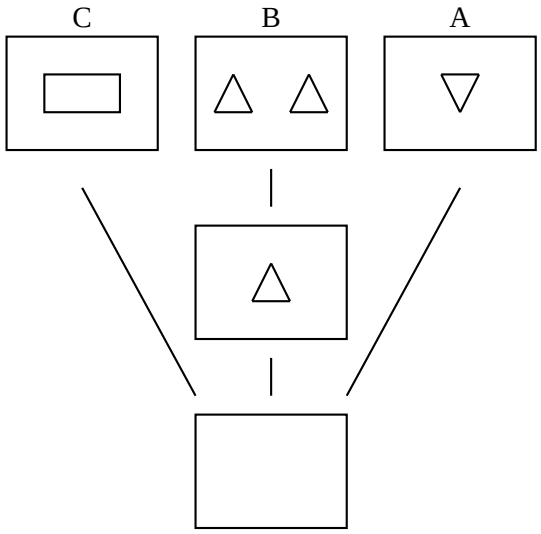


Figure 9.3. In this version, an upside down triangle will be considered as strongly different from a normal one. Only the three objects at the top edge are real, the other are inventions of the mind, they are predictions, they are ghosts because most of times they cannot be found.

The next solution would look more plausible for those that imagine that behind appearances there exists a program, a piece of software, that dictates how things must be done. So, we must not look at appearances but to the instructions, which are: include a rectangle, include a normal triangle, flip the triangle, duplicate the normal triangle:

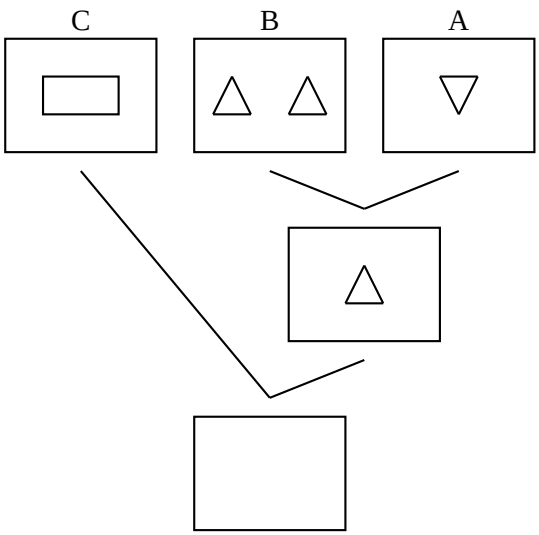


Figure 9.4. This tree is not for the appearances, phenotypes, but for the instructions that created them, genotypes.

We have been able of pointing to two diverse interpretations of the same data with only three objects. If these objects were fossils or living beings, a brilliant freshman would say:

The two theories predict different form as fossils must appear in the rock strata. According to the first theory, we might expect to find that specimens with the normal triangle and those with an upside down triangle indwelling separated geographical regions. But according to the second theory, the two species might inhabit neighboring or even overlapping regions. So, let nature say the last word and let us go to the field!

The problem with this suggestion is that it is not realizable: it is already difficult to find any one fossil, so how much improbable would be to find a fossil with predefined characteristics. So, this option is ruled out. This deficit of experimentable predictions is a crux for cladistics. So, why do we need to make predictions that end in ghosts that in general cannot be detected?

The answer is that predicted ghosts or intermediate forms have been sometimes found and this is seen as a great triumph. A triumph? Yes: in first place, to make an interesting prediction and corroborate it is considered as the trademark of science and in second place, the fulfillment of some of these predictions were used in 2003 to beat the defenders of Intelligent Design, who pretended that they had a scientific proof of the belief that living beings were created by an intelligent person (Padian, 2005). For this last reason alone, ghosts and cladistics will remain as a cherished part of science.

Cladistics is also favored by our point of view, which is summarized as follows:

- Principle of evolution: the developing of software, the genome included, is an extremely complex, daunting task. Therefore, an optimal strategy to develop software is to reuse it as much as possible.
- Principle of the sweat: there is no magic, only work. This is certain for me, for you, for natural, artificial or applied evolution, for expert system, for magicians, and for God (I apologize if this hurt your feelings). The complexity of software development is transcultural.
- Principle of the track: do not pretend that you made a piece of software if you cannot show a history of your pitfalls, of your intermediate successes and releases and a very long list of your actualizations on a daily schedule.

So, cladistics does it good in relation with some very few intermediate releases, while molecular evolution responds for the actualizations on a daily schedule. But, where are the tracks of the developing process that created perfection? We feel that we cannot understand what this question might mean if we do not discuss it in reference to a concrete example.

141 *From bones to my left hand*

Now, expressions of the sort '*creation of perfection*', '*evolution of perfection*', '*finely terminated products*' are subjective expressions and will remain so in spite of all attempts to define it in terms of optimization problems. We like this situation: we believe that when we go to see a dancer of ballet and we make joy of his dexterity, we are making a judgment using the capability of the brain to synthesize high level **beliefs**. In our community, beliefs are good in the measure that they are used as fuel to produce material that could be judged by everyone on a objective basis.

So, what do we have to offer to those than want to see how would look a world in which not finely terminated products might appear?

We have explored in the present volume the simulation of a left hand: we determined a basic building block, a cube, which then was used to build a complex system, a hand, that can be further refined in many forms. Now, our strategy has nothing of a miracle, it is just the easiest way of dealing with complexity, by using and re-using building blocks. so, we expect that any path leading to the existence of our left hand also goes from elementary building blocks to more complex organisms to still more complexity to an organism with a left hand such as the one I have.

The first surprise we find is that the bones of vertebrates are not made of wood. To use wood would had meant significant saving of effort in the construction of building blocks and complex derived units. Instead, a bone is a slight perturbation of a rock. In fact, the major components of bones are calcium oxide $CaCO_3$ and calcium phosphate $Ca_3(PO_4)_2$, substances that abound in nature in the form of rocks. Calcium oxide appears in limestone, marble and chalk. Calcium phosphate appears as phosphate rocks.

Let us suppose now that we have an organism with a proto-bone. Surely it should had appeared some 600 million years by the same time when mollusks appeared, because their shell is also composed of calcium oxide. Precisely, a mollusk and a vertebrate represent a change of strategy: the shell of the mollusk gives structure and offers shelter, while bones offer structure, tools and weapons. So, we expect the first vertebrate to be a mollusk with a boned structure that could had been used as a lance, a sword, a club, a fin and a tool for digging. We imagine that such an unicorn would be able to survive quite fine. So, we predict that the fossil record at the level of the first mollusk strata, clams and oysters, or maybe a bit latter, must be filled with fossils of these animals.

A next stage in evolution could be that of a mollusk with an articulated finger, an empowered tentacle that can be used to explore at remote distance suspicious environments, or to rich otherwise forbidden source of food. Since there are so many options to build such a tentacle, we expect a very great diversity of species of animals with empowered tentacles. At the same time, we expect a gradual lost of the proportion of the body that is covered by the shell thanks to a compensation offered by those empowered tentacles.

Once a finger or empowered tentacle was a reliable unit, we may imagine many trials to make a good limb used for swimming. Of course, it must be plain. According to **lamarckism**, it is plain by good luck. According to **neodarwinism**, it is plain because this structure was the winner among many, many possible ones: Would someone show a suitable fossil?

Our simulated hand has a thumb, whose only difference with the other fingers is that it has two visible phalanges instead of three. But our thumb is not opposable, as the thumb of my left hand. We follow in our design the law of the least effort and that is what we produced. Let us refer to our hand as *plane*.

Our laziness can be made into a prediction: when one finger was ready, the structure of a plane hand is immediate. Plane hands could be useful to swing, so we expect to see plane hands in the early strata of the fossil record of boned fishes. After the plane hand, more complications must come, say, a system of levers to empower the swimming capability of plane hand. We also can expect a system for grasping, a hand with opposable finger. This is plainly reasonable because some mollusks have pliers.

The process can be depicted a follows:

Finger, articulated tentacled → Plane hand → An empowering system of levers and a hand with opposable thumb.

There are many options with many diverse functions and degrees of functionality to build an opposable thumb given that one has the know-how to make a plane hand. So, we expect a very rich diversity of attempts that would represent revolutions whose existence should had been represented quite accurately in the fossil record.

Now, let us think of the vertebral column. This marvel represents an unification of the whole structure of the organism. It is so smart an invention that its transcendence go shoulder with shoulder with the great unifications of science, say the building of a single theory of elementary particles. This theory is still in the making: we have too many models, each one with local descriptive power, but we lack a vertebral column for a single theory. Therefore, we expect to find in the fossil record too many variants as the unification of the system of bones goes from completely separated units to a modern skeleton that defines a single unit.

And after many, many strata along the rocks, we would find those beings that are remarkably similar to us.

We stress that when we say that there must be many, many variants, we are using what we know about the evolution vs SAT struggle: evolution is good to find approximate solutions to some complex problems but it must pay a high price for perfection, i.e., it must make many, many trials to achieve something good. This prediction is mandatory.

142 ***Exercise*** *Develop your own prediction of the evolution of vertebrates and contrast your prediction with the fossil record and comparative anatomy. Make it clear what is your guiding principle. Ours was that of maximal laziness + many, many random trials to get something new but functional.*

Now, let us go to the fossil record and the evolutionary theory to see what they have to tell us. Some points that seem interesting to us are (Puver et al, 2001):

- First boned animals appeared in the sea. Terrestrial boned animal appeared lately.
- All boned animals, marine and terrestrial, have a vertebral column.
- Sharks predate boned fishes and terrestrial animals. Sharks have no bones, instead their skeleton is cartilaginous. On the other hand, vertebrate embryos

are also cartilaginous while ossification is gained step by step ending when growth is stopped but some structures, like the nose, always remain in cartilage form. Therefore, the **biogenesis law** (a living being repeats along its embryonic development the evolutionary history of its species) forces evolution to posit sharks as an ancestor to vertebrates.

- A plane hand with 8 fingers appears in the Ichthyostega. But before it, a very complex system of levers was developed, such in the Eusthenopteron. The number of fingers diminishes along the record fossil from 8 to 7 to 6 to 5 or less.

- The first amphibians had plane hands at the end of well suited forelimbs(Laurin, 2009: please, pay attention to the opening drawing), while modern amphibians have hands with opposable thumbs. Nevertheless, a complaint about the scarcity of fossils has been expressed (Laurin, 2006).

- There are some small animals some 3cms long called lancets (Branchiostomma lanceolata) that have a notechord, which plays the role of a vertebral column of a single piece. The flexibility + consistency given by the notechord is useful for them to hide their bodies in the sediment of the sea while their mouth is open to the water. They are believed to exist even before sharks and so are drawn in the evolutionary trees as an ancestor to sharks, which are also drawn as ancestors of all boned species. On the other hand, embryos of higher vertebrates also have a notechord, from which their vertebral column develops. This fits the **biogenesis law** of evolution.

We can notice now that our complexity based predictions are completely void. We highlight three main failures:

- There is no unicorn and no further evolution of it.

- A plane hand or limb does not appear in the early stages of evolution, instead we see that a system of levers and extensions was well developed at once.

- There is no evolution of the unification of the bone system into a single skeleton that spins around a vertebral column. From a notechord, which is an elastic bar, one passes directly to well terminated vertebral columns.

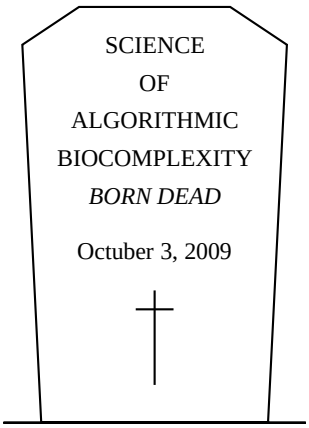


Figure 9.5. A science must have simple theoretical structure that enables the clean computation of testable predictions that are corroborated by experiment and field observation.

So, our science of evolutionary algorithmic biocomplexity is born dead: we began by recognizing that the extreme high complexity of the development of software, the genome include, is objective and transcultural and so we readily make strong predictions that included viable but half-cooked products. Our predictions failed.

The failure of our mandatory predictions gives the correct meaning of the next subjective expressions that serve as summary: we lack a description of the history of life from the stand point of a software developer. We see only finely terminated products.

143 ***The principle of maximum laziness***

The development of software is a very complex task. This means that an intelligent being must commit many bugs whose correction creates more bugs while evolution must do a lot, a huge lot of trials to punch a good solution.

Nevertheless, in some occasions duplications with small changes could amount to variability an/or terrible selective advantage. Say, the incredibly fine senses of animals are explained in some part as the result of structures that have a multiplicity of the very same basic organs that we have. For instance, the organ of the smell of a dog may have 100 times more smelling cells than us.

In general, changes can be given a measure of complexity in such a way that one expects easy changes to occur earlier than more complex ones. This is the

principle that allows us to reconstruct evolutionary trees. Officially, this principle is known as the principle of maximum parsimony.

How do we associate a complexity to a piece of code? There are many forms of doing that and the simplest one is to count the number of lines of the code. Is this definition too coarse for you? Try a modification: use a suitable set of rules to simplify the pieces of code as much as possible and then count the number of lines.

We humans are very good to estimate the complexity of code even before it has been designed: we always are reticent to build complex pieces and when many tasks are to be done, we always begin with the simplest one. This is the principle of laziness that has commanded our path until here. Concretely, we have delayed as much as possible the task of making a hand with an opposable thumb. Why?

The problem is that our hands are marvelous: with minor but very smart fine changes, great marvel are produced. In fact, a human hand is a slight modification of a plane hand but with the bonus of having an opposable thumb. Or, our hand has an opposable thumb but it is still a plane hand. So, it actually plays the two roles.

Why do we say that our hand is a marvel? The reason is that the principle of laziness propels us to try, as in the code below, a hand with opposable thumb that is not a plane hand and that cannot be given the role of mimicking a plane hand. The human hand is also different than the foot of a bird, which is plain but the thumb points contrariwise to the other fingers and all close toward the common inside while the thumb of the human hand points in the same direction of the other fingers.

144 ***The code for a hand with an opposable thumb follows:***

```
/*Program D144 OpposableThumb
 *
 * Our hands are plain and all fingers point in
 * the same direction but the thumb can  be turned
 * so that one can tightly hold things with it.
 * Feet of birds are also plain and have an opposable
 * thumb, which points contrariwise to the other fingers.
 * In humans, the hand follows in the same direction
 * of the arm while the foot of birds is perpendicular
 * to the leg.
 *
 * The hand presented in here is somehow in the middle
 * of a hand of a human and a feet of a bird. We imagine
 * that it is by far simple than that of a human and
```

```
 * that it is very good to grasp things, so we expect
 * that it must appear somewhere in the fossil record
 * that shows the evolutionary history of our lineage.
 *
 * The code offers a half cooked product.
 * In complex projects, half cooked products,
 * determined by local maxims,
 * are mandatory long living
 * innumerable intermediate states.
 * Where are the myriads of half cooked products
 * of the evolutionary process?
 */

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

import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.MouseListener;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.JPopupMenu;
import javax.swing.ButtonGroup;

import java.awt.Polygon;
import java.util.Random;


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

 //The application is instantiated
 static OpposableThumb p = new OpposableThumb();
```

```
//A class with important data is instantiated
 private static Data d = p.new Data();
//The painting class is declared and instantiated
private static Canvas c = p.new Canvas();
//Popup menu is  declared and instantiated
static pppMenu pp = p.new pppMenu();

public static void main( String args[] )
{
 //The JFrame is constructed with a title
 JFrame f = new JFrame( "OpposableThumb" );

 //The popUp Menu is added to the frame
 f.add( pp, BorderLayout.CENTER );
 f.add( new JLabel("Punch a finger with the mouse"),
                          BorderLayout.SOUTH );
 f.setDefaultCloseOperation( JFrame.DISPOSE_ON_CLOSE );
 c.setBackground( Color.pink );
 f.setSize( 600, 700 );
 f.setVisible( true );
 //PopUp menu is added to f
 f.add( c, BorderLayout.CENTER );
}

//Data that serve communication
//among different classes is reunited into an object.
private class Data
{
 private static final long serialVersionUID = 1L;
 //Flag for clicked button
 private  boolean clickedLeft;
 private  boolean clickedRight;
 private int optionColor ;
 private int optionMove ;
 private double ampliFactor;
 private int ex;
 private int ey;
 private Color color;
 private  boolean first;
```

```
public Data()
{
 //Which button of the mouse has been clicked
 clickedLeft = false;
 clickedRight = false;
 //Option of the popUp menu
 optionColor = 0;
 //Option for movement: up vs down
 optionMove =0;
 //Graphic enlarging factor
 ampliFactor = 3;
 //Coordinates of clicking event
 ex = 0;
 ey = 0;
 //Initial color for background
 color = Color.pink;
 //Flag for first call to paint
 first = true;

}

}

//==================================================
//=========pppMenu CLASS============================
//==================================================

//This inner class enables the use of right clicking
//to activate a pop up menu.
//Sensing of the mouse is also enabled.
public class pppMenu extends JPanel
{
 private static final long serialVersionUID = 1L;
 private JRadioButtonMenuItem ItemsColor[];
 private JRadioButtonMenuItem ItemsMove[];
 private JPopupMenu popupMenu;


public pppMenu()
```

```
{
  //**********Color section

  //Handler for color menu items
  ColorHandler colorHandler = new ColorHandler();
  //Names of used colors
  String optionColor[] = {"Pink", "Black", "Blue" };
  //Colors are chosen by clicking a button
  ButtonGroup colorGroup = new ButtonGroup();
  //Creates pop-up menu
  popupMenu = new JPopupMenu();
  //Buttons for selecting colors
  ItemsColor = new JRadioButtonMenuItem[ 3 ];
  //The menu is constructed: it has 6 buttons
  //Three buttons are for color and other 3 for the move
  for ( int i = 0; i < ItemsColor.length; i++ )
  {
   ItemsColor[ i ] =
         new JRadioButtonMenuItem( optionColor[ i ] );
   //add item to pop-up menu
   popupMenu.add( ItemsColor[ i ] );
   //add item to button group
   colorGroup.add( ItemsColor[ i ] );
   //add handler
   ItemsColor[ i ].addActionListener( colorHandler );
  }
  ItemsColor[ 0 ].setSelected(true);

  popupMenu.addSeparator();

  // **********Move section

  // Handler for color menu items
  MoveHandler moveHandler = new MoveHandler();
  //Names of used moves
  String optionMove[] = { "Up", "Down"};
  //Moves are chosen by clicking a button
  ButtonGroup moveGroup = new ButtonGroup();

  //Buttons for selecting moves
```

```
   ItemsMove = new JRadioButtonMenuItem[ 2 ];


   //The menu is constructed: it has 2 buttons
   for ( int i = 0; i < ItemsMove.length; i++ )
   {
    ItemsMove[ i ] =
          new JRadioButtonMenuItem( optionMove[ i ] );
    //add item to pop-up menu
    popupMenu.add( ItemsMove[ i ] );
    //add item to button group
    moveGroup.add( ItemsMove[ i ] );
    //add handler
    ItemsMove[ i ].addActionListener( moveHandler );
   }
   ItemsMove[ 0 ].setSelected(true);


   //The popUp menu is accessed by  releasing
   //of the right button.
   //Attention: the mouse listener is added to the canvas c
   c.addMouseListener
    (
      new MouseAdapter()
      {
       //Detector of right button releasing operation
       public void mouseReleased( MouseEvent event )
       {
        if  ( event.isPopupTrigger() )
         popupMenu.show(
         event.getComponent(),event.getX(),event.getY() );
        }
       }
     );
   // Detection of the mouse
   MouseHandler handlerr = new MouseHandler();
   //Sensors are added to the canvas c
   c.addMouseListener( handlerr );

  }
```

```
//If a color  event has been detected, here is the reaction
private class ColorHandler implements ActionListener
{
 public void actionPerformed( ActionEvent event )
 {
  //Determines which option was selected
  for ( int i = 0; i < ItemsColor.length; i++ )
   {
    if ( event.getSource() == ItemsColor[ i ] )
         d.optionColor = i;
   }
  d.clickedRight = true;
  c.repaint();
 }
}

//If a move event has been detected, here is the reaction
private class MoveHandler implements ActionListener
{
 public void actionPerformed( ActionEvent event )
 {
  //Determines which option was selected
  for ( int i = 0; i < ItemsMove.length; i++ )
   {
    if ( event.getSource() == ItemsMove[ i ] )
     d.optionMove = i;
   }
  d.clickedRight = true;
  c.repaint();
 }
}

//Movements of the mouse are detected
private class MouseHandler implements MouseListener
{
public void mouseEntered( MouseEvent event )
{
//Some invisible code goes in here.
}
```

```
public void mouseClicked( MouseEvent event )
{
 d.ex = event.getX();
 d.ey = event.getY();
 //The clicked finger and phalange are identified
 c.identify();
 c.repaint();
 d.clickedLeft = true;
 c.repaint();
}

public void mousePressed( MouseEvent event )
{
 /*System.out.format ("The mouse was pressed " +
"at [%d, %d] \n", event.getX(), event.getY() );*/
}

public void mouseReleased( MouseEvent event )
{
 /*System.out.format ("The mouse was released " +
"at [%d, %d] \n", event.getX(), event.getY() );*/
}

public void mouseExited( MouseEvent event )
{
 /*System.out.format ("The mouse exited the frame at" +
" [%d, %d] \n", event.getX(), event.getY() );*/
}

}  //end of inner class MouseHandler


} // end of class  pppMenu


 //===========================================
 //============== CLASS CANVAS ===============
 //===========================================
```

```
/*Inner class Canvas

We display a proto-hand which is sensible to
punching by the mouse.

Our protohand has 5  fingers, a palm and an arm.
Fingers have 3  phalanges except the thumb, which has 2.

Since we use finite precision, all our moves are
approximations. Therefore, a multiplication of errors
is expected if we use modifications in tandem.
That is why  every scene is built ab initio by
modification of a unique unmodifiable cube at the origin.
The basic hand is located and moves at the origin
and points downwards over the screen.

The inner class Hand is encoded beginning from line 873
The inner class Finger is presented in line 934.
A finger is composed of 3 cubes. The class Cube
appears in line 1047.

*/


public class Canvas extends JPanel
{

 public static final long serialVersionUID = 1L;

/*
Our basic unit is  the unitary cube at the origin.
It is composed of 8 points or vertexes = oPoints3d[][],
12 edges = edges[][] and 6 faces = faces[][].
Points are numbered from 0 to 7.
These numbers are used in
edges, which  record the pairs of
points that are joined by a line.
A face is determined by its 4 corners.

All other cubes are transformations of this
```

```
 fundamental cube. The form as this is done
 appears in the class Cube.
 */


 //Constructor
 /*This constructor is necessary when one have various
 nested classes. In our case, we have the class Cube,
 which is used by the class Finger,
 which is used by the class Hand,
 which is used by the class Canvas,
 which is used by the main class.
 */
  public Canvas( )
  {

  }

// The points of the fundamental cube
 private     double oPoints3d[][]=
 {
  {0,0,0},
  {1,0,0},
  {0,1,0},
  {0,0,1},
  {1,1,0},
  {1,0,1},
  {0,1,1},
  {1,1,1}
 };

 /*
// The edges of the fundamental cube
 private static int edges[][] =
    {{0,1},
     {0,2},
     {0,3},
     {1,4},
     {1,5},
     {2,6},
```

```
    {2,4},
    {3,5},
    {3,6},
    {4,7},
    {5,7},
    {6,7}};

*/

// The faces of the fundamental cube
private   int faces[][] =
   {{0,1,4,2},
    {0,2,6,3},
    {0,3,5,1},
    {3,5,7,6},
    {1,4,7,5},
    {2,6,7,4}};


 //Measures of the window
 private    int   xWin = 600;
 private    int   yWin = 700;
 private   int nScenes;

 //Number of cubes in the finger
 private     int nPhalanges = 3;
 private     Boolean moveArm = true;

 Random r  = new Random();

 //The constructor of hand is invoked here.
 //h and hCopy are instance of the main class Hand
 private  Hand  h =  new Hand();
 private  Hand  hCopy =  new Hand();
 //An auxiliary  instance of Finger
 private   Finger f1=   new Finger(nPhalanges);
 //Finger and phalange
 private int f, ph;
 //A constant that determines how much a phalange
 //must go up or down.
```

```
  private double deltaUp;



  // The window is enlarged.
  // This method is fired automatically
  // at the beginning
  public void init()
  {
   setSize(xWin,yWin);
  }

 //Clicked finger and phalange are identified.
 //80% exact.
 private    void  identify()
 {

  int px, py;
  int min =999999;
  for(int j = 4; j>=0;j--)
   {
if(j==4) nPhalanges = 2;
    else nPhalanges = 3;
for(int m = 0; m < nPhalanges; m++)
 {
  //A working cube cc is initialized
  Cube  cc = new Cube(m);
  //The working cube is re-set.
  cc = h.finger[j].phalange[m];

  //Projection is calculated
  cc.shade2d =  cc.round(cc.cubeProj );
  int d1 = 0;
  for(int l =0; l<4; l++)
   {
  px = 100-cc.shade2d[faces[1][l]][1];
  py =-150-cc.shade2d[ faces[1][l]][2];
  d1 = d1+ (px-d.ex) * (px-d.ex)
                  + (py-d.ey)*(py-d.ey);
   }
```

```
  if (d1<min)
   {
min = d1;
    f = j;
    ph = m;
   }
}
   }
 // System.out.println("Finger=" + f + " phalange=" +ph);
}

 //face ii of the cube is drawn
 private    void drawface(Graphics g, Cube cc, int ii)
 {
  for(int l =0; l<4; l++)
   {
    cc.pointsX[l] = 100-cc.shade2d[faces[ii][l]][1];
    cc.pointsY[l] =-150-cc.shade2d[ faces[ii][l]][2];
    //System.out.println(pointsX[l] + " " + pointsY[l]);
   }
   //A polygon is defined by two vectors and an integer
   //number:
   //The first vector lists the x-coordinates of the
   //corners;
   //the second lists  the y-coordinates. The integer is
   //the number of points.
   //If the last point is different than the first,
   //the first point is closed with the last.
   Polygon face = new Polygon(cc.pointsX,
                      cc.pointsY, 4);
   g.fillPolygon(face);
 }

 //Arm is drawn
 private    void drawA(Graphics g)
 {
  //System.out.println("drawA");
  //  A working cube cc is initialized
  Cube  cc = new Cube(1);
```

```
 //The working cube is re-set.
 cc = h.arm ;

 cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
 //Projection is calculated
 cc.shade2d =  cc.round(cc.cubeProj );
 //printShade();
 //The cube is drawn
 cc.calculateColor();
 for(int i=0;i<6;i++)
{
    if (cc.coloredFace[i])
   {
    g.setColor( new Color( cc.colors[i][0],
                  cc.colors[i][1],
                  cc.colors[i][2]));
    drawface(g,cc,i);
   }
 }
//worked cube is kept to memory
h.arm = cc;
}

//Arm is drawn
private    void drawArm(Graphics g)
{
 //System.out.println("drawArm");
 //A working cube cc is initialized
 Cube  cc = new Cube(1);
 //The working cube is re-set.
 cc = h.arm ;
 //Movement begins
 if (moveArm) f1.modifyCube(cc,0);
 //worked cube is kept to memory
 h.arm = cc;
 drawA(g);
}

//Basic palm is transformed
private Cube repositPalm(Cube cc)
```

```
 {
  cc.oPosition[0]  =  h.arm.cube[3][0];
  cc.oPosition[1]  =  h.arm.cube[3][1]+5*d.ampliFactor;
  cc.oPosition[2]  =   h.arm.cube[3][2];
  cc.cube = cc.translateCube(cc.cube, cc.oPosition);
  return cc;
}

 //Palm is updated
 private    Cube updatePalm(Cube cc)
 {
  cc = f1.inclination(cc,1);
  cc = f1.rotation(cc,1);
  //cc.printCube(cc.cube, " after rotation");
  cc = repositPalm(cc);
  //cc.printCube(cc.cube , " after relocation");
  for(int i = 0; i<3;i++)
  cc.vTrans[i] = cc.vTrans[i] + cc.deltaTrans[i];
  cc.cube = cc.translateCube(cc.cube,  cc.vTrans);
  //cc.printCube(cc.cube, " after translation");
  cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
  //cc.printCube(cc.cubeProj,  " after projection");
  return cc;
 }

 //Draw palm
 private    void drawP(Graphics g)
 {
  //System.out.println("drawP");
  //A working cube cc is initialized
  Cube  cc = new Cube(1);
  //The working cube is re-set.
  cc = h.palm ;
  cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
  //Projection is calculated
  cc.shade2d =  cc.round(cc.cubeProj );
  //printShade();
  //The cube is drawn
  cc.calculateColor();
  for(int i=0;i<6;i++)
```

```
  {
     if (cc.coloredFace[i])
    {
     g.setColor( new Color( cc.colors[i][0],
                    cc.colors[i][1],
                    cc.colors[i][2]));
     drawface(g,cc,i);
    }
  }
   //Memory is updated
   h.palm = cc;
  }

  //Draw palm
  private    void drawPalm(Graphics g)
  {
   //A working cube cc is initialized
   Cube  cc = new Cube(1);
   //The working cube is re-set.
   cc = h.palm ;
   //Basic palm is translated
   if (moveArm) cc = updatePalm(cc);
   h.palm = cc;
   drawP(g);
  }

  //Thumb is built
  //Thumb is finger number 4
  private void buildThumb(Graphics g)
  {
   for(int j=0; j<3;j++)
h.finger[4].phalange[0].oPosition[j]
                =  h.palm.cube[3][j];

   h.finger[4].phalange[0].oPosition[1]
     = h.finger[4].phalange[0].oPosition[1]
       -Math.round(10*1.2*d.ampliFactor);
   nPhalanges = 2;
   for(int m = 0; m < nPhalanges; m++)
{
```

```
 //A working cube cc is initialized
 Cube  cc = new Cube(m);
 //The working cube is re-set.
 cc = h.finger[4].phalange[m];
 //Translation and rotation of the basic arm
 if (!(moveArm))
   {
    cc.deltaPhi[m] = 0;
    cc.deltaTheta = 0;
    }
 cc = h.finger[4].modifyCube(cc,m);

 //worked cube is kept to memory
 h.finger[4].phalange[m] = cc;
}
  if (moveArm)
  {
   h.finger[4].phalange[0].phi[0]
               = h.finger[4].phalange[0].phi[0]+2*deltaUp;
   h.finger[4].phalange[1].phi[1]
               = h.finger[4].phalange[1].phi[1]+3*deltaUp;
  }
  }


  //Fingers are posited below the palm
  private    void locateFingers()
   {
    for(int i=0; i<4;i++)
  for(int j=0; j<3;j++)
   h.finger[i].phalange[0].oPosition[j]
                                  =  h.palm.cube[3][j];
    for(int i=0; i<4;i++)
  h.finger[i].phalange[0].oPosition[1]
                 = h.finger[i].phalange[0].oPosition[1]
                                  -10*i*d.ampliFactor ;

   }

  private void paintFinger(Graphics g, int j)
```

```
{
if(j==4) nPhalanges = 2;
  else nPhalanges = 3;
  for(int m = 0; m < nPhalanges; m++)
  {
//A working cube cc is initialized
Cube  cc = new Cube(m);
//The working cube is re-set.
   cc = h.finger[j].phalange[m];

   //Projection is calculated
   cc.shade2d =  cc.round(cc.cubeProj );
   //printShade();
   //The cube is drawn
   cc.calculateColor();
   for(int i=0;i<6;i++)
    {
  if (cc.coloredFace[i])
   {
    g.setColor( new Color( cc.colors[i][0],
                  cc.colors[i][1],
                  cc.colors[i][2]));
    drawface(g,cc,i);
   }
    }
   //worked cube is kept to memory
   h.finger[j].phalange[m] = cc;
  }
}

//Draw fingers
private    void drawF (Graphics g)
{
 //System.out.println("drawF");
 locateFingers();
 buildThumb(g);
 for(int j = 3; j>=0;j--)
paintFinger(g,j);
 paintFinger(g,4);
}
```

```
//Draw fingers
private    void drawFingers(Graphics g)
{
 locateFingers();
 nPhalanges = 3;
 for(int j = 3; j>=0;j--)
 {
  for(int m = 0; m < nPhalanges; m++)
  {
//A working cube cc is initialized
Cube  cc = new Cube(m);
//The working cube is re-set.
   cc = h.finger[j].phalange[m];
   //Translation and rotation of the basic arm
    if (!(moveArm))
     {
     cc.deltaPhi[m] = 0;
     cc.deltaTheta = 0;
     }
    cc = h.finger[j].modifyCube(cc,m);

   //worked cube is kept to memory
   h.finger[j].phalange[m] = cc;
  }
 }
 drawF(g);
}

//Arm, palm, finger are reposited
private void moveAPF(Graphics g)
{
drawAP(g);
 drawFingers(g);
}

//A pahalange goes up
private    void  showUp(Graphics g, int j, int m)
{
```

```
 h.finger[j].phalange[m].phi[m]
             = h.finger[j].phalange[m].phi[m]-deltaUp;
moveAPF(g);
}

//A phalange goes back down
private    void  showDown(Graphics g, int j, int m)
{
 h.finger[j].phalange[m].phi[m]
             = h.finger[j].phalange[m].phi[m]+deltaUp;
 moveAPF(g);
}


//Phalange is moved
private    void  movePhalange(Graphics g)
{
 //System.out.println("movePhalange");
 moveArm = false;
 locateFingers();
 coloring(g);
 if (d.optionMove == 0) showUp(g,f,ph);
 if (d.optionMove ==1) showDown(g,f,ph);
 d.clickedLeft = false;
 hCopy = h;
}

// Forarm and palm are drawn
private    void drawAP(Graphics g)
{
 drawArm(g);
 drawPalm(g);
}

//Hand
private    void drawHand(Graphics g)
{
 drawAP(g);
 drawFingers(g);
}
```

```
  //The color chosen of the menu is executed
  private    void coloring(Graphics g)
  {
  //System.out.println("coloring");
  switch (d.optionColor  )
      {
       case 0:   d.color = Color.pink; break;
       case 1:  d.color = Color.black; break;
       case 2:  d.color = Color.blue; break;
      }

c.setBackground(d.color);
g.setColor(d.color);
   g.fillRect(0,0,xWin,yWin);
   //System.out.println("color = " + d.color);
  }

  //Basic hand is transformed into the actual one
  private    void presentation(Graphics g)
  {
   //Loop for the scenes
  for(int k = 0; k<nScenes; k++)
   {
  //System.out.println("scene " + k);
  drawHand(g);
  //delay  + erase drawing
  if (k < nScenes-1)
       coloring(g);
   }//end of for nScenes
  }



 //The hand that opens the session is built
  private    void initialPosition(Graphics g)
  {
   //Repeatability is enabled
   r.setSeed(45);
   //Number of scenes of the presentation.
```

```
   nScenes = 4;
   //System.out.println("First time");
   d.first = false;
   deltaUp = 0.5;
   coloring(g);
   presentation(g);
   hCopy = h;
  }


  //When paintComponent() is called,
  //everything is repainted
  //using information in memory
  private    void refresh(Graphics g )
  {
//System.out.println("refresh");
coloring(g);
 h = hCopy;
 drawA(g);
 drawP(g);
 drawF(g);
  }

  //This method has no effect apart from
  //remedying an unknown bug (maybe in Java itself).
  //Exercise: block it out to see the result
  private void beta(Graphics g)
  {
   //System.out.println("Beta");
   c.setBackground(Color.BLACK);
   c.setBackground(Color.cyan);
   d.clickedRight = false;
  }

  //This is the main method
 public void paintComponent(Graphics g)
  {
   //if this method is called for the first time
   if (d.first) initialPosition(g) ;
   else //second and following calls
```

```
 {
  //System.out.println("Repaint");

  if (d.clickedLeft) movePhalange(g);

  if (d.clickedRight)
   {
    beta(g);
   }
 }
 refresh(g);
}//end of paintComponent(G)

//=========================================================
//=============Beginning of Inner class Hand============
//=========================================================

public class Hand
{
 private Cube arm = new Cube(1);
 private Cube palm = new Cube(1);
 private Finger finger[] = new Finger[5];

 //When memory is allotted for a hand,
 //all variables must be initialized.
 //Lower classes are initialized by default,
 //but some  redefinitions could be necessary.
 Hand()
 {
  //The arm is initialized
   final Cube c1 =    new Cube(1);
   arm = c1;
      //Dimensions and location of the arm.
   arm.scale[0] = 8;
   arm.scale[1] = 30;
   arm.scale[2] = 100;
   for(int i=0; i<8;i++)
   for(int j=0; j<3;j++)
      arm.points3d[i][j] = -d.ampliFactor*arm.scale[j]
                                          *oPoints3d[i][j];
```

```
   //The  palm is initialized
   final Cube c2 =    new Cube(1);
   palm = c2;
   //Dimensions and location of the palm.
   palm.scale[0] = 8;
   palm.scale[1] = 40;
   palm.scale[2] = 60;
   for(int i=0; i<8;i++)
for(int j=0; j<3;j++)
 palm.points3d[i][j] =  -d.ampliFactor
                * palm.scale[j]*oPoints3d[i][j];


   //Fingers are initialized
   for(int i=0; i<5;i++)
   {
   final  Finger f =   new Finger(nPhalanges);
   finger[i] = f;
   }

 }
}
//=============End of Inner class Hand============

//================================================
//=============Beginning of Inner class Finger======
//================================================

/*Variables and methods of the class Finger are
encapsulated in the same unit.
All definitions are non static, i.e., they are
used as erasable workbenches and resettable tools.

A finger is a coordinated set of 3 cubes.
The class cube is defined below.

All methods that are of exclusive use for fingers are
encapsulated here.
```

```
  */
  public class Finger
  {

   //Our finger is a composition of 3 cubes or phalanges
   private   Cube[] phalange    = new Cube[3] ;


   //When memory is alloted for a finger,
   //all variables must be initialized
   Finger(int nPhalanges)
   {
   for(int m =0; m < nPhalanges; m++)
     {
      final Cube cc =    new Cube(m);
      phalange[m] = cc;
     }
   }
//  The only allowed movement of a phalange is
   //inclination, forth and back, in within bounds.
   //Inclination of given phalange governs the
   //rest of the finger
   private   Cube inclination(Cube cc, int m)
   {
    //Inclination is equal to a rotation.
    //Axis of rotation: the y axis.
    cc.axisRot[0] = 0;
    cc.axisRot[1]=1;
    cc.axisRot[2]=0;
    switch(m)
     {
  case 0: cc.phi[0]  = cc.phi[0]  + cc.deltaPhi[0] ;
            break;
  case 1: cc.phi[1]  = cc.phi[0] + cc.phi[1]
                                   + cc.deltaPhi[1] ;
            break;
  case 2: cc.phi[2]  = cc.phi[1] + cc.phi[2]
                                   + cc.deltaPhi[2] ;
            break;
     }
```

```
 cc.cube = cc.rotCube(cc.points3d, cc.axisRot,
                              cc.phi[m] );
 return cc;
}

   //For visualization, the finger is rotated
//around the z axis.
private   Cube rotation(Cube cc, int m)
{
 //Rotation about the z axis.
 cc.axisRot[0] = 0;
 cc.axisRot[1]=0;
 cc.axisRot[2]=1;
    cc.theta = cc.theta + cc.deltaTheta;
 cc.cube = cc.rotCube(cc.cube, cc.axisRot, cc.theta);
 return cc;
}

 //Each phalange must have its initial position that
 //depends on the position of the phalanges below it
 private   Cube rePosit(Cube cc, int m)
 {
   //System.out.println("rePosit(m)" + m);
 switch(m)
  {
   case 0:  {
      } break;
   case 1: {
cc.oPosition[0]  =   phalange[0].cube[3][0];
cc.oPosition[1]  =   phalange[0].cube[3][1];
    cc.oPosition[2]  =   phalange[0].cube[3][2];
   break;
       }
   case 2: {
cc.oPosition[0]  =   phalange[1].cube[3][0];
cc.oPosition[1]  =   phalange[1].cube[3][1];
cc.oPosition[2]  =   phalange[1].cube[3][2];
   break;
       }
  }
```

```
    cc.cube = cc.translateCube(cc.cube, cc.oPosition);
    return cc;
   }

   //The cube is inclined, rotated, translated, projected
   private    Cube modifyCube(Cube cc, int m)
   {
    cc = inclination(cc,m);
     cc = rotation(cc,m);

    //cc.printCube(cc.cube, " after rotation");
    rePosit(cc,m);
    //cc.printCube(cc.cube , " after relocation");
    for(int i = 0; i<3;i++)
    cc.vTrans[i] = cc.vTrans[i] + cc.deltaTrans[i];
    cc.cube = cc.translateCube(cc.cube,  cc.vTrans);
    //cc.printCube(cc.cube, " after translation");
    cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
    //cc.printCube(cc.cubeProj,  " after projection");
    return cc;
   }
  }//End of class Finger


  //=============End of Inner class Finger============

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

  //=============Beginning of Inner class Cube============


  public   class Cube
  {
   private double points3d[][] = new double[8][3];
   private   int shade2d[][] = new int[8][3];

//   All cubes are derived from the basic cube by rotations
//   and translations
   private double cube[][]  = new double[8][3];
   private   double  cubeProj[][] = new double[8][3];
```

```
   private double oPosition[] = new double[3];
   private   double axisRot[] = new double[3];
   private   double axisProj[]  = new double[3];
   private   double deltaTrans[]  = new double[3];
   private   double vTrans[]  = new double[3];
   private   double theta  = 0;
   private   double deltaTheta;

   private   double basisRot[][], basisProj[][]
                = new double[3][3];
   private   double matrixRotz[][],
   matrixRot[][],
   matrixProj[][] = new double[3][3];
//   The matrix of a projection along the z-axis
   private     double J[][]=
         { {1,0,0},
           {0,1,0},
           {0,0,0}};

   private    double scale[] = new double[3];

   int colors[][] = new int[6][3];
   private   int pointsX[]  = new int[6];
   private   int pointsY[]  = new int[6];
//   How to color faces
   private    double maxX;
   private   int nExtreme;
   private   boolean maxPoints[]  = new boolean[8];
   private   boolean coloredFace[]  = new boolean[6];
   //Angle of inclination of phalange
   private   double phi[] = new double[3];;
   private   double deltaPhi[] = new double[3];;

  //Every implementation of Cube is initialized here.
  //Parameter m is used for the initial position
  //of cube number m
  Cube(int m)
  {
   //Original  form
```

```
scale[0] = 8;
scale[1] = 8;
scale[2] = 30;
for(int i=0; i<8;i++)
for(int j=0; j<3;j++)
points3d[i][j] =  -d.ampliFactor*scale[j]*oPoints3d[i][j];
defineColors();
//Original position
for(int i=0; i<3;i++)
 oPosition[i] = 0;
//Each phalange has its own angle of inclination
//with respect to its support
for(int i=0; i<3;i++)
{
 phi[i] = 0;
 deltaPhi[i]  = -0.1*(m+1);
}
//  The 3d cube flies along vectTrans[]
//Each cube has its own direction of translation
for(int i=0; i<3;i++)
 vTrans[i] = 0;
deltaTrans[0] =  0;
deltaTrans[1] =  0;
deltaTrans[2] =  0;
//  printCube();
//  The 3d-graphic is rotated along the axisRot
//  an angle theta that is incremented by deltaTheta.
//  Each cube has its own velocity and direction of
//  rotation.
theta = 0;
deltaTheta = -0.1;
axisRot[0] = 0;
axisRot[1]=0;
axisRot[2]=1;
//  The 3d-graphic is projected over the screen
//  which is perpendicular to the x-axis.
axisProj[0] = 1;
axisProj[1]=0;
axisProj[2]=0;
```

```
   }


  Random r  = new Random();

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

// The multiplication of two matrixes
 private  double[][] matrixMult(double p[][], double q[][])
 {
  double[][] helpM = new double[3][3];

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

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

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

```
  helpV[i]  =helpV[i]  + m[i][j]*v[j];
  return helpV;
 }


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

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

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

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

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


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

  helpT = thirdVector(v);
  helpS = secondVector(v);
  helpF = firstVector(helpT,helpS);
```

```
  for(int i=0; i<3;i++)
  {
   helpM[i][0] = helpF[i];
   helpM[i][1] = helpS[i];
   helpM[i][2] = helpT[i];
  }
 return helpM;
 }


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

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

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

 /*A 3d cube exists only in the imagination:
 * it is not seen.
 * One sees a shade or projection of it over the screen.
 * The shade is defined by 2d points. The edges of
 * the 2d shade are inherited from the edges of the
 * 3d cube.
 * The 3d cube is projected into a 2d plane.
 * The resultant figure must be rendered to the screen.
 * A projection is done along  an axis that crosses
 * through 0.
 * Our projections are observed from infinite:
 * we use no perspective corrections. */
 private     double[][] mProj(double w[] )
 {
  basisProj = basisB(w);
  double[][]  helpM = new double[3][3];
  helpM = matrixMult(J,matrixTranspose(basisProj));
  helpM = matrixMult(basisProj,helpM);
  //printMatrix(helpM, "Proj");
  //Test: Proj*Proj must be equal to Proj
  /*helpM = matrixMult(helpM,helpM);
  printMatrix(helpM, "Proj* Proj");*/
  return helpM;
 }

// The cube c is rotated around axis v an angle theta
 private double[][] rotCube(double c[][],
                            double v[], double theta)
 {
  double[]  helpV = new double[3];
  double cc[][] = new double[8][3];
  matrixRot = mRotation( v,theta);
  //printMatrix(matrixRot, "matrixRot");
```

```
  //The 8 points of the cube must be rotated
  for(int i=0; i<8; i++)
  {
   for(int j=0; j<3; j++)
   helpV[j] = c[i][j];
   helpV = matrixVector(matrixRot, helpV);
   for(int j=0; j<3; j++)
   cc[i][j] = helpV[j];
  }
  //printCube(cc, " after rot");
  return cc;
 }



// The cube c is translated   deltaV
 private double[][] translateCube(double c[][],
                                          double deltaV[])
 {
  double cc[][] = new double[8][3];
  for(int i=0; i<8; i++)
  for(int j=0; j<3; j++)
  cc[i][j] = c[i][j] + deltaV[j];
  return cc;
 }


// The cube c is projected along axis w
 private double[][] projCube(double c[][],  double w[])
 {
  double[]  helpV = new double[3];
  double cc[][] = new double[8][3];
  matrixProj = mProj(w);
  for(int i=0; i<8; i++)
   {
    for(int j=0; j<3; j++)
    helpV[j] = c[i][j];
    helpV = matrixVector(matrixProj, helpV);
    for(int j=0; j<3; j++)
    cc[i][j] = helpV[j];
```

```
   }
  return cc;
 }

// The rendering procedure accepts integer values
 private     int[][] round(double c[][])
 {
  int[][] helpM = new int [8][3];
  for(int i=0;i<8;i++)
  for(int j=0; j<3; j++)
  helpM[i][j] = (int) Math.round(c[i][j]);
  return helpM;
 }

 private   void printCube(double c[][], String a)
 {
  System.out.println("This is the cube" +a);
  for(int i=0;i<8;i++)
  System.out.println(c[i][0] + " " + c[i][1]+ " "
                                        + c[i][2]);
 }

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

 private void printShade()
 {
  System.out.println("This is the shade");
  for(int i=0;i<8;i++)
  System.out.println(shade2d[i][1] + " "
                                   + shade2d[i][2]);
 }

// Colors to paint the faces are defined
 private int[][] defineColors()
```

```
 {
  for(int l = 0; l<6;l++)
   {
    //We use random colors.
    //A color can be defined by three numbers,
    //each one from 0 to 255.
    //We has 6 faces, so we need
    //a matrix to keep colors.

    int c1,c2,c3;
    c1 = r.nextInt(226);
    c2 = r.nextInt(226);
    c3 = r.nextInt(226);
    //Colors are memorized
    colors[l][0]= c1;
    colors[l][1]= c2;
    colors[l][2]= c3;
   }
  return colors;
 }

// what faces must be colored
 private    void marked()
 {
  for(int f = 0; f<6; f++)
  coloredFace[f] = false;
  for(int f = 0; f<6;f++)
  {
   for(int p = 0; p<4;p++)
   {
    for(int c = 0; c<8; c++)
     {
      if (maxPoints[c] == true)
      {
        if (faces[f][p] == c )
        coloredFace[f] = true;
        //System.out.println("coloring");
      }
    }
   }
```

```
 }
}


// The x coordinate of all points are measured
// and the extreme ones are recorded.
private    void calculateColor()
{
 //The maximum value of coordinate x
 maxX =-1000;
 for(int l = 0; l<8; l++)
  {
   if (cube[l][0] > maxX)
    {
     maxX = cube[l][0];
    }
   }
 nExtreme = 0;
 //Number of extreme points
 for(int l = 0; l<8;l++)
  {
   maxPoints[l] = false;
   if (Math.abs(cube[l][0] - maxX )< 2)
   {
    nExtreme = nExtreme +1 ;
    maxPoints[l] = true;
   }
  }
  marked();
 }
 }//=============End of inner class Cube==============
}
}//end of main class
```

145 ***Exercise*** *Run the program and try to figure out which modifications were made to program Reflex, 76, to get this new code. Please, pay special attention to*

the next methods of the program OpposableThumb:
buildThumb(Graphics g)
locateFingers()
paintFinger(Graphics g, int j)
drawF (Graphics g)
These methods relocate the thumb to its new position.

146 ***Exercise*** *Improve the program to clean it from its appearance of being half cooked. Verify else refute that going from half cooked products to perfection demands huge amounts of computation that in this case is made by your mind and includes diverse trials, errors and further corrections that produce more errors.*

147 ***Throwing and clubbing***

Evolution explains the hominid hand as follows: ' It has been proposed that the hominid lineage began when a group of chimpanzee-like apes began to throw rocks and swing clubs at adversaries, and that this behaviour yielded reproductive advantages for millions of years, driving natural selection for improved throwing and clubbing prowess.. ..The resulting selection for improved throwing and clubbing prowess, prolonged over millions of years, led to numerous anatomical changes throughout the body, including those that characterize the evolution of the human hand.' (Young, 2003).

148 ***Exercise*** *Decide else refute the proposition that our half cooked hand with opposable thumb endows its carrier with reproductive advantage over those that carry a plane hand. If your answer is affirmative, as we think, an evolutionary gate is open for further exploration. Try to figure out why these hands do not appear in the fossil record. We predict that the sequence must come form a plane hand to a hand with opposable thumb to a hominid hand that can play both roles, plain and with opposable thumb. Explain why evolution goes almost contrariwise: from the hand of a lemur that has an opposable thumb but cannot be accommodated as a plane hand to a very complex hand of an hominid.*

149 ***Half cooked products***

Our hand is far from being a highly functional unit and hands of this sort are possible by myriads. How do we explain the absence from the fossil record and from modern populations the existence of half cooked hands with opposable thumbs? One might try various types of answers:

- By **lamarckism**: the need creates the organ.

- By randomness: mutations are at random but some times they can render the impression of being directed.
- By the poorness of the record fossil: half cooked products existed but they disappeared as the majority of fossils.
- By magical **neodarwinism**: evolution amounts to a computing machine whose capability surpasses that of all known or imagined computers and so its intermediate states cannot be registered.
- By intelligent design: intelligent beings solve problems in simulated, possibly digital, worlds and then they implemented their solutions in terminated forms.
- By magical creationism: God is an almighty magician and so one instant is for him more than enough to solve any problem or to execute any task.
- By Biblical creationism: God must work as hard as any one of us to develop software but as it is usual among developers, He also hid half cooked and twisted products that appeared through the development process.

Some observations are in order:

- Lamarkism is the default explanation of modern evolutionism. When evolutionism claims 'randomness', one must read **lamarckism**.
- It is true that sometimes randomness can mimic directed evolution. But, how many times? Some 2 or 3 among 10^{1000}. So, invoke randomness to explain the eye but not to explain the hand, the eye, heart, and the kidneys.
- According to what we know about SAT (a relatively simple problem in comparison with SOFT, software design), half cooked products are in general overrepresented with respect to full solutions. Hence, if fossils of terminated products are found, it is impossible not to find fossils of half cooked products.
- Evolution is nowadays a familiar form of solving problems and we find no one single reason tho think that it is a sort of super-quantum-computing: half cooked products whose corrections lead to even worst solutions are always expected. Notice that we do not deny that evolution is powerful to solve all problems. Instead, we refuse to believe that evolution can solve complex problems without leaving an indelible track way of its infinitely many half cooked products.

- Products made by intelligent beings might be free from half cooked quality only if they can model reality up to full precision, 100%. But this is impossible: after the theorems of G*ö*edel about the immanent incompleteness of mathematics, this belief is anymore acceptable. So, intelligent beings can produce model solutions but their implementation are guaranteed to have disfunctionalities, whose correction will cause more bugs. Therefore, perfection that lacks a long history of correction of errors is not expected from intelligent designers.

- Judaism and Christianity are religions in which G-d is not a magician. In these religions, G-d must work and work very hard no less than you or me to get what He desires. Nevertheless, we do not include G-d in the class of intelligent beings because He has plenty of time, while these beings not. Unlimited time can serve to solve problems but also to prepare beforehand computing facilities for specific classes of problems, so G-d is also armed with more computing power than intelligent beings. Dear Reader, I thank your kindness in letting me to express myself.

150 *Look at your left hand!*

Our study of SAT has allowed us to give force to some of our arguments. Nevertheless, the heart of the theme we are discussing needs no more than a look at our left hand.

Dear Reader, please, verify by direct experimentation on your left hand that your fingers are moved by a system of cables that are fastened to the forearm (not to the hand). To that aim, open and close your hand, while you observe what happens to the muscles of your forearm. You will notice that they also move.

Let us try to have our hand forming a right angle with the forearm. We also can move our fingers. But the system of cables does not pass through the minimum distance. Why? It is because the system of cables must run through a path that is delimited by special strips in the hand. The idea is a common one in mechanics and is depicted by the accompanying drawing.

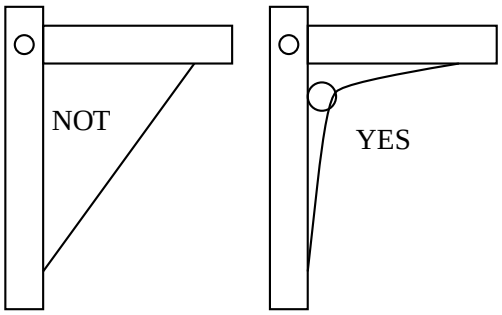


Figure 9.6. The systems of cables that allow you to move your hands and feet do not go through the path with minimum distance because they are forced to pass close to the hand-forearm or foot-leg structures. Please, look at an anatomy handbook to see the actual form as this is done.

How did this arrangement arise?

Let us consider the possibility that it arouse by evolution. We are unable of imagining that the aforementioned arrangement arouse at once in fully terminated functional form. We demand therefore to see the many trials that evolution made to arrive to this wonderful design. For instance, it would be delightful to see a fossil in which some cables are under the strip and others not. We do not find such a witness in human fossils, nor in those of chimps, nor in those of birds. Nor in those of amphibians. Nor in those of the first vertebrate fishes with articulated fins that already show a complex empowering structure.

151 *Playing the flute*

Flute is the problem that consists in designing a hand that could correctly posit fingers to play a flute that is tuned with a musical scale, say, that of birds. Flutes exist in most aborigine cultures and continues to be important feature in modern technologies, so there exists a lot of diversity of flutes.

The hand that we have simulated up to date is not capable of playing the flute. Why? Because traversal movements of the fingers are not enabled. So, we feel the duty of discussing the the problem of the evolution of the hand at the moment when its fingers acquire the capability of traversal movement. The discussion is oriented through exercises. The reason, dear Reader, is that these questions are very important for many people but a lot of discussion must be done before a consensus could be attained. Anyway, you can find the opinion of the Author in the Answers. The technical details of the comparison of the ape hand with that of the man can be read in Young (2003).

152 ***Exercise*** *Decide whether or not the ability to make traversal movements of fingers furnishes carriers with adaptive advantage and favorable reproductive differential potential by comparison with those that lack that power. Carefully distinguish among the cultural environment, say, a modern vs ancestral one. Prove else refute that the evolution of the hand and of the brain go together or by separated paths.*

153 ***Exercise*** *Figure out how evolution would begin with a plane hand to proceed to the development of a hand with traversal movements of fingers and with opposable thumb. Carefully distinguish between lamarkism and* ***neo-darwinism****. Compare your answer with the record fossil and comparative anatomy.*

154 ***Exercise*** *The* ***biogenesis law*** *of evolution reads: an embryo must repeat the evolution of its evolutionary lineage. This law is considered to be a relevant factor that gives power to the evolutionary theory. Test this law on the following predictions:*

1. *The phalanges of the chimpanzee are curved, well adapted to arboreal way of living. By contrast, those of man are not. Please, verify that in the human embryo, phalanges are first curved and then become straight.*

2. *The ape has four hands, the man only two. Hence, the human embryo must also have four hands in its early states.*

3. *Verify that you can make circular movements with your thumb. The ape cannot. This implies that you have 3 or 4 more muscles than the ape, because circular movement needs the coordination of at least three muscles. They must be nerved with a very precise computing and transmission system which enable your thumb to make very accurate movements. Therefore, the human embryo must have a state in which it has a thumb that lacks the exclusive to man muscles.*

4. *Can you make circular movements with you fingers others than the thumb?*

Discuss the implication of your answers.

155 ***Exercise*** *The process of ontogenesis of the hand does not differentiate fingers since the pure beginning. Instead, the hand begins to develop as if it were the foot of a duck, whose fingers are united by membranes. At the last stage, the membranes are digested by apoptosis, programmed death of cells, and the fingers gain their individuality (Puver et al, 2001).*

a . Does this process fit the ***biogenesis law*** *of evolution?*

b . Notice that apoptosis digests precisely what hinders the independence of the fingers. Explain how this arouse, first by ***lamarckism****, next by* ***neodarwinism****.*

9.4 Conclusion

We saw in Vol II that the genome is software and that evolution is software developing. SOFT or the problem of software developing is much more expensive than SAT but nevertheless it is at the foundation of modern culture, so we have gathered a lot of experience in the task of solving it. The most simple law about software developing is that the path from a first version to a terminated one is plagued with every sort of pitfalls and half cooked products. We also firmly believe that natural evolution is natural, i.e, it must conform to this law. Nevertheless, we see that natural products are the epitome of perfection and half cooked products are not found in nature or in the record fossil. That is why we claim that the software of the genomes corresponds to a version for the final user, from which every trace of the evolution toward perfection has been deleted.

Chapter 10

Conclusion

```
Believe or perish.
```

Our experience with Java for the study of evolution has shown us that problems might become extremely complex demanding high doses of creativity, expertise and endurance. Where is the energy to get all that? Our proposal is that the energy that is necessary to remain alive in this deal can be taken from personal **convictions**, whatever they could be. So, let us review some important points that enlighten the whole landscape of beliefs and **convictions**.

10.1 Beliefs

We discuss the arising of beliefs, the purpose of their existence, the relation of science with beliefs.

156 ***Beliefs are given for free***

Imagine a world in which **convictions** should be purchased at a very high price. Only some privileged people could enjoy one. But our world is quite contrary to that because the human mind seems to be a fabric of beliefs, which represent nascent **convictions**: any person can generate beliefs about any field in which he or she has some background information. Typical is the forecasting of weather. More to the point, the generation of beliefs is automatic: we have saw in chapter 2 that beliefs arise in the mind by similar mechanisms to those that generate visual illusions: they are produced by an automatic refined neural machinery and reported to the consciousness as if they were facts of nature (Rodríguez, 2007).

We contend that the machinery that synthesizes beliefs works on minimal inputs. The next are two examples that illustrate that some few objective elements are enough to generate beliefs that are relevant for our studies:

- A look at the left hand is enough for many minds to generate a belief in that we were created.
- The change of fossils of the same species along rock strata is sufficient to trigger the idea of evolution of species.

157 ***The purpose of beliefs***

A problem with the neural machinery is that it does not report the algorithm that was used to synthesize a given belief. This is a serious problem because beliefs enslave us. Nevertheless, the synthesis of beliefs does not demand huge amounts of data and changes of mind are not rare. Therefore, we must conclude that beliefs are synthesized for just one purpose: to motivate the brain to work further. For this reason, every belief or **conviction** is welcome in our community. Every one is invited to express him or herself succinctly. But the challenge is to extract wisdom and force from own **convictions**. Wisdom to formulate deep insights whose understanding could be enhanced with Java and with science in general and force to endure in the study of proposed problems.

A natural action for a person that believes is to try to understand oneself and to make predictions that might help to test beliefs. For instance,

- If one tries to understand what the left hand has that it compels many persons to believe that they were created, one discovers that its anatomical structure cannot be explained by chance, say the disposition of muscles in do-undo pairs is cosily understood as a plan by an engineer (chapter 4).
- If one tries to understand why the change in within an species motivates many people to belief that species can evolve one into another, then one can notice that the entrails of a hen, a fish, a sheep, and a human are all very similar. On the other hand, we all exist thanks to reproduction but nevertheless we are somehow different from parents. The record fossil shows us what we cannot witness during a human life span: that life evolves. Putting all these facts together, we get the evolutionary theory: all living beings are variants of some few ancestors.

158 ***Beliefs and science***

The official **purpose of science** is to study proposals to expand them deeply enough to produce important predictions that could be tested by further observation or by experiment (Popper, 1935). So, any belief can be the object of study of science. But a scientific rejection of an idea does not imply that a single mind will abandon it. Why?

The reason is that the human brain is capable of managing contradictory information (see chapter 2), so it must be clear for everybody once and for all times that a scientific rejection of a belief does not imply disbelief. Example:

Many people believe in God and in the evolutionary theory at the same time:

- This is contrary to science: the single point of science is that we are alone, that God is not with us, that Jesus will not come, that we appeared by chance, change and differential surviving.

- This is also against religion: a human being is a living soul, a fusion of matter and spirit, a creation that will be judged by Jesus in the final day. Now, if we appeared by chance, who does have the right to judge us? Nobody.

So, the evolutionary theory and religion reject one another. Now, all this is very clear to everybody. Nevertheless, people that believe in both God and the evolutionary theory are counted by millions, including some authorities of evolution: see a confession in Ayala et altri (2008).

Therefore, we conclude that to change minds is not and cannot be a **purpose of science**. In regard with beliefs, science has very concrete tasks, such as the next ones:

1. To study the process of generation of beliefs.

2. To understand how a belief passes from being an idea of a thinker to a **conviction** of a group.

3. To study how beliefs are nurtured and maintained by social groups.

4. To describe how and why beliefs and **convictions** evolve, to understand their reproduction, extinction and replacement.

5. To assess the scientific strength or weakness of a given belief.

10.2 World views

When a belief is elaborated, tested and refined, we get a **conviction**. There are **convictions** that serve as polarizing centers for the whole personality and way of life: these are the world views.

159 ***convictions are powerful***

Religions generate world views that are dressed in powerful **convictions** that awake strong feelings: many people consider that their religious ideologies have helped them to survive with success in this terrific world and so they give to their **convictions** the status of life donors, of gods. So, when their convictions are attacked, they feel that they have been attacked and in consequence a violent defense might be observed. As we see, this is a matter of personal surviving and so one can expect the very same type of reaction from any person that shares any world view. In general, we feel so proud of our convictions that we would like that our sons and daughters had the very same convictions that we have and so we indoctrinate them. Why? Because we hope, as a minimum, that this will enhance their probabilities of surviving and being happy.

In our community, world views are necessary: we cannot imagine a person that can endure in this deal without the force given by firm **convictions**.

160 ***Rival world views***

Since every world view has been tested a lot of times by many, many people and with very diverse procedures, it is amazing to find world views that are contrary one to another. Most astonishing is to find world views that differ in their conception of the very reality: materialism teaches that hadrons, leptons and radiation is all to existence while spiritualism claims that the spirit, which is not material, is more important than the body, because the former is eternal while the second perishes.

How is this craziness possible?

The scientific study of the problem of the arising of rival world views is a complex theme but very exiting. In particular, it seems that this phenomenon is not only possible but also mandatory according to first principle of modern physics and mathematics (Primas, 1983). Moreover, other non fundamental reasons also conspire to produce people with different perceptions of reality:

1. In the process of knowing we distinguish three entities: consciousness (the entity that knows), reality (the object that is known), senses and mind (the

neural machinery that perceives the reality, studies it and reports the main conclusions to the consciousness). The inform presented by the mind to the consciousness is called the artificial inner reality. This artificial reality is not a faithful representation of the exterior reality but a highly elaborated inform that discriminates reality in units of representation. Say, when we see, we do not perceive myriads of photons, instead we perceive a world with individualized objects, say, trees, persons and computers.

2. Beliefs are mental constructs that arise when the mind can recognize a given pattern in the data. But a price must be paid: some aspects of reality or of the inner being are declared valid while many other objective or mental facts are ignored or are given the status of noise. Example: Christendom stands up because accepted facts and noise are easily distinguished one from another:

    - Accepted fact: Many people witness that God gives peace to the heart.
    - Noise: the history of the world witnesses that the peace and wealthy of believers flourishes over the blood and misery of their victims: just give a look to Africa or to the blooding veins of Latin America(Galeano, 1988).

3. Mature **convictions** always have tenets that are very important for many persons:

    - Religion: *To remember His holy covenant, the oath He (God) swore to our father Abraham: to rescue us from the hand of our enemies, and to enable us to serve him without fear in holiness and righteousness before him all of our days* (Luke 1: 73-75).
    - Evolution: *There is grandeur in this view of life, with its several powers, having been originally synthesized by chance into a few forms or into one; and that, whilst this planet has gone cycling on according to the fixed law of gravity, from so simple a beginning endless forms most beautiful and most wonderful have been, and are being, evolved.* (Darwin, updated by the Author.)

4. Mature **convictions** always have features that are object of science and that are in agreement with logic and experimentation:

    - One of the most astonishing prophesies of the Bible is found in the book of Genesis: *In the sweat of thy face shalt thou eat bread, till thou return unto the ground; for out of it wast thou taken: for dust thou art,*

and unto dust shalt thou return (The King James Version). From the anthropological stand point, this assertion is no more than and advice given by the elder members of a well developed agricultural society with a two fold purpose: when everything goes well, to battle the thirst of producing richness and rather to explore non profitable forms of being happy; and, when everything goes wrong, to return to the shelter of God, the father of all us. But, since the Bible is a Holy Book, this assertion also contains the prophesy that we always will experience the need to work hard to survive or to maintain our status. This prediction is challenging because it is quite contrary to what one might reasonably expect: social and scientific development together with peace in the world will eventually produce rest for every human being from being enslaved to work and instead non obsessive personal realization might be looked for. But this fails to happen. Among the many data that might be used to explain why we always must work so hard we find the dismantling of the Soviet Union (the faith in social order) together with the insufficiency of democracy (the faith in regulated egoism and creativity) to keep itself safe from tyranny or from socialist outbursts, such as those we see today in Latin America.

- A fundamental tenet of modern evolutionary theory that is corroborated at every moment in the study of viruses and of human kin relations is that mutation is at random and in normal conditions of modern populations it happens in average at a very low rate. The relative successful battle against pandemic diseases cannot be imagined without the knowledge and application of these principles.

5. Mature **convictions** are stable against the arguments of rival proposals. Operationally, what one sees is that challenging insights that are accepted in a given world view are considered by followers of rival **convictions** as stupidities:

   - Religion is stable against atheism and science: *The fool hath said in his heart, There is no God. They are corrupt, they have done abominable works, there is none that doeth good*, (Psalm 14:1, The King James Version).
   - Science is stable against religion: *Does God bring cold and winter? So, let Him bring cold and winter in summer!* (Russian joke of the socialist era meaning that God has no power at all and that progress must come from a coherent scientifically optimized effort of all the people).

In general, a model that has been built with the purpose of reinforcing the evolutionary theory ipso facto reinforces creationism. The reason is that evolution is today a tool to solve the most diverse problems in science, technology and art. So, a creationist might look at an evolutionary model and conclude: if humans can use evolution for their purposes, how much God can use it for his own ones. And viceversa, if a creationist will produce a model, if any, it will be good in the measure in which it reinforces materialism, otherwise it will be classified as noise and therefore discarded. This shows in action the stability of world visions: they must be capable of absorbing and digesting any important feature of rival world views.

6. Mature **convictions** are stable against inner instabilities:

   - Religion: *For my thoughts are not your thoughts, neither are your ways my ways, saith the LORD. For as the heavens are higher than the earth, so are my ways higher than your ways, and my thoughts than your thoughts* (Isaias, 55:8-9). This is a very polite form of saying that God is a logical contradiction and that He is not with us on the day that we most need Him. Anyway, believers do not cease claiming God for help, company and blessing for their children.
   - Science: *Live and let die because there is nothing else to life*. This description of the hopeless of materialism does not hinder people from living and letting die, eating and drinking and telling that they are happy.

7. Mature world views are stable against the criticisms of science:

   - We read in the Genesis that God made Earth and Heavens not all at a sudden but through a process. In particular He made light before sun, moon and stars. This is contrary to science: modern science cannot imagine daylight without thermonuclear reactions in the sun in which hydrogen is burnt to make helium. Believers could say that the purpose of the Genesis was not to produce a scientific paper but to warn everyone that to worship the sun, the moon and stars, as the followers of horoscope do, is a great sin because these are creations made by the true and only God.
   - We all are amazed by the achievements of science and technology, but science is from the social point of view a subculture that is weak enough to be shameful: its relation with religion looks more as an alliance than as the battle dictated by the international literature and that

endorses science as a world view. Nevertheless, with very few exceptions, scientists happily go around.

10.3 The role of Java

Let us make now a clear cut assessment of the role of Java for the study of evolution to help us to understand the beliefs that rule us.

161 ***Java as a neutral tool***

We observe, in first instance, that Java for the study of evolution can play the role of a neutral tool that can be used with any purpose by followers of a whatever world view:

- Our exploration of the magic of mouse clicking in chapter 5 allows us to correctly understand the God of Moses, which is human-like and so His creative acts are dressed in hard work in the very same way as we can program magic acts in a computer if only we can make a program that execute for us the magic we are looking for.
- We have reviewed in chapter 6 a model called biomorphs, proposed by Dawkins (1986), which points to the immense possibilities of evolution to create morphological diversity, a crucial point in supporting the belief in the evolution of species. Biomorphs is also a piece of computing art, an area that we have enriched with our own version of an evolutionary linguistic model that creates paintings (see the same chapter).

162 ***Java is powerful for the study of complexity***

Given that there are and must be rival world views, we must ask ourselves: which are the weaknesses of the evolutionary theory because of which it cannot defeat rival religious world views in the market of ideologies?

Our **conviction** is that the crucial weakness of evolution is that it deals very poorly with complexity. Let us explain this more clearly.

Evolution can be compared to a machine that must reach the highest peak over a landscape. The fuel is chance and natural selection is the driver. Chance alone is not reliable but the duet composed of chance and natural selection is. Now, this assertion is not a dogma. It is an expectation that must be studied and corroborated else falsified.

What does Java for the study of evolution say? Our main points are:

1. We have reviewed in chapter 7 the Shakespeare model of complexity, which proposes a phrase that could be one of Shakespeare and that must be guessed, and we have seen that it is a very easy challenge for evolution. So, we follow Dawkins (l.c., 1986), the proponent of this model, in his conclusion that evolution is powerful to create complexity. Because our Java simulations allows us to exactly see what the Shakespeare model relies on, we could criticize its immanent simplicity and propose more realistic models.

2. As we have seen in Vol II, the genome is software and evolution is software developing. Therefore, the correct model of biological evolution is SOFT or the problem of software developing, which consists in developing a piece of software that implements a given function. Our encounter with this model was a failure: SOFT is not a problem to be studied with hurry using a single desktop. So, we must look for a model of intermediate level of complexity, it must be complex enough to be interesting but solvable with a single desktop.

3. To clearly understand that there are more complex problems than Shakespeare and that evolution can solve them, we proposed in chapter 8 to study SAT, the satisfaction problem of mathematical logic. We have witnessed that evolution can solve it but only if exaggerated large number of trials are done wasting huge amounts of resources. Something similar happens with SOFT: the human being is at present the only reliable agent capable of synthesizing software but evolution is step by step becoming a help in this hard enterprize. The problem is with resources: large lapses of time or/and enormous volumes of memory.

4. We postulate that evolution can solve any problem, SAT or SOFT or whatever, if only we have enough resources.

5. The common denominator of complex problems is that to arrive to the solution of a problem, half cooked products persist during long times, which are longer the more complex are the problems.

Our interpretation of these conclusions is the following:

- It is sane to propose the evolutionary theory as an explanation of biological complexity.

- If we are ready to follow science, our proposal produces a prediction: any evolutionary path must be plagued with every sort of pitfalls and half cooked products. Therefore, to scientifically explain life by selection of random mutations one must exhibit full hands of half-cooked products as in the fossil record as in modern populations.

- This prediction is not corroborated by nature: while every scientists postulates that mutation is at random, in agreement with our simulations, we see that natural products are the epitome of perfection and half cooked products are not found in nature or in the record fossil. Instead, when we find a half-cooked, twisted, deformed or hunchbacked being, we rapidly invokes a mutation from normal parents but no one succeeds to see in it a local optimum, i.e., a half cooked product synthesized by evolution in its process of gaining perfection.

- We conclude that modern evolutionary theory is the perfect implementation of **lamarckism**: the necessity creates the mutation and so no half cooked product is expected.

- We are aware of the fact that our conclusions are challenging and that are very difficult to study. That is why we consider that **convictions** are a treasure for everybody because they give endurance to remain in this deal: believe or perish.

163 ***Examples***

Let us illustrate why modern evolutionary theory is no more than **lamarckism**, according to which the need creates the mutation, or that uphill mutations always easily happen. To that aim, let us consider three examples:

1. Let us pay attention to the anatomical localization of the eyes in a bird, say a pigeon: they do not look to the front, instead they look to the side. Why? Those birds whose anatomical organization of the eyes allowed them to detect on the fly predators coming from behind and to know what they have in front left more progeny than those than not. Therefore an evolutionary process was triggered that ended in the present anatomical organization. Now: this explanation is not **neodarwinism**, it is no more than disguised **lamarckism**. The reason is that it lacks the proof that the selected mutants were generated at random: we lack the fossil evidence proving that there was a process in which eyes migrated from somewhere to the actual position and very specially we lack the evidence in the record fossil or in natural populations of mandatory malformations.

2. Humans that lie in bed because of a terminal illness know with a precision of a week or less that they are going to die and in the very last two or three days they cease to feel pain or sorrow. We must solve two questions. First: How do they know that they are up to die if they never have died before? Second:

How could had been selected for those aforementioned characteristics given that in normal conditions no dead person leaves progeny? The answer might very simple as the following: by the same token as there exist programmed cell death, apoptosis, there also exists programmed organismic death, which is accompanied with release of endorphims and other relaxing hormones. A dead in peace is impressive for us as it was impressive for ancient people, creating a trend of selection of the progeny of the dead person that enjoyed those characteristics. Now, an explanation of this sort is disguised **lamarckism**. The reason is that it lacks the proof that selected mutants were generated at random: where are the living populations whose members die amidst cries of war? Such a characteristic should had been very valuable for ancient populations pressed by violence and war, a situation that is very usual among humans. Therefore, we must expect from us to die among cries of war. This is false even for warriors.

3. Our hands are plain but their thumbs are fully opposable and we can separate fingers so that we can play the flute. How did such a wonder come into existence? By evolution: an evolutionary trend exists beginning from the fins of fishes to early reptiles to early mammals to apes to human beings. Now, this accepted explanation is disguised **lamarckism**. The reason is that it lacks the proof that selected mutants were generated at random: where are the expected thousand of malformations as in the record fossil as in modern populations that should had appeared were mutations at random? We consider that this particular problem deserves to be studied at dept by our community and to that aim we have made the first steps in chapters 3,5 and 9.

10.4 Evolution and computing science

We are not against the idea that **lamarckism** could be the correct explanation of our existence. But computing science is. Let us see why:

Evolution that explains complexity and function is similar to a rise across a landscape with the aim of searching for one of the highest peaks. Thus, evolution computes the solution of an optimization problem. Lamarckism might be the correct solution if problems were easy for evolution: it would find the solution without much effort, leaving few wastes, which were not enough to generate intermediate fossil records or suboptimal surviving populations. The problem with this proposal is that evolution is today a current approach to problem solving in computing science. And everything that we know about it says us that ordinary problems are difficult for evolution as well as for any other computing paradigm or

methodology. This means that if mutation is at random, full hands of intermediate half cooked products are expected. But these are not shown in the fossil record neither in living populations. That is why **lamarckism** is ruled our by computing science.

164 *Epilogue*

The evolutionary theory is nowadays, the beginning of the XXI century, considered to be the major treasure of modern science and in fact nothing in science makes sense if not in the light of a very extended evolutionary theory: from elementary particles to atoms to molecules to cells to organisms to ecosystems to planets to solar systems to galaxies to clusters to the whole universe, everything fits an evolutionary process which is depicted by the Big Bang theory and that explains our coming into existence and our unavoidable collapse.

Now, it is our duty to inform evolutionists that they pay a price for their **conviction**: they must close their eyes to the fact that the statistical properties of randomness, studied by the theory of probabilities and tested million of times everyday in ordinary life and in the laboratory, are not observed by the chance invoked by the evolutionary theory to sustain its whole building. As a consequence, modern evolutionary theory is no more than mere **lamarckism**, in which improving mutants appear without a struggle for life among many, many suboptimal variations.

Chapter 11

Glossary

Algorithmic genomics: It is the science whose foundation is the fact that the genome is software. It is computing science that is applied to the study of the genome.

Evolutionary algorithmic bio-complexity: This is a chapter of algorithmic genomics which applies to the evolutionary process the laws that have been formulated in the study of the software that has been developed by human beings. The more important law is that the development of software is a very difficult task. Therefore, any path along a developing process must be filled in a rich story of intermeiate products. We have therefore a prediction: the record fossil and extant populations must be filled in half-cooked products. This science was born dead because this mandatory predictions fails.

API: An application programming interface. It is a specialized library mainly of methods or modules to ease the building of applications in a specific field.

Belief: An idea that is taken as true or real.

Biogenesis law: 'Embriogenesis recapitulates evolution' or 'every living thing in its individual development passes through a series of constructive stages like those in the evolutionary development of the race to which it belongs'(Patten, 1958). This is also known in molecular biology as Haeckel's evolutionary paradigm of embryonic development, where an intermediate in the assembly of a complex represents a form that appeared in its own evolutionary history' (Levi, 2008).

Cladistics: Clad means branch, so the purpose of cladistics is to organize data in a binary tree whose structure is determined by relations of maximal similitude and minimal difference.

Christianity: The spiritualist religion that claims that spirits are organized in two political parties: to be with God and with His Son Jesus or to be against Them. (There are religions that are not spiritualist because they have as god a living creature, say, Maradona).

Conjunctive form: An expression like $(\sim p \vee \sim q \vee \sim r) \wedge (\sim p \vee q \vee \sim r)$. It is an *And* package of propositions, every one of which is string of variables, negated or not, joined by the *OR* operator. Each *OR* string is called a clause. The normal conjunctive form is full when every clause contains all variables of the discourse. The formula $(\sim p \vee \sim q \vee \sim r) \wedge (\sim p \vee q \vee \sim r)$ has two clauses over three variables.

Consciousness: the perception of the self amidst a world and a society. It observes and perceives. For some people, consciousness resides in the spirit. For modern science, consciousness is an illusion created by the brain.

Conviction: A belief that has been tested thoroughly and that can be adopted as a dogma by a group.

Darwinism: The philosophical envelope of the historical development of the ideas of Darwin, whose fundamental contribution was to suggest that the variability of populations plays and adaptive role and so the environment selects those individuals that carry the appropriate conditions to survive and reproduce further. In the long range, he expected to have that populations evolve and new species arise. Darwin did not have a mechanism for the origin of variability. This vacuum was filled in by Mendel.

Demon: Evil spirit that torture people and to relieve them, demons demand worshiping and sacrifices that maybe of a human person, say, a baby.

Daemon: In computer science, it is program that executes a task as soon as it is needed. This is one of the most impressive implementations of the OOP philosophy.

Disjunctive form: A proposition of the form $(p \wedge q \wedge r) \vee (p \wedge \sim q \wedge r)$. It is an OR package of propositions, every one of which is string of variables, negated or not, joined by the AND operator. Each AND string is called a clause. The normal disjunctive form is full when every clause contains all variables of the discourse. The formula $(p \wedge q \wedge r) \vee (p \wedge \sim q \wedge r)$ has two clauses over three variables.

Dogma Belief adopted not by testing but because of authority. Dogmas are very important for surviving: they serve as world views and because they are received as inheritance, one must not worry because of testing of beliefs but instead ones receives truths already made in terminated form. Many mathematical results are dogma for those that hate mathematics. Materialism is a dogma for many scientists that do not want to waste their time discussing questions that for them are ridiculous, say, why little children fear the darkness (they can feel ghosts but they experiment relief if they feel the light.)

Idealism: The philosophical position in which the mind and its inner artificial reality is blamed for strong interference against the possibility to know the exterior reality -if any. So, the being is no more real that our thought allows it to be.

Inner artificial reality: The report of the mind to the consciousness. It is composed of a depiction of the exterior reality together with sensations about the organisms, mud and beliefs.

Ghosts: Faint creatures that can speak by telepathy, trespass walls and cause the hair of the skin to bristle and that up to date are registered by persons but not by reliable experimental protocols. Because of this last reason, their real existence is denied by many people and it is not recognized by science.

Lamarkism: A thesis proposed by Lamarck at the beginning of the XIX century claiming that *the need creates the organ*. While modern science openly rejects lamarckism, evolutionists of the XX century have peacefully made a living thanks to a mutated form of lamarckism: *the need creates the mutation*.

Matter: in materialism it is a synonym of reality. It is composed of atoms, electromagnetic radiation, elementary particles.

Materialism: the philosophical system that claims mainly that mind and consciousness are byproducts of matter and that matter is registrable by matter.

Mind: the processor that builds an inner artificial reality that is the only object of perception of the consciousness.

Neodarwinism: The school of evolutionary thought resulting from blending the ideas of Darwin with those of Mendel. Darwin explained the role of natural selection to filter extant variability as the agent of evolution. Mendel experiments and ideas lead to the discovery of genes as the carriers of inheritable variation and of mutation which is the ultimate source of variability.

Java 3D API: The API of Java to ease the programming of three dimensional graphics.

Oracle: Mythical programs that solve a specific problem immediately and without cost for the User. An oracle is simulated in computer science by a real program that solves the problem beforehand and so it is ready when needed.

Philosophy: the mania to formulate and discuss questions about the being, the consciousness and the way to acquire and generates knowledge.

Reality: what exists outside of the mind and independent of it.

Religion: from the Latin word re-ligare, to be strongly tied to God like with a laze.

SAT or the satisfiability problem of mathematical logic: Consists in studying the assignment of truth values to the variables of a proposition with the purpose of making it true.

SOFT: The problem of developing software. The developer could be a conscious being, like a human, or an unconscious process, like evolution. If the con-

scious being has a purpose, we speak of design. The evolutionary theory reads: evolution is the developer that is responsible for the existence of the genome, the natural software. Evolution has no purposes, so it is not a designer, but acquires an emergent purpose: to gain more and more surviving capability, so evolution has a purpose and its developed software must comply with a function: augment differential surviving and reproduction. Therefore, all what human designers know about software design is also valid for natural evolution.

Spirit: A conscious being that is not composed of the matter that is listed in the periodic table. Spirits communicate with people through voices, visions and dreams. Some persons can see them but most people cannot. According to many cultures, demons are spirits that induce and reinforce bad actions in the people no matter that they can be felt or not. According to Christianity, all beings with a spirit, humans, angels and demons, will be judged by Jesus, whose name means salvation.

Spiritualism: The belief that reality is composed not only of matter but that also contains spirits. In particular, every human being has a soul that after dead becomes a ghost. Let us notice that magic, miracles and the like have plenty of witness in all cultures and spiritualism is mainly a mechanistic theory about them: all paranormal phenomena are tied to a spirit. For instance, some people are famous because when they read the tarot cards to know the luck of a person, their prophecies inexorably get fulfilled. How does this divination happen? The fatalist interpretation says: they read the future. The spiritualist interpretation replies: the future does not exist, so it cannot be read. Instead, it is created and the form as this is done can be influenced, to some step, by spirits. Judaism explicitly forbids divination. Christian believers associate divination with evil spirits, even though they have prophets that enjoy to predict the future in almost the very same fuzzy way as tarot readers. If you see somebody that claims to be a prophet of God, please, challenge him or her to express very concrete and specific prophecies. For instance, instead of saying: you will be happy today, let that prophet say: a girl with name Felicity will approach you today at 10:11 at the cafeteria and you will be very happy to meet her.

Chapter 12

Answers to exercises

Problems of Chapter 2

11 The code for a cube that rotates and is translated follows:

```
/* Program D11, FlyingCube
   A flying 3d cube that  rotates.
   Translation is added.
   */

import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;

public class FlyingCube extends Applet
 {
  private static final long serialVersionUID = 1L;


  /*
   Our basic cube is the unitary cube at the origin.
   It is composed of 8 points or vertexes = points3d
   and 12 edges.
   Points are numbered from 0 to 7.
   These numbers are used in
   edges, which  records the pairs of
    points that are joined by a line. */
```

```
private static   double points3d[][]=
                          { {0,0,0},
                         {1,0,0},
                         {0,1,0},
                         {0,0,1},
                         {1,1,0},
                         {1,0,1},
                         {0,1,1},
                         {1,1,1}
                        };

private static int edges[][] =
                              {{0,1},
                            {0,2},
                            {0,3},
                            {1,4},
                            {1,5},
                            {2,6},
                            {2,4},
                            {3,5},
                            {3,6},
                            {4,7},
                            {5,7},
                            {6,7}};

/*A 3d cube exists only in the imagination:
 * it is not seen.
 * One sees a shade or projection of it over the  screen.
 * The shade is defined by 2d points. The edges of
 * the 2d shade are inherited from the edges of the
 * 3d cube.
 */
private static int shade2d[][] = new int[8][3];

//All cubes are derived from the basic cube by rotations
//and translations
private static double cube[][]  = new double[8][3];
private static double  cubeProj[][] = new double[8][3];
```

```
  private static double axisRot[] = new double[3];
  private static double axisProj[]  = new double[3];
  private static double deltaTrans[]  = new double[3];
  private static double vTrans[]  = new double[3];
  private static double theta  = 0;
  private static double deltaTheta;

  private static double basisRot[][], basisProj[][]
                                        = new double[3][3];
  private static double matrixRotz[][],
                        matrixRot[][],
                        matrixProj[][] = new double[3][3];
  //The matrix of a projection along the z-axis
  private static   double J[][]=  { {1,0,0},
                                    {0,1,0},
                                    {0,0,0}
                                   };
  private static  double scale;
  private static int delay ;
  private static int nScenes ;

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

  //The multiplication of two matrixes
  private static  double[][] matrixMult(double p[][],
                                double q[][])
  {
  double[][] helpM = new double[3][3];

  for(int i=0; i<3; i++)
for(int j=0; j<3; j++)
```

```
 helpM[i][j] =0;

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

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

 //------------------------------

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

```
return helpV;
 }

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

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

 //The first vector is the crux product of the other 2.
 private static  double[] firstVector(double[] v,
                                double[] w)
 {
 double a,b,c;
 double[] helpV = new double[3];
```

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

  }


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

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

return helpM;
  }

 //------------------------------


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

```
 mRotz[0][2] = 0;
 mRotz[1][0] = -Math.sin(theta);
 mRotz[1][1] = Math.cos(theta);
 mRotz[1][2] = 0;
 mRotz[2][0] = 0;
 mRotz[2][1] = 0;
 mRotz[2][2] = 1;
 return mRotz;
}

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

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

 //The 3d cube is projected into a 2d plane.
 //The resultant figure must be rendered to the screen.
 //A projection is done along  an axis that crosses
 //through 0.
 //Our projections are observed from infinite:
 //we use no perspective corrections.
 private  static  double[][] mProj(double w[] )
 {
```

```
  basisProj = basisB(w);
  double[][]  helpM = new double[3][3];
  helpM = matrixMult(J,matrixTranspose(basisProj));
  helpM = matrixMult(basisProj,helpM);
  //printMatrix(helpM, "Proj");
  //Test: Proj*Proj must be equal to Proj
  /*helpM = matrixMult(helpM,helpM);
   printMatrix(helpM, "Proj* Proj");*/
  return helpM;
  }

  //The cube c is rotated around axis v an angle theta
  private  static   double[][] rotCube(double c[][],
                        double v[], double theta )
  {
double[]  helpV = new double[3];
double cc[][] = new double[8][3];
matrixRot = mRotation( v,theta);
//printMatrix(matrixRot, "matrixRot");
//The 8 points of the cube must be rotated
for(int i=0; i<8; i++)
{
 for(int j=0; j<3; j++)
   helpV[j] = c[i][j];
 helpV = matrixVector(matrixRot, helpV);
 for(int j=0; j<3; j++)
 cc[i][j] = helpV[j];
}
//printCube(cc, " after rot");
return cc;
  }



//The cube c is translated   deltaV
  private static double[][] translateCube(double c[][],
                                     double deltaV[])
  {
```

```
   double cc[][] = new double[8][3];
   for(int i=0; i<8; i++)
    for(int j=0; j<3; j++)
   cc[i][j] = c[i][j] + deltaV[j];
   return cc;
  }


  //The cube c is projected along axis w
  private static double[][] projCube(double c[][],
                                  double w[])
  {
double[]  helpV = new double[3];
double cc[][] = new double[8][3];
matrixProj = mProj(w);
for(int i=0; i<8; i++)
{
 for(int j=0; j<3; j++)
   helpV[j] = c[i][j];
 helpV = matrixVector(matrixProj, helpV);
 for(int j=0; j<3; j++)
 cc[i][j] = helpV[j];
}
return cc;
  }

  //The rendering procedure accepts integer values
  private  static  int[][] round(double c[][] )
  {
  int[][] helpM = new int [8][3];
for(int i=0;i<8;i++)
 for(int j=0; j<3; j++)
helpM[i][j] = (int) Math.round(c[i][j]);
return helpM;
  }

  private static void printCube(double c[][], String a)
  {
System.out.println("This is the cube" +a);
    for(int i=0;i<8;i++)
```

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

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

  private void printShade()
  {
System.out.println("This is the shade");
    for(int i=0;i<8;i++)
     System.out.println(shade2d[i][1] + " "+shade2d[i][2]);
  }


  private static  void initialize()
  {
scale = 50;
for(int i=0; i<3;i++)
vTrans[i] = 0;
for(int i=0; i<8;i++)
 for(int j=0; j<3;j++)
  points3d[i][j] = scale*points3d[i][j];
//printCube();
//The 3d-graphic is rotated along the axisRot
//an angle theta that is incremented by deltaTheta
theta = 0;
deltaTheta = 0.01;
axisRot[0] = 1;
axisRot[1]=1;
axisRot[2]=1;
    //The 3d-graphic is projected over the screen
    // which is perpendicular to the x-axis.
axisProj[0] = 1;
axisProj[1]=0;
```

```
axisProj[2]=0;
//The 3d cube flies along vectTrans[]
deltaTrans[0] = 0;
deltaTrans[1] = -1;
deltaTrans[2] = 0;
nScenes = 500;
//delay between two scenes
delay =  10;
  }

public void paint (Graphics g)
{
 System.out.println("Cube rotates");
 initialize();
 int a,b,c,d=0;
 //background color
 g.setColor(Color.pink);
 g.fillRect(0,0,200,200);
 //Loop for the scenes
 for(int k = 0; k<nScenes; k++)
 {
  theta = theta + deltaTheta;

  cube = rotCube(points3d,axisRot,theta);
  //printCube(cube, " after rotation");
  for(int i = 0; i<3;i++)
  vTrans[i] = vTrans[i] + deltaTrans[i];
  cube = translateCube(cube, vTrans);

  cubeProj = projCube(cube,axisProj);
  //printCube(cubeProj, " after projection");
  shade2d =  round(cubeProj );
  //printShade();
  //draw cube
  g.setColor(Color.blue);
   for(int i=0;i<12;i++)
   {
a =   100-shade2d[ edges[i][0]  ] [1];
b =   100-shade2d[ edges[i][0]  ] [2];
c =   100-shade2d[ edges[i][1]  ] [1];
```

```
d =   100-shade2d[ edges[i][1]  ] [2];

/*System.out.println(edges[i][0] + " " + edges[i][1]);
System.out.println( "i= " +i + " " +a + " "
                                  +b+ " " + c+" " +d);*/
g.drawLine(a,b,c,d);
   }
   //delay
   if (k < nScenes-1)
    {
     try
      {
   Thread.sleep( delay );
      }
      catch (InterruptedException e)
      {
   e.printStackTrace();
      }

     //erase cube: we repaint the cube in
     //the background color.
     g.setColor(Color.pink);


    for(int i=0;i<12;i++)
     {
  a =   100-shade2d[ edges[i][0]  ] [1];
  b =   100-shade2d[ edges[i][0]  ] [2];
  c =   100-shade2d[ edges[i][1]  ] [1];
  d =   100-shade2d[ edges[i][1]  ] [2];
  g.drawLine(a,b,c,d);
     }//end of for i
  }//end of if
 }//end of for nScenes
}//end of paint
}//end of class
```

13 An opaque solid cube is encoded as follows:

```
/* Program D13,  cubeSolid
   A flying rotating 3d solid cube.
   To generate the sensation of a solid cube,
   we color only those faces that are closer to the
   observer. To this aim, we measure the x-coordinate
   of each vertex and determine the extreme ones.
   One extreme vertex is chosen and the three
   adjacent faces are colored.
   */

import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Polygon;
import java.util.Random;

public class cubeSolid extends Applet
 {
  private static final long serialVersionUID = 1L;


  /*
   Our basic cube is the unitary cube at the origin.
   It is composed of 8 points or vertexes = points3d
   and 12 edges.
   Points are numbered from 0 to 7.
   These numbers are used in
   edges, which  records the pairs of
   points that are joined by a line.
   A face is determined by its 4 corners */


  private static   double points3d[][]=
                           { {0,0,0},
                         {1,0,0},
                         {0,1,0},
                         {0,0,1},
```

```
                        {1,1,0},
                        {1,0,1},
                        {0,1,1},
                        {1,1,1}
                       };

  private static int edges[][] =
                              {{0,1},
                           {0,2},
                           {0,3},
                           {1,4},
                           {1,5},
                           {2,6},
                           {2,4},
                           {3,5},
                           {3,6},
                           {4,7},
                           {5,7},
                           {6,7}};

  private static int faces[][] =
                              {{0,1,4,2},
                           {0,2,6,3},
                           {0,3,5,1},
                               {3,5,7,6},
                               {1,4,7,5},
                               {2,6,7,4}};


  /*A 3d cube exists only in the imagination:
   * it is not seen.
   * One sees a shade or projection of it over the  screen.
   * The shade is defined by 2d points. The edges of
   * the 2d shade are inherited from the edges of the
   * 3d cube.
   */
  private static int shade2d[][] = new int[8][3];

  //All cubes are derived from the basic cube by rotations
  //and translations
```

```
private static double cube[][]  = new double[8][3];
private static double  cubeProj[][] = new double[8][3];

private static double axisRot[] = new double[3];
private static double axisProj[]  = new double[3];
private static double deltaTrans[]  = new double[3];
private static double vTrans[]  = new double[3];
private static double theta  = 0;
private static double deltaTheta;

private static double basisRot[][], basisProj[][]
                                      = new double[3][3];
private static double matrixRotz[][],
                      matrixRot[][],
                      matrixProj[][] = new double[3][3];
//The matrix of a projection along the z-axis
private static   double J[][]=  { {1,0,0},
                                  {0,1,0},
                                  {0,0,0}
                                 };
private static  double scale;
private static int delay ;
private static int nScenes ;

static int colors[][] = new int[6][3];
private static int pointsX[]  = new int[6];
private static int pointsY[]  = new int[6];
//How to color faces
private static  double maxX;
private static  int maxXPoint;
private static  int maxZPoint;

private static int nExtreme;
private static boolean maxPoints[]  = new boolean[8];
private static boolean coloredFace[]  = new boolean[6];

static Random r  = new Random();

//The transpose of a matrix
private static  double[][] matrixTranspose(double m[][])
```

```
  {
double[][] helpM = new double[3][3];
 for(int i=0; i<3; i++)
  for(int j=0; j<3; j++)
   helpM[i][j] =m[j][i];
return helpM;
 }

  //The multiplication of two matrixes
  private static  double[][] matrixMult(double p[][],
                                 double q[][])
  {
  double[][] helpM = new double[3][3];

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

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

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

  //------------------------------
```

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

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

```
  x=0;
     }
  double norm = Math.sqrt(x*x + y*y + z*z);

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

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

  }


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

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

```
 helpM[i][1] = helpS[i];
 helpM[i][2] = helpT[i];
}

return helpM;
  }

  //------------------------------


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

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

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

  //The 3d cube is projected into a 2d plane.
  //The resultant figure must be rendered to the screen.
  //A projection is done along  an axis that crosses
  //through 0.
  //Our projections are observed from infinite:
  //we use no perspective corrections.
  private  static  double[][] mProj(double w[] )
  {
  basisProj = basisB(w);
  double[][]  helpM = new double[3][3];
  helpM = matrixMult(J,matrixTranspose(basisProj));
  helpM = matrixMult(basisProj,helpM);
  //printMatrix(helpM, "Proj");
  //Test: Proj*Proj must be equal to Proj
  /*helpM = matrixMult(helpM,helpM);
   printMatrix(helpM, "Proj* Proj");*/
  return helpM;
  }

  //The cube c is rotated around axis v an angle theta
  private  static   double[][] rotCube(double c[][],
                        double v[], double theta)
  {
double[]  helpV = new double[3];
double cc[][] = new double[8][3];
matrixRot = mRotation( v,theta);
//printMatrix(matrixRot, "matrixRot");
//The 8 points of the cube must be rotated
for(int i=0; i<8; i++)
{
 for(int j=0; j<3; j++)
   helpV[j] = c[i][j];
```

```
 helpV = matrixVector(matrixRot, helpV);
 for(int j=0; j<3; j++)
 cc[i][j] = helpV[j];
}
//printCube(cc, " after rot");
return cc;
  }



//The cube c is translated   deltaV
  private static double[][] translateCube(double c[][],
                                     double deltaV[])
  {

   double cc[][] = new double[8][3];
   for(int i=0; i<8; i++)
    for(int j=0; j<3; j++)
   cc[i][j] = c[i][j] + deltaV[j];
   return cc;
  }


  //The cube c is projected along axis w
  private static double[][] projCube(double c[][],
                                  double w[])
  {
double[]  helpV = new double[3];
double cc[][] = new double[8][3];
matrixProj = mProj(w);
for(int i=0; i<8; i++)
{
 for(int j=0; j<3; j++)
   helpV[j] = c[i][j];
 helpV = matrixVector(matrixProj, helpV);
 for(int j=0; j<3; j++)
 cc[i][j] = helpV[j];
}
return cc;
```

```
  }

  //The rendering procedure accepts integer values
  private  static  int[][] round(double c[][] )
  {
  int[][] helpM = new int [8][3];
for(int i=0;i<8;i++)
 for(int j=0; j<3; j++)
helpM[i][j] = (int) Math.round(c[i][j]);
return helpM;
  }

  private static void printCube(double c[][], String a)
  {
   System.out.println("This is the cube" +a);
   for(int i=0;i<8;i++)
    System.out.println(c[i][0] + " " + c[i][1]+ " "
                                     + c[i][2]);
  }

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

  private void printShade()
  {
System.out.println("This is the shade");
    for(int i=0;i<8;i++)
     System.out.println(shade2d[i][1] + " "
                              + shade2d[i][2]);
  }

  //Colors to paint the faces are defined
  static private void defineColors()
  {
  for(int l = 0; l<6;l++)
```

```
{
 //We use random colors.
 //A color can be defined by three numbers,
     //each one from 0 to 255.
     //We has 6 faces, so we need
     //a matrix to keep colors.

  int c1,c2,c3;
  c1 = r.nextInt(226);
  c2 = r.nextInt(226);
  c3 = r.nextInt(226);
  //Colors are memorized
  colors[l][0]= c1;
  colors[l][1]= c2;
  colors[l][2]= c3;
}
  }

  //what faces must be colored
  private static  void marked()
  {
for(int f = 0; f<6; f++)
 coloredFace[f] = false;
     for(int f = 0; f<6;f++)
  {
for(int p = 0; p<4;p++)
 {
  for(int c = 0; c<8; c++)
   {
if (maxPoints[c] == true)
 {
if (faces[f][p] == c )
coloredFace[f] = true;
//System.out.println("coloring");
 }
   }
 }
  }
}
```

```
  //The x coordinate of all points are measured
  //and the extreme ones are recorded.
  private static  void calculateColor()
  {
//The maximum value of coordinate x
maxX =-1000;
for(int l = 0; l<8; l++)
{
  if (cube[l][0] > maxX)
   {
  maxX = cube[l][0];
  maxXPoint = l;
   }
}
//System.out.println("maxX, maxZ "+ maxX + " " + maxZ);
//System.out.println("maxXPoint, maxZPoint "+
//                         maxXPoint + " " + maxZPoint);
 nExtreme = 0;
//Number of extreme points
for(int l = 0; l<8;l++)
 {
  maxPoints[l] = false;
  if (Math.abs(cube[l][0] - maxX )< 2)
   {
    nExtreme = nExtreme +1 ;
    maxPoints[l] = true;
   }
 //faces to be colored
 }
marked();
  }

  //Parameters are set up.
  private static  void initialize()
  {
scale = 50;
defineColors();
for(int i=0; i<3;i++)
vTrans[i] = 0;
```

```
for(int i=0; i<8;i++)
 for(int j=0; j<3;j++)
  points3d[i][j] = scale*points3d[i][j];
//printCube();
//The 3d-graphic is rotated along the axisRot
//an angle theta that is incremented by deltaTheta
theta = 0;
deltaTheta = 0.1;
axisRot[0] = 1;
axisRot[1]=1;
axisRot[2]=1;
    //The 3d-graphic is projected over the screen
    // which is perpendicular to the x-axis.
axisProj[0] = 1;
axisProj[1]=0;
axisProj[2]=0;
//The 3d cube flies along vectTrans[]
deltaTrans[0] = 0;
deltaTrans[1] = -1;
deltaTrans[2] = 0;
nScenes = 500;
//delay between two scenes
delay =  100;
  }

  //This is the main method
public void paint (Graphics g)
{
 System.out.println("Solid flying rotating cube");
 initialize();


 //background color
 g.setColor(Color.pink);
 g.fillRect(0,0,400,400);
 //Loop for the scenes
 for(int k = 0; k<nScenes; k++)
 {
  theta = theta + deltaTheta;
```

```
  cube = rotCube(points3d,axisRot,theta);
  //printCube(cube, " after rotation");
  for(int i = 0; i<3;i++)
   vTrans[i] = vTrans[i] + deltaTrans[i];

  cube = translateCube(cube, vTrans);
  //printCube(cube, " after translation");
  cubeProj = projCube(cube,axisProj);
  //printCube(cubeProj, " after projection");
  shade2d =  round(cubeProj );
  //printShade();

  //draw cube
  calculateColor();
  for(int i=0;i<6;i++)
   {
if (coloredFace[i])
{
  g.setColor( new Color( colors[i][0], colors[i][1],
                              colors[i][2]));




 for(int l =0; l<4; l++)
  {
   pointsX[l] = 100-shade2d[ faces[i][l]  ] [1];
   pointsY[l] = 100-shade2d[ faces[i][l]  ] [2];
   //System.out.println(pointsX[l] + " " + pointsY[l]);
  }
     //A polygon is defined by two vectors and an integer
 //number:
 //The first vector lists the x-coordinates of the
 //corners;
 //the second lists  the y-coordinates. The integer is
 //the number of points.
 //If the last point is different than the first,
 //the first point is closed with the last.
 Polygon face = new Polygon(pointsX, pointsY, 4);
 g.fillPolygon(face);
```

```
}
   }
   //delay
   if (k < nScenes-1)
    {
     try
      {
   Thread.sleep( delay );
      }
      catch (InterruptedException e)
      {
   e.printStackTrace();
      }

     //erase cube: we repaint everything in
     //the background color.
      g.setColor(Color.pink);
      g.fillRect(0,0,200,400);
  }//end of if
 }//end of for nScenes
}//end of paint
}//end of class
```

17 The phenomenon of telepathic voices that some people listen at their heads are explained variously and each explanation generates its own therapeutic style. Let us consider the following materialist position in which the definition of matter is taken from modern science, which considers it to be composed of the elements listed in the chemical periodic table plus protons, electrons and electromagnetic radiation:

- A fundamental task of the brain is to serve as a simulation machine. So, we can dream when we sleep and we can have imaginations when we are awaken. Usually, one can stop these fantasies but many people report imaginations, images and/or voices, that seem to have will one their own. Such voices and visions can be explained by considering that imaginations can have a force of representation and that one can stop only those that come with weak force while one cannot stop those that are very strong: in some persons, some type of imaginations always come very strong and so they

seem to come from another being. Other persons might have troubles to distinguish own thoughts from dreaming and so a splitting of personality appears. In general, consciousness is a byproduct of the neural activity alone: the brain is composed of circuits that are organized in a hierarchy and a circuit at the top level can be called consciousness. This organization is dictated by genetics + local conditions. By many diverse causes, the expected uniqueness of top level circuits is not produced and the personality gets split. It can split in two or in three or in five personalities or eventually no dominant circuits exits and the patient follows in comma or something like that.

- The ensuing therapy is then to silence the emergent competing circuits. This can be done by electric shock, because competing circuits are more excitable than the rest and so more suitable to be harmed by electricity. Or the appropriate neuronal receptors can be blocked with given medicaments. In case of very strong disorders, ablation is recommended.

- Assessment: Treatment with electric shock must be repeated every two weeks. Medicaments with neuronal activity create addition with psychopathic effects. Ablation might deprive the person from his or her native personality.

Let us consider now the position of Christianity as a representative of spiritualism:

- The consciousness of a human being resides in the spirit, which is immortal (this does not preclude the possibility that a robot could have consciousness). Apart form human spirits, there are angels, spirits that love God and that obey his voice. There are also demons, spirits that hate God and that rejoice with the suffering of persons and that even can cause illness and pain and that can cause miracles through witches, priests of alien gods or possessed persons. The spirit of a person has a tight communication with his or her body and using the same mechanism other spirits can also interfere with the normal functioning of that communication: they produce visions and voices. An undesirable voice come from a demon or from the soul of a dead person and sometimes from a living witch.

- Therapy: to cast demons in the name of The Lord Jesus.

- Assessment: To the best of our knowledge, relief is minimal, comes very slowly and can be transitory. The impotency of the community of believers in this regard is well known from its very beginning, as it is told by Saint Mathew 17: 14-21. Nevertheless, group therapy seems to produce some simple but nonetheless important results after a long treatment, say, after 50

therapies once per week: a person that is close to getting mad and committing suicide is converted in a person that can battle on his or her own for having life to the full.

Problems of Chapter 3

27 The code for completely independent cubes follows:

```
/*Program D27 freeCubes

We present a third  OOP version of
a flying rotating solid cube:
the inner class Cube is
used as a prototype to build many cubes.
Each cube flies with its own
direction, velocity and size.
To that aim, the parameter m is used to
define the variables of each cube.
In our variations, edges must not have
the same measure.
*/

import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Polygon;
import java.util.Random;

public class freeCubes extends Applet
{

  private static final long serialVersionUID = 1L;


/*
Our basic cube is the unitary cube at the origin.
It is composed of 8 points or vertexes = oPoints3d[][],
12 edges = edges[][] and 6 faces = faces[][].
Points are numbered from 0 to 7.
```

```
These numbers are used in
edges, which  records the pairs of
points that are joined by a line.
A face is determined by its 4 corners.

All other cubes are transformations of this
fundamental cube. The form as this is done
appears in the class Cube.

The Cube as a prototype is defined as
an inner class in line 243.

*/

//The points of the fundamental cube
private static   double oPoints3d[][]=
{
 {0,0,0},
 {1,0,0},
 {0,1,0},
 {0,0,1},
 {1,1,0},
 {1,0,1},
 {0,1,1},
 {1,1,1}
};

/*
//The edges of the fundamental cube
private static int edges[][] =
   {{0,1},
    {0,2},
    {0,3},
    {1,4},
    {1,5},
    {2,6},
    {2,4},
    {3,5},
    {3,6},
    {4,7},
```

```
    {5,7},
    {6,7}};

*/

//The faces of the fundamental cube
private static int faces[][] =
   {{0,1,4,2},
    {0,2,6,3},
    {0,3,5,1},
    {3,5,7,6},
    {1,4,7,5},
    {2,6,7,4}};



 //Working instance of CubesOOP
 static  CubesOOP CubesOOP   = new CubesOOP();

 //Measures of the window
 private    int   xWin = 500;
 private    int   yWin = 400;
 private   int nScenes ;
 private   int delay ;

 //Up to 10 cubes can be built
 private   Cube[] vCube = new Cube[10];

//The window is enlarged.
//This method is fired automatically
//at the beginning
 public void init()
  {
   setSize(xWin,yWin);
  }

 //Each cube must have its initial position
 private   void rePosit(Cube cc, int m)
 {
  //System.out.println("rePosit(m)" + m);
```

```
 cc.oPosition[0] = 0;
 cc.oPosition[1] = 15 *m;
 cc.oPosition[2] = -60 *m;
 cc.cube = cc.locateCube(cc.cube, cc.oPosition);
}

//The cube is rotated, translated and projected
private    void modifyCube(Cube cc, int m)
{
 cc.theta = cc.theta + cc.deltaTheta;
 cc.cube = cc.rotCube(cc.points3d, cc.axisRot, cc.theta);
 //cc.printCube(cc.cube, " after rotation");
 rePosit(cc,m);
 //cc.printCube(cc.cube , " after relocation");
 for(int i = 0; i<3;i++)
 cc.vTrans[i] = cc.vTrans[i] + cc.deltaTrans[i];
 cc.cube = cc.translateCube(cc.cube,  cc.vTrans);
 //cc.printCube(cc.cube, " after translation");
 cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
 //cc.printCube(cc.cubeProj,  " after projection");
}


//Regulation of the velocity of movement
private    void delay()
{
try
    {
  Thread.sleep(delay );
    }
    catch (InterruptedException e)
    {
  e.printStackTrace();
    }
}

//face ii of the cube is drawn
private    void drawface(Graphics g, Cube cc, int ii)
{
for(int l =0; l<4; l++)
```

```
  {
cc.pointsX[l] = 100
             -cc.shade2d[faces[ii][l]  ] [1];
 cc.pointsY[l] = 100
            -cc.shade2d[ faces[ii][l]  ] [2];
   //System.out.println(pointsX[l] + " " + pointsY[l]);
  }
  //A polygon is defined by two vectors and an integer
   //number:
   //The first vector lists the x-coordinates of the
   //corners;
   //the second lists  the y-coordinates. The integer is
   //the number of points.
   //If the last point is different than the first,
   //the first point is closed with the last.
   Polygon face = new Polygon(cc.pointsX,
                      cc.pointsY, 4);
   g.fillPolygon(face);
 }

 //This is the main method
 public void paint (Graphics g)
 {
  //Number of scenes of the presentation.
  nScenes = 500;
  //Delay between two scenes
  delay =  120;
  //Number of cubes
  int nCubes = 5;
  System.out.println("Solid flying rotating OOP cubes");

  //background color
  g.setColor(Color.pink);
  g.fillRect(0,0,400,400);
  //Cubes are initialized
  for(int m =0; m < nCubes; m++)
   {
    final Cube cc =    new Cube(m);
    vCube[m] = cc;
   }
```

```
  //Loop for the scenes
  for(int k = 0; k<nScenes; k++)
  {
   //System.out.println("scene " + k);
   //nCubes are displayed
   for(int m = 0; m < nCubes; m++)
   {
//A working cube cc is initialized
Cube  cc = new Cube(m);
//The working cube is re-set.
    cc = vCube[m];
    //Movement beginds
    modifyCube(cc,m);
    //Projection is calculated
    cc.shade2d =  cc.round(cc.cubeProj );
    //printShade();
    //The cube is drawn
    cc.calculateColor();
    for(int i=0;i<6;i++)
     {
    if (cc.coloredFace[i])
     {
      g.setColor( new Color( cc.colors[i][0],
                    cc.colors[i][1],
                    cc.colors[i][2]));
      drawface(g,cc,i);
     }
      }
    vCube[m] = cc; //worked cube is kept to memory
    }
    //delay  + erase drawing
    if (k < nScenes-1)
     {
      delay();
      //erase cube: we repaint everything in
      //the background color.
       g.setColor(Color.pink);
       g.fillRect(0,0,xWin,yWin);
     }//end of if
  }//end of for nScenes
```

```
 }//end of paintOneCube(g,m)


 //=============Inner class Cube==============

 /*Variables and methods of the class cube are
   encapsulated in the same unit.
   Please, imagine a word populated with cubes, spheres,
   buildings, dogs and cats. Each class is defined
   following the pattern in here.
   All definitions are non static, i.e., they are
   used as erasable workbenches and resettable tools. */
 public   class Cube
 {
  private double points3d[][] = new double[8][3];
  private   int shade2d[][] = new int[8][3];

//  All cubes are derived from the basic cube by rotations
//  and translations
  private double cube[][]  = new double[8][3];
  private   double  cubeProj[][] = new double[8][3];

  private double oPosition[] = new double[3];
  private   double axisRot[] = new double[3];
  private   double axisProj[]  = new double[3];
  private   double deltaTrans[]  = new double[3];
  private   double vTrans[]  = new double[3];
  private   double theta  = 0;
  private   double deltaTheta;

  private   double basisRot[][], basisProj[][]
                = new double[3][3];
  private   double matrixRotz[][],
  matrixRot[][],
  matrixProj[][] = new double[3][3];
//  The matrix of a projection along the z-axis
  private     double J[][]=
        { {1,0,0},
          {0,1,0},
          {0,0,0}};
```

```
  private    double scale;

  int colors[][] = new int[6][3];
  private   int pointsX[]  = new int[6];
  private   int pointsY[]  = new int[6];
//  How to color faces
  private    double maxX;
  private   int nExtreme;
  private   boolean maxPoints[]  = new boolean[8];
  private   boolean coloredFace[]  = new boolean[6];


 //Every implementation of Cube is initialized here.
 //Parameter m is used for the initial position
 //of cube number m
 Cube(int m)
 {
  scale = 50;
  defineColors();
  //Original position
  for(int i=0; i<3;i++)
   oPosition[i] = 300;
  oPosition[2] = 100*m;
  for(int i=0; i<3;i++)
  vTrans[i] = 0;
  //Original  form
  for(int i=0; i<8;i++)
  for(int j=0; j<3;j++)
  points3d[i][j] = (m+1)* 0.1*scale*oPoints3d[i][j]*
                   (j+1);
  //  printCube();
  //  The 3d-graphic is rotated along the axisRot
  //  an angle theta that is incremented by deltaTheta.
  //  Each cube has its own velocity and direction of
  //  rotation.
  theta = 0;
  deltaTheta = 0.004 + 0.04*m* Math.pow(-1, m);
  axisRot[0] = 1;
  axisRot[1]=1;
```

```
  axisRot[2]=1;
  //  The 3d-graphic is projected over the screen
  //  which is perpendicular to the x-axis.
  axisProj[0] = 1;
  axisProj[1]=0;
  axisProj[2]=0;
  //  The 3d cube flies along vectTrans[]
  //Each cube has its own direction of translation
  deltaTrans[0] =  -1*(m-2);
  deltaTrans[1] =  -1*(m-2)*m;
  deltaTrans[2] =  -1*(m-2);

  }


 Random r  = new Random();

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

//The multiplication of two matrixes
private  double[][] matrixMult(double p[][], double q[][])
{
 double[][] helpM = new double[3][3];

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

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

```
 return helpM;
}

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



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

//The second vector is perpendicular to the first
private    double[] secondVector(double[] v)
{
 double[] helpV = new double[3];
```

```
 double   x,y,z;
 double a = v[0];
 double b = v[1];
 double c = v[2];
 x=b;
 y = -a;
 z=0;
 if (b==0)
  {
   x=c;
   z = -a;
   y=0;
  }
if (a==0)
 {
  y=c;
  z = -b;
  x=0;
 }
 double norm = Math.sqrt(x*x + y*y + z*z);

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

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

```
}


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

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


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

//A rotation is defined by an axis v that crosses
//through 0,
```

```
//and a counterclockwise angle theta of rotation
private double[][] mRotation(double v[], double theta)
{
 matrixRotz= mRot(theta);
 basisRot = basisB(v);
 //printMatrix(matrixRotz, "Rotz");
 //printMatrix(basisRot, "BasisRot");
 double[][]  helpM = new double[3][3];
 helpM = matrixMult(matrixRotz,
 matrixTranspose(basisRot));
 helpM = matrixMult(basisRot,helpM);

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

/*A 3d cube exists only in the imagination:
* it is not seen.
* One sees a shade or projection of it over the screen.
* The shade is defined by 2d points. The edges of
* the 2d shade are inherited from the edges of the
* 3d cube.
* The 3d cube is projected into a 2d plane.
* The resultant figure must be rendered to the screen.
* A projection is done along  an axis that crosses
* through 0.
* Our projections are observed from infinite:
* we use no perspective corrections. */
private     double[][] mProj(double w[] )
{
 basisProj = basisB(w);
 double[][]  helpM = new double[3][3];
 helpM = matrixMult(J,matrixTranspose(basisProj));
 helpM = matrixMult(basisProj,helpM);
 //printMatrix(helpM, "Proj");
 //Test: Proj*Proj must be equal to Proj
 /*helpM = matrixMult(helpM,helpM);
```

```
 printMatrix(helpM, "Proj* Proj");*/
 return helpM;
}

//The cube c is rotated around axis v an angle theta
private double[][] rotCube(double c[][],
                           double v[], double theta)
{
 double[]  helpV = new double[3];
 double cc[][] = new double[8][3];
 matrixRot = mRotation( v,theta);
 //printMatrix(matrixRot, "matrixRot");
 //The 8 points of the cube must be rotated
 for(int i=0; i<8; i++)
 {
  for(int j=0; j<3; j++)
  helpV[j] = c[i][j];
  helpV = matrixVector(matrixRot, helpV);
  for(int j=0; j<3; j++)
  cc[i][j] = helpV[j];
 }
 //printCube(cc, " after rot");
 return cc;
}


//The cube c is translated   deltaV
private double[][] locateCube(double c[][],
                                        double oPosition[])
{
 double cc[][] = new double[8][3];
 for(int i=0; i<8; i++)
 for(int j=0; j<3; j++)
 cc[i][j] = c[i][j] + oPosition[j];
 return cc;
}


//The cube c is translated   deltaV
private double[][] translateCube(double c[][],
```

```
                                        double deltaV[])
{
 double cc[][] = new double[8][3];
 for(int i=0; i<8; i++)
 for(int j=0; j<3; j++)
 cc[i][j] = c[i][j] + deltaV[j];
 return cc;
}


//The cube c is projected along axis w
private double[][] projCube(double c[][],  double w[])
{
 double[]  helpV = new double[3];
 double cc[][] = new double[8][3];
 matrixProj = mProj(w);
 for(int i=0; i<8; i++)
  {
   for(int j=0; j<3; j++)
   helpV[j] = c[i][j];
   helpV = matrixVector(matrixProj, helpV);
   for(int j=0; j<3; j++)
   cc[i][j] = helpV[j];
  }
 return cc;
}

//The rendering procedure accepts integer values
private     int[][] round(double c[][])
{
 int[][] helpM = new int [8][3];
 for(int i=0;i<8;i++)
 for(int j=0; j<3; j++)
 helpM[i][j] = (int) Math.round(c[i][j]);
 return helpM;
}

private   void printCube(double c[][], String a)
{
 System.out.println("This is the cube" +a);
```

```
 for(int i=0;i<8;i++)
 System.out.println(c[i][0] + " " + c[i][1]+ " "
                                        + c[i][2]);
}

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

private void printShade()
{
 System.out.println("This is the shade");
 for(int i=0;i<8;i++)
 System.out.println(shade2d[i][1] + " "
                                  + shade2d[i][2]);
}

//Colors to paint the faces are defined
private int[][] defineColors()
{
 for(int l = 0; l<6;l++)
  {
   //We use random colors.
   //A color can be defined by three numbers,
   //each one from 0 to 255.
   //We has 6 faces, so we need
   //a matrix to keep colors.

   int c1,c2,c3;
   c1 = r.nextInt(226);
   c2 = r.nextInt(226);
   c3 = r.nextInt(226);
   //Colors are memorized
   colors[l][0]= c1;
   colors[l][1]= c2;
   colors[l][2]= c3;
```

```
  }
 return colors;
}

//what faces must be colored
private    void marked()
{
 for(int f = 0; f<6; f++)
 coloredFace[f] = false;
 for(int f = 0; f<6;f++)
 {
  for(int p = 0; p<4;p++)
  {
   for(int c = 0; c<8; c++)
    {
     if (maxPoints[c] == true)
     {
       if (faces[f][p] == c )
       coloredFace[f] = true;
       //System.out.println("coloring");
     }
   }
  }
 }
}


//The x coordinate of all points are measured
//and the extreme ones are recorded.
private    void calculateColor()
{
 //The maximum value of coordinate x
 maxX =-1000;
 for(int l = 0; l<8; l++)
  {
   if (cube[l][0] > maxX)
    {
     maxX = cube[l][0];
    }
   }
```

```
 nExtreme = 0;
 //Number of extreme points
 for(int l = 0; l<8;l++)
  {
   maxPoints[l] = false;
   if (Math.abs(cube[l][0] - maxX )< 2)
   {
    nExtreme = nExtreme +1 ;
    maxPoints[l] = true;
   }
  }
  marked();
 }


 }//=============End of inner class Cube==============

}//end of class freeCubes
```

32 A finger that opens while rotates and then reverses its movement:

```
/*Program D32 fingerCloseOpen

This is an OOP version of a finger
that closes and then reverses the movement.

A finger is a   set of 3 modified cubes
that are posited in tandem and that move
coordinately.

Since we use finite precision, all our moves are
approximations. Therefore, a multiplication of errors
is expected if we use modifications in tandem.
That is why  every scene is built ab initio by
modification of constant entities at the origin.
The finger is located and move at the origin
and points downwards over the screen.
```

```
The inner class Finger is presented in line 248.
A finger is composed of 3 cubes. The class Cube
appears in line 359

The main program follows.

*/

import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Polygon;
import java.util.Random;

public class fingerCloseOpen extends Applet
{

  public static final long serialVersionUID = 1L;


 //This constructor is necessary when one have various
 //nested classes. In our case, we have a phalange,
 // which is used by the class Finger,
 // which is used by the main procedure in here.
 public fingerCloseOpen()
 {

 }


/*
Our basic unit is a cube is the unitary cube at the origin.
It is composed of 8 points or vertexes = oPoints3d[][],
12 edges = edges[][] and 6 faces = faces[][].
Points are numbered from 0 to 7.
These numbers are used in
edges, which  records the pairs of
points that are joined by a line.
A face is determined by its 4 corners.
```

```
All other cubes are transformations of this
fundamental cube. The form as this is done
appears in the class Cube.


*/

//The points of the fundamental cube
private static   double oPoints3d[][]=
{
 {0,0,0},
 {1,0,0},
 {0,1,0},
 {0,0,1},
 {1,1,0},
 {1,0,1},
 {0,1,1},
 {1,1,1}
};

/*
//The edges of the fundamental cube
private static int edges[][] =
   {{0,1},
    {0,2},
    {0,3},
    {1,4},
    {1,5},
    {2,6},
    {2,4},
    {3,5},
    {3,6},
    {4,7},
    {5,7},
    {6,7}};

*/

//The faces of the fundamental cube
private static int faces[][] =
```

```
  {{0,1,4,2},
   {0,2,6,3},
   {0,3,5,1},
   {3,5,7,6},
   {1,4,7,5},
   {2,6,7,4}};



 //Measures of the window
 private    int   xWin = 500;
 private    int   yWin = 400;
 private   int nScenes ;
 private   int delay ;
 //Number of cubes in the finger
 private  static  int nPhalanges = 3;

 //The constructor of fingerCloseOpen is invoked here.
 //A is now the instance of fingerCloseOpen
 private static fingerCloseOpen A = new fingerCloseOpen();
 //Our finger is f1: it is an instance of Finger
 private static Finger f1 =  A.new Finger(nPhalanges);

//The window is enlarged.
//This method is fired automatically
//at the beginning

 public void init()
 {
  setSize(xWin,yWin);
 }

 //Regulation of the velocity of movement
 private    void delay()
 {
 try
     {
   Thread.sleep(delay );
     }
```

```
     catch (InterruptedException e)
     {
   e.printStackTrace();
     }
 }

 //face ii of the cube is drawn
 private    void drawface(Graphics g, Cube cc, int ii)
 {
 for(int l =0; l<4; l++)
  {
cc.pointsX[l] = 100
             -cc.shade2d[faces[ii][l]  ] [1];
 cc.pointsY[l] = 100
            -cc.shade2d[ faces[ii][l]  ] [2];
   //System.out.println(pointsX[l] + " " + pointsY[l]);
  }
  //A polygon is defined by two vectors and an integer
   //number:
   //The first vector lists the x-coordinates of the
   //corners;
   //the second lists  the y-coordinates. The integer is
   //the number of points.
   //If the last point is different than the first,
   //the first point is closed with the last.
   Polygon face = new Polygon(cc.pointsX,
                      cc.pointsY, 4);
   g.fillPolygon(face);
 }

 private    void dynamics(Graphics g, Finger f1)
 {
 //Loop for the scenes
  for(int k = 0; k<nScenes; k++)
  {
   //System.out.println("scene " + k);
   //nCubes are displayed
   for(int m = 0; m < nPhalanges; m++)
   {
//A working cube cc is initialized
```

```
Cube  cc = new Cube(m);
//The working cube is re-set.
    cc = f1.phalanges[m];
    //Movement begins
    f1.modifyCube(cc,m);
    //Projection is calculated
    cc.shade2d =  cc.round(cc.cubeProj );
    //printShade();
    //The cube is drawn
    cc.calculateColor();
    for(int i=0;i<6;i++)
     {
    if (cc.coloredFace[i])
     {
      g.setColor( new Color( cc.colors[i][0],
                     cc.colors[i][1],
                     cc.colors[i][2]));
      drawface(g,cc,i);
     }
      }
        //worked cube is kept to memory
    f1.phalanges[m] = cc;
    }
    //delay  + erase drawing
    if (k < nScenes-1)
     {
      delay();
      //erase cube: we repaint everything in
      //the background color.
       g.setColor(Color.pink);
       g.fillRect(0,0,xWin,yWin);
     }//end of if
  }//end of for nScenes
 }

 //This is the main method
 public void paint (Graphics g)
 {
  //Number of scenes of the presentation.
  nScenes = 20;
```

```
 //Delay between two scenes
 delay =  1000;

 System.out.println("Bending finger");

 //background color
 g.setColor(Color.pink);
 g.fillRect(0,0,400,400);

 dynamics(g, f1);
 //To reverse the movement, we invert the angles
 //of rotation.
 for(int i=0; i<3;i++)
 {

  f1.phalanges[i].deltaPhi[i]  =  0.1*(i+1);
  f1.phalanges[i].deltaTheta =-f1.phalanges[i].deltaTheta;
 }
 delay();
 //erase cube: we repaint everything in
 //the background color.
  g.setColor(Color.pink);
  g.fillRect(0,0,xWin,yWin);
 dynamics(g, f1);
}//end of paintOneCube(g,m)



//=============Beginning of Inner class Finger============
/*Variables and methods of the class Finger are
encapsulated in the same unit.
Please, imagine a word populated with fingers, eyes,
hearts and so on. Each class is defined
following the pattern in here.
All definitions are non static, i.e., they are
used as erasable workbenches and resettable tools.

A finger is a coordinated set of 3 cubes.
The class cube is defined below.
```

```
 All methods that are of exclusive use for fingers are
 encapsulated here.
 */

 public class Finger
 {

 //Our finger is a composition of 3 cubes or phalanges
 private   Cube[] phalanges   = new Cube[3] ;


 //When memory is alloted for a finger,
 //all variables must be initialized
 Finger(int nPhalanges)
 {
 for(int m =0; m < nPhalanges; m++)
   {
    final Cube cc =    new Cube(m);
    phalanges[m] = cc;
   }
 }
// The only allowed movement of a phalange is
 //inclination, forth and back, in within bounds.
 //Inclination of given phalange governs the
 //rest of the finger
 private   Cube inclination(Cube cc, int m)
 {
  //Inclination is equal to a rotation
  //Axis of rotation: the y axis.
  cc.axisRot[0] = 0;
  cc.axisRot[1]=1;
  cc.axisRot[2]=0;
  switch(m)
   {
case 0: cc.phi[0]  = cc.phi[0]  + cc.deltaPhi[0] ;
           break;
case 1: cc.phi[1]  = cc.phi[0] + cc.phi[1]
                                 + cc.deltaPhi[1] ;
           break;
case 2: cc.phi[2]  = cc.phi[1] + cc.phi[2]
```

```
                                    + cc.deltaPhi[2] ;
           break;
  }
cc.cube = cc.rotCube(cc.points3d, cc.axisRot,
                              cc.phi[m] );
  return cc;
 }

    //For visualization, the finger is rotated
//around the z axis.
private   Cube rotation(Cube cc, int m)
{
 //Rotation about the z axis.
 cc.axisRot[0] = 0;
 cc.axisRot[1]=0;
 cc.axisRot[2]=1;
     cc.theta = cc.theta + cc.deltaTheta;
 cc.cube = cc.rotCube(cc.cube, cc.axisRot, cc.theta);
 return cc;
}

 //Each phalange must have its initial position
 //that depends on the position of the phalanges below it
 private   void rePosit(Cube cc, int m)
 {
  //System.out.println("rePosit(m)" + m);
 cc.oPosition[0]  = 0;
   cc.oPosition[1]  = 0;
   cc.oPosition[2]  =0;
 switch(m)
  {
   case 0:   break;
   case 1: {
cc.oPosition[0]  =  f1.phalanges[0].cube[3][0];
cc.oPosition[1]  =  f1.phalanges[0].cube[3][1];
    cc.oPosition[2]  =  f1.phalanges[0].cube[3][2];
   break;
       }
   case 2: {
cc.oPosition[0]  =  f1.phalanges[1].cube[3][0];
```

```
cc.oPosition[1]  =  f1.phalanges[1].cube[3][1];
cc.oPosition[2]  =  f1.phalanges[1].cube[3][2];
   break;
       }
  }

  cc.cube = cc.locateCube(cc.cube, cc.oPosition);

 }

 //The cube is inclined, rotated, translated, projected
 private    void modifyCube(Cube cc, int m)
 {
  cc = inclination(cc,m);
  cc = rotation(cc,m);

  //cc.printCube(cc.cube, " after rotation");
  rePosit(cc,m);
  //cc.printCube(cc.cube , " after relocation");
  for(int i = 0; i<3;i++)
  cc.vTrans[i] = cc.vTrans[i] + cc.deltaTrans[i];
  cc.cube = cc.translateCube(cc.cube,  cc.vTrans);
  //cc.printCube(cc.cube, " after translation");
  cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
  //cc.printCube(cc.cubeProj,  " after projection");
 }

 }
 //=============End of Inner class Finger============

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

 //=============Beginning of Inner class Cube============


 public   class Cube
 {
  private double points3d[][] = new double[8][3];
  private   int shade2d[][] = new int[8][3];
```

```
//  All cubes are derived from the basic cube by rotations
//  and translations
  private double cube[][]  = new double[8][3];
  private   double  cubeProj[][] = new double[8][3];

  private double oPosition[] = new double[3];
  private   double axisRot[] = new double[3];
  private   double axisProj[]  = new double[3];
  private   double deltaTrans[]  = new double[3];
  private   double vTrans[]  = new double[3];
  private   double theta  = 0;
  private   double deltaTheta;

  private   double basisRot[][], basisProj[][]
               = new double[3][3];
  private   double matrixRotz[][],
  matrixRot[][],
  matrixProj[][] = new double[3][3];
//  The matrix of a projection along the z-axis
  private     double J[][]=
        { {1,0,0},
          {0,1,0},
          {0,0,0}};

  private    double scale[] = new double[3];

  int colors[][] = new int[6][3];
  private   int pointsX[]  = new int[6];
  private   int pointsY[]  = new int[6];
//  How to color faces
  private    double maxX;
  private   int nExtreme;
  private   boolean maxPoints[]  = new boolean[8];
  private   boolean coloredFace[]  = new boolean[6];
  //Angle of inclination of phalange
  private   double phi[] = new double[3];;
  private   double deltaPhi[] = new double[3];;

 //Every implementation of Cube is initialized here.
 //Parameter m is used for the initial position
```

```
//of cube number m
Cube(int m)
{
 //Original  form
 scale[0] = 8;
 scale[1] = 8;
 scale[2] = 30;
 for(int i=0; i<8;i++)
 for(int j=0; j<3;j++)
 points3d[i][j] =  -scale[j]*oPoints3d[i][j];
 defineColors();
 //Original position
 for(int i=0; i<3;i++)
  oPosition[i] = 0;
 //Each phalange has its own angle of inclination
 //with respect to its support
 for(int i=0; i<3;i++)
 {
 phi[i] = 0;
 deltaPhi[i]  = -0.1*(m+1);
 }
 //  The 3d cube flies along vectTrans[]
 //Each cube has its own direction of translation
 for(int i=0; i<3;i++)
 vTrans[i] = 0;
 deltaTrans[0] =  0;
 deltaTrans[1] =  0;
 deltaTrans[2] =  0;
 //  printCube();
 //  The 3d-graphic is rotated along the axisRot
 //  an angle theta that is incremented by deltaTheta.
 //  Each cube has its own velocity and direction of
 //  rotation.
 theta = 0;
 deltaTheta = 0.1;
 axisRot[0] = 0;
 axisRot[1]=0;
 axisRot[2]=1;
 //  The 3d-graphic is projected over the screen
 //  which is perpendicular to the x-axis.
```

```
  axisProj[0] = 1;
  axisProj[1]=0;
  axisProj[2]=0;


  }


 Random r  = new Random();

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

//The multiplication of two matrixes
private  double[][] matrixMult(double p[][], double q[][])
{
 double[][] helpM = new double[3][3];

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

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

//The multiplication of a matrix times a vector
private double[] matrixVector(double m[][], double v[])
{
 double[]  helpV = new double[3];
```

```
 for(int i=0; i<3; i++)
 helpV[i] =0;
 for(int i=0; i<3; i++)
 for(int j=0; j<3; j++)
 helpV[i]  =helpV[i]  + m[i][j]*v[j];
 return helpV;
}



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

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

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

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

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


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

```
 double[]   helpT, helpS, helpF = new double[3];

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


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

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

```
 double[][]  helpM = new double[3][3];
 helpM = matrixMult(matrixRotz,
 matrixTranspose(basisRot));
 helpM = matrixMult(basisRot,helpM);

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

/*A 3d cube exists only in the imagination:
* it is not seen.
* One sees a shade or projection of it over the screen.
* The shade is defined by 2d points. The edges of
* the 2d shade are inherited from the edges of the
* 3d cube.
* The 3d cube is projected into a 2d plane.
* The resultant figure must be rendered to the screen.
* A projection is done along  an axis that crosses
* through 0.
* Our projections are observed from infinite:
* we use no perspective corrections. */
private     double[][] mProj(double w[] )
{
 basisProj = basisB(w);
 double[][]  helpM = new double[3][3];
 helpM = matrixMult(J,matrixTranspose(basisProj));
 helpM = matrixMult(basisProj,helpM);
 //printMatrix(helpM, "Proj");
 //Test: Proj*Proj must be equal to Proj
 /*helpM = matrixMult(helpM,helpM);
 printMatrix(helpM, "Proj* Proj");*/
 return helpM;
}

//The cube c is rotated around axis v an angle theta
private double[][] rotCube(double c[][],
                           double v[], double theta)
```

```
{
 double[]  helpV = new double[3];
 double cc[][] = new double[8][3];
 matrixRot = mRotation( v,theta);
 //printMatrix(matrixRot, "matrixRot");
 //The 8 points of the cube must be rotated
 for(int i=0; i<8; i++)
 {
  for(int j=0; j<3; j++)
  helpV[j] = c[i][j];
  helpV = matrixVector(matrixRot, helpV);
  for(int j=0; j<3; j++)
  cc[i][j] = helpV[j];
 }
 //printCube(cc, " after rot");
 return cc;
}


//The cube c is translated   deltaV
private double[][] locateCube(double c[][],
                                        double oPosition[])
{
 double cc[][] = new double[8][3];
 for(int i=0; i<8; i++)
 for(int j=0; j<3; j++)
 cc[i][j] = c[i][j] + oPosition[j];
 return cc;
}


//The cube c is translated   deltaV
private double[][] translateCube(double c[][],
                                        double deltaV[])
{
 double cc[][] = new double[8][3];
 for(int i=0; i<8; i++)
 for(int j=0; j<3; j++)
 cc[i][j] = c[i][j] + deltaV[j];
 return cc;
```

```
}


//The cube c is projected along axis w
private double[][] projCube(double c[][],  double w[])
{
 double[]  helpV = new double[3];
 double cc[][] = new double[8][3];
 matrixProj = mProj(w);
 for(int i=0; i<8; i++)
  {
   for(int j=0; j<3; j++)
   helpV[j] = c[i][j];
   helpV = matrixVector(matrixProj, helpV);
   for(int j=0; j<3; j++)
   cc[i][j] = helpV[j];
  }
 return cc;
}

//The rendering procedure accepts integer values
private     int[][] round(double c[][])
{
 int[][] helpM = new int [8][3];
 for(int i=0;i<8;i++)
 for(int j=0; j<3; j++)
 helpM[i][j] = (int) Math.round(c[i][j]);
 return helpM;
}

private   void printCube(double c[][], String a)
{
 System.out.println("This is the cube" +a);
 for(int i=0;i<8;i++)
 System.out.println(c[i][0] + " " + c[i][1]+ " "
                                        + c[i][2]);
}

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

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

private void printShade()
{
 System.out.println("This is the shade");
 for(int i=0;i<8;i++)
 System.out.println(shade2d[i][1] + " "
                                + shade2d[i][2]);
}

//Colors to paint the faces are defined
private int[][] defineColors()
{
 for(int l = 0; l<6;l++)
  {
   //We use random colors.
   //A color can be defined by three numbers,
   //each one from 0 to 255.
   //We has 6 faces, so we need
   //a matrix to keep colors.

   int c1,c2,c3;
   c1 = r.nextInt(226);
   c2 = r.nextInt(226);
   c3 = r.nextInt(226);
   //Colors are memorized
   colors[l][0]= c1;
   colors[l][1]= c2;
   colors[l][2]= c3;
  }
 return colors;
}

//what faces must be colored
private    void marked()
{
```

```
 for(int f = 0; f<6; f++)
 coloredFace[f] = false;
 for(int f = 0; f<6;f++)
 {
  for(int p = 0; p<4;p++)
  {
   for(int c = 0; c<8; c++)
    {
     if (maxPoints[c] == true)
     {
       if (faces[f][p] == c )
       coloredFace[f] = true;
       //System.out.println("coloring");
     }
   }
  }
 }
}


//The x coordinate of all points are measured
//and the extreme ones are recorded.
private    void calculateColor()
{
 //The maximum value of coordinate x
 maxX =-1000;
 for(int l = 0; l<8; l++)
  {
   if (cube[l][0] > maxX)
    {
     maxX = cube[l][0];
    }
   }
 nExtreme = 0;
 //Number of extreme points
 for(int l = 0; l<8;l++)
  {
   maxPoints[l] = false;
   if (Math.abs(cube[l][0] - maxX )< 2)
   {
```

```
    nExtreme = nExtreme +1 ;
    maxPoints[l] = true;
   }
  }
  marked();
 }


 }//=============End of inner class Cube==============


}//end of class fingerCloseOpen
```

36 The code for a hand with a proto-thumb:

```
/*Program D36 protoThumb

Our protohand has 5  fingers, a palm and an arm.
Fingers have 3  phalanges except the thumb, which has 2.

Since we use finite precision, all our moves are
approximations. Therefore, a multiplication of errors
is expected if we use modifications in tandem.
That is why  every scene is built ab initio by
modification of a unique unmodifiable cube at the origin.
The hand is located and move at the origin
and points downwards over the screen.

The inner class Hand is encoded beginning from line 379
The inner class Finger is presented in line 444.
A finger is composed of 3 cubes. The class Cube
appears in line 555

The main program follows.

*/

import java.applet.Applet;
```

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


public class protoThumb extends Applet
{

 public static final long serialVersionUID = 1L;

 /*This constructor is necessary when one have various
  nested classes. In our case, we have the class Cube,
   which is used by the class Finger,
   which is used by the class Hand,
   which is used by the main class (protoThumb) in here.*/
 public protoThumb()
 {

 }


/*
Our basic unit is  the unitary cube at the origin.
It is composed of 8 points or vertexes = oPoints3d[][],
12 edges = edges[][] and 6 faces = faces[][].
Points are numbered from 0 to 7.
These numbers are used in
edges, which  records the pairs of
points that are joined by a line.
A face is determined by its 4 corners.

All other cubes are transformations of this
fundamental cube. The form as this is done
appears in the class Cube.


*/

//The points of the fundamental cube
```

```
private static   double oPoints3d[][]=
{
 {0,0,0},
 {1,0,0},
 {0,1,0},
 {0,0,1},
 {1,1,0},
 {1,0,1},
 {0,1,1},
 {1,1,1}
};

/*
//The edges of the fundamental cube
private static int edges[][] =
   {{0,1},
    {0,2},
    {0,3},
    {1,4},
    {1,5},
    {2,6},
    {2,4},
    {3,5},
    {3,6},
    {4,7},
    {5,7},
    {6,7}};

*/

//The faces of the fundamental cube
private static int faces[][] =
   {{0,1,4,2},
    {0,2,6,3},
    {0,3,5,1},
    {3,5,7,6},
    {1,4,7,5},
    {2,6,7,4}};
```

```
 //Measures of the window
 private    int   xWin = 600;
 private    int   yWin = 500;
 private   int nScenes ;
 private   int delay ;
 //Number of cubes in the finger
 private  static  int nPhalanges = 3;

 //The constructor of hand is invoked here.
 //A is now the instance of the main class hand
 private static protoThumb A = new protoThumb();
 //The inner class Hand is instantiated
 private static Hand  h = A.new Hand();
 //An auxiliary  instance of Finger
 private static Finger f1=  A.new Finger(nPhalanges);



//The window is enlarged.
//This method is fired automatically
//at the beginning
 public void init()
 {
  setSize(xWin,yWin);
 }

 //Regulation of the velocity of movement
 private    void delay(int delay)
 {
 try
     {
   Thread.sleep(delay );
     }
     catch (InterruptedException e)
     {
   e.printStackTrace();
     }
 }
```

```
 //face ii of the cube is drawn
 private    void drawface(Graphics g, Cube cc, int ii)
 {
 for(int l =0; l<4; l++)
  {
   cc.pointsX[l] = 200-cc.shade2d[faces[ii][l]][1];
   cc.pointsY[l] = 200-cc.shade2d[ faces[ii][l]][2];
   //System.out.println(pointsX[l] + " " + pointsY[l]);
  }
  //A polygon is defined by two vectors and an integer
   //number:
   //The first vector lists the x-coordinates of the
   //corners;
   //the second lists  the y-coordinates. The integer is
   //the number of points.
   //If the last point is different than the first,
   //the first point is closed with the last.
   Polygon face = new Polygon(cc.pointsX,
                      cc.pointsY, 4);
   g.fillPolygon(face);
 }

 private    void drawArm(Graphics g)
 {
// A working cube cc is initialized
Cube  cc = new Cube(1);
//The working cube is re-set.
   cc = h.arm ;
   //Movement begins
   f1.modifyCube(cc,0);
   cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
   //Projection is calculated
   cc.shade2d =  cc.round(cc.cubeProj );
   //printShade();
   //The cube is drawn
   cc.calculateColor();
   for(int i=0;i<6;i++)
    {
  if (cc.coloredFace[i])
```

```
   {
    g.setColor( new Color( cc.colors[i][0],
                   cc.colors[i][1],
                   cc.colors[i][2]));
    drawface(g,cc,i);
   }
     }
   //worked cube is kept to memory
   h.arm = cc;
 }

 private Cube repositPalm(Cube cc)
 {
  cc.oPosition[0]  =  h.arm.cube[3][0];
  cc.oPosition[1]  =  h.arm.cube[3][1]+5;
  cc.oPosition[2]  =   h.arm.cube[3][2];
  cc.cube = cc.translateCube(cc.cube, cc.oPosition);
  return cc;
}

 private    Cube updatePalm(Cube cc)
 {
 cc = f1.inclination(cc,1);
  cc = f1.rotation(cc,1);

  //cc.printCube(cc.cube, " after rotation");
   cc = repositPalm(cc);
  //cc.printCube(cc.cube , " after relocation");
  for(int i = 0; i<3;i++)
  cc.vTrans[i] = cc.vTrans[i] + cc.deltaTrans[i];
  cc.cube = cc.translateCube(cc.cube,  cc.vTrans);
  //cc.printCube(cc.cube, " after translation");
  cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
  //cc.printCube(cc.cubeProj,  " after projection");
 return cc;
 }

 private    void drawPalm(Graphics g)
 {
// A working cube cc is initialized
```

```
Cube  cc = new Cube(1);
//The working cube is re-set.
   cc = h.palm ;
   //Movement begins
   cc = updatePalm(cc);
   cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
   //Projection is calculated
   cc.shade2d =  cc.round(cc.cubeProj );
   //printShade();
   //The cube is drawn
   cc.calculateColor();
   for(int i=0;i<6;i++)
    {
  if (cc.coloredFace[i])
   {
    g.setColor( new Color( cc.colors[i][0],
                  cc.colors[i][1],
                  cc.colors[i][2]));
    drawface(g,cc,i);
   }
     }
   //worked cube is kept to memory
   h.palm = cc;
 }

 //Fingers are posited below the palm
 private    void locateFingers()
  {
   for(int i=0; i<5;i++)
for(int j=0; j<3;j++)
 h.finger[i].phalange[0].oPosition[j]
                                  =  h.palm.cube[3][j];
   for(int i=0; i<5;i++)
h.finger[i].phalange[0].oPosition[1]
        = h.finger[i].phalange[0].oPosition[1] -10*i ;

  }


 private    void drawFingers(Graphics g)
```

```
 {
  locateFingers();
  for(int j = 4; j>=0;j--)
  {
  if(j==4) nPhalanges = 2;
  else nPhalanges = 3;
   for(int m = 0; m < nPhalanges; m++)
   {

//A working cube cc is initialized
Cube  cc = new Cube(m);
//The working cube is re-set.
    cc = h.finger[j].phalange[m];
    //Movement begins
    cc = h.finger[j].modifyCube(cc,m);
    //Projection is calculated
    cc.shade2d =  cc.round(cc.cubeProj );
    //printShade();
    //The cube is drawn
    cc.calculateColor();
    for(int i=0;i<6;i++)
     {
  if (cc.coloredFace[i])
   {
    g.setColor( new Color( cc.colors[i][0],
                  cc.colors[i][1],
                  cc.colors[i][2]));
    drawface(g,cc,i);
   }
     }
    //worked cube is kept to memory
    h.finger[j].phalange[m] = cc;
   }
  }
 }

 private    void drawHand(Graphics g)
 {
drawArm(g);
 drawPalm(g);
```

```
 drawFingers(g);
 }


 private    void dynamics(Graphics g)
 {
 //Loop for the scenes
  for(int k = 0; k<nScenes; k++)
  {
   //System.out.println("scene " + k);

drawHand(g);
    //delay  + erase drawing
    if (k < nScenes-1)
     {
      delay(delay);
      //erase cube: we repaint everything in
      //the background color.
       g.setColor(Color.pink);
       g.fillRect(0,0,xWin,yWin);
     }//end of if
  }//end of for nScenes
 }

 //This is the main method
 public void paint (Graphics g)
 {
  //Number of scenes of the presentation.
  nScenes = 20;
  //Delay between two scenes
  delay =  800;

  System.out.println("A proto-hand with thumb");
  //An instance of Hand is initialized


  //background color
  g.setColor(Color.pink);
  g.fillRect(0,0,xWin,yWin);
  dynamics(g);
```

```
  //To reverse the movement, we invert the angles
  //of rotation
  for(int j=0; j<3;j++)
   {
h.arm.deltaPhi[j] = -h.arm.deltaPhi[j];
h.palm.deltaPhi[j] = -h.palm.deltaPhi[j];
   }

  h.arm.deltaTheta = -h.arm.deltaTheta;
  h.palm.deltaTheta = -h.palm.deltaTheta;
  for(int i=0; i<5;i++)
  {
   for(int j=0; j<3;j++)
{
     h.finger[i].phalange[j].deltaPhi[j]  =
            -h.finger[i].phalange[j].deltaPhi[j];
     h.finger[i].phalange[j].deltaTheta =
             -h.finger[i].phalange[j].deltaTheta;
}
  }
  delay(delay);
  //erase cube: we repaint everything in
  //the background color.
   g.setColor(Color.pink);
   g.fillRect(0,0,xWin,yWin);
  dynamics(g);

 }//end of paint



 //=============Beginning of Inner class Hand============

 public class Hand
 {
 private Cube arm = new Cube(1);
 private Cube palm = new Cube(1);
 private Finger finger[] = new Finger[5];
```

```
 //When memory is alloted for a hand,
 //all variables must be initialized.
 //Lower classes are initialized by default,
 //but some  redefinitions could be necessary.
 Hand()
 {
  //The arm is initialized
   final Cube c1 =    new Cube(1);
   arm = c1;
       //Dimensions and location of the arm.
   arm.scale[0] = 8;
   arm.scale[1] = 30;
   arm.scale[2] = 100;
   for(int i=0; i<8;i++)
   for(int j=0; j<3;j++)
       arm.points3d[i][j] =  -arm.scale[j]*oPoints3d[i][j];


   //The  palm is initialized
   final Cube c2 =    new Cube(1);
   palm = c2;
   //Dimensions and location of the palm.
   palm.scale[0] = 8;
   palm.scale[1] = 40;
   palm.scale[2] = 60;
   for(int i=0; i<8;i++)
for(int j=0; j<3;j++)
  palm.points3d[i][j] =  -palm.scale[j]*oPoints3d[i][j];


   //Fingers are initialized
   for(int i=0; i<5;i++)
   {
   final  Finger f =   new Finger(nPhalanges);
   finger[i] = f;
   }
```

```
 }
 }
 //=============End of Inner class Hand============

 //=============Beginning of Inner class Finger===========
 /*Variables and methods of the class Finger are
 encapsulated in the same unit.
 All definitions are non static, i.e., they are
 used as erasable workbenches and resettable tools.

 A finger is a coordinated set of 3 cubes.
 The class cube is defined below.

 All methods that are of exclusive use for fingers are
 encapsulated here.
 */
 public class Finger
 {

 //Our finger is a composition of 3 cubes or phalanges
 private   Cube[] phalange    = new Cube[3] ;


 //When memory is alloted for a finger,
 //all variables must be initialized
 Finger(int nPhalanges)
 {
 for(int m =0; m < nPhalanges; m++)
   {
    final Cube cc =    new Cube(m);
    phalange[m] = cc;
   }
 }
// The only allowed movement of a phalange is
 //inclination, forth and back, in within bounds.
 //Inclination of given phalange governs the
 //rest of the finger
 private   Cube inclination(Cube cc, int m)
```

```
 {
  //Inclination is equal to a rotation
  //Axis of rotation: the y axis.
  cc.axisRot[0] = 0;
  cc.axisRot[1]=1;
  cc.axisRot[2]=0;
  switch(m)
   {
case 0: cc.phi[0]  = cc.phi[0]  + cc.deltaPhi[0] ;
          break;
case 1: cc.phi[1]  = cc.phi[0] + cc.phi[1]
                               + cc.deltaPhi[1] ;
          break;
case 2: cc.phi[2]  = cc.phi[1] + cc.phi[2]
                               + cc.deltaPhi[2] ;
          break;
  }
cc.cube = cc.rotCube(cc.points3d, cc.axisRot,
                            cc.phi[m] );
  return cc;
 }

    //For visualization, the finger is rotated
//around the z axis.
private   Cube rotation(Cube cc, int m)
{
 //Rotation about the z axis.
 cc.axisRot[0] = 0;
 cc.axisRot[1]=0;
 cc.axisRot[2]=1;
     cc.theta = cc.theta + cc.deltaTheta;
 cc.cube = cc.rotCube(cc.cube, cc.axisRot, cc.theta);
 return cc;
}

 //Each phalange must have its initial position
 //that depends on the position of the phalanges below it
 private   Cube rePosit(Cube cc, int m)
 {
   //System.out.println("rePosit(m)" + m);
```

```
 switch(m)
  {
   case 0:  {
      } break;
   case 1: {
cc.oPosition[0]  =   phalange[0].cube[3][0];
cc.oPosition[1]  =   phalange[0].cube[3][1];
    cc.oPosition[2]  =   phalange[0].cube[3][2];
   break;
       }
   case 2: {
cc.oPosition[0]  =   phalange[1].cube[3][0];
cc.oPosition[1]  =   phalange[1].cube[3][1];
cc.oPosition[2]  =   phalange[1].cube[3][2];
   break;
       }
  }
  cc.cube = cc.translateCube(cc.cube, cc.oPosition);
  return cc;
 }

 //The cube is inclined, rotated, translated, projected
 private    Cube modifyCube(Cube cc, int m)
 {
  cc = inclination(cc,m);
  cc = rotation(cc,m);

  //cc.printCube(cc.cube, " after rotation");
  rePosit(cc,m);
  //cc.printCube(cc.cube , " after relocation");
  for(int i = 0; i<3;i++)
  cc.vTrans[i] = cc.vTrans[i] + cc.deltaTrans[i];
  cc.cube = cc.translateCube(cc.cube,  cc.vTrans);
  //cc.printCube(cc.cube, " after translation");
  cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
  //cc.printCube(cc.cubeProj,  " after projection");
  return cc;
 }

 }
```

```
 //=============End of Inner class Finger============

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

 //=============Beginning of Inner class Cube============



 public   class Cube
 {
  private double points3d[][] = new double[8][3];
  private   int shade2d[][] = new int[8][3];

//  All cubes are derived from the basic cube by rotations
//  and translations
  private double cube[][]  = new double[8][3];
  private   double  cubeProj[][] = new double[8][3];

  private double oPosition[] = new double[3];
  private   double axisRot[] = new double[3];
  private   double axisProj[]  = new double[3];
  private   double deltaTrans[]  = new double[3];
  private   double vTrans[]  = new double[3];
  private   double theta  = 0;
  private   double deltaTheta;

  private   double basisRot[][], basisProj[][]
                = new double[3][3];
  private   double matrixRotz[][],
  matrixRot[][],
  matrixProj[][] = new double[3][3];
//  The matrix of a projection along the z-axis
  private     double J[][]=
        { {1,0,0},
          {0,1,0},
          {0,0,0}};

  private    double scale[] = new double[3];
```

```
  int colors[][] = new int[6][3];
  private   int pointsX[]  = new int[6];
  private   int pointsY[]  = new int[6];
//  How to color faces
  private    double maxX;
  private   int nExtreme;
  private   boolean maxPoints[]  = new boolean[8];
  private   boolean coloredFace[]  = new boolean[6];
  //Angle of inclination of phalange
  private   double phi[] = new double[3];;
  private   double deltaPhi[] = new double[3];;

 //Every implementation of Cube is initialized here.
 //Parameter m is used for the initial position
 //of cube number m
 Cube(int m)
 {
  //Original  form
  scale[0] = 8;
  scale[1] = 8;
  scale[2] = 30;
  for(int i=0; i<8;i++)
  for(int j=0; j<3;j++)
  points3d[i][j] =  -scale[j]*oPoints3d[i][j];
  defineColors();
  //Original position
  for(int i=0; i<3;i++)
   oPosition[i] = 0;
  //Each phalange has its own angle of inclination
  //with respect to its support
  for(int i=0; i<3;i++)
  {
  phi[i] = 0;
  deltaPhi[i]  = -0.1*(m+1);
  }
  //  The 3d cube flies along vectTrans[]
  //Each cube has its own direction of translation
  for(int i=0; i<3;i++)
  vTrans[i] = 0;
  deltaTrans[0] =  0;
```

```
  deltaTrans[1] =  0;
  deltaTrans[2] =  0;
  //  printCube();
  //  The 3d-graphic is rotated along the axisRot
  //  an angle theta that is incremented by deltaTheta.
  //  Each cube has its own velocity and direction of
  //  rotation.
  theta = 0;
  deltaTheta = -0.1;
  axisRot[0] = 0;
  axisRot[1]=0;
  axisRot[2]=1;
  //  The 3d-graphic is projected over the screen
  //  which is perpendicular to the x-axis.
  axisProj[0] = 1;
  axisProj[1]=0;
  axisProj[2]=0;


  }


 Random r  = new Random();

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

//The multiplication of two matrixes
private  double[][] matrixMult(double p[][], double q[][])
{
 double[][] helpM = new double[3][3];

 for(int i=0; i<3; i++)
```

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

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

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



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

```
 return helpV;
}

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

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

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

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


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

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


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

```
 mRotz[2][1] = 0;
 mRotz[2][2] = 1;
 return mRotz;
}

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

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

/*A 3d cube exists only in the imagination:
* it is not seen.
* One sees a shade or projection of it over the screen.
* The shade is defined by 2d points. The edges of
* the 2d shade are inherited from the edges of the
* 3d cube.
* The 3d cube is projected into a 2d plane.
* The resultant figure must be rendered to the screen.
* A projection is done along  an axis that crosses
* through 0.
* Our projections are observed from infinite:
* we use no perspective corrections. */
private     double[][] mProj(double w[] )
{
```

```
 basisProj = basisB(w);
 double[][]  helpM = new double[3][3];
 helpM = matrixMult(J,matrixTranspose(basisProj));
 helpM = matrixMult(basisProj,helpM);
 //printMatrix(helpM, "Proj");
 //Test: Proj*Proj must be equal to Proj
 /*helpM = matrixMult(helpM,helpM);
 printMatrix(helpM, "Proj* Proj");*/
 return helpM;
}

//The cube c is rotated around axis v an angle theta
private double[][] rotCube(double c[][],
                           double v[], double theta)
{
 double[]  helpV = new double[3];
 double cc[][] = new double[8][3];
 matrixRot = mRotation( v,theta);
 //printMatrix(matrixRot, "matrixRot");
 //The 8 points of the cube must be rotated
 for(int i=0; i<8; i++)
 {
  for(int j=0; j<3; j++)
  helpV[j] = c[i][j];
  helpV = matrixVector(matrixRot, helpV);
  for(int j=0; j<3; j++)
  cc[i][j] = helpV[j];
 }
 //printCube(cc, " after rot");
 return cc;
}



//The cube c is translated   deltaV
private double[][] translateCube(double c[][],
                                         double deltaV[])
{
 double cc[][] = new double[8][3];
 for(int i=0; i<8; i++)
```

```
 for(int j=0; j<3; j++)
 cc[i][j] = c[i][j] + deltaV[j];
 return cc;
}


//The cube c is projected along axis w
private double[][] projCube(double c[][],  double w[])
{
 double[]  helpV = new double[3];
 double cc[][] = new double[8][3];
 matrixProj = mProj(w);
 for(int i=0; i<8; i++)
  {
   for(int j=0; j<3; j++)
   helpV[j] = c[i][j];
   helpV = matrixVector(matrixProj, helpV);
   for(int j=0; j<3; j++)
   cc[i][j] = helpV[j];
  }
 return cc;
}

//The rendering procedure accepts integer values
private     int[][] round(double c[][])
{
 int[][] helpM = new int [8][3];
 for(int i=0;i<8;i++)
 for(int j=0; j<3; j++)
 helpM[i][j] = (int) Math.round(c[i][j]);
 return helpM;
}

private   void printCube(double c[][], String a)
{
 System.out.println("This is the cube" +a);
 for(int i=0;i<8;i++)
 System.out.println(c[i][0] + " " + c[i][1]+ " "
                                        + c[i][2]);
}
```

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

private void printShade()
{
 System.out.println("This is the shade");
 for(int i=0;i<8;i++)
 System.out.println(shade2d[i][1] + " "
                                  + shade2d[i][2]);
}

//Colors to paint the faces are defined
private int[][] defineColors()
{
 for(int l = 0; l<6;l++)
  {
   //We use random colors.
   //A color can be defined by three numbers,
   //each one from 0 to 255.
   //We has 6 faces, so we need
   //a matrix to keep colors.

   int c1,c2,c3;
   c1 = r.nextInt(226);
   c2 = r.nextInt(226);
   c3 = r.nextInt(226);
   //Colors are memorized
   colors[l][0]= c1;
   colors[l][1]= c2;
   colors[l][2]= c3;
  }
 return colors;
}
```

```
//what faces must be colored
private    void marked()
{
 for(int f = 0; f<6; f++)
 coloredFace[f] = false;
 for(int f = 0; f<6;f++)
 {
  for(int p = 0; p<4;p++)
  {
   for(int c = 0; c<8; c++)
    {
     if (maxPoints[c] == true)
     {
       if (faces[f][p] == c )
       coloredFace[f] = true;
       //System.out.println("coloring");
     }
   }
  }
 }
}


//The x coordinate of all points are measured
//and the extreme ones are recorded.
private    void calculateColor()
{
 //The maximum value of coordinate x
 maxX =-1000;
 for(int l = 0; l<8; l++)
  {
   if (cube[l][0] > maxX)
    {
     maxX = cube[l][0];
    }
   }
 nExtreme = 0;
 //Number of extreme points
 for(int l = 0; l<8;l++)
  {
```

```
   maxPoints[l] = false;
   if (Math.abs(cube[l][0] - maxX )< 2)
   {
    nExtreme = nExtreme +1;
    maxPoints[l] = true;
   }
  }
  marked();
 }
 }//=============End of inner class Cube==============


}//end of main class
```

Problems of Chapter 5

53 The white muscles of the belly do not come by pairs. Nevertheless, these muscles somehow work in coordination with the diaphragm and intercostal muscles to simulate do-undo movements.

Problems of Chapter 6

103 To guess a goal with n letters by mere randomness, one must guess each letter among an alphabet with 26 entries. So, the probability is $1/26$. Because all letters are guessed independently, the total probability is $p = (1/26)^n$, henceforth, the expected number of trials to make a guessing is $1/p = 26^n$. The number of trials in our simulation seems to fit such an exponential dependence in the number of letters of the input.

Problems of Chapter 9

146 The next version improves a bit an anterior version that presented an opposable thumb:

```
/*Program D146 OppThumbImproved
 *
 * Our hands are plain and all fingers point in
 * the same direction but the thumb can  be turned
```

```
 * so that one can tightly hold things with it.
 * Feet of birds are also plain and have an opposable
 * thumb, which points contrariwise to the other fingers.
 * In humans, the hand follows in the same direction
 * of the arm while the foot of birds is perpendicular
 * to the leg.
 *
 * The hand presented in here is somehow in the middle
 * of a hand of a human and a feet of a bird. We imagine
 * that it is by far simple than that of a human and
 * that it is very good to grasp things, so we expect
 * that it must appear somewhere in the fossil record
 * that shows the evolutionary history of our lineage.
 *
 * Half cooked products, achieved in local maxims,
 * are in complex projects mandatory long living
 * countless intermediate states.
 *
 * This version improves a bit an anterior one.
 */

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

import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.MouseListener;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.JPopupMenu;
import javax.swing.ButtonGroup;

import java.awt.Polygon;
import java.util.Random;
```

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

 //The application is instantiated
 static OppThumbImproved p = new OppThumbImproved();
 //A class with important data is instantiated
  private static Data d = p.new Data();
 //The painting class is declared and instantiated
 private static Canvas c = p.new Canvas();
 //Popup menu is  declared and instantiated
 static pppMenu pp = p.new pppMenu();

 public static void main( String args[] )
 {
  //The JFrame is constructed with a title
  JFrame f = new JFrame( "Opposable Thumb improved" );

  //The popUp Menu is added to the frame
  f.add( pp, BorderLayout.CENTER );
  f.add( new JLabel("Punch a finger with the mouse"),
                           BorderLayout.SOUTH );
  f.setDefaultCloseOperation( JFrame.DISPOSE_ON_CLOSE );
  c.setBackground( Color.pink );
  f.setSize( 600, 700 );
  f.setVisible( true );
  //PopUp menu is added to f
  f.add( c, BorderLayout.CENTER );
 }

 //Data that serve communication
 //among diferent classes is reunited into an object.
 private class Data
 {
  private static final long serialVersionUID = 1L;
  //Flag for clicked button
  private  boolean clickedLeft;
  private  boolean clickedRight;
```

```
 private int optionColor ;
 private int optionMove ;
 private double ampliFactor;
 private int ex;
 private int ey;
 private Color color;
 private  boolean first;


public Data()
{
 //Which button of the mouse has been clicked
 clickedLeft = false;
 clickedRight = false;
 //Option of the popUp menu
 optionColor = 0;
 //Option for movement: up vs down
 optionMove =0;
 //Graphic enlarging factor
 ampliFactor = 3;
 //Coordinates of clicking event
 ex = 0;
 ey = 0;
 //Initial color for background
 color = Color.pink;
 //Flag for first call to paint
 first = true;

}

}

//==================================================
//=========pppMenu CLASS============================
//==================================================

//This inner class enables the use of right clicking
//to activate a pop up menu.
//Sensing of the mouse is also enabled.
public class pppMenu extends JPanel
```

```
{
 private static final long serialVersionUID = 1L;
 private JRadioButtonMenuItem ItemsColor[];
 private JRadioButtonMenuItem ItemsMove[];
 private JPopupMenu popupMenu;


public pppMenu()
{
  //**********Color section

  //Handler for color menu items
  ColorHandler colorHandler = new ColorHandler();
  //Names of used colors
  String optionColor[] = {"Pink", "Black", "Blue" };
  //Colors are chosen by clicking a button
  ButtonGroup colorGroup = new ButtonGroup();
  //Creates pop-up menu
  popupMenu = new JPopupMenu();
  //Buttons for selecting colors
  ItemsColor = new JRadioButtonMenuItem[ 3 ];
  //The menu is constructed: it has 6 buttons
  //Three buttons are for color and other 3 for the move
  for ( int i = 0; i < ItemsColor.length; i++ )
  {
   ItemsColor[ i ] =
         new JRadioButtonMenuItem( optionColor[ i ] );
   //add item to pop-up menu
   popupMenu.add( ItemsColor[ i ] );
   //add item to button group
   colorGroup.add( ItemsColor[ i ] );
   //add handler
   ItemsColor[ i ].addActionListener( colorHandler );
  }
  ItemsColor[ 0 ].setSelected(true);

  popupMenu.addSeparator();

  // **********Move section
```

```
// Handler for color menu items
MoveHandler moveHandler = new MoveHandler();
//Names of used moves
String optionMove[] = { "Up", "Down"};
//Moves are chosen by clicking a button
ButtonGroup moveGroup = new ButtonGroup();

//Buttons for selecting moves
ItemsMove = new JRadioButtonMenuItem[ 2 ];


//The menu is constructed: it has 2 buttons
for ( int i = 0; i < ItemsMove.length; i++ )
{
 ItemsMove[ i ] =
       new JRadioButtonMenuItem( optionMove[ i ] );
 //add item to pop-up menu
 popupMenu.add( ItemsMove[ i ] );
 //add item to button group
 moveGroup.add( ItemsMove[ i ] );
 //add handler
 ItemsMove[ i ].addActionListener( moveHandler );
}
ItemsMove[ 0 ].setSelected(true);


//The popUp menu is accessed by  releasing
//of the right button.
//Attention: the mouse listener is added to the canvas c
c.addMouseListener
 (
   new MouseAdapter()
   {
    //Detector of right button releasing operation
    public void mouseReleased( MouseEvent event )
    {
     if  ( event.isPopupTrigger() )
      popupMenu.show(
      event.getComponent(),event.getX(),event.getY() );
     }
```

```
       }
      );
   // Detection of the mouse
   MouseHandler handlerr = new MouseHandler();
   //Sensors are added to the canvas c
   c.addMouseListener( handlerr );

  }

//If a color  event has been detected, here is the reaction
private class ColorHandler implements ActionListener
{
 public void actionPerformed( ActionEvent event )
 {
  //Determines which option was selected
  for ( int i = 0; i < ItemsColor.length; i++ )
   {
    if ( event.getSource() == ItemsColor[ i ] )
         d.optionColor = i;
   }
  d.clickedRight = true;
  c.repaint();
 }
}

//If a move event has been detected, here is the reaction
private class MoveHandler implements ActionListener
{
 public void actionPerformed( ActionEvent event )
 {
  //Determines which option was selected
  for ( int i = 0; i < ItemsMove.length; i++ )
   {
    if ( event.getSource() == ItemsMove[ i ] )
     d.optionMove = i;
   }
  d.clickedRight = true;
  c.repaint();
 }
}
```

```
//Movements of the mouse are detected
private class MouseHandler implements MouseListener
{
public void mouseEntered( MouseEvent event )
{
//Some invisible code goes in here.
}

public void mouseClicked( MouseEvent event )
{
 d.ex = event.getX();
 d.ey = event.getY();
 //The clicked finger and phalange are identified
 c.identify();
 c.repaint();
 d.clickedLeft = true;
 c.repaint();
}

public void mousePressed( MouseEvent event )
{
 /*System.out.format ("The mouse was pressed " +
"at [%d, %d] \n", event.getX(), event.getY() );*/
}

public void mouseReleased( MouseEvent event )
{
 /*System.out.format ("The mouse was released " +
"at [%d, %d] \n", event.getX(), event.getY() );*/
}

public void mouseExited( MouseEvent event )
{
 /*System.out.format ("The mouse exited the frame at" +
" [%d, %d] \n", event.getX(), event.getY() );*/
}

}  //end of inner class MouseHandler
```

```
} // end of class  pppMenu


 //===========================================
 //============== CLASS CANVAS ===============
 //===========================================

 /*Inner class Canvas

 We display a proto-hand which is sensible to
 punching by the mouse.

 Our protohand has 5  fingers, a palm and an arm.
 Fingers have 3  phalanges except the thumb, which has 2.

 Since we use finite precision, all our moves are
 approximations. Therefore, a multiplication of errors
 is expected if we use modifications in tandem.
 That is why  every scene is built ab initio by
 modification of a unique unmodifiable cube at the origin.
 The basic hand is located and moves at the origin
 and points downwards over the screen.

 The inner class Hand is encoded beginning from line 869
 The inner class Finger is presented in line 933.
 A finger is composed of 3 cubes. The class Cube
 appears in line 1043.

 */


 public class Canvas extends JPanel
 {

  public static final long serialVersionUID = 1L;

 /*
 Our basic unit is  the unitary cube at the origin.
 It is composed of 8 points or vertexes = oPoints3d[][],
```

```
 12 edges = edges[][] and 6 faces = faces[][].
 Points are numbered from 0 to 7.
 These numbers are used in
 edges, which  record the pairs of
 points that are joined by a line.
 A face is determined by its 4 corners.

 All other cubes are transformations of this
 fundamental cube. The form as this is done
 appears in the class Cube.
 */


 //Constructor
 /*This constructor is necessary when one have various
 nested classes. In our case, we have the class Cube,
 which is used by the class Finger,
 which is used by the class Hand,
 which is used by the class Canvas,
 which is used by the main class.
 */
  public Canvas( )
  {

  }

// The points of the fundamental cube
 private     double oPoints3d[][]=
 {
  {0,0,0},
  {1,0,0},
  {0,1,0},
  {0,0,1},
  {1,1,0},
  {1,0,1},
  {0,1,1},
  {1,1,1}
 };

 /*
```

```
// The edges of the fundamental cube
 private static int edges[][] =
    {{0,1},
     {0,2},
     {0,3},
     {1,4},
     {1,5},
     {2,6},
     {2,4},
     {3,5},
     {3,6},
     {4,7},
     {5,7},
     {6,7}};

 */

 // The faces of the fundamental cube
 private   int faces[][] =
    {{0,1,4,2},
     {0,2,6,3},
     {0,3,5,1},
     {3,5,7,6},
     {1,4,7,5},
     {2,6,7,4}};


  //Measures of the window
  private    int   xWin = 600;
  private    int   yWin = 700;
  private   int nScenes;

  //Number of cubes in the finger
  private     int nPhalanges = 3;
  private     Boolean moveArm = true;

  Random r  = new Random();

  //The constructor of hand is invoked here.
  //h and hCopy are instance of the main class Hand
```

```
  private  Hand  h =  new Hand();
  private  Hand  hCopy =  new Hand();
  //An auxiliary  instance of Finger
  private   Finger f1=   new Finger(nPhalanges);
  //Finger and phalange
  private int f, ph;
  //A constant that determines how much a phalange
  //must go up or down.
  private double deltaUp;



  // The window is enlarged.
  // This method is fired automatically
  // at the beginning
  public void init()
  {
   setSize(xWin,yWin);
  }

 //Clicked finger and phalange are identified.
 //80% exact.
 private    void  identify()
 {

  int px, py;
  int min =999999;
  for(int j = 4; j>=0;j--)
   {
if(j==4) nPhalanges = 2;
    else nPhalanges = 3;
for(int m = 0; m < nPhalanges; m++)
 {
  //A working cube cc is initialized
  Cube  cc = new Cube(m);
  //The working cube is re-set.
  cc = h.finger[j].phalange[m];

  //Projection is calculated
  cc.shade2d =  cc.round(cc.cubeProj );
```

```
 int d1 = 0;
 for(int l =0; l<4; l++)
  {
 px = 100-cc.shade2d[faces[1][l]][1];
 py =-150-cc.shade2d[ faces[1][l]][2];
 d1 = d1+ (px-d.ex) * (px-d.ex)
                 + (py-d.ey)*(py-d.ey);
  }

  if (d1<min)
   {
min = d1;
    f = j;
    ph = m;
   }
}
   }
 // System.out.println("Finger=" + f + " phalange=" +ph);
}

 //face ii of the cube is drawn
 private    void drawface(Graphics g, Cube cc, int ii)
 {
  for(int l =0; l<4; l++)
   {
    cc.pointsX[l] = 100-cc.shade2d[faces[ii][l]][1];
    cc.pointsY[l] =-150-cc.shade2d[ faces[ii][l]][2];
    //System.out.println(pointsX[l] + " " + pointsY[l]);
   }
   //A polygon is defined by two vectors and an integer
   //number:
   //The first vector lists the x-coordinates of the
   //corners;
   //the second lists  the y-coordinates. The integer is
   //the number of points.
   //If the last point is different than the first,
   //the first point is closed with the last.
   Polygon face = new Polygon(cc.pointsX,
                      cc.pointsY, 4);
   g.fillPolygon(face);
```

```
}

//Arm is drawn
private    void drawA(Graphics g)
{
 //System.out.println("drawA");
 //  A working cube cc is initialized
 Cube  cc = new Cube(1);
 //The working cube is re-set.
 cc = h.arm ;

 cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
 //Projection is calculated
 cc.shade2d =  cc.round(cc.cubeProj );
 //printShade();
 //The cube is drawn
 cc.calculateColor();
 for(int i=0;i<6;i++)
{
    if (cc.coloredFace[i])
   {
    g.setColor( new Color( cc.colors[i][0],
                   cc.colors[i][1],
                   cc.colors[i][2]));
    drawface(g,cc,i);
   }
 }
//worked cube is kept to memory
h.arm = cc;
}

//Arm is drawn
private    void drawArm(Graphics g)
{
 //System.out.println("drawArm");
 //A working cube cc is initialized
 Cube  cc = new Cube(1);
 //The working cube is re-set.
 cc = h.arm ;
 //Movement begins
```

```
   if (moveArm) f1.modifyCube(cc,0);
   //worked cube is kept to memory
   h.arm = cc;
   drawA(g);
  }

  //Basic palm is transformed
  private Cube repositPalm(Cube cc)
  {
   cc.oPosition[0]  =  h.arm.cube[3][0];
   cc.oPosition[1]  =  h.arm.cube[3][1]+5*d.ampliFactor;
   cc.oPosition[2]  =   h.arm.cube[3][2];
   cc.cube = cc.translateCube(cc.cube, cc.oPosition);
   return cc;
 }

  //Palm is updated
  private    Cube updatePalm(Cube cc)
  {
   cc = f1.inclination(cc,1);
   cc = f1.rotation(cc,1);
   //cc.printCube(cc.cube, " after rotation");
   cc = repositPalm(cc);
   //cc.printCube(cc.cube , " after relocation");
   for(int i = 0; i<3;i++)
   cc.vTrans[i] = cc.vTrans[i] + cc.deltaTrans[i];
   cc.cube = cc.translateCube(cc.cube,  cc.vTrans);
   //cc.printCube(cc.cube, " after translation");
   cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
   //cc.printCube(cc.cubeProj,  " after projection");
   return cc;
  }

  //Draw palm
  private    void drawP(Graphics g)
  {
   //System.out.println("drawP");
   //A working cube cc is initialized
   Cube  cc = new Cube(1);
   //The working cube is re-set.
```

```
   cc = h.palm ;
   cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
   //Projection is calculated
   cc.shade2d =  cc.round(cc.cubeProj );
   //printShade();
   //The cube is drawn
   cc.calculateColor();
   for(int i=0;i<6;i++)
  {
     if (cc.coloredFace[i])
    {
     g.setColor( new Color( cc.colors[i][0],
                    cc.colors[i][1],
                    cc.colors[i][2]));
     drawface(g,cc,i);
    }
  }
   //Memory is updated
   h.palm = cc;
  }

  //Draw palm
  private    void drawPalm(Graphics g)
  {
   //A working cube cc is initialized
   Cube  cc = new Cube(1);
   //The working cube is re-set.
   cc = h.palm ;
   //Basic palm is translated
   if (moveArm) cc = updatePalm(cc);
   h.palm = cc;
   drawP(g);
  }

  //Thumb, finger number 4, is built
  private void buildThumb(Graphics g)
  {
   //Position of the thumb
   for(int j=0; j<3;j++)
h.finger[4].phalange[0].oPosition[j]
```

```
                =  h.palm.cube[3][j];

   h.finger[4].phalange[0].oPosition[1]
     = h.finger[4].phalange[0].oPosition[1]
       -Math.round(10*1.2*d.ampliFactor);
   nPhalanges = 2;

   for(int m = 0; m < nPhalanges; m++)
{
 //A working cube cc is initialized
 Cube  cc = new Cube(m);
 //The working cube is re-set.
 cc = h.finger[4].phalange[m];
 //Horizontal position
 if (!(moveArm))
   {
    cc.deltaPhi[m] = 0;
    cc.deltaTheta = 0;
   }
 cc = h.finger[4].modifyCube(cc,m);

 //worked cube is kept to memory
 h.finger[4].phalange[m] = cc;
}

   if (moveArm)
  {
   //Vertical initial position
   h.finger[4].phalange[0].phi[0]
           = h.finger[4].phalange[0].phi[0]+3*deltaUp;
  h.finger[4].phalange[1].phi[1]
           = h.finger[4].phalange[1].phi[1]+4*deltaUp;
  for(int i=0; i<2;i++)
      {
   h.finger[4].phalange[0].deltaPhi[i] = 0;
   h.finger[4].phalange[1].deltaPhi[i] = 0;
   }
  }
   //deltaUp = 0;
  }
```

```
//Fingers are posited below the palm
private    void locateFingers()
 {
  for(int i=0; i<4;i++)
for(int j=0; j<3;j++)
 h.finger[i].phalange[0].oPosition[j]
                                  =  h.palm.cube[3][j];
  for(int i=0; i<4;i++)
h.finger[i].phalange[0].oPosition[1]
               = h.finger[i].phalange[0].oPosition[1]
                                  -10*i*d.ampliFactor ;

 }

private void paintFinger(Graphics g, int j)
{
if(j==4) nPhalanges = 2;
  else nPhalanges = 3;
  for(int m = 0; m < nPhalanges; m++)
  {
//A working cube cc is initialized
Cube  cc = new Cube(m);
//The working cube is re-set.
   cc = h.finger[j].phalange[m];

   //Projection is calculated
   cc.shade2d =  cc.round(cc.cubeProj );
   //printShade();
   //The cube is drawn
   cc.calculateColor();
   for(int i=0;i<6;i++)
    {
  if (cc.coloredFace[i])
   {
    g.setColor( new Color( cc.colors[i][0],
                   cc.colors[i][1],
                   cc.colors[i][2]));
    drawface(g,cc,i);
```

```
    }
     }
    //worked cube is kept to memory
    h.finger[j].phalange[m] = cc;
   }
}

//Draw fingers
private    void drawF (Graphics g)
{
 //System.out.println("drawF");
 locateFingers();
 buildThumb(g);
 for(int j = 3; j>=0;j--)
paintFinger(g,j);
 paintFinger(g,4);
}


//Draw fingers
private    void drawFingers(Graphics g)
{
 locateFingers();
 nPhalanges = 3;
 for(int j = 3; j>=0;j--)
 {
  for(int m = 0; m < nPhalanges; m++)
  {
//A working cube cc is initialized
Cube  cc = new Cube(m);
//The working cube is re-set.
   cc = h.finger[j].phalange[m];
   //Translation and rotation of the basic arm
    if (!(moveArm))
     {
     cc.deltaPhi[m] = 0;
     cc.deltaTheta = 0;
     }
    cc = h.finger[j].modifyCube(cc,m);
```

```
   //worked cube is kept to memory
   h.finger[j].phalange[m] = cc;
  }
 }
 drawF(g);
}

//Arm, palm, finger are reposited
private void moveAPF(Graphics g)
{
drawAP(g);
 drawFingers(g);
}

//A phalange goes up
private    void  showUp(Graphics g, int f, int ph)
{
 h.finger[f].phalange[ph].phi[ph]
             = h.finger[f].phalange[ph].phi[ph]-deltaUp;
moveAPF(g);
}

//A phalange goes back down
private    void  showDown(Graphics g, int f, int ph)
{
 h.finger[f].phalange[ph].phi[ph]
             = h.finger[f].phalange[ph].phi[ph]+deltaUp;
 moveAPF(g);
}


//Phalange is moved
private    void  movePhalange(Graphics g)
{
 //System.out.println("movePhalange");
 moveArm = false;
 locateFingers();
 coloring(g);
 if (d.optionMove == 0) showUp(g,f,ph);
 if (d.optionMove ==1) showDown(g,f,ph);
```

```
   d.clickedLeft = false;
   hCopy = h;
  }

  // Forearm and palm are drawn
  private    void drawAP(Graphics g)
  {
   drawArm(g);
   drawPalm(g);
  }

  //Hand
  private    void drawHand(Graphics g)
  {
   drawAP(g);
   drawFingers(g);
  }

  //The color chosen of the menu is executed
  private    void coloring(Graphics g)
  {
  //System.out.println("coloring");
  switch (d.optionColor  )
      {
       case 0:   d.color = Color.pink; break;
       case 1:  d.color = Color.black; break;
       case 2:  d.color = Color.blue; break;
      }

c.setBackground(d.color);
g.setColor(d.color);
   g.fillRect(0,0,xWin,yWin);
   //System.out.println("color = " + d.color);
  }

  //Basic hand is transformed into the actual one
  private    void presentation(Graphics g)
  {
   //Loop for the scenes
  for(int k = 0; k<nScenes; k++)
```

```
   {
  //System.out.println("scene " + k);
  drawHand(g);
  //delay  + erase drawing
  if (k < nScenes-1)
       coloring(g);
   }//end of for nScenes
  }



 //The hand that opens the session is built
  private    void initialPosition(Graphics g)
  {
   //Repeatability is enabled
   r.setSeed(45);
   //Number of scenes of the presentation.
   nScenes = 4;
   //System.out.println("First time");
   d.first = false;
   deltaUp = 0.25;
   coloring(g);
   presentation(g);
   hCopy = h;
  }


  //When paintComponent() is called,
  //everything is repainted
  //using information in memory
  private    void refresh(Graphics g )
  {
//System.out.println("refresh");
coloring(g);
 h = hCopy;
 drawA(g);
 drawP(g);
 drawF(g);
  }
```

```
 //This method has no effect apart from
 //remedying an unknown bug (maybe in Java itself).
 //Exercise: block it out to see the result
 private void beta(Graphics g)
 {
  //System.out.println("Beta");
  c.setBackground(Color.BLACK);
  c.setBackground(Color.cyan);
  d.clickedRight = false;
 }

 //This is the main method
public void paintComponent(Graphics g)
 {
  //if this method is called for the first time
  if (d.first) initialPosition(g) ;
  else //second and following calls
  {
   //System.out.println("Repaint");

   if (d.clickedLeft) movePhalange(g);

   if (d.clickedRight)
    {
     beta(g);
    }
  }
  refresh(g);
 }//end of paintComponent(G)

 //=======================================================
 //=============Beginning of Inner class Hand============
 //=======================================================

 public class Hand
 {
  private Cube arm = new Cube(1);
  private Cube palm = new Cube(1);
  private Finger finger[] = new Finger[5];
```

```
 //When memory is allotted for a hand,
 //all variables must be initialized.
 //Lower classes are initialized by default,
 //but some  redefinitions could be necessary.
 Hand()
 {
  //The arm is initialized
   final Cube c1 =    new Cube(1);
   arm = c1;
      //Dimensions and location of the arm.
   arm.scale[0] = 8;
   arm.scale[1] = 30;
   arm.scale[2] = 100;
   for(int i=0; i<8;i++)
   for(int j=0; j<3;j++)
      arm.points3d[i][j] = -d.ampliFactor*arm.scale[j]
                                          *oPoints3d[i][j];


   //The  palm is initialized
   final Cube c2 =    new Cube(1);
   palm = c2;
   //Dimensions and location of the palm.
   palm.scale[0] = 8;
   palm.scale[1] = 40;
   palm.scale[2] = 60;
   for(int i=0; i<8;i++)
for(int j=0; j<3;j++)
 palm.points3d[i][j] =  -d.ampliFactor
                * palm.scale[j]*oPoints3d[i][j];


   //Fingers are initialized
   for(int i=0; i<5;i++)
   {
   final  Finger f =   new Finger(nPhalanges);
   finger[i] = f;
   }

 }
```

```
  }
  //=============End of Inner class Hand============

  //================================================
  //=============Beginning of Inner class Finger======
  //================================================

  /*Variables and methods of the class Finger are
  encapsulated in the same unit.
  All definitions are non static, i.e., they are
  used as erasable workbenches and resettable tools.

  A finger is a coordinated set of 3 cubes.
  The class cube is defined below.

  All methods that are of exclusive use for fingers are
  encapsulated here.
  */
  public class Finger
  {

   //Our finger is a composition of 3 cubes or phalanges
   private   Cube[] phalange    = new Cube[3] ;


   //When memory is alloted for a finger,
   //all variables must be initialized
   Finger(int nPhalanges)
   {
   for(int m =0; m < nPhalanges; m++)
     {
      final Cube cc =    new Cube(m);
      phalange[m] = cc;
     }
   }
//  The only allowed movement of a phalange is
   //inclination, forth and back, in within bounds.
   //Inclination of given phalange governs the
   //rest of the finger
   private   Cube inclination(Cube cc, int m)
```

```
 {
  //Inclination is equal to a rotation.
  //Axis of rotation: the y axis.
  cc.axisRot[0] = 0;
  cc.axisRot[1]=1;
  cc.axisRot[2]=0;
  switch(m)
   {
case 0: cc.phi[0]  = cc.phi[0]  + cc.deltaPhi[0] ;
           break;
case 1: cc.phi[1]  = cc.phi[0] + cc.phi[1]
                                + cc.deltaPhi[1] ;
           break;
case 2: cc.phi[2]  = cc.phi[1] + cc.phi[2]
                                + cc.deltaPhi[2] ;
           break;
   }
 cc.cube = cc.rotCube(cc.points3d, cc.axisRot,
                             cc.phi[m] );
 return cc;
}

   //For visualization, the finger is rotated
//around the z axis.
private   Cube rotation(Cube cc, int m)
{
 //Rotation about the z axis.
 cc.axisRot[0] = 0;
 cc.axisRot[1]=0;
 cc.axisRot[2]=1;
    cc.theta = cc.theta + cc.deltaTheta;
 cc.cube = cc.rotCube(cc.cube, cc.axisRot, cc.theta);
 return cc;
}

 //Each phalange must have its initial position that
 //depends on the position of the phalanges below it
 private   Cube rePosit(Cube cc, int m)
 {
   //System.out.println("rePosit(m)" + m);
```

```
 switch(m)
  {
   case 0:  {
      } break;
   case 1: {
cc.oPosition[0]  =   phalange[0].cube[3][0];
cc.oPosition[1]  =   phalange[0].cube[3][1];
    cc.oPosition[2]  =   phalange[0].cube[3][2];
   break;
       }
   case 2: {
cc.oPosition[0]  =   phalange[1].cube[3][0];
cc.oPosition[1]  =   phalange[1].cube[3][1];
cc.oPosition[2]  =   phalange[1].cube[3][2];
   break;
       }
  }
  cc.cube = cc.translateCube(cc.cube, cc.oPosition);
  return cc;
 }

 //The cube is inclined, rotated, translated, projected
 private    Cube modifyCube(Cube cc, int m)
 {
  cc = inclination(cc,m);
   cc = rotation(cc,m);

  //cc.printCube(cc.cube, " after rotation");
  rePosit(cc,m);
  //cc.printCube(cc.cube , " after relocation");
  for(int i = 0; i<3;i++)
  cc.vTrans[i] = cc.vTrans[i] + cc.deltaTrans[i];
  cc.cube = cc.translateCube(cc.cube,  cc.vTrans);
  //cc.printCube(cc.cube, " after translation");
  cc.cubeProj = cc.projCube(cc.cube, cc.axisProj);
  //cc.printCube(cc.cubeProj,  " after projection");
  return cc;
 }
}//End of class Finger
```

```
  //=============End of Inner class Finger============

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

  //=============Beginning of Inner class Cube============


  public   class Cube
  {
   private double points3d[][] = new double[8][3];
   private   int shade2d[][] = new int[8][3];

//   All cubes are derived from the basic cube by rotations
//   and translations
   private double cube[][]  = new double[8][3];
   private   double  cubeProj[][] = new double[8][3];

   private double oPosition[] = new double[3];
   private   double axisRot[] = new double[3];
   private   double axisProj[]  = new double[3];
   private   double deltaTrans[]  = new double[3];
   private   double vTrans[]  = new double[3];
   private   double theta  = 0;
   private   double deltaTheta;

   private   double basisRot[][], basisProj[][]
                = new double[3][3];
   private   double matrixRotz[][],
   matrixRot[][],
   matrixProj[][] = new double[3][3];
//   The matrix of a projection along the z-axis
   private     double J[][]=
         { {1,0,0},
           {0,1,0},
           {0,0,0}};

   private    double scale[] = new double[3];

   int colors[][] = new int[6][3];
```

```
   private   int pointsX[]  = new int[6];
   private   int pointsY[]  = new int[6];
//   How to color faces
   private    double maxX;
   private   int nExtreme;
   private   boolean maxPoints[]  = new boolean[8];
   private   boolean coloredFace[]  = new boolean[6];
   //Angle of inclination of phalange
   private   double phi[] = new double[3];;
   private   double deltaPhi[] = new double[3];;

  //Every implementation of Cube is initialized here.
  //Parameter m is used for the initial position
  //of cube number m
  Cube(int m)
  {
   //Original  form
   scale[0] = 8;
   scale[1] = 8;
   scale[2] = 30;
   for(int i=0; i<8;i++)
   for(int j=0; j<3;j++)
   points3d[i][j] =  -d.ampliFactor*scale[j]*oPoints3d[i][j];
   defineColors();
   //Original position
   for(int i=0; i<3;i++)
    oPosition[i] = 0;
   //Each phalange has its own angle of inclination
   //with respect to its support
   for(int i=0; i<3;i++)
   {
    phi[i] = 0;
    deltaPhi[i]  = -0.1*(m+1);
   }
   //  The 3d cube flies along vectTrans[]
   //Each cube has its own direction of translation
   for(int i=0; i<3;i++)
    vTrans[i] = 0;
   deltaTrans[0] =  0;
   deltaTrans[1] =  0;
```

```
   deltaTrans[2] =  0;
   //  printCube();
   //  The 3d-graphic is rotated along the axisRot
   //  an angle theta that is incremented by deltaTheta.
   //  Each cube has its own velocity and direction of
   //  rotation.
   theta = 0;
   deltaTheta = -0.1;
   axisRot[0] = 0;
   axisRot[1]=0;
   axisRot[2]=1;
   //  The 3d-graphic is projected over the screen
   //  which is perpendicular to the x-axis.
   axisProj[0] = 1;
   axisProj[1]=0;
   axisProj[2]=0;


   }


  Random r  = new Random();

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

// The multiplication of two matrixes
 private  double[][] matrixMult(double p[][], double q[][])
 {
  double[][] helpM = new double[3][3];

  for(int i=0; i<3; i++)
  for(int j=0; j<3; j++)
```

```
   helpM[i][j] =0;

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

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


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

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

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

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

```
  double norm = Math.sqrt(a*a + b*b + c*c);
  helpV[0] = a/norm;
  helpV[1]  = b/norm;
  helpV[2]  = c/norm;
  return helpV;
 }


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

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


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

```
  return mRotz;
 }

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

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

 /*A 3d cube exists only in the imagination:
 * it is not seen.
 * One sees a shade or projection of it over the screen.
 * The shade is defined by 2d points. The edges of
 * the 2d shade are inherited from the edges of the
 * 3d cube.
 * The 3d cube is projected into a 2d plane.
 * The resultant figure must be rendered to the screen.
 * A projection is done along  an axis that crosses
 * through 0.
 * Our projections are observed from infinite:
 * we use no perspective corrections. */
 private     double[][] mProj(double w[] )
 {
  basisProj = basisB(w);
  double[][]  helpM = new double[3][3];
```

```
  helpM = matrixMult(J,matrixTranspose(basisProj));
  helpM = matrixMult(basisProj,helpM);
  //printMatrix(helpM, "Proj");
  //Test: Proj*Proj must be equal to Proj
  /*helpM = matrixMult(helpM,helpM);
  printMatrix(helpM, "Proj* Proj");*/
  return helpM;
 }

// The cube c is rotated around axis v an angle theta
 private double[][] rotCube(double c[][],
                            double v[], double theta)
 {
  double[]  helpV = new double[3];
  double cc[][] = new double[8][3];
  matrixRot = mRotation( v,theta);
  //printMatrix(matrixRot, "matrixRot");
  //The 8 points of the cube must be rotated
  for(int i=0; i<8; i++)
  {
   for(int j=0; j<3; j++)
   helpV[j] = c[i][j];
   helpV = matrixVector(matrixRot, helpV);
   for(int j=0; j<3; j++)
   cc[i][j] = helpV[j];
  }
  //printCube(cc, " after rot");
  return cc;
 }



// The cube c is translated   deltaV
 private double[][] translateCube(double c[][],
                                          double deltaV[])
 {
  double cc[][] = new double[8][3];
  for(int i=0; i<8; i++)
  for(int j=0; j<3; j++)
  cc[i][j] = c[i][j] + deltaV[j];
```

```
  return cc;
 }


// The cube c is projected along axis w
 private double[][] projCube(double c[][],  double w[])
 {
  double[]  helpV = new double[3];
  double cc[][] = new double[8][3];
  matrixProj = mProj(w);
  for(int i=0; i<8; i++)
   {
    for(int j=0; j<3; j++)
    helpV[j] = c[i][j];
    helpV = matrixVector(matrixProj, helpV);
    for(int j=0; j<3; j++)
    cc[i][j] = helpV[j];
   }
  return cc;
 }

// The rendering procedure accepts integer values
 private     int[][] round(double c[][])
 {
  int[][] helpM = new int [8][3];
  for(int i=0;i<8;i++)
  for(int j=0; j<3; j++)
  helpM[i][j] = (int) Math.round(c[i][j]);
  return helpM;
 }

 private   void printCube(double c[][], String a)
 {
  System.out.println("This is the cube" +a);
  for(int i=0;i<8;i++)
  System.out.println(c[i][0] + " " + c[i][1]+ " "
                                        + c[i][2]);
 }

 private   void printMatrix(double m[][], String a)
```

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

 private void printShade()
 {
  System.out.println("This is the shade");
  for(int i=0;i<8;i++)
  System.out.println(shade2d[i][1] + " "
                                   + shade2d[i][2]);
 }

// Colors to paint the faces are defined
 private int[][] defineColors()
 {
  for(int l = 0; l<6;l++)
   {
    //We use random colors.
    //A color can be defined by three numbers,
    //each one from 0 to 255.
    //We has 6 faces, so we need
    //a matrix to keep colors.

    int c1,c2,c3;
    c1 = r.nextInt(226);
    c2 = r.nextInt(226);
    c3 = r.nextInt(226);
    //Colors are memorized
    colors[l][0]= c1;
    colors[l][1]= c2;
    colors[l][2]= c3;
   }
  return colors;
 }

// what faces must be colored
 private    void marked()
```

```
{
 for(int f = 0; f<6; f++)
 coloredFace[f] = false;
 for(int f = 0; f<6;f++)
 {
  for(int p = 0; p<4;p++)
  {
   for(int c = 0; c<8; c++)
    {
     if (maxPoints[c] == true)
     {
       if (faces[f][p] == c )
       coloredFace[f] = true;
       //System.out.println("coloring");
     }
   }
  }
 }
}


// The x coordinate of all points are measured
// and the extreme ones are recorded.
private    void calculateColor()
{
 //The maximum value of coordinate x
 maxX =-1000;
 for(int l = 0; l<8; l++)
  {
   if (cube[l][0] > maxX)
    {
     maxX = cube[l][0];
    }
   }
 nExtreme = 0;
 //Number of extreme points
 for(int l = 0; l<8;l++)
  {
   maxPoints[l] = false;
   if (Math.abs(cube[l][0] - maxX )< 2)
```

```
    {
     nExtreme = nExtreme +1 ;
     maxPoints[l] = true;
    }
   }
   marked();
  }
  }//=============End of inner class Cube==============
 }
 }//end of main class
```

152 Human children learn late in their lives to move their fingers transversally. That is why we can suppose that transversal movements represent a refinement that aims at finely tuned skills that might be good to make precision works, such as the confection of tools and weapons which will trigger a tremendous adaptive advantage. Such precision movements are useful only when a very developed brain direct them and they could be the more valuable the more developed is the cultural environment in which life goes.

153 In **lamarckism**, one adds needed things and delete what one does not need. All in one step. So, things always look perfect.

Mutation happens at random in **neodarwinism**. If we already have a plane hand that functions, one needs a perturbation of the neuroanatomical structure that must create mutation in the next items: sites of insertion for muscles, appropriate neural connections together with suitable brain substructures that are well coordinated by the will. All that is too much complex a problem for us, so let us pay attention to a minor detail: how does a record fossil of the hand look at those strata when the traversal movements of fingers is added?

Our point of departing is that mutation is random. What does it mean to be random when the hand must be re-structured? Let us see:

Let us depict a hand as an ensemble of bones and muscles. The structure of such hand is defined by rules of the type: muscle number i has a point of insertion in bone $j1$, extreme $k1$, face $l1$ and the other insertion site in bone $j2$, extreme $k2$, face $l2$. The re-structuring of the hand by means of random mutation implies that one must produce a random perturbation of a given structure. Some possible changes are:

- Add a muscle without insertion points.

- Add a muscle with only one insertion point.
- Add a muscle with two insertions points on the same bone.
- Delete a muscle that is important for closing the hand.
- Delete an insertion point.
- Change the insertion point of a muscle to another one over the same bone but upon a wrong face.
- Use a muscle to tie the thumb with the little finger.
- Add a second thumb.
- Delete a phalange.
- Insert a phalange.
- Add a finger to the hand with independent neural connections.
- Delete a finger.
- Branch a finger with k subfingers.
- Branch the arm with l hands.
- Multiply arms.

We expect the fossil record and living populations to be filled in these and other mutants. The reason is that some of these mutations will produce minor damage to adaptability and so they might be tolerated and the carriers would reproduce.

What do we find in nature? We find with relative frequency that the hand is not protected from the work of mutation and various abnormalities have been observed, say, hands with an extra little finger or an extra thumb, which could be deprived of independent movement because they are half cooked products, say, they do not have independent nervous animation, or lack bones. As we see, evolution is capable of producing half cooked products. Our point is that the history of life must be filled in thousands of such type of products that trace the work of evolution when it went from incipient forms and functions to the overdeveloped ones that we see all around.

Can we associate observed mutations that produce abnormalities with the evolution from the ape that has no traversal movement of its fingers to the man which can play the flute?

No, we cannot because we have every right to expect evolution to be visible, i.e., we expect to find a history in the fossil record and/or living populations that shows that the hand of present populations is the winner among many, many trials. The next drawing show us some of them:

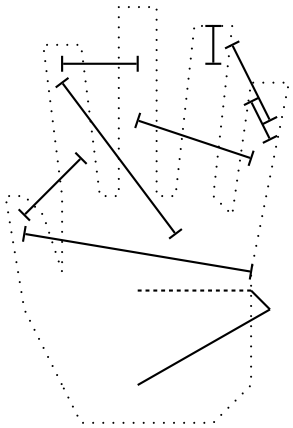


Figure 12.0. According to neo-darwininsm, the structure of the hand of present humans is the winner among a long history of organization trials generated at random and selected by functionality. Some trials that show muscles with their insertion points are shown here. The process of ontogenesis might delete the muscles that join separate fingers but not their points for insertion. So, it is natural to expect some of these trials in the record fossil and in modern populations: Where are they?

There is no evidence of such a history neither in the record fossil or in actual populations. For this reason we contend that the life we know is not explained by an evolutionary process in situ. It is just a vision that corresponds to a final user not to a process of developing.

154 Human embryos always have two hands and two feet but never four hands. Remaining questions are mysterious to us.

Since the four handed human embryo does not exist at any stadium of development, we consider that the **biogenesis law** is no law at all but only a good statistical generalization that demands anyway an explanation. An acceptable theory must predict both when this law is fulfilled and when it is falsified. This is a good opportunity to convert personal **convictions** into a program of investigation. At the mean time, the actual form of evolutionary theory seems to be a **conviction** against evidence.

155 In regard with the ontogenesis of the hand, we remark:

a . According to the evolutionary theory, moving from water to dry land did not happen all at a sudden, instead, it was made step by step: we have in first place fishes with lungs, then amphibians, then fully terrestrial animals. Amphibians have membranes among their fingers so that they could easily swing. So, the existence of membranes among fingers in the human embryo is a fact that fits the **biogenesis law** of evolution.

b . In **lamarckism**, the correct solution arises in terminated form in one step. In neo-darwinism, mutations happen at random with respect to function. So, we must consider many trials that produce plausible solutions that must compete one with another. Therefore, present structure arouse as the winner after a very long evolutionary process that must left tracks in the fossil record and in actual populations. Some examples of expected forms follow:

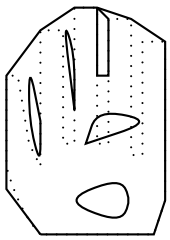


Figure 12.1. The hand of the embryo has membranes among fingers, in punctured lines, as in a duck. These membranes are later on digested by apoptosis, programmed cellular death. The correct mode of digestion concedes individuality to fingers without hurting the rest of the hand. This correct mode must be, according to neo-darwinism, the winner among many, many trials, one of which is presented here.

Bibliography

[1] Ayala F and other 14 authors (2008) Science, Evolution and Creationism, National Academy of Science USA, Institute of Medicine http://books.nap.edu/html/11876/SECbrochure.pdf. Cited 10 Jan 2008.

[2] Ball P (1995) Towards the synthetic dragonfly. *Nature* 375(6527):101–102.

[3] Darnell J, Lodish H, Baltimore D (1986) *Molecular cell biology*. Scientific American Books.

[4] Darwin C (1859) *The origin of species* Murrap, Londres.

[5] Dawkins R(1976) *The selfish gene* Oxford: Oxford University Press.

[6] Dawkins R (1986) The Blind Watchmaker. W.W.Norton & Company, NY, London.

[7] Dawkins R (2006)The God delusion.Houghton Mifflin Company. http://search.barnesandnoble.com/God-Delusion/Richard-Dawkins/e/9780618680009#EXC. Cited 6 May 2008.

[8] Dembski W (1998) *The Design Inference: Eliminating Chance through Small Probabilities*, Cambridge U. Press, New York.

[9] Dettman J, Sirjusingh C, Kohn L, Anderson J (2007) Incipient speciation by divergent adaptation and antagonistic epistasis in yeast. *Nature*, 445: 82–85.

[10] Drabek Christine and Braunl Thomas(1998) L-systems. ftp://ftp.informatik.uni-stuttgart.de/pub/lgrammar"¿ http://www.informatik.uni-stuttgart.de/ipvr/bv/lgrammar/lgrammar.html Cited Nov 2007

[11] Eclipse (classic) (2007) http://www.eclipse.org/downloads/. Cited 16 Dec 2009.

[12] Editor of Nature (2008) Editorial: Spread the word, *Nature* 451, 108.

[13] Fraser C M and other 26 authors (1995). The minimal gene complement of Mycoplasma genitalium. Science 270:397-403.

[14] Finkel T, Serrano M, Blasco M (2007) The common biology of cancer and ageing *Nature*, 448: 767–774.

[15] Galeano E (1988) Las venas abiertas de América Latina. Siglo XXI de Colombia Limitada.

[16] Gogniat A (1998) Biomorphs
http://www.cs.bgu.ac.il/ sipper/biomorphs/ Cited 8 july 2007.

[17] GoldenFeld N, C Woese (2007) Biology's next revolution *Nature*, 445: 369.

[18] Haag-Liautard C, Dorris M, Maside X, Macaskill S, Halligan D, Charlesworth B, Keightley P (2007) Direct estimation of per nucleotide and genome deleterious mutation rates in Drosophila. *Nature*, 445: 82–85.

[19] Hermann J, Marti-Arbona M, Fedorov A, Fedorov E, Almo S, Shoichet B, Raushel F (2007) Structure-based activity prediction for an enzyme of unknown function *Nature*, 448: 775–779.

[20] Hochbaum D (1995) Various notions of approximations: Good, better, Best, and More. *Approximation Algorithms for NP-hard Problems*, edited by Dorit S. Hochbaum, PWS Publishing Company, Boston, 346–394.

[21] Hutchison III Clyde A , Scott N Peterson, Steven R Gill, Robin T Cline, Owen White, Claire M Fraser, Hamilton O Smith, J Craig Venter (1999) Global Transposon Mutagenesis and a Minimal Mycoplasma Genome. Science Vol. 286. no. 5447, pp. 2165 - 2169, DOI: 10.1126/science.286.5447.2165

[22] José-José (2009) Linear Algebra. Unpublish. In Spanish.

[23] Kauffman (1995) At home in the universe. Oxford University Press, NY.

[24] Laurin, M. 2006. Scanty evidence and changing opinions about evolving appendages. Zoologica Scr. 35: 667-668.

[25] Laurin, Michel. 2009. Terrestrial Vertebrates. Stegocephalians: Tetrapods and other digit-bearing vertebrates. Version 22 August 2009.
`http://tolweb.org/Terrestrial_Vertebrates/14952/2009.08.22`
in The Tree of Life Web Project, http://tolweb.org/

[26] Levy E, Boeri E, Robinson C, Teichmann S (2008) Assembly reflects evolution of protein complexes. *Nature* 453, 1262-1265.

[27] Melott A (2002) Intelligent Design Is Creationism in a Cheap Tuxedo, Physics Today Online, Vol 55, June 2002, Opinion, http://www.aip.org/pt/vol-55/iss-6/current.html. Cited 11 Jan 2008.

[28] Monasson R, Zecchina R, Kirkpatrick S, Selman B, Troyansky L (1999) Determining computational complexity from characteristic phase transitions', Nature 400: 133–137.

[29] Mushegian A R and Koonin E V (1996) A minimal gene set for cellular life derived by comparison of complete bacterial genomes. Proc Natl Acad Sci U S A. 93(19): 10268–10273. PMCID: PMC38373.

[30] Padian K (2005) The testimony of Kevin Padian in Kitzmiller v. Dover, Edited by Nick Matzke, National Center for Science Education http://tinyurl.com/2nlgar cited 10 Jan 2008.

[31] Patel N H (2006) Evolutionary biology: How to build a longer beak. *Nature* 442, 515–516 (3 August 2006)

[32] Patten B (1958) *Foundations of embryology.* McGraw-Hill, N.Y.

[33] Popper R (1935) *The Logic of Scientific Discovery.* Hutchinson, London, 1959.

[34] Primas H (1983) *Chemistry, Quantum Mechanics and Reductionism.* Springer-Verlag, Berlin.

[35] Purves W, Sadava D, Orians G, Craig H (2001) *Life, the science of biology.* W.H. Freeman and Co. NY.

[36] Radman M, Taddei F, Matic I (2000) Evolution-driving genes. *Res. Microbiol.* 151 (2000) 91–95.

[37] Ray T (1990) Evolution and optimization of digital organisms. In Brown H (Editor), *Proceedings of the 1990 IBM Supercomputing Competition: Large Scale Computing Analysis and Modeling Conference.* MIT Press.

[38] Rodríguez J (2003) NP-harsh problems in education, *International Journal of Mathematical Education in Science and Technology*, v34, n4, p537-547. EJ770423.

[39] Rodríguez J (2007) On peacebuilding and individual psychology. *Antípoda* 5, 209–237. In Spanish.

[40] Rodriguez J (2008) *Electromagnetismo y geometría* arXiv:0806.1492 http://front.math.ucdavis.edu/0806.1492 Cited 12 june 2008.

[41] Rodriguez J (2010) Estadística y decisiones. In Spanish, 240 pages, 60 figures.

[42] Sun Microsystems (2009) Java Standard Edition, http://sun.com.com. Cited 16 Dec 2009. http://www.java.com/. Cited 16 Dec 2009.

[43] The Chimpanzee Sequencing and Analysis Consortium (25 authors) (1995) Initial sequence of the chimpanzee genome and comparison with the human genome. *Nature* 437, 69-87 (1 September 2005)

[44] Whitfield J (2007) We are a family *Nature*, 446: 247–249.

[45] Young R (2003) Evolution of the human hand: the role of throwing and clubbing. J Anat. 2003 January; 202(1): 165-174. doi: 10.1046/j.1469-7580.2003.00144.x. PMCID: PMC1571064

Index

Dawkins , 299
MAX-K-SAT=Maximum K-satisfiability, 552
mind , 35

belief, 36, 37, 40–43, 218, 461, 603, 653
biogenesis law, 606, 650, 800, 801

chance, 408
cladistics, 599, 602
class, 2
complexity, 469
conjunctive normal form, 512
consciousness, 35
console, 4
conviction, v, 294, 295, 401, 459, 461–463, 466, 475, 590, 653–660, 662, 664, 800
copy, 5
creationism, 297

Darwin, 160, 295, 432, 461, 462
delete, 5
dogma, 40

Eclipse, 1, 2
evolJava community, 3
evolution, 294
Evolutionism, 293
evolutionism, 296

Flute, 649
formula, 467

genome project, 295
ghosts, 39
gravity, 147

How to stop a program?, 406

Java, 1
    installing, 1
Java 3D API, 34

L-systems, 358
lamarckism, 511, 590, 591, 604, 646, 647, 651, 662–664, 798, 801
laws of Morgan, 513
Laws of software design, 250
laziness, 607

materialism, 37, 38
minimum genome, 469
model problem Shakespeare, 464
mouse detector, 221

neodarwinism, 511, 604, 647, 650, 651, 662, 798

OOP, 227
oracle, 475

paint, 242
pasting, 5
philosophy, 35
Pop-up menus, 227
probabilistic algorithm, class of problems, 513

proposition, 466
purpose of science, 461, 655

realism, 37
reality , 35
retro-engineering problem , 405

SAT, 468, 667
SAT = Satisfaction problem of mathematical logic, 463
satisfiable, 467
select all, 5
selection, 295
SOFT, 597
spiritualism, 40
systematics, 295

turtle, 358

vertebrates, 599
visual art, 358

What is complexity?, 464
what is perfection?, 599