JAVA FOR THE STUDY OF EVOLUTION

VOLUME I version 3

FUNDAMENTALS

José del Carmen Rodríguez Santamaría
The EvolJava Community

Dedication

To all those that renounce to the cosiness of fuzzy thought for the sake of precise knowledge and light and all the terrible suffering that these bring forth.

In plain words:

This work is dedicated to all those that experience a battle of his or her scientific beliefs against his or her intuition. On one side, science teaches that evolution is the one and only explanation of our existence while the intuition claims that so complex and wonderful life that surrounds us merits much more than evolution to be explained. Following the claims of their intuition in this regard many people invent or accept a God Creator. The Author is included here: he is a convinced Christian.

The price that a believer in the Creator must pay is to appear stupid else to tolerate two almost perfectly separated mental worlds: one in which science reigns and another in which faith gives forgiveness of sins, blessing for the enemies, joy and peace. So, these two worlds seem to complement each other and that is why some scientist are also believers. Nevertheless, this form of being is footed on fuzzy knowledge. In fact, evolution is a crude reality -just look how much we differ one from another- but the Modern Evolutionary Theory is just a dream in the mind of scientists. Why? Let us see: that theory claims that every form of life appeared by descent from previous ones and that the first appeared by self organization from inanimate matter. Now, it is true that we have an infinitude of data that fits this pattern -just compare the entrails of a pig or horse with ours. But it is false that we have proved that all relevant data fit the theory or even that it is close to do that. Quite to the contrary, the Evolutionary Theory is obviously false:

Every one knows that complexity is reign supreme of this world. This means that to achieve something interesting one must make many trials no matter how much one knows the problem at hand. Now, very complex problems exist and this means that huge amounts of verifiable trials must be done before one achieves an important goal. Example: to give a report of the many difficulties that one is

surviving to pay the mortgage is a witness that one is not associated with black knights. By the same token, if you do not want to make evolution into something magical you must show that the fossil record is filled in crooked forms and that modern populations are replenished with malfunctions and malformations. Now, demanded witness does not exist. Therefore, the Evolutionary Theory is false.

Dear Reader: the previous falsifying reasoning is subjective. This means that it is based on personal experiences together with an special way of interpretation. The problem is that its claims are so concrete that it is a duty to compute them accurately. That is the reason of our project: claimed falsification must arise naturally from the study of evolution. But it happens that this task is so complex that more and more problems appear at every step. Therefore, we have adopted the politics of enjoying complexity. This refers as to the human nature as to specific calculations.

In regard with the human nature, the following incident happened to me two months ago: when my reserve in my prepay phone account was small but undetermined I bought to a local reseller a recharge of 20000 Colombian pesos. This together with my credit balance, predicted that I should have around 22000 pesos at all. The phone company used to send me a message reporting my payments. But nothing was received at that occasion. So, I concluded that the payment was not made by the reseller. To verify my belief, I checked my balance and I read a cipher that was roughly 2500. I waited for a whole week and I neither received a message from my company nor my balance changed so, I went to the reseller and presented a complaint. As a direct proof of it, I invited him to check for my balance. But just when both of us looked at it, we read together a cipher that was around 25000 pesos. Additionally, the reseller was able to retrieve from the System the exact hour and day when he made the payment. To my shame, I was forced to concluded that the payment was done by the reseller but that my readings of the balance were wrong: my perception of the balance was distorted by my belief that the absence of a message from the phone company was a fair indicator of no payment.

Let me to grasp two lessons from this experience. First: perception is routinely modified by false or incomplete beliefs to twist and distort reality. Second: we need one another to exit such a deceiving atmosphere. Both claims are vividly illustrated by the animation *Cul de Bouteille (Specky four eyes)* directed by Jean-Claude Rozec ([12], 2010). The animation (filmaffinity, [5] 2011) presents a boy that consciously joins his extreme shortsightedness to his color blindness to a vivid imagination with the aim of perceiving a world inhabited by fantastic creatures. But after all, seen creatures are not that fantastic, they belong in the literature for children given by the culture that educated him. So, we see that education favors specific distortions in perception.

Now that we know that the human nature includes a capability of perception that can be distorted by beliefs, we present an invitation to explore Java to see how much it can help us to prove our falsifying claims about the Evolutionary Theory. In this first volume we have the fundamentals of Java and explain why the genome is software. We hope that going through proposed exercises will help the Reader to grasp a tremendous reality: software design is among the hardest tasks ever invented. What does it mean? It means that you cannot get a functional program unless you make many errors or bugs whose correction generate more bugs. To grasp this subjective experience is very important: since the genome is software you immediately will pursue a tracking of the bugs along the evolutionary process that according to science explains our existence. This will give you force to continue over a very long path that is in the making. So, you are welcome to the greatest pleasure of science: creating the unknown.

José Rodríguez
July/2015

PREFACE

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

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

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

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

Our motivations follow.

The conviction that propels science is that the fundamental laws of nature are quantitative and allow for the prediction of quantitative outputs. Thus, an evolutionist needs to acquire expertise in quantitative analysis. But that is very difficult no matter what approach one could use. For that reason, quantitative studies have been restricted to people with a huge knowledge of mathematics and/or programming that have hidden their wisdom behind insurmountable barriers of sophistication.

Fortunately, the advance of programming science has provided us with many languages that allow the study of quantitative questions at not a too high price. Experts are specially attracted by C^{++} over Unix or Linux. But as a rule, all experts use to forgive how difficult was to begin, and so few people understand the details of their work. Nevertheless, we feel that technology has matured enough to enable

the popularization of high-level programming. Our proposal in this book to prove that assertion is to use JAVA: it is a direct and simplified descendant of C^{++} that has been designed for attacking complex problems and to enabling programming in the web. Additionally, it can be acquired without fees. Moreover, its code is now open, so it has become an international human patrimony in permanent updating.

We will learn to use Java to calculate both analytical and statistical models and to make simulations. An **analytical model** is a set of equations and restrictions depicting some important aspects of some folds of reality. In **a statistical or probabilistic model**, we deal with random variables. On the other hand, a **simulation** consists in the building of an artificial world in the memory of a computer with exactly the form we like, to study it and to report the results.

It is important for an evolutionist to realize that there are various approaches to programming and that programming science has evolved over time. So, we begin with simple programs that report their output to the console, later we introduce Applets to make a modern interface, next we pass to visual simulations in which we learn the fundamental features about Object Oriented Programming.

Dear Reader: programming is a very difficult task. So, it is important to follow a method of study of the present material that causes the least trauma. The next could be a good methodology:

- Learn to run the programs that are contained in the present material.
- Try to understand their design.
- Execute on each program the modifications suggested by the exercises.
- In the measure that you gain confidence, take all programs one by one and give yourself the task to design a program with exactly the same function.
- Devise your own tasks and develop the corresponding code. Make sure that the designed program fills exactly the function it is intended to fill in.
- Our approach to Java is very intuitive so, it is insufficient to cope with precise questions. Besides, Java evolves. Nevertheless, the web is filled in very good tutorials, say, (Downey, [3] 2012). Explore the web with the tag *java tutorial*.
- Keep in mind that the exercises and their solutions are fundamental part of the text. So, answers by the author have been reported.

Version 2: more readability + less bugs + migration to NetBeans 8.02/2015. Version 3: a blatant omission was corrected.

Bogotá, Colombia, *José Rodríguez*
December 2009

Contents

Dedication i

PREFACE v

Codex xi

1 Starting up 1
- 1.1 General terminology . . . 1
- 1.2 Installing Java . . . 3
- 1.3 Purposes and projects . . . 5
- 1.4 Our first program . . . 7
- 1.5 Review . . . 9

2 Integer numbers 11
- 2.1 Elementary operations . . . 11
- 2.2 Structured programming . . . 13
- 2.3 The if-control-structure . . . 18
- 2.4 Review . . . 20

3 Using Excel and Calc 21
- 3.1 The tabulator . . . 21
- 3.2 The For Instruction . . . 23
- 3.3 Review . . . 27

4 Decimal numbers 29
- 4.1 Type double . . . 29
- 4.2 The for-control structure revisited . . . 30
- 4.3 Review . . . 32

5 Texts and strings **33**
5.1 Operators . . . 33
5.2 Structured versions . . . 43
5.3 Review . . . 46

6 Arrays **47**
6.1 Numeric arrays . . . 47
6.2 String arrays and the genetic code . . . 50
6.3 Review . . . 59

7 The random generator **61**
7.1 Random numbers . . . 61
7.2 Random DNA . . . 65
7.3 Positional effects . . . 66
7.4 Review . . . 71

8 The challenge of complexity **73**
8.1 Recombinant collisions . . . 82
8.2 Conclusions . . . 89

9 Semantics **91**
9.1 From strings to words . . . 91
9.2 Conclusions . . . 96

10 Evolution on letter-zymes **97**
10.1 Evolution . . . 97
10.2 Review . . . 113

11 More on Java **115**
11.1 Software design . . . 115
11.2 Project interconnection . . . 117
11.3 Objects . . . 122
11.4 Interactive programming . . . 122
11.5 Disk management . . . 125
11.6 Communicating with Excel and Calc . . . 131
11.7 Multithreading . . . 134
11.8 Review . . . 140

12 Distributions **141**
12.1 The psudo-random generator 142
12.2 The uniform distribution 148
12.3 The binomial distribution 153
12.4 The normal distribution 160
12.5 Fitting anyone distribution 163
12.6 Conclusions 172

13 The Pandora box **173**
13.1 General view 173
13.2 Competence among models 174
13.3 Evolution creates determinism 188
13.4 Conclusions 205

14 Markov chains **207**
14.1 Implementation 207
14.2 Conclusion 217

15 Lotka Volterra **219**
15.1 Deterministic model 220
15.2 A stochastic version 222
15.3 A logistic correction 225
15.4 Conclusion 228

16 Moths **229**
16.1 A model without mutation 230
16.2 A model with mutation 233
16.3 In hindsight 237
16.4 Conclusion 238

17 A Java interface **239**
17.1 Dialogs 239
17.2 My first Applet 241
17.3 A new style of programming 242
17.4 Conclusion 247

18 A virtual lab **249**
18.1 Implementation 249
18.2 Conclusion 259

19 Simulation in statistics **261**
19.1 One factor anova 265
19.2 A block design 274
19.3 An incomplete block design 284
19.4 Conclusion 294

20 Genetic drift **297**
20.1 One population 297
20.2 Many populations 301
20.3 Conclusion 306

21 Graphics and movement **307**
21.1 Graphics 307
21.2 Movement 310
21.3 Conclusion 316

22 Prototypes in OOP(Object Oriented Programming) **317**
22.1 Philosophical directives 317
22.2 Implementation 318
22.3 Conclusion 331

23 Bugs **333**
23.1 Inner classes 333
23.2 Conclusion 344

24 Conclusions **345**

Glossary **349**

Solutions **355**

Codex

Programming languages have at presently a very tight syntax, and Java is not an exception. So one must learn the formalism used for each case. We can find here the most usual expressions and the reference number where they appear along the text for the first time.

7. Printing a message to the console:
`System.out.println("I` am starting up! ”);

13. Printing the value of variable *x*:
`System.out.println("`Value of x=” + x);

18. Declaration of a procedure:
public static int sum(int x, int y)

26. The if structure: REMEMBER THE DOUBLE ==

```
if (y == 0)
System.out.print("The division by zero is forbidden");
else
 {
// Report
System.out.print(" The integer division of x =" + x );
System.out.println( " by  y = " + y + " is " + x/y);
 }
```

31. The Tabulator: `"\t"`

34. The for structure:

```
for(int i = 1; i<7; i++)
 {
  System.out.println(i + tab + i*i);
 }
```

50. Declaration of a String

```
String a = "Hello";
```

51. Concatenation of Strings:

```
String s4 = s1+s2+s3;
```

53. Replace operator:

```
s2 = s1.replace("ion", "ate");
c.replace(a,b);
```

55. Conversion of number to string. If the number is of type double, we use:

```
s1  = String.valueOf(r2);
```

When the number is of type int, we use the next procedure:

```
int i = 3;
Integer n = i;
String s = n.toString();
System.out.println(s);
```

60. Copy operator

```
s2 = s1.substring(4);
s3 = s1.substring(4,8);
```

66. The length of a String:
`int` n = s1.length();

81. Declaration of an array with integral entries

```
int data[];  // data is the name.
data = new int[ 10 ];
```

83. Declaration and assignation of a linear array Data

```
int  Data[] = {1,5,2,4,7,8,9,5,6,3,5};

int n = Data.length; //the length of the array Data.
```

87. To compare two strings a and d:

```
a.equals(d)
```

To declare an array with two columns:

```
String  GenCode[][] = { {"AUU","Ile"},
                        {"UUA","Leu"},
                        {"UAU","Tyr"},
                        {"AUA","Arg"},
                        {"AAU","Asn"},
                        {"AAA","Lys"},
                        {"UUU","Phe"},
                        {"UAA","Term"}
                       };
```

97. Random numbers and chars

```
import java.util.Random; //At the beginning of the program
Random r  = new Random(); // somewhere at the beginning
```

```
//A random number from 1 to 5
r.nextInt(6)   //In within a method

//This is a char or letter
char n = 'j';

//A random upper case char:
char ch =   ( char ) ( r.nextInt( 26 ) + 65 );

//A random lower case char:
ch =  ( char ) ( r.nextInt( 26 ) + 97 );

//Other variants
Random r = new Random (seedValue);

r.nextLong();

r.nextFloat();

r.nextDouble();

r.nextGaussian();
```

105. The case control structure:

```
// switch multiple selection structure
 public static void permutator(int k)
 {
       switch (k)
{
case 0: zero(); break;
case 1: one(); break;
case 2: two(); break;
case 3: three(); break;
case 4: four(); break;
case 5: five(); break;
case 6: six(); break;
```

```
case 7: seven() ; break;
case 8: eight(); break;
case 9: nine(); break;
}
 }
```

109. The while control structure:

```
while (this condition holds)
  {
    do this and that
  }
```

116. Char c is converted into string s

```
String s = c.toString();
```

??. Chaptering a project

```
//Program 145
//Chaptering a project,
//Chapter 1: a toolbox

public class A145
{

static void subtask1()
{
 System.out.println("I am subtask1");
}


static void task1()
{
 System.out.println("I am task 1");
 subtask1();
```

```
}

static void task2()
{
 System.out.println("I am task 2");
}

static void task3()
{
 System.out.println("I am task 3");
}
}



//Program 146
//Chaptering a project,
//Chapter 2:
//the head that uses the elements of
//the toolbox of chapter 1.
public class A146
{
public  static void main(String[] args)
 {
 // My general idea consists in three tasks
 A.task1();
 toolbox.task2();
 toolbox.task3();
 }
}
```

158. Disk and multithreading management: go to **??**.

199. Mathematical functions:

```
Math.pow(x,y); // number x to the power y.
Math.abs(z); //Absolute value of z.
```

208. random values and rounding:

```
r.nextGaussian() //normal distribution

  int c = (int) Math.floor(5*d) + 25; //rounding from below.
```

228. A number from 0 to 9 is converted to a char.

```
//Integer number i is converted to char z,
//when z is printed it looks just like i.
char z=  ( char ) ( i +48  );
```

230. A char is withdrawn from a String s and a char is given a numerical code:

```
  char c  =   s.charAt(0);
int sign = Character.getNumericValue(c);
```

280. How to capture data using panes and Applets: see program **??**

282. My first Applet, see program **??**.

310. Rounding from above

```
int d = (int)  Math.ceil(  1.2);
```

322. The negation of a boolean clause:

```
if (!(i==2)) |(!(j==2))
```

342. Drawing rects and ovals:

```
g.drawRect(10,15,100,100);
  g.fillOval(150,20,20,20);
  g.setColor(Color.red);
```

354. Prototypes in OOP (Object Oriented Programming). Go to **??** in the main text.

361. Random colors:

```
c1 = r.nextInt(226);
c2 = r.nextInt(226);
c3 = r.nextInt(226);
g.setColor( new Color(c1,c2,c3));
```

Not in the text: to print tables, use the instruction ”printf” that means ”print with format” (inherited from C). Suppose we want to print the next information:

```
Min = 121  ( 0,86 ) Max = 127  ( 0,91 )
```

To achieve that, we duplicate the information, but we replace the value of the variables by the format we want to be printed. The first variable, minFitnes = 121, is an integer, we use the format ”%d”. The second number is inf = 0.86, it is of type double, so we use the format ”%.2f” to indicate that the precision must be set to two figures. Then we add the names of the variables to be printed. Names are separated by a comma not by a plus. We get:

```
"Min = %d  ( %.2f )", minFitness , inf
```

If we want a change of car, we use `"\n"` that means ”new line”. The code that prints the complete information look like this:

```
System.out.printf( "Min = %d  ( %.2f ) "
            , minFitness , inf );
System.out.printf( "Max = %d  ( %.2f ) \n"
           , maxFitness , sup );
```

or like this, where we try to take care of the occupied number of spaces:

```
System.out.printf(
"Min = %7d  ( %10.2f ) Max = %7d  ( %10.2f ) \n "
       , minFitness , inf,  maxFitness , sup );
```

Chapter 1

Starting up

`Beginning from total ignorance.`

1 ***Purpose.*** *We learn the general terminology, how to acquire Java and NetBeans or Eclipse, how to install them, and how to run a first program.*

1.1 General terminology

There are some usual terms that we specify below.

2 ♣ ***General definitions***

In first place, let us notice that we, human beings, use languages to communicate one with another. A **language** is composed of verbal instructions, a code of interpretation and a semantic content. A **verbal instruction** is a string of sounds or letters that belong in an **alphabet**. Verbal instructions must be deciphered according to a **code of interpretation**, which appears, say, in a dictionary. The deciphered verbal instructions convey information about the world or about actions that must be undertaken and it is in this way that they acquire **semantic content**.

A **computer** is a devise that has the potential ability of executing verbal instructions. A **program** is a set of verbal instructions, written in a specific **programming language**, that can be interpreted and executed by a computer. **Code** is a set of instructions that eventually contain correctly written programs. **Software** is a set of programs that execute a task and very specially that create a friendly interface with the human. **Hardware** is the wiring that converts programs into specific actions. A **robot** is a machine that has a computer that has been programmed to have certain independence.

A **developer** is a person that is committed to the design of software, i.e., of programs with a specific function, which are defined before beginning but that maybe modified along the process. It is absolutely sure that the first trial of a code is inviable. Errors are called **bugs**. But if one manages to produce a viable code, most surely it will not do what one so eagerly expects. The process of adjusting the code to the predefined purpose is called **debugging**.

It is important to realize that the **genome** contains, apart from other things, a set of verbal instructions to program ribosomes to build specified proteins. All this can be read in (Rothamsted Research, [13], 2001)). The code that allows the deciphering of the verbal instructions contained in the genome is the **genetic code**, a dictionary that contains the correspondence among codons and amino acids. This dictionary is written in the t-RNAs: these molecules have in one extreme a codon and in the other the corresponding amino acid. Amino acids are assembled together by the ribosomes into proteins, which enable surviving in one of two ways: they may serve as structural items or as enzymes to catalyze chemical reactions. Thus, **surviving** is the feature that allows the genome to acquire a semantic content. A **cell** is a robotic entity: it is directed by the genome that is software and has a lot of independence.

So, the genome is for us the natural example of software. Let us emphasize this: **the natural example of software is the genome**. Thus, the evolution of the genome is a sub discipline of a more general matter: the study of the evolution of software in general. In fact, our study of **Java**, our platform of software design, will rise many questions of biological interest and especially in relation with the genome.

The name Java comes from a family of ants that lives in the island of Java. In a single tree but in different leaves one might find various nests of the same colony. The colony has various queens in proportion to the number of nests, which also determines the number of adult workers and immatures. But the numbers of males and of new queens go a path on their own.

To use Java directly is not as easy as using it in duo with NetBeans or Eclipse, which are precious interfaces of Java with the developer. All what follows refers to this ensemble that is generically referred to as Java. In strict sense, **Java is a great project** whose grandiosity might overwhelm at anytime. To begin, let us attempt a structural approach to it.

3 ♣ *Definitions.* *A **method** in Java is a portion of code that is designed to fill a determined function. A set of methods maybe gathered in a **class** to execute a great task. A class must be saved in a file with the same name, a relation that is case-sensitive. Classes are organized in books that are called **projects**.*

Java is the answer of our time to various and complex challenges posed to programming science. In this regard, we can understand the next points quite easily:

1. Automation: Usual repetitive tasks must be generalized, automated and implemented into a method.
2. Reusability: Any method must be ready to be incorporated into more larger methods.
3. Mutability: Any method must be modifiable to fit special needs.
4. Individuality: Any user shall have the opportunity to work according to his or her own style and needs.
5. Proficiency: the searching, calling or modification of a method must be millions of times easier than designing it ab initio.
6. Social proneness: one shall not get apart from the academic community or from the whole world. Java is the worldwide master in integration of programming workers.
7. Easiness of use: Java + NetBeans or Eclipse make of high level computing a realizable and rewarding task.

We will learn to install **Java**, to learn how to use **NetBeans or Eclipse** to run an existing Java program, and to manage the fundamental ingredients of programming that are directed to study models and evolution.

The basic items to simulate **evolution** are strings and numbers. That is so because DNA is a string of characters whose alphabet is A G T C, while many variables of phenotypic interest are naturally encoded in the form of numbers.

1.2 Installing Java

Java was created by Sun Microsystems in 1995. Sun Microsystems was acquired by Oracle in 2010. Sun distributed Java for free and Oracle has continued with the same tradition.

Java is not accessed by developers directly but through a friendly interface. The most popular interfaces are *Eclipse* of IBM and *NetBeans* of Oracle and both are distributed freely. The easiest to install is *NetBeans* because one has the possibility to download a bundle with *NetBeans* and the latest version of Java. Our material will be developed for NetBeans but some few guides will be included for Eclipse.

Java needs a folder for work. Create and name one, for instance *workspace3*. You can do this before installing your interface or during installation.

To download *NetBeans* + *Java* prompt Google with `netbeans jdk bundle` and look for a link to Oracle. Follow the link and arrive to the distribution, which by July/2015 is *JDK 8u51 with NetBeans 8.0.2*. Accept license agreement, choose your system (Windows, Linux or Mac), load and install the bundle on your desktop.

You might decide to suffer a bit by choosing a separate installation of Java and NetBeans. Or you would desire to experiment with Eclipse that also demands a separate installation. In both cases the idea is the same: get from Oracle (Oracle, [11], 2015) the latest version of Java, which by July/2015 is *Java SE (Standard Edition) 8u51*. Java comes with a lot of tools, which together form a JDK (Java Development Kit). Follow the links, match your system, say Windows or Linux, download Java and extract it to an appropriate folder.

Next, get an appropriate version of your interface to Java. We consider that the best version of NetBeans is the latest one, 8.02 at July/2015, and that of Eclipse is *Eclipse Luna* . The reason is that both can be used to work with *JavaFX*, the modern Java form of making graphics and animations.

To download Eclipse Luna proceed as follows: go to Eclipse home (Eclipse, [4], 2015) and follow the links:

Downloads → Eclipse IDE (Integrated Development Kit) for Java Developers → Releases → Luna Packages → Eclipse Luna SR2 Packages → Eclipse IDE for Java Developers → Download Links: Choose your system say, Windows 32-bit → choose a mirror for download. Download and extract your Eclipse to the same folder of Java.

To launch NetBeans or Eclipse one click or maybe two over an icon with a link to the execution file will suffice.

The web is full in tutorials about Java, NetBeans and Eclipse. Prompt Google with the key words that include your target + tutorial. In general, tutorials serve when they are combined with experimentation and suffering. For work in the long range, the complete guide to NetBeans would be very useful (NetBeans, [9] 2014).

The first task is to play with the new toy. If you want to be a great master, you must learn to enjoy playing permanently. By the way, give a try to your kids: most surely, they are looking for something really cool. Incite them to look here and there and try out whatever comes to the mind.

A very clear tutorial to create and run the first application or program in NetBeans is (NetBeans, [10] 2015). The same ideas can guide you to create your first program in Eclipse because both platforms are designed on the same basis and at last they are very similar. A series of videos to teach both Eclipse and Java are presented by (dextercowley, [2] 2015). Besides, Eclipse presents in its welcome window specific instructions to create a first program. Go to *Tutorials* → *Create a Hello world application*. Follow the instructions and try to create your first program.

If you fail to create your first program, use our methodology, which comes below.

4 *Guarantee.* *If, after some two hours of work, you cannot follow the instructions of the present chapter to run the first program, it is mandatory that you write me because you are guaranteed to have a happy start.*

Furthermore, we present many programs (evoljava, [1] 2015) that are guaranteed to work perfectly on Java SE 8u51 and both NetBeans 8 and Eclipse Luna Service Release 2 (4.4.2). Make your guarantee effective by expressing your discomfort to

`jose@evoljava.com`

It is rather difficult to understand how a program is designed but we have made an effort to explain why each program works, and the reader is committed to learn how to modify them to create variations that fulfill slightly modified functions. So, the first step is to learn how to copy a program into NetBeans or Eclipse that it could be run.

1.3 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 could be just to study the material of volume I. This intention defines a project. Then, we must communicate to Java our intention.

The organization of files in Java is in the form of a tree and is as follows: a project has two branches: Source Packages and Libraries. Source packages include all packages devised by the User. These packages are populated with files that may content say, a document written in plain text or maybe Java code in whose case their names have *.java* as suffix. Libraries are provided by NetBeans.

5 ***Opening a project***

To open a project follow the sequence of options beginning from the main menu: $File \rightarrow newproject \rightarrow Java \rightarrow Next$ A window will pop up for you with an invitation to fill the relevant data about your project. Give it a name: *eJVol1*. To end, click *Finish*.

By default, a source package is automatically generated together with a seed program. These have the same name as the project but may differ by some letters in upper vs lower case.

Now that we have a purpose, a project, and a seed program, let us copy and run a previously written program.

6 ***Copying a program***

Programs are not typed except when they are created. Instead, they are transcribed by a copy-paste procedure, for which you have four 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 and page numbers, so they must be identified and deleted. The identification is done automatically 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, which can be opened with a Latex editor. 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

4. If you are a teacher, you might prefer to import the accompanying zipped package with all programs in this volume. This option is not recommended for beginners: suffering is possibly the best teacher.

Copy now the program A7 EJVol1 below and replace the seed program with it. To that purpose:

1. Select the material you want to copy and use Ctrl + C to copy it to clipboard.

2. Select the seed program that was generated by NetBeans.

3. Apply Ctrl + V over the seed program to replace it with program A7.

1.4 Our first program

Our first programs will have the next structure:

```
//Comments begin with double slash.
/*
 This is also a comment.
*/

// This is the structure of a program.
//The name in Java of this program is myName.


public class myName
{
 //The next is the head of the program
  public static void main(String[] args)
  {
    //Here are the instructions.
  }
}
```

We have below a program whose purpose is to write a message to the console. The **console** is a special window to report results and errors. If whatever cause hides the console, one can restore its visibility as follows:

For NetBeans: In the main menu follow the path: *Windows* → *Output*. Else use Ctrl + 4.

For Eclipse: In the main menu follow the path: *Windows* → *ShowView* → *Console*. Else Click Alt+Shift + Q, then click C.

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

```
//Program   A7  EJVol1
// This is my first program.
//It prints a message to the console.
// printSomething is the name of this class.
//A7 is the reference number of this program along this series.
```

```
//Package declaration
package ejvol1;


//Class declaration
public class EJVol1
{

//The next 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! ");

}
}
```

We use in this program the method **System.out.println**. This method prints a message to the console and posits the console cursor at the beginning of the next line. One can use 'print' to write something but remaining in the same line.

8 ***Exercise*** *In the anterior code, decide which sentences end with a ";". Try to explain why. Observe that parentheses and keys always come by pairs. Java allows placing each parenthesis with freedom. Our directive is to enable a quick visualization of blocks. Point to the lines that contain a commentary and to those that contain code properly, i.e., instructions that Java must decipher and execute. What distinguishes code from commentaries?* *Answer*

9 ***Running our program***

To copy and run this class, go to the Run menu. We must before everything else to save and build the project and then run the file. To save: click over the Save All icon. To build the project: run + Build Project. To run the file: Run File.

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

10 ***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.*

Let us face to our first exercise in modifying a program. We are artificially evolving software. Java interfaces are very gentle with the developer and they mark errors in red. To **correct an error**, drag the cursor to the error and wait a moment that NetBeans pops up a message with the explanation of the nature of the error and possibly with a suggestion to correct it. Errors are also marked at the navigating vertical bars that surround the editor window. If a yellow light also appears, right click on it to read more suggestions for error correction. A suggestion can be executed at once by double clicking on it.

11 ***Exercise*** *Modify our first program in such a way as to write in the console your name and birthday. So, you will need to know how to append a file to a project:*

1. *Click on* New File *and a window will pop up.*
2. *Select your project. Verify that you are working with Java and that you will generate a Java Class. Click Next. In the emerging window fill in the name of the new class:* printMyProgram. *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* printMyProgram.java.
3. *You can now modify your new class at pleasure.*

Cut way: Under the Projects tab, look for your project. Expand its tree to find Source packages*, the package* ejvol1*, and the class* EJVol1.java*. Right click on the package* ejvol1*. A contextual menu will pop up and choose* New → Java Class*. Fill in the name and then click on* Finish*.* *Answer*

1.5 Review

We have learned the fundamental terminology and we have seen why the genome is our natural example of software: it contains verbal instructions that must be

decoded by t-RNAs and executed by the ribosomes. Computation is as old as life. We have learned to acquire and install Java and its companion NetBeans, where we have created our first project to which we appended run and modified our first program.

Chapter 2

Integer numbers

$....-3,-2,-1,0,1,2,3....$

12 ***Purpose.*** *Length is measured in centimeters and memory is measured in* ***bits****. In spite of the tremendous technological developments, memory continues to be a scare resource. So, to observe prudence in its management is an elementary duty. That is why a variable must be declared as belonging to a type, whose first information is to say how many bits must be allotted to record an instance of that variable. Our first type is* ***Integer****. Dealing with integers, we get acquainted with the control instructions* ***for*** *and* ***if*** *and with the structuring of programs.*

2.1 Elementary operations

13 ***Zero is the beginning. Integer numbers*** *in Java begins with zero: 0, 1, 2, 3, ..., but they are not allowed to reach infinity. Nevertheless, integers can be positive or negative and the sign is encoded just as in English. Because of memory restrictions, integers must be bounded somehow. Later, we will investigate the upper limit. In the next program, we learn to add two numbers.*

In a program, a variable must come with the declaration or specification of its type. This must be done just once. In the next program, we learn how to declare an **integer variable** and how to **add** two numbers.

```
//Program A13 SumOfIntegers
//This program introduces integers
//and their addition.

package ejvol1;
```

```
public class SumOfIntegers
{
    public static void main(String[] args)
    {
        //Declaration and initialization
        // of variables as integers.
        int x = 2;
        int y = 3;
        //Computation
        int result  = x+y; //result is declared as integer
         // Report
        System.out.println(" The sum of x =" + x + " and ");
        System.out.println( " y = " + y + " is " + result );
    }
}
```

14 ***Exercise*** *Copy the code to the clipboard, next open in project eJVol1, package ejvol1 a class, name it as A13, next replace the automatic generated text by that kept in the clipboard. Run the code. The output appears in the console. Verify the result. Play wit the code: change initial values of x and/or y.*

15 ***STOPPING A PROGRAM.*** *To stop the executing machine when a program is running, click on the terminator icon, in live red. If the red is very pale, the activated program has been stopped. Warning: if the machine gets too slow, there are possibly some programs that are running in background. To make sure that all launches have been canceled, go to menu Run, Stop Build/Run and terminate running programs.*

16 ***Exercise*** *To write on the console, one uses the instructions*

```
System.out.println(" The sum of x =" + x + " and ");
System.out.println( " y = " + y + " is " + Result );
```

The syntax reads: "print to the console(this and this)" but instead of "and" one uses a "+". Let us notice that text among quotations appear verbatim, but variables are replaced by their values. Modify the previous program that the output appears in one line instead of two. Answer

17 ***Exercise*** *Modify the previous program to make a multiplication of two numbers. To* ***multiply*** *apply the* ***star*** ** operator as in "a*b". Indicate also a* ***division***

*(use /) and a **subtraction** (use -). Warning: 6 /4 = 1.5, but when one declares the result to be integer, the division is ONE. The usual division (6 /4 = 1.5) will be recovered in the next chapter.* Answer

2.2 Structured programming

18 ***Motivation.*** *In the next code we will see the structuring style of programming, which was the great invention of the 1970s: you can build great structures by recursion to building blocks that are known to perfectly function. This style is the implementation of a simple but very powerful directive to try to break complexity: divide, name and conquer.*

A structured program is a program that is compound of various subunits, modules, subroutines, functions or methods, each of which fills in a determined function. Structuring a program makes things easier but also has a price in relation with syntax troubles.

```
/*
 * Program A18 StructuringProgramming
 * Some code is encapsulated into a function
 * or method.
 */
package ejvol1;


public class StructuredProgramming
{

    //The next is  a method.
    //It encodes for a function with input
    //two integers and output their sum.
    //The output is also integer and must be declared
    //explicitly before the name of the method.

    //The output of the method appears after
    //the reserved word "return".
    //The declaration of a method or a class
    //does not terminate with semicolon ";"
    public static int sum(int x, int y)
    {
```

```
        return  x+y;
    }


    //This is the main method.
    //This is the head of the class.
    public static void main(String[] args)
    {
        //Declaration and initialization
        //of variables
        int x = 2;
        int y = 3;
        //Invocation of a method
        int z = sum(x,y);
        // Report to the console
        System.out.print(" The sum of x =" + x + " and ");
        System.out.println( " y = " + y + " is " + z);
    }
}//End of program A18 StructuringProgramming
```

19 **Exercise** *Develop a program that executes the arithmetics of integers, in which each binary function is encapsulated into a method.* Answer

20 **Exercise** *Change the position of the methods of the previous program and try to run it to see what happens.* Answer

21 ***Choose a style that helps.*** *Java does not perceive any change in the relative position along the text of the different methods. So, one can posit them in whatever position one likes. Nevertheless, programming is a difficult task and it helps to keep track of the hierarchy of the methods. Our style is: the more dominant is a method the lower is its position in the text. Please, feel free to develop your own style, for instance, the head or main method must be up and the subordinated methods below in order of decreasing importance.*

22 ***A program can be structured as much as desired. Let us see the previous program with one more method:***

```
/*
 * Program A22 StructuringProgramming3
 * Encapsulation may become a mania.
 */
```

```
package ejvol1;


public class StructuredProgramming3
{

    //The next is  a method.
    //It encodes for a function with input
    //two integers and output their sum.
    //The output is also integer and must be declared
    //explicitly before the name of the method.

    //The output of the method appears after
    //the reserved word "return".
    //The declaration of a method or a class
    //does not terminate with semicolon ";"
    public static int sum(int x, int y)
    {
        return  x+y;
    }


    public static int multiplication(int x, int y)
    {
        return  x*y;
    }

    public static int subtraction(int x, int y)
    {
        return  x-y;
    }

    public static int division(int x, int y)
    {
        return  x/y;
    }

    //This method reports to the console
    // some information about x,y and
    //arithmetic operations with them
```

```
    public static void report(int x, int y,
            int z1, int z2, int z3, int z4)
    {
        System.out.print(" The sum of x =" + x + " and ");
        System.out.println( " y = " + y + " is " + z1);
        System.out.print(" The multiplication of x =" + x );
System.out.println( " and  y = " + y + " is " + z2);
System.out.print(" The integer division of x =" + x );
System.out.println( " by  y = " + y + " is " +  z3);
System.out.print("  x =" + x );
System.out.println( " minus  y = " + y + " is " + z4);
    }

    //This is the main method.
    //This is the head of the class.
    public static void main(String[] args)
    {
        //Declaration and initialization
        //of variables
        int x = 2;
        int y = 3;
        //Invocation of a method
        int z1 = sum(x,y);
        int z2 = multiplication(x,y);
        int z3 = division(x,y);
        int z4 = subtraction(x,y);
        //Report to the console
        report( x, y, z1,  z2,  z3,  z4);

    }
}//End of program A22 StructuringProgramming3
```

23 ***The degree of structuring depends on the style of the programmer. Let us consider two styles. First: a method and its commentaries shall not surpass the capacity of the editor window. Second: a method is worth dividing when the divisions are natural according to their function; otherwise division generates more work, whose product operates as noise when the need of revision arises.***

24 ***Structuring of the genome.***

The information in the genome is also structured. A nice proposal to understand the style of structuring is this: if a piece of code can be reused to complement another part of the genome, it is worth being separated. The separating symbols are known as **introns** and the useful code as **exons**. This is so because the DNA is transcribed to RNA and the introns are deleted from the RNA into the nucleus and so they remain internal to the nucleus, while the exons can leave the nucleus to be translated by ribosomes. Apart from this structuring, the cell also has spatial structuring in diverse **organelles**.

There was a time in programming science when programming was not structured. A program consisted just of code that was deciphered and executed line by line one after another. By the same token, there was a time in the history of the Earth when the genome was not structured and there were no organelles. In that time, life was dominated by prokaryotes. Structured information appeared with eukaryotes, cells that contained diverse organelles, a nucleus separated by a membrane and structured genomes.

One may suppose that the appearing of eukaryotes occurred with the intrusion of certain prokaryote inside another. The intruding bacterium could have evolved into a nucleus with a membrane, while the recipient could have lose its nucleus but gained the ability to host another parasitic cells that became organelles.

One can plague science with suppositions of this kind. They look attractive but lack any quantitative prediction and, more to the point, are naive: eukaryotes are important more because of their structuring programming than by their spatial compartmentalization.

Which is the mystery enclosed by structured programming?

Be it in reference to natural or to artificial software, structured programming amounts to a revolution that boosts the combinatorial power of possible changes: without structuring, one can combine letters but with structuring programming one can combine modules, methods or exons.

One combines letters to create exons and genes, one combines genes to create genomes. Thus, eukaryotes differ more by the form and tempo of regulation of their genes than by genes themselves. But prokaryotes may preferentially differ by their genes, whose regulation is quite simple (but very effective).

Now, we are forced to ask: Was the appearing of structuring of the genome triggered by bacteria parasitism? To answer this question, it is instructive to look at the evolution of software engineering.

Structuring programming appeared in software engineering because it was created to fill in three needs. First, to try to kill **complexity** using a strategy of divide + name + conquer. Second: to get profit of modular design. This type of design is

the one used in modern times by students: they copy from Internet one paragraph from here and other from there, concatenate them together and go to the teacher with a wonderful work. In software engineering, modular design is called **software reusability**, a terminology that is gaining popularity in evolutionary biology. Third reason: to shorten compilation times. With the advent of high level programming languages, the compilation or translation to machine language was a highly consuming time activity because debugging forced the developer to compile a program many times. But with modular design, small pieces of code could be perfectly assembled and compiled once and forever. In this way, the terrible load of debugging + compiling was lightened at least a bit.

The first reason is not applicable to nature, because it lacks any notion of complexity. The second reason possibly may be applied to nature if one shows that structuring the genome is an evolutionary stable strategy, i.e. that it is competitive in evolutionary terms. If that were the case, the permanence until our days of an already extant structuring of the genome could be understood. But this says nothing about its appearing. This problem is very interesting because the evolutionary competitiveness of structured genomes is by no means clear:

Certainly, a modular approach to problem solving can do more with less and the regulation capacity also grows. But the cost of control also grows and, because of length considerations, it is by far much more expensive to synthesize the DNA of a eukaryote than that of a prokaryote.

Can we, the EvolJava community, enrich these wonderful discussions with simulations that could lead to brilliant experiments? Dear Reader: our responsibility in this regard is very high because we indeed have the possibility to illuminate somehow these terrible problems.

2.3 The if-control-structure

25 ***Exercise*** *Develop else use a previous program to divide two numbers and to study division by zero. Verify that Java refuses to execute a division by zero. This must be so, because a division by zero leads to contradictions. Prove this.* *Answer*

26 ***Using conditional execution.*** *To prevent the spoiling of a program by a division by zero, one can use an* ***if-(then) -else*** *control condition as in the following program:*

Dear Reader, please, notice and remember once and forever that the if-structure tests a logical condition and if it refers to equality between two numbers, **the sign of equality comes twice**!

```
//Program A26 DivisionByZero2
//This program divides two integers.
//It is robust against
//division by zero.

package ejvol1;


public class DivisionByZero2
{
    public static void main(String[] args)
    {
        //Declaration and initialization
        // of variables
        int x = 7;
        int y = 0;
        //Logical testing: == instead of =
        if (y == 0) //If divison by zero is proposed
          System.out.println(
                  "The division by zero is forbidden");
        else //If y is different than 0, do this:
          {
           //Computation and report
           System.out.print(" The integer division of x ="
                                + x );
           System.out.println( " by y = " + y + " is " + x/y);
          }
     }
}
```

27 ***Exercise*** *Run the previous program and play with the code: test it for various values of the variable y. What do you observe?* Answer

28 ***Robustness as an impossibility.*** *To design software with guaranteed robustness against all the possible inputs given by the client is impossible in practical terms. That is why companies produce beta versions that are released for free in the hope that users could report the most usual problems. The same happens in biology: death + reproduction +* ***evolution*** *could be understood as a solution to the practical impossibility of devising perfect robustness against the milieu and self-destructing operations. Can this intrigue be made into a theory about the extinction of species?*

29 ***Challenge*** *Develop a structured program that makes the following:*

1. Writes your name, birth date, last employment and the date.

2. Calculates you age in years, in weeks and in days.

3. If you are younger than 22, write your favorite sport. If you are older than 22, write your favorite team.

2.4 Review

We have become acquainted with integers and its fundamental operations: addition, multiplication, division and subtraction. We learned also about structured programming and discovered that our natural example of software, the genome, also comes structured. Nevertheless, structuring may appear by diverse reasons, say by reusability, as it probably happens in the cell, or as a mean to try to kill complexity by divide + name + conquer, such as it happens with software designed by human beings. We learned to use the if-else-control structure to make programs robust against division by zero. The problem of devising robust programs for every input is so complex that companies prefer to release free beta versions that clients could report most frequent problems. It seems to us that this problem also concerns evolution and the extinction of species.

Chapter 3

Using Excel and Calc

`Java + Excel or Calc of Open Office.`

30 ***Purpose.*** *We must learn now how to combine the calculating power of JAVA with the graphical facility of* ***Excel****. Instead of Excel one can use* ***Calc of Open Office****, which is distributed for free. Instead of Windows, one also can work with Solaris, or UNIX or Linux. Excel and Calc allow the possibility of working with huge volumes of data, a reason that forces us to learn the for-control structure of Java.*

3.1 The tabulator

The output of the program just below contains the first 7 numbers and their corresponding squares. The idea is that we want to report the output graphically with the help of Excel, so we must make sure that the output is correctly written in the format of Excel, which also fits that of Open Office. The problem is that our output has two columns and we must guarantee that when copied to the clipboard and then pasted into an Excel or Calc sheet, one shall see two columns instead of one. Inserting a **tab** signal among the entries of different columns does this. The instruction responsible for that operation contains the encoding of a tab as follows:

```
"\t"
```

That symbol could appear as in the next instruction which prints the number 1 followed by a tab and next by the number 5:

```
System.out.println("1 \t 5");
```

In this program, the method or instruction

```
System.out.println
```

is used to write down on the console a text. The technical name for a text is string. To copy one string after another, the symbol + shall be used, than means concatenation. So, the last three instructions of the next code are equivalent:

```
String tab = "\t";
System.out.println("7" + tab + "49 ");
System.out.println("7 \t  49 ");
System.out.printf("7" ,  "\t" , "49 \n ");
```

All these instructions write down string 7, next a tab space is left and then the string 49 is written. JAVA distinguishes between the string ”7” and the number 7. Nevertheless, both are written in the console as number 7 and so both can be pasted into a sheet of Excel or Calc as numbers. The method *printf* means print with format, an option that is used here with the command `\n` that means move to the next line.

31 ***A concrete example that uses tabulators.***

```
//Program A31 Tab
// How to use the tab
// to print columns of numbers
// in a format appropriate for
// Excel or for Open Office Calc.

package ejvol1;

public class Tab
{

public static void main(String[] args)
    {
// Here a tab sign is defined.
String tab = "\t";

  // Report to the console
```

```
System.out.println("1" + tab + "1");
System.out.println("2" + tab + "4");
System.out.println("3" + tab + "9 ");
System.out.println("4" + tab + "16 ");
System.out.println("5" + tab + "25 ");
System.out.println("6" + tab + "36 ");
System.out.println("7" + tab + "49 ");
}
}
```

32 ***Exercise*** *Run the program, copy the output to the clipboard, paste it to a sheet of Excel and invoke over these data the menu Insert* $\rightarrow$ *graphics* $\rightarrow$ $XY\,(Dispersion)$. *You shall see the graphic of* $y = x^2$, *which is a parabola. If you work with Calc of Open Office, please follow the next instructions: Run the program, copy the output to the clipboard, paste it to a sheet of Calc with the right-click instruction* $PasteSpecial \rightarrow unformatted \rightarrow unicode + tab \rightarrow OK$. *To make a graphic: select the 2 columns of data, invoke Insert* $\rightarrow$ $char \rightarrow XYchar \rightarrow lines\ only \rightarrow create$. *You shall see the graphic of* $y = x^2$.

3.2 The For Instruction

Since we are expected to work with a large quantity of numbers, it is a good occasion to see the use of the **for** instruction in JAVA. The for instruction has the following pseudo-syntax:

33 *for(beginning with this number; while this condition holds; increment the counting index by one after each loop) do this and that.*

As an instance, the next block

```
for(int i = 1; i<7; i++)
 {
  System.out.println(i + tab + i*i);
 }
```

begins the counting index with 1, prints 1 and its square, increments the counting index in one, gets 2, compares it with 7, decides that it must continue looping, prints a 2 with its square and so on until the square of six.

In this piece of code we find for the first time the symbol $i++$. This is part of the new syntax, (after C++), where we have the next instructions:

$i+=1$: increment i in one.

$++i$: increment i in one, then use the new value of i in the expression, where i is located.

$i++$: use the current value of i in the expression where i is posited and then increment the value of i in one.

As one can guess, these symbols and many other facilities of Java allow great proficiency in software development. That is not our aim. Our purpose is to learn to study diverse models and to simulate evolution and every kind of related problems. Thus, we follow the next directives: a) Learn the maximum of biology and evolution with a maximum reuse of elementary things in programming. b) Learn something only when it pays a lot. It is unfortunate, nevertheless, to witness later that the minimum to be learned actually is a huge! Why must this be so? It is a question that also concerns evolution, since **evolution** is before everything else the creation of software!

We will use the notation

$i = i + a$

that means: add a to the current value of i.

This old fashioned notation is both clearly and effective. But in a for loop, we will use $i++$ because it means exactly the same as $++i$ and as $i = i + 1$. So, one must not care about possible confusions. In that way, we do not pose an excessive load on the reader and we make this material accessible and comprehensible for more people.

34 ***The code that contains a for loop follows:***

```
/*
 * Program A34 ForLoop
 */
//Introduction of the for-control structure.
//The "for" instruction causes a loop
//to deal with repetitive tasks.
package ejvol1;


public class ForLoop
{

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

```
    {
        String tab = "\t";

        for(int i = 1; i<7; i++)
        {
            System.out.println(i + tab  + i*i);
        }
    }
}
```

The letter *i* is a variable that controls the loop that repeats the operation of printing into the console the squares of the first 6 numbers.

35 ***Exercise*** *Run the program, recover your parable and play with the code.*

36 ***Exercise*** *Follow the instructions.*
In the previous program instead of the instruction

```
for(int i = 1; i<7; i++)
```

use the next one

```
for(int i = 1; i<=7; i++)
```

Check out the effect of the change. Answer

37 ***Exercise*** *Modify the previous program to calculate the squares but beginning from zero and going until 10 included. Show the corresponding graphic.* Answer

38 ***Exercise*** *Modify the previous program to calculate the cubes but beginning from zero and going until 20 included. Show the corresponding graphic.* Answer

39 ***Challenge*** *The division of i by 2 is indicated as i/2. Try and fail to modify the previous program to make it calculate the half of the first 15 integer numbers. The reason of your failure is that if the output of a division of an integer by two is declared as integer, it is an operation that rounds the result. To conserve decimals fractions, we need to use another type of numbers that appear in the next chapter.*

40 ***Integers are finite***

Integers go in many languages from approximately -32000 to + 32000, one by one.

To make an estimation of the allowed range of the integers in JAVA, let us use the next program. We will know that the limit is surpassed because when adding a positive number to another positive number, we get a negative number. Our program reports the first N multiples of 1000. We begin with $N = 10$.

```
/*
 * Program A40 ForLoop5
 * The upper bound of integers in Java.
 * Assign to N a large value.
 * Check for a negative output.
 * To stop the program, click over a red square
 * in the left bar.
 */
package ejvol1;


public class ForLoop5
{

    public static void main(String[] args)
    {
        String tab = "\t";
        int N= 10;
for(int i = 1; i<=N; i++ )
        {
            System.out.println(i + tab + 1000 * i);
        }
    }
}
```

41 ***Exercise*** *Run the program to see if 10000 is actually an integer in JAVA. Gradually increase the value of N to see when the result bends to negative values. Verify that 10000000000 is an integer.*

3.3 Review

We have managed to connect the output of Java, when it is of the int type, to Excel and Calc. Now, we enjoy the possibility to make graphics of our outputs. In prevision of high volumes of data, we have learned the use of the for control structure of Java and of some of its variations. As an application, we found that the number 10000000000 is admitted as integer. This is a facility that is not present in many languages.

Chapter 4

Decimal numbers

```
-17.234
```

42 ***Purpose*** *We learn to manage decimal numbers in Java.*

A decimal number is a string of digits among which a comma or a period seems to float. That is why they are encoded as belonging in the **float type**. But Java also has the type *double*, which is enough for all our needs.

4.1 Type double

Let us to learn how Java encodes decimal number with enough decimal ciphers (if they are not enough for some very accurate calculations, Java can provide you with an appropriate type of encoding).

43 ***Declaration***

JAVA allows the declaration of a decimal number as **double**, which means that we deal with a number with many decimal significant figures. How that works is seen in the next program.

```
//Program A43 TypeDouble
//The type double,
//to encode decimal numbers.
//Square roots are calculated.

package ejvol1;
```

```
public class TypeDouble
{

    public static void main(String[] args)
    {
        String tab = "\t";
        for(int i = 0; i<=15; i++)
        {
            double r = i;
            //Square root of r
            double o = Math.sqrt(r);
            System.out.println(i + tab + o);
        }
    }
}
```

44 ***Exercise*** *Run the previous program. Try to graphic its output in Excel or Calc. Explain.* *Answer*

45 ***Exercise*** *Modify the previous program to test the capacity of the console. Try with 100 as an upper limit of the counting index i. Use next 200,.. .., 1000, 2000,.. .. Do not let the console to misguide you: you must be able to see as the first as the last line without mutilations.* *Answer*

4.2 The for-control structure revisited

46 ***Variation of the for statement: let us pay attention to the next variation of the for statement:***

```
//Program A46 ForLoop6
// Variation of the for statement.
package ejvol1;


public class ForLoop6
{
    public static void main(String[] args)
    {
```

```
        String tab = "\t";
            for(int i = 0; i<=100 ; i= i + 7 )
            {
                System.out.println(i + tab + 1000 * i);
            }
    }
}
```

47 ***Exercise*** *Run the code and find the change in the output. Next, find the causal change in the code.* Answer

48 ***Fractional increments***

In the next program, we see a variation of the for structure in which we see that index variables admit decimal increments.

```
//Program A48 ForLoop7
//Variation of the for statement
//with fractional increments.

package ejvol1;


public class ForLoop7
{
    public static void main(String[] args)
    {
        String tab = "\t";
        //The index is declared as "double",
        //so it accepts a decimal  expression
        //in its increment.
        for(double i = 1; i<=2.01 ; i= i + 0.01 )
        {
            System.out.println(i + tab + 2*i);
        }
    }
}
```

49 ***Exercise*** *Run the program, study its output and find the function of each instruction. Verify your theories by composing a program to calculate the squares*

of the real numbers beginning with one and ending with 2 with a step of 0.05. *Answer*

4.3 Review

We have get acquainted with the treatment of Java to decimal numbers when they are declared as type *double*. In combination with a for-control structure, our new type allows for nice possibilities. It is also found that JAVA loses precision beyond 14 significant figures. We must tolerate that imprecision else look for other resources that are also provided by Java. Additionally, we discovered that our decimal numbers are incomprehensible for some versions of Excel and Calc because Java uses a period to indicate decimal fractions while Excel and Calc use a comma. Java has an answer for this and for every problem but we will devise our own approach in the next chapter.

Chapter 5

Texts and strings

```
ATTTTGGCCCTTTAAA
```

50 ***Purpose.*** *We have been acquainted with numbers, which are appropriate to deal with quantitative aspects of biology and evolution. But a DNA string is written with an alphabet that has four letters or* ***chars****: ACGT. Texts are best dealt in programming with* ***strings****. Thus, we need to understand various operators to manipulate strings to manage molecular biology and evolution.*

The type ***String*** *is is declared in Java as a* ***String****. Any string must be among quotes:*

```
String a = "Hello";
```

Once a string has been defined, it can be invoked by its name and modified by a handful of operators. Our purpose is to learn to work with them.

5.1 Operators

51 ***Concatenation.*** *The first operation with strings is concatenation, which consists in appending a string at the end of another. As it was noted above, the* ***concatenation*** *operator is "+". Let us see again how it works:*

```
//Program A51 Concatenation
//Concatenation of strings.

package ejvol1;
```

```
public class Concatenation
{

    public static void main(String[] args)
    {
        //  Declaration of type of variables.
        //  A declaration allots memory resources.
        String s1, s2, s3;
        //  Assignation
        s1 = "Java ";
        s2 = "for ";
        s3 = "biologists";
        //  Report to the console
        System.out.println(s1 + s2 +s3); //action
        //  Declaration and assignment
        String s4 = s1+s2+s3;
        //Variation
        System.out.println(s4 + "!"); //action
    }

}
```

52 ***Exercise*** *Run the previous program and modify it to write down from shorter substrings the phrase: 'My aim is to be the best simulator of evolution in the history of the world!'.* *Answer*

53 ***Replace operator.*** *The fundamental string operator* ***replace*** *allows us to replace a part of a string by another string. Its syntax can be typified as follows*

```
s2 = s1.replace("ion", "ate");
```

where the substring ion is replaced by ate in string s1 and the result is kept in string s2. In general, the operator

```
c.replace(a,b);
```

replaces a by b inside c. This operator is seen in action in the next program:

```
//Program A53 Replace
//Replace operator of strings.
```

```
//Replacement of all instances of a substring
//with another one.

package ejvol1;


public class Replace
{

    public static void main(String[] args)
    {
        //Declaration of type of variables
        String s1, s2;
        // Assignation
        s1 = "Mutation is the key to evolution.";
        System.out.println(s1); //action

        s2 = s1.replace("ion", "ate");
        System.out.println(s2); //action
    }
}
```

54 ***Exercise*** *Run and modify the program to fit your imagination.* *Answer*

55 ***Period vs comma***

As an application of the replace operator, let us solve the problem of the incompatibility of Java with some versions of Excel and Open Office in regard with **decimal numbers**. All we shall do is to change a point by a comma using the replace operator. We use also the method

```
double r2 = 0.00234;
String s1 = String.valueOf(r2);
```

to **change a number to string** . We need this to change a point by a comma inside a number and that must be done as a string. The program follows:

```
//Program A55 Replace
//Change of a period by a comma inside decimal numbers
package ejvol1;
```

```
public class Replace3
{
    public static void main(String[] args)
    {
        String tab = "\t";
        String s1, s2,  s4, s5 ;

        for(double r1 = 1; r1<=2.05 ; r1= r1 + 0.05 )
        {
            double r2, r3 ;
            r2= r1;


            //Conversion of   number r2 into the string s1
               s1 = String.valueOf(r2);
            //Replacement in s1 of "." with ","
            s2=s1.replace(".", ",");

            //Number r3 is the square of r1.
            r3= r1*r1;

           //conversion of number r3
           //into a string:
           s4 = String.valueOf(r3);
           //A period is replace by a comma
           s5=s4.replace(".", ",");
           //Report
           System.out.println(s2 + tab + s5);
        }
   }
}
```

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

57 ***Exercise*** *Study the previous program and digest it. To make sure of that, try to shorten and simplify the code.* *Answer*

58 ***Exercise*** *Devise a program that translates from German into English. Use the next dictionary:*

Auto = car;
bin = am
er = he;
ein = a
Fahrer = driver;
fahren = drive;
gut = good;
hast = has;
ich = I;
ist = is
müsse = must;
schlecht = bad;
teuer = expensive;

Test your program on the following phrases: Er hast ein Auto , er ist gut Fahrer. Ich bin schlecht Fahrer. Ich müsse fahren. This exercise seems to show that to devise a translator shall be a game for children. But experience shows that no one has been able to design a translator qualified enough to compete with human beings. Explain why? Answer

59 ***RNA translation***

The problem of translation is also present in genetics: the genetic material is naturally described by strings of words. In the nucleus, a DNA string must be transcribed to a messenger RNA, which in its turn is processed into intron-free material and then translated in the cytoplasm into a protein chain. These operations are a marvel of the biological machinery. But from the stand point of languages, excision of introns and translation are just the result of the replacement operator. We will see this in more detail in the next chapter.

60 ***Copying***

Another important operation with strings is to **copy** a part or **substring** of a string. The next two instructions do that:

```
s2 = s1.substring(4);
s3 = s1.substring(4,8);
```

We command in the first instruction to copy into string s2 that part of string s1 that begins at the fourth character (Warning: Java begins counting with zero). While in the second, we command to copy in s3 the part of s1 that begins at the

fourth character and ends at the eighth one excluded. A viable program that illustrates how these operators work follows:

```
//Program A60
//Copying a substring.
//Warning: Java begins conting with 0
package ejvol1;


public class Copy
{
    public static void main(String[] args)
    {
        String s1, s2, s3;
        s1 = "Selection is the key to function.";
        System.out.println("Original string = " + s1);

        // Let us copy into s2 the part of s1
        // that begins at the fourth character:
        s2 = s1.substring(4);
        System.out.println("Copied substing2= " +s2);

        // Let us copy into s3 the part of s1
        // that begins at the fourth character
        // and ends   at the eighth one excluded:
        s3 = s1.substring(4,8);
        System.out.println("Copied substing3= " + s3);
    }
}
```

61 ***Exercise*** *Run the code to see how it works. Play with modifications of the point of beginning and of that of termination to form many diverse sub strings.*

62 ***Let us look at the next program to learn how to combine what we know to COPY AND PASTE a string into a text.***

```
//Program A62 CopyPaste
// This program copies a substring
// from sOrigin and pastes it into sTarget.
```

```
package ejvol1;


public class CopyPaste
{

    public static void main(String[] args)
    {
        String sOrigin, sTarget, sMiddle, sInitial, sFinal;
        sOrigin = "abcdefghij";
        sTarget = "1234567890";
        System.out.println("Origin = " + sOrigin);
        System.out.println("Target = " + sTarget);
        // Let us copy into sMiddle
        // 4 characters of sOrigin
        // beginning with the second char:
        // initial point included: 2,
        // terminal point excluded: 6.
        sMiddle = sOrigin.substring(2, 6);
        System.out.println("sMiddle = " + sMiddle);

        //We copy into sInitial the first 3 chars  of sTarget
        sInitial = sTarget.substring(0,3);
        System.out.println("sInitial = " + sInitial);
        //We copy into sFinal the last part of sTarget
        //beginning with the third char
        sFinal = sTarget.substring(3);
        System.out.println("sFinal = " + sFinal);
        // We concatenate sInitial,sMiddle and sFinal.
        String  sOutput = sInitial+sMiddle + sFinal;
        System.out.println("output = " + sOutput);
    }
}
```

63 ***Exercise*** *Run the code and verify its fairness. Disentangle the function of each sentence. Modify the program else write one on your own to copy from "How does function appear in evolution?" the substring "function" into the appropriate place of the string " appears as a byproduct of reproduction under selection".*
Answer

64 ***Deletion.*** *Let us illustrate how to* ***delete the first 3 characters*** *of a string. Deletion is a by-product of the replacement operator: the substring to be deleted is replaced by a void substring.*

```
//Program A64 Deletion
// This program deletes from s1
// its first 3 characters.

package ejvol1;


public class Deletion
{
    public static void main(String[] args)
    {
        String s1, s2, s3;
        s1 = "Evolution = mutation + selection";
        System.out.println(s1);

        s2 = s1.substring(0,3);
        System.out.println("Substring to delete = " + s2);
        //s2 is replaced by the void string
        s3 = s1.replace(s2, "");
        System.out.println("Substring deleted: "  + s3);
    }
}
```

65 ***Exercise*** *Run the code to see what it does. Play with it to delete the first k characters of s1.*

66 ***Let us learn how to delete the last 3 characters of a string.***

We need to use a method of Java that reports the **length of a string**:

```
// The next method measures
// the number of characters of a string.
int n = s1.length();
```

A viable program that contains this method follows:

```
//Program A66
// This program deletes from s1
// the last 3 characters.

package ejvol1;


public class Length
{
    public static void main(String[] args)
    {
        String s1, s2;
        s1 = "Evolution = mutation + selection";
        System.out.println(s1);
        // The next method measures
        // the number of characters of a string.
        int n = s1.length();
        // Copy procedure
        s2 = s1.substring(0,n-3);
        System.out.println(s2);
    }
}
```

67 ***Exercise*** *Run the code to see what it does. Play with it to delete the last k characters of s1. Explain why the code works.* Answer

68 ***Exercise*** *Let l be the length of a string s. Design a code that deletes 3 characters from s beginning from the* $(l-4)$*th one.* Answer

69 *Duplication*

A very powerful mutation is **duplication**. Some people think that duplication is the door to evolution: duplicated material can be mutated while prior function is still alive in the original string. To test this belief is a challenge for our community. Let us implement a simple duplication:

```
//Program A69 Duplication
//This program duplicates the substring
//of a from points 3 to 7.

package ejvol1;
```

```
public class Duplication
{
    public static void main(String[] args)
    {
        String a, c, s1, s2;
        a = "Duplication is the key to amplification";
        System.out.println("Original string = " + a);
        //String to duplicate
        c= a.substring(3, 7);
        System.out.println("Substring to duplicate = " + c);
        //We copy  the first part of a
        s1 = a.substring(0, 7);
        System.out.println("The left part of c is = " + s1);

        //We copy  the right part of a
        s2 = a.substring(7);
        System.out.println("The Right part of a is = " + s2);

        //We concatenate the left part of c
        // to  a to the right part of c
        String Duplication = s1   + c + s2;
        System.out.println("Duplication = " + Duplication);
    }
}
```

70 ***Exercise*** *Run this program, develop and run your own variations.*

71 ***Inversion***

Inversions of portions of chromosomes correlate with sub-speciation. So, it is supposed that inversion is a powerful evolutionary agent. This belief also poses a challenge to our community. Let us learn how to do an **inversion** of a sequence:

```
//Program A71 Inversion
//This program inverts string a.

package ejvol1;
```

```
public  class Inversion
{
    public static void main(String[] args)
    {
        String a = "Inversion is important in sub-speciation";

        int Start;
        String charI;
        //We get the length of a
        int la = a.length();
        //Output
        String  d = "";
        //To invert a, we copy each character of a,
        //one by one, and paste them together
        //in reversed order:

        for(int i = 0; i < la ; i++)
        {
            Start = i;
            charI = a.substring(Start,Start + 1);
            //Char is pasted to the left
            d = charI + d;
        }
    System.out.println("Original sequence =" +  a);
    System.out.println("Inverted sequence =" +  d);
  }
}
```

72 ***Exercise*** *Run the code and verbalized its design. Design a program to invert just the substring "important" in within the string "Inversion is important in sub-speciation" while conserving the rest intact.* Answer

5.2 Structured versions

Programming is a complex task: it is very probably to make bugs but it is very hard to correct them. So, to facilitate programming and to enable re-using, a structuring approach is widely used. Let us rewrite two or three of our previous programs in structured form.

73 ***Concatenation***

```
//Program A73 Concatenation3
//Concatenation.
//Structured version.

package ejvol1;

public class Concatenation3
{
    // This method concatenates a and b
    // to produce a result that shall be invoked
    //as glue(a,b) and that is of String type.
    public static String  glue(String a, String b)
    {
        // The reserved word "return"
        // points to the output of the method glue.
        return a+b;
    }

    // This is the head of the program.
    public static void main(String[] args)
    {
        String a = "pqrst";
        String b = "123";
        // The output of glue is placed in string c
        String c = glue(a,b);
        // Report
        System.out.println("a =   " + a);
        System.out.println("b =   " + b);
        System.out.println("a + b =   " + c);
    }
}
```

74 ***Exercise*** *Run the program, study it. Modify the program to concatenate three strings. Choose one between two solutions: first, to use the method glue(a,b) twice, or second, to devise a new glue(a,b,c).* Answer

75 ***Replacement***

Let us remember that the replace operator

```
c.replace(a,b);
```

replaces a by b inside c. A structured version of the replace operator follows.

```
//Program 75 Replace5
//Replacement as a method.

package ejvol1;


public class Replace5
{
    // This method replaces string a by b
    // inside c.
    public static String
                replacement(String a, String b, String c)
    {
        return c.replace(a,b);
    }

    // This is the head of the program.
    public static void main(String[] args)
    {
        String a = "def";
        String b = "123";
        String c = "abcdefghi";
        // The method replacement is invoked
        String sOutput = replacement(a,b,c);
        // Report
        System.out.println("a =   " + a);
        System.out.println("b =   " + b);
        System.out.println("c =   " + c);
        System.out.println("Replacement done =  " + sOutput);
    }
}
```

76 ***Exercise*** *Run the program and play with your own modifications.*

77 ***Exercise*** *Design a program with two methods, glue and replacement. Method glue concatenates a to b and in its output c is replaced by d. Next, make the report into a separate method.* Answer

78 ***Redundancy***

We have defined various operations with strings and one can guess that many more can be invented. But at the same time, we can notice that some of the defined operations were built as a composition of others. So, our operations are redundant. More complicated is to select the basic operations. There are various choices and one that has proved to be very popular is to choose copy+paste, which are the text operators of any text processor.

79 ***Mutation and computing***

Our operations with strings also correspond to mutations. It is common to see in mutations a source of noise and troubles but mutations have another flavor that has been unveiled by computing science: one can make calculations with mutations, i.e., with string operators. So, we will speak of the **noise-computing duality of mutation**.

5.3 Review

The type in Java to deal with texts is *String*. We have worked out many important string manipulations: concatenation, deletion, copy+paste, inversion, replacement. These are more than enough to simulate evolution because all other manipulations could be derived from the ones studied here. Indeed, copy+paste is all we need to simulate any mutation. These are the fundamental operators of any text processor.

Chapter 6

Arrays

```
Tables are called arrays.
```

80 ***Purpose.*** *In programming languages, tables are called* ***arrays****. Tables can have one, two, three or more columns and one, two or more dimensions. We become acquainted in this chapter with one and two-dimensional arrays. A numeric array with one column may represent numerical data and so we learn to calculate their mean and variance. We also consider arrays of strings to model DNA translation.*

6.1 Numeric arrays

Numerical data can be arranged into an array to easy use of for-loops.

81 ***A linear array.*** *In the following program we use a one-dimensional array to posit the squares of the first 10 numbers (unsigned integer numbers in Java begin from zero).*

The **declaration of an array** can be made in one step but one might prefer two:

```
// Declaration of an array with integer entries
   int data[];  // data is the name.

// Assignation of memory capacity of the array:
// ten entries, beginning from zero.
// data[i] can be defined for i from zero to 9
// data[10] is not defined
  data = new int[ 10 ];
```

The whole program follows:

```
// Program 81 Array
// We introduce one-dimensional arrays.
package ejvol1;


public class Array
{
   public static void main( String args[] )
   {
    // Declaration of an array with integer entries
       int data[];  // data is the name.

    // Assignation of the capacity of the array:
    // ten entries, beginning from zero.
    // data[i] can be defined for i from zero to 9
    // data[10] is not defined
      data = new int[ 10 ];

    // Assignation of each entry
    for(int i = 0; i<=9; i=i+1)
    {
        data[i] = i*i;
    }

    // The array is written to the console.
    for(int i = 0; i < 10; i++)
        System.out.println(  i + " " + data[i] );
   }
}
```

82 ***Exercise*** *Run the code and verbalize its functioning. Modify it to calculate the cubes of the first 12 numbers.* Answer

83 ***The mean and variance.*** *Let us see how one can calculate the* ***mean and variance*** *of some data. In the next program we can see another form as an* ***array*** *can be* ***declared and initialized****:*

```
 //Declaration and assignation of the array Data
```

```
int  Data[] = {1,5,2,4,7,8,9,5,6,3,5};
```

In this instruction, we declare that Data[0]=1, Data[1]= 5, ..., Data[10]= 5. Moreover, we use the instruction

```
int n = Data.length;
```

to keep in the integer variable *n* the length of the array Data. In that way, one can modify the data at pleasure without worrying about further bookkeeping.

The whole program follows:

```
//program A83 Arrays3
//Mean and variance of some data.
package ejvol1;


public class Arrays3
{
   public  static void main( String args[] )
   {
        //Declaration and assignation of the data array
        int  Data[] = {1,5,2,4,7,8,9,5,6,3,5};
        // Report of data
        System.out.println( "Data are" );
        // To know the length of the array : Data.length
        for ( int i = 0; i < Data.length; i++ )
        {
            System.out.print( Data[ i ]);
            System.out.println(); //pass to the next datum
        }
        //Let us calculate the mean
        double sum = 0;
        int n = Data.length;
        for ( int i = 0; i < n; i++ )
            sum = sum + Data[i];
        double mean = sum / n;
        System.out.println( "Sum \t "+ sum);
        System.out.println( "Mean \t "+ mean);
        //Let us calculate the variance
        double sum2 = 0;
```

```
        for ( int i = 0; i < n; i++ )
         sum2 = sum2 + (Data[i]-mean)*(Data[i]-mean);
        double var = sum2 / n;
        System.out.println( "Variance \t "+ var);
        System.out.println( "Deviation \t "+ Math.sqrt(var));
    }
}
```

84 ***Exercise*** *Run the code and verbalize its functioning. Explain how one can make sure that all data are included in the sum, mean and variance. Modify the code to calculate the mean and variance of own data of type double.* Answer

6.2 String arrays and the genetic code

Let us use arrays to implement the **translation** procedure that is defined by the **genetic code**.

85 ***Exercise*** *a) Copy the next chain into a Word document: AUU-UUA-UUA-UAU-AUA-UAU-AUA-UAU-AUA-UAU-AUA-UAU-AUU-UAU-AUA- UAU-AUU-UAA.*

b) Use the Replace function of Word to find the corresponding amino acid chain, if the dictionary of translation is the following:

AUU : Ile (Isoleucine)
UUA: Leu (Leucine)
UAU: Tyr (Tyrosine)
AUA : Arg (Arginine)
AAU : Asn (Asparagine)
AAA: Lys (Lysine)
UUU: Phe (Phenylalanine)
UAA: Term (Chain terminating codon) Answer

86 ***Exercise*** *Type your own RNA strings with only A and U and use the replace function of a word processor to get the protein equivalents.*

87 *Translation*

To make a full **translator** of RNA messages into proteins according to the genetic code, let us use now two dimensional arrays to mimic the work done by t-RNAs and ribosomes.

The following comments help to understand the next code in Java:

1. A two dimensional table or array is addressed with two indexes, the first for the row, the second for the column. Concretely, if the array is genCode, we use genCode[2][5] to denote the entry in the second row and the fifth column. Remember that anyone index of any array always begins with zero.

2. In the genetic code we use strings. To compare strings a and d, we use the expression

```
a.equals(d)
```

If a and d are equal one to the another, **a.equals(d)** returns ”true” else it returns ”false”.

The instruction

```
if ( a.equals(d) )
    {
action
    }
```

means that if a and d are equal as strings, then the action is done else the action is ignored.

It is worth noting that the corresponding structure for **comparing numbers** *a* and *d* is as follows:

```
if ( a == d )
    {
action
    }
```

In general, each type has its own operators.

The following code translates from RNA strings to amino acid chains using a reduced code or dictionary.

Now a note in regard with bugs: programming is a difficult task, so much that one cannot expect to redact at once a program that does exactly what one wants. Errors are called bugs. But software is known through its output and only through them. So, many errors are discovered only at or after execution time.

As Elipse as NetBeans have debugging facilities that are very powerful weapons to crack bugs. A tutorial for NetBeans can be found in (IDR Solutions, [6] 2014).

Additionally, one can include in the code various sentences, whose only purpose is to trace actions: they are not comments, they are printing instructions for Java that are included to track bugs and are silenced with double back slash once the bugs are found and corrected. These tracers instructions are called **debugger lines**.

Let us also observe the variations in the printing function: we use 'println()' when we want a change of car, but we write 'print()' when we want no change of car.

```
//Program A87 GeneticCode
// A simplified genetic code
package ejvol1;

public class GeneticCode
{
   public  static void main( String args[] )
   {
        //Declaration and assignation of the array
        //There are 2 columns. Capacity unspecified.
        String  genCode[][] = { {"AUU","Ile"},
                    {"UUA","Leu"},
                    {"UAU","Tyr"},
                    {"AUA","Arg"},
                    {"AAU","Asn"},
                    {"AAA","Lys"},
                    {"UUU","Phe"},
                    {"UAA","Term"}
                   };

        //  Report of the code
        System.out.println( "The genetic code is" );
        //  One loop for the rows.
        //  To know the length of the array : genCode.length
        for ( int i = 0; i < genCode.length; i++ )
        {
            System.out.print( genCode[ i ][ 0 ]  +
                  " " +
                  genCode[ i ][ 1 ] );
            System.out.println(); //pass to the next line
```

```
    }

    //  Now we simulate the action of a ribosome.
    //  Test RNA string
    String c = "AUUUUAUUAUAUAUAUAUAU"+
             "AUAUAUAUAUAUUUAUAUAU"+
             "AUAUUUAA";

    String dash = "-";
    //  We get the length of c:
    int lc = c.length();
    //  We get the number of codons:
    int nCodons = lc / 3;
    System.out.println( "Length   = " + lc);
    System.out.println( "Number of codons = " + nCodons);
     //Easy reading of c
    String cc = c;
    //Insert a dash to separate one codon from another.
    //Begin with the right end, move to the left,
    //and terminate at t he first codon.
    for(int i = nCodons-1; i >0;  i--)
    {
        String t  = cc.substring(0, 3*i);
        //Debugger line
        //System.out.println("t = " + t);
        //Dash is inserted
        cc = t + dash +  cc.substring(3*i);
        //System.out.println("cc = " + cc);
    }




    //  We translate codon by codon.
    //  The output sequence z is initialized
    String z = "";
    int start;
    for(int i = 0; i< nCodons ; i++)
{
        //We copy the ith codon of c into a
```

```
            start = 3 * i;
            String a = c.substring(start,start+3);

            //We revise the Genetic code to know
            //the corresponding row m where a is located:
            int m = 0; //default initialization

            for( int  j = 0; j < genCode.length; j++ )
            {
                if ( a.equals(genCode[ j ][ 0 ]) )
                {
                    m = j;
                    // If the codon is located in
                    //the dictionary, exit the loop:
                    break;
                }

            }
            //Debugging line
            //System.out.println( a + " " + m);

            //We consult the dictionary in the other column
            //and synthesize z by adding the equivalent value
            z = z + "-" + genCode[m][1];

}
        z = z.substring(1);
        //Report
        System.out.println( "RNA and protein sequences:");
        System.out.println( cc + "\n" + z);
   }
}//End of program A87 GeneticCode
```

88 ***Exercise*** *Run the code and verify that it works as it is due. Compose a new input string on A and U, place it instead of c, and check out the answer.*

89 ***Structuring***

As we already knows, when the complexity of design grows, it is very helpful to structure the code into various methods. Each module or method is expected to

be understood all at once. Let us see the structured version of the previous code. As an aside, the meaning of the **static** modifier is the following: it deals with the management of memory. The assignation of memory could be forever; in that case it is declared as static. We generally work with permanent assignations. When a variable is unmodifiable, it is declared final.

90 *Let us remake previous program into a structured one.*

```
// Program 90
// A simplified genetic code
public class A90
{
// Declaration and assignation of the array
// There are 2 columns. Capacity unspecified.

static  String  genCode[][] = { {"AUU","Ile"},
             {"UUA","Leu"},
             {"UAU","Tyr"},
             {"AUA","Arg"},
             {"AAU","Asn"},
             {"AAA","Lys"},
             {"UUU","Phe"},
             {"UAA","Term"}
            };

// Report of the code

 public static void report()
 {
    System.out.println( "The genetic code is" );
    //  One loop for the rows.
    //  To know the length of the array : genCode.length
     for ( int i = 0; i < genCode.length; i++ )
     {
         System.out.print( genCode[ i ][ 0 ]  +
         " " +
         genCode[ i ][ 1 ] );
         System.out.println(); //pass to the next row
     }
```

```
 }


//  We revise the Genetic code to know
//  the corresponding row m where codon a is located:
 public static int rowNumber(String a)
 {
    //  Debugging tracer
    // System.out.println( a);

    int m = 0; //default initialization

    for( int  j = 0; j < genCode.length; j++ )
   {
     if ( a.equals(genCode[ j ][ 0 ]) )
      {
    m = j;
    // Debugging tracer
    // System.out.println( "Match found");
    //     If the codon is located in the dictionary
    //      exit the loop:
    j=genCode.length;
    }
    //   Debugging help
    //   System.out.println( a + " " + d + j +m);
}
 return m;
}


 public static void translation(String c)
 {
    // We get the length of c:
    int lc = c.length();
    //  We get the number of codons:
    int nCodons = lc / 3;
System.out.println( "Length of c = " + lc);
System.out.println( "Number of codons = " + nCodons);
//  We translate codon by codon.
```

```
//  The output sequence z is initialized
String z = " ";
int start;
for(int i = 1; i<= nCodons ; i++)
 {
  // We copy the ith codon of c into a
     start = 3 * (i-1);
     String a = c.substring(start,start+3);
     int m = rowNumber(a);
  // We consult the dictionary in the other column
  // and synthesize z by adding the equivalent value
     z = z + "-" + genCode[m][1];
   }
//We report z
System.out.println( "The protein chain is " + z);

 }

   public  static void main( String args[] )
   {
   // The code is rewritten to check functioning.
       report();
    // Assignation of test DNA string
       String c = "AUUUUAUUAUAUAUAUAUAU"+
                  "AUAUAUAUAUAUUUAUAUAU"+
                  "AUAUUUAA";
       System.out.println( "The original sequence is " + c);
       translation(c);
   }
}
```

91 ***Exercise*** *Run the code and verify that its output and function coincide with those of the previous unstructured program.*

92 ***Challenge*** *Modify the anterior code to cover the genetic code completely.*

93 ***Navigating helps.*** *One can modify a program without understanding it all. But, to understand large programs it is convenient to know how to navigate across them. The reason is that to gain an overall insight into a large program, one must*

tie ends that are far away in the text of the program. These jumping operations could become tedious. To use the help that NetBeans offers for that purpose, one can follow the next procedure:

1. *Click on the source of your program to activate it.*
2. *Look for the Navigator tab, that has a compass, and activate it.*
3. *Observe a tree with the structure of your program in the corresponding window.*
4. *Select a method you want to look at and right click on* `Go to Source`*. In the Editor Window you will see the desired method.*

Additionally, if you are reading the source and want to go to whatever element you want, put the cursor over it and right click to `Navigate, Go to Source`*. One also can put bookmarks in the left vertical bar. Click over the name of method and right click to* `Call Hierarchy`*.*

Eclipse provides at least 4 proficient helps to **navigate** through a program:

1) Choose a pivot point, an important one, in your program and in front of it right click on the most left margin of the editor and choose the option 'Add Bookmark'. Please, choose not a commentary. Once clicked, a dialog will pop up in which the theme considered in your pivot point will appear. Click OK else change the pivot point else edit on your own the name of the bookmark. Once your bookmark gets accepted, a blue marker is displayed in the most right margin of the editor window. If you drag the cursor to it and waits for its conversion to a hand, the name of your bookmark is displayed. The bookmark can be invoked by clicking on it. When several book markers are elected, one can choose among them by name and by relative position. To delete a bookmark, drag the cursor to it over the left most bar, right click, and choose remove bookmark.

2) Right-click on the most right margin of the editor window and choose $Folding \rightarrow CollapseAll$. As a result, Eclipse will hide the body of each method while titles remain visible. So, one can look the overall plan of design. One can expand and shrink at will any method: If a method is compressed, one can expand it by clicking on the corresponding plus sign at the margin. Also, one can hide the body or implementation of a method by clicking on the corresponding minus sign.

3) Eclipse allows you to visit automatically the source code of a method, whose invocation appears at any place: click in the name of the method and next press F3. You will be teletransported to the source of the requested method.

4) In the main menu, choose *window* $\rightarrow$ *Show view* $\rightarrow$ *Outline*. Then command teletransportation to a method by clicking on it. (Sometimes, it is necessary to activate the following sequence: *Window* $\rightarrow$ *Open Perspective* $\rightarrow$ *Java*).

94 ***Exercise*** *Try to understand the previous program. Please, enjoy and introvert the aforementioned navigating options and get aware of what high quality software design means. Intrigue: How can we convert this appreciation into a light to study evolution, which is a blind and mindless developer of software?*

6.3 Review

We have learned the fundamentals of one and two-dimensional arrays, whose entries could be numbers or strings. Numeric arrays represent quantitative data, whose mean and variance were calculated, while string arrays allowed us to simulate the work of ribosomes to synthesize a protein chain beginning with a RNA sequence. We also learned about the facilities to navigate across large programs.

Chapter 7

The random generator

`Generating pseudo-random numbers.`

95 ***Motivation: Java is offered here as a precious tool to make simulations, which consist in the construction of artificial worlds with exactly the characteristics one wants and with the instruments of measurements one chooses. Simulations will be used below to study evolution. Execution of methods in random order is also discussed.***

96 ***Purpose: to get acquainted with the fundamental ingredient of simulations, the random numbers generator, and with its use to generate random strings.***

7.1 Random numbers

97 ♣ ***Definition.*** *A* ***random generator*** *is a code that serves as a controllable source of random numbers, i.e., a sequence of numbers such that the probability of correctly guessing the next numbers is almost nil. Random numbers are used everywhere in simulation: to simulate behaviors and to contrast null hypotheses. In the next program we see how they are put into work.*

The fundamental instructions to work with random numbers follow. In the first sentence we declare that we have the intention of working with random numbers, so we command to bring forth the appropriate tool, kept by Java in a library called util. The second instruction declares that we turn on the generator. The third instruction declares that we will generate random integer numbers from 0 to 5.

```
import java.util.Random; //At the beginning of the program
```

```
Random r  = new Random(); // somewhere at the beginning
r.nextInt(6)   //In within a method
```

We also can generate both upper case and lower case **random chars**:

```
 //A random upper case char:
   char ch =   ( char ) ( r.nextInt( 26 ) + 65 );

//A random lower case char:
        ch =  ( char ) ( r.nextInt( 26 ) + 97 );
```

We include also the type char, which is the official Java name for a letter:

```
 char n = 'j';
```

The whole program follows:

```
//Program A97 RandomGenerator
//Generation of random numbers and chars

package ejvol1;

//The generation of random numbers
//is supported by the class Random,
//so, that class must be activated.
import java.util.Random;


public class RandomGenerator


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

```
//A sequence of random numbers is generated as follows:
//We turn on the random gebnerator:
Random r  = new Random();
//  We invoke the next random number,
// each time it is needed.
//  The parameter 6 means that the random numbers
// will be in within 0 and 5.
// Generated numbers are integers.
System.out.println( "Random number " + r.nextInt(6));
System.out.println( "Random number " + r.nextInt(6));
System.out.println( "Random number " + r.nextInt(6));
System.out.println( "Random number " + r.nextInt(6));
System.out.println( "Random number " + r.nextInt(6));
System.out.println( "Random number " + r.nextInt(6));
System.out.println( "Random number " + r.nextInt(6));
System.out.println( "Random number " + r.nextInt(6));
System.out.println( "Random number " + r.nextInt(6));
System.out.println( "Random number " + r.nextInt(6));


// For a large sequence, a loop is necessary
System.out.print("Simulation of a die: \n");
for ( int i = 1; i <= 20; i++ )
{
    // To simulate a die, we add 1.
    // Observe that the symbol +
    // is used in two very different senses:
    System.out.print( 1 + r.nextInt( 6 ) + "  " );
}
System.out.println();

//This is type char for letters
//Strings are surrounded b two marks,
//chars by one.
char c = 'j';
System.out.println("This is a char = " + c);

// ALPHABET = upper case chars(ascii code):
System.out.print("ALPHABET = " );
for ( int i = 65; i <= 90; i++ )
```

```
        {
            char ch = ( char) (i);
            System.out.print(ch + " ");
        }

        System.out.println( );


        //Alphabet
        //Lower case chars (ascii code):
        // Alphabet = upper case chars(ascii code):
        System.out.print("Alphabet = " );

        for ( int i = 97; i <= 122; i++ )
        {
            char ch = ( char) (i);
            System.out.print(ch + " ");
        }
        System.out.println();

        //A random upper case char:
        char ch =   ( char ) ( r.nextInt( 26 ) + 65 );
        System.out.println("Random Upper char = " +   ch );

        //A random lower case char:
        ch =  ( char ) (  r.nextInt( 26 ) + 97 );
        System.out.println("Random lower char = " + ch );
    }
} //End of Program A97 RandomGenerator
```

98 ***Exercise*** *Run and interpret the code. To test your understanding, compose a program to generate 20 random upper case chars followed by other 20 random lower case chars.* *Answer*

99 ***Challenge*** *Play with the following instructions to verify their function:*

Random r = new Random (**seedValue**); (always initializes the generator in the same form)

r.**nextLong();** (generates a random integer of type long, which is more powerful than type int.)

r.**nextFloat();** and r.**nextDouble();** (generate a decimal number between 0 and 1 with a uniform distribution.)

r.**nextGaussian();** (generates a random decimal number with a Z distribution.)

7.2 Random DNA

100 ***Random DNA generator.*** *It is now straightforward to simulate a* ***random DNA string****. The alphabet of DNA is ATCG.*

```
//Program A100 RandomDNA
//Random  DNA string.
package ejvol1;


import java.util.Random;

public class RandomDNA
{
   public static void main( String args[] )
   {
        //We turn on the random generator:
        Random r  = new Random();
        // Definition of the alphabet
        char Alphabet[] = {'A','T','C','G'};

        // Generation of a DNA string 60 bases long
        for(int i= 0; i< 60; i++)
        {
            char ch =   Alphabet[ r.nextInt( 4 )];
            System.out.print (  ch);
        }
        System.out.println();
   }
}
```

101 ***Exercise*** *Run the code. Verbalize its functioning. Modify it to generate a random RNA string 25 chars long.* Answer

102 ***Challenge.*** *Design a program to generate a protein random chain formed by the 20 amino acids that appear in the genetic code. Choose between an encoding of one amino acid- one letter else one amino acid - three letters.*

103 ***Hard challenge.*** *We have been working with linear strings. In nature, some genetic sequences are circular. Find the best way to encode circular sequences into a linear one in order to generalize all defined string functions and methods to cover both cases, linear and close. Make a suitable program to illustrate your ideas.*

7.3 Positional effects

Natural evolution proficiently enables transposition of DNA fragments. In that vein, jumping genes in maize were announced to the world as a source of variability but next they appeared as possible cause of cancer in humans. In some cases, positional effects correspond to change in the order of execution of genetic instructions. To ape positional changes, we will study in this section how one can automatically program random positional effects of methods.

104 ***Permutation of methods***

Let us do an exercise in which we can vary the order of execution of methods. Our methods generate numbers. Each one of our methods prints just one number from 0 to 9. By calling in line any group of four methods, with repetitions allowed, one can eventually write down all numbers from 0 to 9999. In the next code we write the number 345:

```
//Program A104 RandomMethods
//Permutation of order of execution
//of methods.
package ejvol1;


public class RandomMethods
{

    public static void zero()
    {
     System.out.print(0);
```

```
}

public static void one()
{
 System.out.print(1);
}

public static void two()
{
 System.out.print(2);
}

public static void three()
{
 System.out.print(3);
}

public static void four()
{
 System.out.print(4);
}

public static void five()
{
 System.out.print(5);
}

public static void six()
{
 System.out.print(6);
}

public static void seven()
{
 System.out.print(7);
}

public static void eight()
{
 System.out.print(8);
```

```
    }

    public static void nine()
    {
     System.out.print(9);
    }


    public static void main(String[] args)
    {
        // Let us write 345:
        three();
        four();
        five();
        System.out.println();
    }
}//End of Program A104 RandomMethods
```

It is clear than apart from 345, one would eventually need to generate all numbers from zero to, say, 999. With what we know, achieving that would be a stupendous source of fatigue. So, let us learn the form as Java converts that terrible problem in an elementary task.

105 *The switch structure*

One uses the switch structure when one is presented with a multiple selection problem. The switch structure + parameterized methods are all we need to solve the problem of permutation of methods, as the next program shows. In it we use the switch structure as follows:

```
// switch multiple selection structure
 public static void permutator(int k)
 {
       switch (k)
{
case 0: zero(); break;
case 1: one(); break;
case 2: two(); break;
case 3: three(); break;
```

```
case 4: four(); break;
case 5: five(); break;
case 6: six(); break;
case 7: seven() ; break;
case 8: eight(); break;
case 9: nine(); break;
}
 }
```

This structure is executed by reading the value of k and executing the corresponding case, if any, and the structure is abandoned in the next step. The complete program follows:

```
//Program A105 RandomMethods2
//How to   permutate methods.
//Use of the switch structure.
package ejvol1;


import java.util.Random;


public class RandomMethods2
{
    //Turn on of the random generator
    static Random r  = new Random();

    public static void zero()
    {
        System.out.print(0);
    }

    public static void one()
    {
        System.out.print(1);
    }

    public static void two()
```

```
{
    System.out.print(2);
}

public static void three()
{
    System.out.print(3);
}

public static void four()
{
    System.out.print(4);
}

public static void five()
{
    System.out.print(5);
}

public static void six()
{
    System.out.print(6);
}

public static void seven()
{
    System.out.print(7);
}

public static void eight()
{
    System.out.print(8);
}

public static void nine()
{
    System.out.print(9);
}

// switch multiple selection structure
```

```
    public static void permutator(int k)
    {
        switch (k)
        {
            case 0: zero(); break;
            case 1: one(); break;
            case 2: two(); break;
            case 3: three(); break;
            case 4: four(); break;
            case 5: five(); break;
            case 6: six(); break;
            case 7: seven() ; break;
            case 8: eight(); break;
            case 9: nine(); break;
        }
    }


    public static   void main(String[] args)
    {
        System.out.println("Test 1:");
        for(int k=0; k<=9; k++) permutator(k);
        System.out.println();
        System.out.println("Test 2:");
        for(int k=0; k<=9; k++) permutator(9-k);
        System.out.println();
        System.out.println("Random digits:");
        for(int k=0; k<=9; k++) permutator(r.nextInt(10));
        System.out.println();
    }
}//End of Program A105 RandomMethods2
```

106 ***Exercise*** *Run the code and verify that this program is an automatic random permutator of methods.*

7.4 Review

The fundamental ingredient of simulations, the random numbers generator, is now at our disposal to generate numbers and strings, including DNA and RNA ran-

dom chains. We also managed to produce invocation of methods in random order, simulating in that way positional effects in genetics.

Chapter 8

The challenge of complexity

```
Complexity is what a very good model lacks to be realistic.
```

107 ***Motivation: A very important problem of biology is to study the origin and evolution of complexity. We consider this theme so serious and challenging that we invite the reader to choose it as a profession for a life.***

108 ***Purpose: Complex as the theme could be, we begin its study right here with a try to explain complexity as the result of a sequence of contingency events that happen at random in linear order. Next, we pass to consider complexity as the result of random recombination of strings. We end with a brief discussion.***

109 ***What is complexity?***

To consider rigorous definitions of complexity would take us too far away from our purpose of learning Java for Biologists. So, let us consider a parable: complexity is similar to richness. Let us investigate whether or not one can get rich as a result of betting, in which one can lose or gain the same insignificant amount with equal probability. One may become rich if one recurrently wins more times than those that lose. If we encode a winning event as 1 and a losing event as 0, one is getting rich in a case like 1101110011111001101111.

When the same event comes a number of successive times such as 111111111, we say that we have a **run** of, say, winners. Our previous run of winners is 9 units long. Simulation helps us to capture how easy or hard it could be to have large runs. This is studied in the next program, which introduces for the first time the control instruction **while**, whose syntax obeys the following scheme:

```
while (this conditions holds)
  {
    do this and that
  }
```

A while loop must contain a form of breaking the required condition to loop. In the next program, that requirement is implemented so: we define a variable stop to be zero at the entrance. Next, we define that the run must be of ones. Thus, the first zero breaks the run and the variable stop acquires the value one and the loop is terminated.

The whole program follows:

```
//Program A109 Runs
/* We study random runs
 of a distinguished face of a coin.
  */
package ejvol1;


import java.util.Random;

public class Runs
{
    //Turning on of the random generator
    //This declaration is prior to any method,
    //it is global definition and covers
    //All methods.
    static Random r  = new Random();

    public static void main(String[] args)
    {
        // Number of trials
        int trials = 10;
        System.out.println( "Length of runs");
        for(int i = 1; i<= trials; i++)
        {
            // l is the landed face of a coin;
```

```
            int l;
            int runLength = 0;
            // stop = 1 says when a run gets   broken.
            int stop = 0;

            while (stop == 0)
            {
                // a coin is thrown
                l = r.nextInt(2);
                //System.out.println( l);
                // Our runs must be of ones.
                if (l==1)
                {
                    runLength = runLength + 1;

                }
                else stop = 1; //break the loop
            }
            //System.out.println( );
            System.out.println("   " +  i + "\t" + runLength);
            }
        }
}//End of Program A109 Runs
```

110 ***Exercise*** *Run, play and decipher the code. Draw your conclusions about the probability of getting long runs as a function of the number of trials.* Answer

111 ***Purposes and blindness***

Our next purpose is to get data about the minimum number of trials to get a run k events long. This is seen in the next variation to previous program.

```
//Program A111 Runs2
/* We study runs at random.
 We gauge the   number of trials
 that are necessary  to get for the first
 time  a run k units long, for varying k.
  */
package ejvol1;
```

```
import java.util.Random;

public class Runs2
{
    //Turning on of the random generator
    //Global declaration
    static Random r  = new Random();

    //This method throws coins
    //and returns the length of the first run.
    //That length might be a very large number,
    //so the output is encoded as long,
    //which is an integer of high capacity.
    private static long run()
    {
        // l is the landed face of a coin;
      int l;
      int runLength = 0;
      // stop=1 says when a run gets   broken.
      int stop = 0;

      while (stop == 0)
      {
            l = r.nextInt(2);
            // Our runs must be of ones.
            if (l==1)
            {
                runLength = runLength + 1;
                //System.out.print( l);
            }
            else stop = 1;
      }
      //This is the report of the method
      return runLength;
    }


    //This method reports the number of
    //necessary trials to get for the first time
    //a run of the asked length.
```

```
private static long tryAndTry(int askedLength)
{
    long runLength = 0;
    long count =0;
     while (runLength < askedLength )
    {
        count = count+1;
        runLength = run() ;
    }
    return  count;
}

// The askedLength is varied
public static void main(String[] args)
{
    int rounds = 20;
    for(int j = 1; j<= rounds; j++)
    {
        // We count the number of trials
        // that are necessary to get a run
        // askedLength units long.
        int askedLength = j;

        long count = tryAndTry(askedLength);
        System.out.println(askedLength  + "\t " +count);
    }
   }
}//End of Program A111 Runs2
```

112 ***Exercise*** *Run the code and decipher it. Make a graphic of the output. Pursue a regression analysis to know whether or not the required number of trials to get a run k units long is an exponential function in the input k.* Answer

The results of the last exercise demand to explain complexity with more than naive suppositions. In particular, the mechanism of small transactions that can be won else lost with equal probability is not recommended to explain it. At this moment in time, one must think of other proposals. The next candidates to explain **complexity** are **natural selection**, once evolution has been enabled, and **self organizing evolvable systems**, to form complexity ab initio. We will implement in our course various simulations of evolution but **self organization** will not be touched at.

We have expressed our reticence to accept that one can get rich by betting insignificant amounts or, equivalently, that complexity could be explained as a result of tiny reorganizations that happen at random in linear or sequential order. But our betting mechanism could be a correct explanation for an intriguing fact of nature: the existence of cosmic rays. They are important for us because they come with tremendously high kinetic energy, which is similar to having high richness.

113 *Cosmic rays*

Runs allow us to imagine a mechanism to explain the origin of **cosmic rays**. In effect, a cosmic ray is, for instance, a proton that comes with a **very high kinetic energy**. How did that proton acquire its energy? It is not at all clear why, because many complexities are at hand. One proposal is that in the cosmic space the original proton traveled at low speed but because of collisions with photons of low energy, the proton acquired energy bit by bit, i.e., quantum by quantum.

From the fact that we can see stars continuously, we know that there is a permanent flux of photons from every star. In general, photons abide in very great quantities everywhere. On the other hand, charged particles are too few in comparisons with photons. So, one might imagine that they interact meanly with photons. So, we can imagine that any proton is alone in a sea of photons, with which it interacts through collisions. It is not excluded that the net result is that some protons may go over a run of happy events in which they acquire more and more energy.

So, suppose we have a run of the type 1111111111. This run simulates a proton with 10 quanta of kinetic energy. Suppose now that a photon of very low energy comes and that it interacts with the proton. The result of the interaction could be that the proton gains one quantum of energy else loses one. To study what could happen, we design the next program, which reports the maximum number of quanta of energy per generation in a given population of protons. In each generation, every proton suffers a shock with a photon, which can either lend or take out one quantum of energy.

```
//Program A113 CosmicRays
/* We study a population of protons
    in the cosmic space
    that interacts with photons  by collisions.
    From a collision, a proton can gain
    else lose one quantum of energy.
    We are interested in knowing
    the trend of elongation of runs,
```

```
    i.e. of high energy,
    which might be similar to high complexity. */

package ejvol1;



import java.util.Random;

public class CosmicRays
{
    // Pre-declaration of global variables.
    // They are used all throughout the whole class.

    // Protons are numbered and
    // their energy is  kept in the array
    // energy[].
    // The number of protons must be
    // less than limit

    static int limit = 2000;
    static int energy[ ];//Predeclaration
    // The population has
    // a constant number of protons:
    static int nProtons= 100 ;

    static int rounds ;

    // We keep track of the minimum and
    // maximum energy attained in the population
    static int ReportMin[], ReportMax[];


    // Turning on of the random generator
    static Random r  = new Random();

    /* We generate nProtons protons
        with no energy */
    private static void Initialization( )
```

```
{
    //Formal declaration of our arrays.
    energy= new int[limit];
    ReportMin = new int[100001];
    ReportMax = new int[100001];
    for(int i = 1; i<= nProtons; i++)
    {
        ReportMin[i] = 0;
        ReportMax[i] = 0;
    }

    //System.out.println("ORIGINAL POPULATION");
    for(int i = 1; i<= nProtons; i++)
    {
        //Initialization  in zero,
        energy[i]=0;
    }
}


//For each generation, the min and max are recorded
private static void Observation(int Gen)
{
    int m, Min, Max;

    Max = 0;
    Min = 1000000000;
    for(int j = 0; j< nProtons; j++)
    {
        m =  energy[j];
        /*System.out.println(  "The proton  "
                + j + " has energy " + m); */
        if (m > Max)  Max = m;
        if (m < Min)   Min = m;
    }
    /* System.out.println(  "Generation " + Gen
     + " minEnergy = " + Min + " maxEnergy = " + Max); */
    ReportMin[Gen] = Min;
    ReportMax[Gen] = Max;
}
```

```
// This method  executes collisions
// among a finite number of protons
// and abounding photons.
private  static  void  dynamics(int gen)
{
    for(int i = 0; i< nProtons; i++)
    {
        // proton i collides
        // with a photon.
        // The proton can gain
        // a quantum of energy
        // with probability 0.5
        // and can lose a quantum, if any,
        // with equal probability.

        int j = r.nextInt(2);
        if (j==1)
           energy[i] = energy[i]+1;
        else
            energy[i] = Math.max( energy[i]-1,0);
        Observation(gen);
    }
}



//For the given generation, min and max numbers of
//energy quanta among all protons is reported.
public static void finalreport()
{
    System.out.println( "GEN, MIN and MAX VALUES");
    for(int Gen = 0; Gen < rounds; Gen++)
        System.out.println(Gen +"\t" +  ReportMin[Gen]
                    +"\t" +  ReportMax[Gen]);
}


//Initialization + dynamics
public static void main(String[] args)
```

```
    {
        // We seed a population of protons
        Initialization();
        // A round is a process in which
        // every proton collides once.
         rounds = 20000 ;
        for(int i = 0; i < rounds; i++)
        {
            int gen = i;
            dynamics(gen);
        }
            finalreport();
    }
}//Program A113 CosmicRays
```

114 ***Exercise*** *Run and decipher the code. Manage to know what happens after 10000 and 20000 generations. Draw your conclusions about the proposed mechanism to explain why there are cosmic rays with exceedingly high energy.* Answer

8.1 Recombinant collisions

These are collisions among DNA molecules that produce exchange of information.

115 ***Motivation***

If we try to translate the suggested process of generation of cosmic rays to biology, we would say that the molecular complexity of informative molecules, DNA and RNA, could have been gained through collisions, in which molecules interchange portions as in recombination. The result of a collision could be to enlarge some strings and to shorten others and if we allow ourselves to take complexity and length of a string as synonyms, we are proposing a model to explain complexity.

Our view is worth a criticism: a cosmic ray cannot radiate in isolation, so it can travel during years and nothing happens to it until it finds a photon, but our informative molecules always can be destroyed by excision or by contamination because of reactions with undue molecules along undue branches.

116 ***Implementation***

Let us implement a simulation in which strings collide one with another, and as a result of the collision they undergo **recombination**. The result of a recombinant

shock could be to enlarge some strings and to shorten others. So, it is not excluded that some strings could grow and grow forever.

In the next program we need to convert a **char to a string**, an operation that is executed by the instruction:

```
// Char c is converted into a string s
  String s = ""+c;
```

The program follows:

```
//Program A116 Chains
/* We study a population of strings
    that interacts by collisions
    that produce recombination.
    We are interesting in knowing
    the trend of elongation of strings.
    The program reports tha max (min) length
    of strings for each given generation.
 */

package ejvol1;



import java.util.Random;

public class Chains
{
    //Pre-declaration of global variables.
    //They are used all throughout the whole class.

    //Number of generations to run the experiment
    static int nGen = 20000;

    //Individuals are kept in the array
    //Individual[]. It is an array of strings.
    //The number of individuals must be
```

```
//less than limit

static int limit = 2000;
static String Individual[ ];//Declaration
// We consider a population
// with a constant number of individuals:
static int nIndiv = 100 ;


// We keep track of the minumal and maximal  length
// attained in the population
static int ReportMin[], ReportMax[];


//Turning on of the random generator
static Random r  = new Random();

//This method generates random chars
private  static  Character  randomChar()
{
    return  ( char ) ( r.nextInt( 26 ) + 65  );
}


/* We generate nIndiv individuals (strings)
with a random number of letters,
less than 10. Sequences are
completely random */
private static void Initialization( )
{

    //Formal declaration of our arrays.
    Individual= new String[limit];
    ReportMin = new int[nGen];
    ReportMax = new int[nGen];
    for(int i = 0; i< nIndiv; i++)
    {
        ReportMin[i] = 0;
        ReportMax[i] = 0;
    }
```

```
        //System.out.println("ORIGINAL POPULATION");
        for(int i = 0; i < nIndiv; i++)
        {
            // Initialization  in blank.
            Individual[i]="";

            // We generate a random number
            // in within 1 and 9.
            int number = r.nextInt(9)+1;

            //An individual is assembled char by char
            for(int j = 0; j < number; j++)
            {
                // We generate a random char
                Character c =  randomChar();
                //System.out.println( "char = " + c);
                // Char c is converted into a string s
                String s = "" + c;
                // s is concatenated to Individual[i]
                Individual[i] = Individual[i]+s;
            }
           /* System.out.println("Individual " +  i);
            System.out.println( Individual[i]);*/
        }
    }

    // This method  looks for a partner
    // different than l.
    private  static  int  chooseInd(int l)
    {

// done = 1 is a flag that says that
// we have found a different partner:
int done = 0;
        int number=l;
        {
            while (done == 0)//partner has not been found
            {
                number = r.nextInt(nIndiv);
```

```
                if (number ==   l)
                      done = 0; //Continue searching
                else done = 1; //break
            }
            return number;
}
    }

    //A collision produces recombination
    private  static   void collision(int l, int k)
    {
        String s = Individual[l];
        String t = Individual[k];
        // A collision corresponds to
        // recombination.
        // We divide the strings in two
        // and recombine pieces:
        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);
            //System.out.println( s + " " + s1 + " " + s2);
        }

        String  t1= t;
        String  t2 = "";
        int n= t.length();
        if (n>2)
        { int n1 = r.nextInt(n-1)+2 ;
            t1= t.substring(0,n1);
            t2= t.substring( n1);
            //System.out.println( t + " " + t1 + " " + t2);
        }
        //Recombination
        Individual[l]=s1.concat(t2);
        Individual[k]=t1.concat(s2);
```

```
}

//Min and max lengths are recorded
private static void Observation(int Gen)
{
    int m, Min, Max;

    Max = 0;
    Min = 32000;
    for(int j = 0; j < nIndiv; j++)
    {
        m =  Individual[j].length();
        /*System.out.println(  "The Individual  "
           + Individual(j) + " has length " + m); */
        if (m > Max)  Max = m;
        if (m < Min)   Min = m;
    }
    /*System.out.println(  "Generation " + Gen
     + " MinLength = " + Min + " MaxLength = " + Max);*/
    ReportMin[Gen] = Min;
    ReportMax[Gen] = Max;
}

//This method  executes collisions among chains
private  static  void  dynamics(int gen)
{
    for(int i = 0; i < nIndiv; i++)
    {
        //individual i collides
        // with another one
        // chosen at random.


        int l = i;
        int k = chooseInd(l);

        String s = Individual[i];
        String t = Individual[k];
        //System.out.println( s + "\t " + t);
```

```
            collision(l,k);
            Observation(gen);
        }
    }


    //Min and max lengths are reported
    public static void finalreport()
    {
        System.out.println( "GEN, MIN and MAX VALUES");
        for(int gen = 0; gen < nGen; gen++)
            System.out.println(gen +"\t" +  ReportMin[gen]
                        +"\t" +  ReportMax[gen]);
    }


    //Initialization and dynamics
    public static void main(String[] args)
    {
        // We seed a population
        Initialization();
        for(int gen = 0; gen < nGen; gen++)
        {
            dynamics(gen);
        }
        finalreport();
    }
}//End of Program A116 Chains
```

117 ***Exercise*** *Run, decipher and play with the code. Express your opinion about the possibilities of this model to explain biological complexity.* *Answer*

118 ***Complexity as a challenge***

In hindsight, we tried to repeat the success of our cosmic ray model and we proposed that the natural version of a collision between DNA and RNA molecules could be recombination. But we failed to share with the success of physics. The reason could be that our type of recombination is powerful to enlarge but it is also powerful to divide.

119 ***A birth-death process***

We have been simulating one of the most popular games in all cultures: to bet on the output of a coin. In rigorous terms, we have simulated a **birth-death process**, which is a stochastic process of fundamental importance. A birth corresponds to a winning event and a death to a losing one.

8.2 Conclusions

We have avoided to work with a formal definition of complexity, instead we formulated a parable that can be grasped by children that are 3 years old: to get complex is similar to get rich. How can one get rich? We study the proposal that to become rich one can bet on small amounts. We rejected this proposal. Why? Because our life is too short for having enough trials but if one were as a proton in cosmic space, which has plenty of time, then one cannot deny that a proton could eventually become a cosmic ray, replenished of high energy.

Let us say it again but in technical jargon: we advice to reject the proposal of explaining complexity as the result of a birth-death process that gathers simplicity but that can lose it. What we have in mind is that a stochastic model that is usually seen as higher mathematics is no more than an ordinary exercise in simulation.

We also considered the possibility that complexity could spring from recombination, but we were incapable to unveil a positive effect. Nevertheless, recombination could be important for gaining other folds of complexity apart from that that is similar to getting a lot of money.

On the other hand, it is amazing that with so simple simulations we found the force to express ourselves with a loud voice in regard with such a terrible problem as the origin of complexity. This shows once and for all times that *simulation + clear, intuitive and deep ideas* conform weapons that are worth the most sincere respect in science.

Chapter 9

Semantics

```
Meaning is to a language what function is to biology.
```

120 ***Motivation and purpose.*** *A* ***word*** *in a human language is a string of phonemes or chars. Words have a* ***meaning****, i.e., they refer to an object or action in the world or in the human culture. Because of memory and computing restrictions, not every string has a meaning. The meaning of a word is similar to the* ***function*** *of a gene in genetics. The study of meanings is done by* ***semantics****.*

The study of biological semantics is very complex and in vain one would separate it from the study of semantics of human languages. As a matter of fact, if the texts of population genetics (Crow and Kimura, 1970; Nagylaki 1992) are difficult to understand for somebody, advice him or her to learn its concepts from linguistics, say, from (Nettle, 1999).

Our purpose in this chapter is to gauge the semantic capacity of a human language.

9.1 From strings to words

121 ***Let us generate strings 4 chars long:***

```
//Program A121 Chars
//Strings 4 char long
package ejvol1;


import java.util.Random;

public class Chars
```

```
{
   public static void main( String args[] )
   {
        //We turn on the random generator:
        Random r  = new Random();
        for(int j= 0; j<20; j++)
        {
            for(int i= 0; i<4; i++)
            {
                //Random lower char
                char ch =   ( char ) ( r.nextInt( 26 ) + 97 );
                System.out.print (ch);
            }
            System.out.println();
        }
   }
}
```

122 ***Exercise*** *Run the code and count the proportion of English words generated by this procedure. If needed, you can copy your words to the clipboard and next paste it to a document of Word in Windows or of Writer in Open Office, where you can test your random strings with the orthographic facility to see which words have a meaning in English. To estimate the proportion of words 4 chars long that have a meaning in English, do this exercise for no less than 2000 randomly generated strings. Explain why a simulation is preferable to a complete generation of all strings.* Answer

123 ***Nonsense***

The previous exercise is crucial: it allows us to conclude that most random strings are senseless in English.

124 ***Challenge*** *Verify that most random strings are senseless in every one language. If necessary, install orthographic correctors for French, Spanish, German, Italian or whatever.*

125 ***Challenge*** *We are trying to extract general rules from an experiment with strings 4 characters long. And what about the shorter or larger strings? Did you notice that many languages have a gap between lengths 8 and 10? How can you verify that? How can you explain that? Please, make the necessary arrangements*

in the previous code and carry suitable experiments to answer posed questions and specially to strengthen the desired conclusion: ***most strings are senseless in anyone language****.*

126 *Prediction*

Let us formulate now a prediction: if any word is submitted to a **mutation process** of random substitution character per character, the resultant series of strings would lose meaning very soon. To test that prediction, let us develop the appropriate code. We need to develop a method to make punctual substitutions. It is found in the next code:

```
//Program A126 Nonsense
//Random substitutions in tandem
package ejvol1;


import java.util.Random;

public class Nonsense
{
   public static void main( String args[] )
   {
        //We turn on the random generator:
        Random r  = new Random();
        //Original string
        String s = "meaning";

        int l = s.length();
        System.out.println (s + " length = " + l);

        // We want 30 substitutions in tandem
        for (int i = 0; i<3000; i++)
        {
            // We choose at random a place
            // to make a substitution.
            int place = r.nextInt( l );
            // We choose a char at random
            char ch =   ( char ) ( r.nextInt( 26 ) + 97 );
            // We make a substitution
```

```
                s = s.substring(0, place) + ch + s.substring(place+1);
                System.out.println (s);
            }
        }
    }
```

127 ***Exercise*** *Run the code and verify the prediction for the chosen word.* *Answer*

128 ***Exercise*** *Make enough experiments with this code to conclude that point mutations in tandem rapidly end in nonsense. Is the length of the original string an important parameter for the discussion of this assertion? Please, make experiments changing the length of the original string. Experiment also with phrases and paragraphs, which you can directly copy from any text into the original string. Notice and report also those happy circumstances in which some substitutions change the meaning of a phrase and produce another phrase with another meaning.* *Answer*

129 ***Various challenges.***

1. Scan some mutational processes in various languages to witness the challenge of the nonsense sea. Prove that there are examples in which mutation is more deleterious in large words that in short ones. Prove that there are examples pointing to the contrary. Explain.

2. Are meanings in the DNA language grouped in small islands surrounded by an infinite sea of nonsense? Let us call this decision problem as the Nonsense Sea Problem of genetics. Discuss the next proposal: even if now meanings are grouped in small islands in an infinite sea of nonsense, this is only the final state of the ordinary evolution of life dominated by a tendency toward specialization of meanings but that began with a very coarse semantics in which almost anything had a meaning. Let us refer to this idea as the Pangaea of semantics. Did biological semantics evolve from a Pangaea to an archipelago of small islands scattered through an infinite sea of nonsense?

3. Define the family of **semantics** and prove that the relation "finer than" orders them. More clearly: Define semantics accurately over the space of strings. Define the relation "finer than" among the class of semantics to differentiate semantics that are more restricting than others. How is related the finesse of a semantics with the expected evolutionary rate of strings? What is a Pangaea, a platform, an island, an isolated island in a semantic sense?. Give examples helping to understand what could be an evolution under the Pangaea Hypothesis.

4. Let s1 and s2 be two strings. a) Prove that d(s1,s2) = number of places where s1 differs from s2, defines a **metric** for short random DNA strings of equal length.

b) Redefine a metric for the complete set of DNA strings. Recall that DNA strands tend to match one another at those places where they are complementary. Use this fact as a cue to make a sound definition of the distance function.

5. The modern works in **DNA computing** account for the creative role of mutation as distinct to that captured by the Theory of Information in which mutation is considered as noisy. Prove that in genetics these two concepts are complementary, and we speak of the noise and computation duality of mutation. Shall the importance of each factor bounce forth and back in any successful application of the evolutionary theory?

6. Is the freedom given by the third codon of the genetic code used to improve semantic self-isolation of each genome? Otherwise DNA would become a mesh hardening cell division, transcription and DNA duplication.

7. Prove that languages must comply with the pragmatic condition of representing reality and it is in that way that they acquire semantics. Therefore, if languages have a sea of nonsense, the reality they describe must have a deeper nonsense sea. What does this mean?

130 ***Exercise*** *From all what we have said, one could expect that languages evolve very slowly. But that is false. In the Papua Island languages evolve just in front of own eyes. How can you reconcile the sea of nonsense with the exceedingly rapid evolution of languages?* Answer

131 ***Challenge*** *One application of the nonsense sea is that when one commits an error of typing, an automatic orthographic facility may suggest a list containing the correction with almost probability one. Design an orthographic corrector with a dictionary with 20 words and 5 clusters. Let us remember that to compare two strings a and b, one uses the following structure:*

```
if (a.equals(b))

  {do this; }

else { do that;}
```

132 ***The inheritance***

Human languages represent an inheritance from incalculable value. The reason is that they solve in a marvelous way the compromise between memory constraints and expressive functionality of the language as a fair representation of a complex

reality. We have in the actuality some various thousands of languages but they are menaced by the trend in globalization observed all around the world.

133 ***Exercise*** *Try to connect our aforementioned linguistic preoccupations with the problem of the evolution of DNA. Specifically: must we expect a sea of nonsense in the semantics of life .* *Answer*

9.2 Conclusions

In many occasions one can detect and correctly correct various types of linguistic errors in a text. Why? Because mutations in the strings that encode meanings rapidly lead to nonsense. That is why we said: a sea of nonsense surrounds meanings. This proposition was verified for English and we have enough confidence to predict that the same must happen as in anyone language as in biology. That seems generally true, but an exception is known: t-RNAs.

Chapter 10

Evolution on letter-zymes

```
Letter-zymes:  strings that are at the same time phenotypes
and genotypes.
```

134 ***Purpose: In this chapter we simulate evolution in the simplest of all worlds, which is composed of letter-zymes: strings that are at the same time phenotypes and genotypes. Moreover, the enzymatic activity to make mutations is giving to oblivion. In that way, we can pay attention to the evolutionary process in abstracto.***

10.1 Evolution

With the developed tools, we can compose our first simulation of evolution. At this moment in time, this simulation is just a game to test our programming skills: the apparent purpose is to devise an evolutionary process that renders a stable population. The considered population consists of strings in which its members need to feed to reproduce and grow but have parasites that cut part of their tails. There is also a hand-given selection against excessive growing. We also simulate an observer that wants to know what happens to the minimum and maximum lengths of the strings.

We will simulate **artificial selection**, as that made by a farmer that ex-profeso wants caves with intermediate weight and kills or sells all the others. By contrast, **natural selection** appears as an indirect or **emergent result** of operations that are not directly tied to reproduction, say, from the ability to distinguish mature fruits from immature ones. Our next simulation is of artificial selection.

135 *Initialization*

The complete code of our simulation could look like a nightmare, so let us introduce it in two or three steps. The first one is just the initialization

```
//Program A135 Letterzymes
//Evolution with letterzymes step 1:
//initialization of  a population.
package ejvol1;



import java.util.Random;

public class Letterzymes
{
    // Declaration of global variables.
    // They are used all throughout the whole class.

    // Individual  is an array of strings.
    static String Individual[ ];//Declaration
    static int NIndiv ; //declaration
    static String PatternParasite, PatternFood;
    // Order  is an array of integers.
    static int Order[];
    static String b;
    static int Gen;
    static int ReportMin[], ReportMax[];


    // Turn on of the random generator
    static Random r  = new Random();

    // This method generates random upper chars
    private  static  Character  randomChar()
    {
        return  ( char ) ( r.nextInt( 26 ) + 65  );
    }
```

```
   /* We generate NIndiv individuals (strings)
ten characters long.
Sequences are completely random */
   private static void Initialization( )
   {
       //Formal declaration of our array.
       Individual = new String[100];
       for(int i = 0; i< NIndiv; i++)
       {
           //Initialization  in blank.
           Individual[i]="";

           //An individual is assembled char by char
           for(int j = 0; j< 10; j++)
           {
               //We generate a random char
               Character c =  randomChar();
               //System.out.println( "char = " + c);
               //Char c is converted into a string s
               String s = ""+c;
               //s is concatenated to Individual[i]
               Individual[i] = Individual[i]+s;
           }
           System.out.println("Individual " +  i);
           System.out.println( Individual[i]);
       }

       //We generate a parasite pattern
       PatternParasite = "PARASITE";
       //We generate a food pattern
       PatternFood = "ALGAE";
   }


   public static void main(String[] args)
   {
       //The population has NIndiv members, NIndiv >10
       NIndiv = 15;
       Individual= new String[NIndiv];
       Initialization( );
```

```
    }
}
```

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

137 ***The heart***

The heart of the simulation is contained in the next method. Let us observe how natural it looks like.

```
//This is the heart of the process
private static void Dynamics()
{
    //All individuals feed
    for(int i = 0; i< NIndiv; i++)
    {
        //The string Algae is digested and inserted
        //into the individual[i] char by char
        /*System.out.println( "The hungry individual "
                    + i + " is " + Individual[i]); */
        feeding(i);
        /*System.out.println(  "The full individual "
             + i + " is " + Individual(i)); */
        //All individuals are tested by the parasite,
        //which corrodes any  large substring
        //similar to it located at any end.
        parasitation(i);
        /*System.out.println(  "Pos-parasitation individual "
             + i + " is " + Individual(i));*/
        //Very long chains are attacked recurrently,
        //the same as very large fishes in the sea.
        Purge(i);
        /*System.out.println(  "The purged  individual "
             + i + " is " + Individual(i)); */
    }
    //The top ten are allowed to  reproduce:
    //a child occupies the place of the shortest individual
    Reproduction();
    //The new population is subjected to mutation
    Mutation();
```

```
}
```

138 ***The complete code follows:***

```
//Program A138 Letterzymes2
//Evolution of a population
//of strings.

//Strings are at the same time
//genotype and phenotype.
//They are like ribozymes.


package ejvol1;



import java.util.Random;

public class Letterzymes2
{

//Pre-declaration of global variables.
//They are used all throughout the whole class.

//Individuals are kept in the array
//Individual[]. It is an array of strings.
//The number of individuals must be
//less than limit

private static final int limit = 1000;
//How many generation to run
private static final int  nGen = 2;
//Define here when a string is long enough
//to be purged
private static final int largeSize = 1;
static private String Individual[ ] = new String[100];
static private int NIndiv = 15;
//Rank  keeps the rank of each individual.
```

```
static private int Rank[] = new int[limit];
static private String b;

static private int ReportMin[], ReportMax[];


//Turn on of the random generator
private static final Random r  = new Random();

//This method generates random chars
private  static  Character  randomChar()
{
    return  ( char ) ( r.nextInt( 26 ) + 65  );
}

/* We generate NIndiv individuals (strings)
 ten characters long.
 Sequences are completely random */
private static void Initialization( )
{
    System.out.println("ORIGINAL POPULATION");
    for(int i = 0; i< NIndiv; i++)
    {
        //Initialization  in blank.
        Individual[i]="";

        //An individual is assembled char by char
        for(int j = 0; j< 10; j++)
        {
            //We generate a random char
            Character c =  randomChar();
            //System.out.println( "char = " + c);
            //Char c is converted into a string s
            String s = ""+c;
            //s is concatenated to Individual[i]
            Individual[i] = Individual[i]+s;
        }
        System.out.println("Individual " +  i);
        System.out.println( Individual[i]);
    }
```

```
    for(int i = 0; i< NIndiv; i++)
        Rank[i]=0;

    ReportMin= new int[nGen];
    ReportMax= new int[nGen];
    for(int gen = 0; gen < nGen; gen++)
    {
        ReportMax[gen]=0;
        ReportMin[gen]=0;
    }



}


//This method takes substring a
//and posits it at place Start inside c
private  static  String
      Insert(String a, int Start, String c)
{

    String  s1, s2;
    // We copy  the left part of c
    s1 = c.substring(0, Start);
    // We copy  the right part of c
    s2 = c.substring(Start);
    // We concatenate the left part of c
    //to a to the right part of c
    String z = s1 + a + s2;
    /*System.out.println("Original c " + c);
    System.out.println("Original a " + a);
    System.out.println("Start " + Start);
    System.out.println("PosInsertion c = " + z);
    */
    return z;
}
```

```
//The letters of "ALGAE" are inserted
//into the string of individual i
private static  void feeding(int i)
{
    String a;
    // We measure the length of Individual[i]
    int l = Individual[i].length();
    //Null strings cannot feed anyhow
    if (l == 0) ;
    else //Non null strings  feed on ALGAE
    {
        a = "A";
        int Start = r.nextInt(l);
        Individual[i] = Insert(a, Start, Individual[i]);
        a = "L";
        l = Individual[i].length();
        Start = r.nextInt(l);
        Individual[i] = Insert(a, Start, Individual[i]);
        a = "G";
        l = Individual[i].length();
        Start = r.nextInt(l);
        Individual[i] = Insert(a, Start, Individual[i]);
        a = "A";
        l = Individual[i].length();
        Start = r.nextInt(l);
        Individual[i] = Insert(a, Start, Individual[i]);
        a = "E";
        l = Individual[i].length();
        Start = r.nextInt(l);
        Individual[i] = Insert(a, Start, Individual[i]);
    }
}


//This Function replaces all occurrences of
//substring a by b inside c
private static String
Substitution(String a, String b, String c)
{
```

```
    String z = c.replace(a,b);
    /*System.out.println("original a " + a);
    System.out.println("Original b " + b);
    System.out.println("Original c " + c);
    System.out.println("replaced c " + z);
    */
    return z;
}


//This function or method deletes all occurrences
//of a from c.
static private String Delete(String a, String c)
{
    b = "";
    return Substitution(a, b, c);
}


//Here we see what a PARA-SITE  does to individuals.
private  static void parasitation(int i)
{
    int l  =  Individual[i].length();
    //The parasite deletes from the
    //beginning of the individual
    //any substring   matching "PARA".
    //We copy to Beginning the first
    //4 chars of Individual[i]
    if ( l > 4)
    {
        String Beginning =  Individual[i].substring(0,4);
        String s = "PARA";
        // We compare Beginning and s
        if (Beginning.equals(s))
            Individual[i] = Individual[i].substring(4);

        //The parasite deletes from the tail
        //of the individual
        //any substring   matching "SITE".
        String Ending =  Individual[i].substring(4);
```

```
        String t = "SITE";
        // We compare Ending and t
        if (Ending.equals(t))  Individual[i] =
                    Individual[i].substring( 0,l - 4);
    }
 }


//In this method individuals are mutilated
//as a measure to hinder overgrowth.
//We delete from the original string
//a substring of a random length beginning with
//a random Start position.
private static void Purge(int i)
{
    String c, s1, s2;
    int lc, Start, n;
    c = Individual[i];
    lc = c.length();
    if (lc > largeSize)
    {
        Start = r.nextInt(lc);
        //We copy  the first part of c
        s1 = c.substring(0, Start);
        // We throw a random number
        n = r.nextInt(lc - Start+1);
        //We copy  the right part of c
        // after Start + n places
        s2 = c.substring( Start+ n);
        //We concatenate the left part of c
        //to  the right part of c
        Individual[i] = s1 + s2;
    }
}


//Individuals are sorted by fitness.
//Fitness = length, as an indicator of complexity.
private static void Sorting()
```

```
{
    int   Champ;
    //Necessary default initialization
    int Fitness[] = new int[100];
    for(int i = 0; i< NIndiv; i++)
        Fitness[i] =0;
    //We define a fitness function equal to the length
    //of the string
    for(int i = 0; i< NIndiv; i++)
    {
        String s = Individual[i];
        Fitness[i] = s.length();
        //System.out.println(Individual[i] + Fitness[i]);
    }
    //We sort individuals by fitness
    for(int i = 0; i< NIndiv;i++)
    {
        Champ = 0;
        for(int j = 0; j< NIndiv;j++)
            if (Fitness[j] > Fitness[Champ])   Champ = j;

        //The array Rank keeps a record of fitness
        //by decreasing order.
        Rank[i] = Champ;
        //The champ leaves the game
        Fitness[Champ] = 0;

    }
    /*
    //Report in decreasing order
    System.out.println("\nSorting :");
    for(int i = 0; i< NIndiv;i++)
        System.out.println(Individual[Rank[i]]);
    */
}


//The top ten fill the world
    private  static void Copying()
    {
```

```
        int   k;
        for(int i = 0; i< 10; i++)
        {
            int  l = 10*(10-i);
            for(int j = 0; j< 10; j++)
            {
                k = l -j-1;
                Individual[k] = Individual[Rank[i]];
                /*System.out.println("Ind " + k
                        + " replaced by top" + i);*/

            }
        }

        //System.out.println(  "THE NEW POPULATION IS");

        // for(int i = 0; i <  NIndiv;i++)
        //System.out.println(  Individual[i]);
    }

private static void Reproduction()
{
    //Individuals are sorted by length
    Sorting();
    //for(int i = 0; i< 10;i++)
    //System.out.println(  Individual(Rank(i))
    //The top ten produce a copy that
    //substitutes the bottom ten.
    Copying();
}

// From individual i
// the fist n  chars are deleted
private static   void Deletion(int i)
{
    int l, n;
    l = Individual[i].length();
    n = 0;
    if (l > 0)  n = r.nextInt(l);
    Individual[i] = Individual[i].substring(n);
```

```
 }

//This method inverts a
static private   String Inversion(String a)
{
    int  la = a.length();
    //Output
    String  d  = "";

    //We copy the first character of a to the last place
    //and so on.
    for(int  i = 0; i < la; i++)
     {
        char charI = a.charAt(i);
        d = charI + d;
     }
    return d;
}

//Local inversion of a substring inside individual i.
private static void innerInversion(int i)
{
    String sLeft, sRight;
    int start;
    int n = -1;
    String c = Individual[i];
    int lc = c.length();
    if (lc > 1)
    {
        start = r.nextInt(lc);
        //We copy  the first part of c
        sLeft = c.substring(0, start);
        // We throw a random number
        n = r.nextInt(lc - start+1);
        //Inner substring
        String a = c.substring(start, start + n);
        String d = Inversion(a);
        //We copy  the right part of c
        // after Start + n places
        sRight = c.substring( start+ n);
```

```
        //We concatenate the left part of c
        //to  the right part of c
        Individual[i] = sLeft + d + sRight;
    }
    /*System.out.println("Input  =   " + c + "  " + n);
    System.out.println("Output  =  " + Individual[i]);
    */
}

//We consider only two types of mutation:
//deletion of the beginning
//and inversion of an inner part.
private static void Mutation()
{
    //Which mutation affects which individual
    //is a matter of gambling.
    //We throw a 0 else a 1
    for(int i = 0; i< NIndiv; i++)
    {
        int Luck = r.nextInt(2);
        if ( Luck == 1) Deletion(i);
        else  innerInversion(i);
    }
}

//This is the heart of the process
private static void Dynamics()
{
    //All individuals feed
    for(int i = 0; i< NIndiv; i++)
    {
        //The string Algae is digested and inserted
        //into the individual[i] char by char
        /*System.out.println( "The hungry individual "
                    + i + " is " + Individual[i]); */
        feeding(i);
        /*System.out.println(  "The full individual "
             + i + " is " + Individual(i)); */
        //All individuals are tested by the parasite,
        //which corrodes any  large substring
```

```
        //similar to it located at any end.
        parasitation(i);
        /*System.out.println(  "Pos-parasitation individual "
             + i + " is " + Individual(i));*/
        //Very long chains are attacked recurrently,
        //the same as very large fishes in the sea.
        Purge(i);
        /*System.out.println(  "The purged  individual "
             + i + " is " + Individual(i)); */
    }
    //The top ten are allowed to  reproduce:
    //a child occupies the place of the shortest individual
    Reproduction();
    //The new population is subjected to mutation
    Mutation();
}


private static void report(int gen )
{
    System.out.println("\nPopulation at gen = " + gen );
    for(int j = 0; j< NIndiv; j++)
    {
        System.out.println("Gen " + gen + " Ind "
                     + j +" " +  Individual[j]);
    }
}

//Min and Max lengths are detected
private static void Observation(int gen)
{
    int m, Min, Max;
    Max = 0;
    Min = 32000000;
    for(int j = 0; j< NIndiv; j++)
    {
        m =  Individual[j].length();
        /*System.out.println(  "The Individual  "
           + Individual(j) + " has length " + m);*/
        if (m > Max)  Max = m;
```

```
        if (m < Min)   Min = m;
    }

    /*System.out.println(  "Generation "  + Gen
    + " MinLength = " + Min + " MaxLength = " + Max);*/
    ReportMin[gen] = Min;
    ReportMax[gen] = Max;
}

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

    for(int gen = 0; gen < nGen; gen++)
    {
        //System.out.println( "Beginning3");
        Dynamics();
        report(gen);
        //The aim of observation is to record
        //the minimum and maximum lengths
        //of the individuals in each generation.
        Observation(gen);
    }
    //Report of the dynamics of min and max indicators.
    System.out.println( "GENERATION MIN AND MAX VALUES");
    for(int Gen = 0; Gen < nGen; Gen++)
        System.out.println( Gen + "\t" + ReportMin[Gen]
        + "\t" + ReportMax[Gen]);

}

}//End of Program A138 Letterzymes2
```

139 ***Exercise*** *Run the program and play with the code. Once and forever get used to test methods one by one in order to verify that they do what is promised: to test a method, devise a suitable input, process it with the method and compare the output with what is expected according to the intention of the Developer. Repeat the procedure many times, a task that sometimes can be done automatically by enlarging the number of generations or of individuals of the population. The appropriate code can be inserted just at a method and be called as soon as it is*

possible. Many tests are already at hand by activating the corresponding debugger lines.

140 ***Exercise*** *In the previous program we have implemented selection and mutation. To simulate evolution completely we need to include recombination. Add it to the previous program. Why it is necessary? Selection has as function to favor the reproduction of the fittest while that of mutation is to create variability. On the other hand, recombination joins together partial solutions that were discovered by different individuals and so recombination is the evolutionary way of expressing the power of the whole population that cooperates to improve performance. Recombination of two strings produces a third one composed of parts of the former two. Example: If we have the string* `abcdefg` *and* `1234567` *then a recombinant might be* `abcd567` *and another* `12cdefg`*. In nature recombination takes two strings as input and produces two strings as output but we will produce only one. The Author forgot to include recombination in this program as in the original as in the second versions of this volume. This blatant error was dragged to other programs because many programs were created by modification of extant ones, i.e., by evolution. (Historically, this same fault was committed by the precursor of this field that is known as Genetic algorithms.)* *Answer*

10.2 Review

We have simulated evolution using a world composed of letterzymes: strings of letters, which play the roles of genotype and phenotype, and that need no catalysts. Evolution is simple but its simulation may look a bit complex. Or, there are many families of mutation, recombination and of selection procedures. And for all of them Java is sufficient. No biology in the horizon, just a test of our Java skills.

Chapter 11

More on Java

`Structuring a program, objects, disk management and multithreading.`

141 ***Purpose: We gather in this chapter four themes. The first deals with a facility of Java + NetBeans (and Eclipse) to attack software design by a top-down approach. The second is about objects. Next we learn the basics of disk management and we end with the introduction of multithreading.***

11.1 Software design

When one is a beginner, one cannot avoid feeling smashed by the complexity of 'simple' programs. The complexity arises because of two reasons: the large number of lines, say 400, and the intricacies of the syntax requirements. It is a correct appreciation that both facts press very hard over any mind. So, one is led to ask: How can I learn to devise such a complex program? The first step is to slightly modify extant material, like in our exercises. The problem with our variations of software is that one not always has an appropriate raw material from which it could be easy to make a modification by descent to fill in personal needs. So, design ab initio is a duty that we cannot avoid forever.

142 ***Top-down approach*** *Our approach to software design is the* ***top-down*** *approach. This means that one has a problem for which one has a very fuzzy almost insignificant idea. One must write it. Then one must fill in the next sublevel of determination. And so on. How it functions can be seen in the next program:*

```
//Program A142 TopDown
```

```
//Top-down approach
//to software design:
//a very fuzzy idea is conceived and written.


package ejvol1;

public class TopDown
{

    static public void subtask1()
    {
        System.out.println("I am subtask1");
    }


    static public void task1()
    {
        System.out.println("I am task 1");
        subtask1();
    }

    static public void task2()
    {
        System.out.println("I am task 2");
    }

    static private void task3()
    {
        System.out.println("I am task 3");
    }

    public  static void main(String[] args)
    {
        // My general idea consists in three tasks
        task1();
        task2();
        task3();
    }
}
```

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

144 ***Intelligence and design.*** *We have said that a top-down approach to software design is our choice. Is that the only possible way to face up software design? Is that the only possible way to face up software design for an intelligent being? Is that the only possible way to face up software design for an intelligent being when it has a very complex problem? These terrible questions that relate design to intelligence will be shown to be related also to non-intelligence thanks to* ***digital or simulated evolution****. The reason is that evolution can also be used to compose software. This special application is known as* ***genetic programming****.*

145 ***Our codex***

Programming is an art, which includes redacting and testing the ideas to solve a problem following a given syntax of a programming language. The only way to learn the syntax is to work with it. Our advice is to keep in mind how other programmers do it. To facilitate that work, we have included a Codex, printed at the beginning of this volume, where the fundamental expressions of Java have been collected.

11.2 Project interconnection

Let us learn to see in a project a functional unit.

146 ***From a flea box to a unity.*** *Up to this moment we have been piling up programs and programs on the same project. What do we have? we have a flea box. Evidently, this is an important use of Java projects. A second one is to use a project as a functional unit with many organs. The key to organism activity is interconnection. The next program shows us how easily this is done in Java:*

```
//Program A146 TopDown2
//This program invokes some methods of another class
//in the same project and package.
package ejvol1;

public class TopDown2
{

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

```
    {
        //A method of class TopDown is invoked
        System.out.println("Invokation of another class"
                + " in the same project and package:");
        TopDown.task1();
        TopDown.task2();
        //?????
        //TopDown.task3();
    }

}
```

147 ***Exercise.*** *Run the program and play with the code. Notice and try explain why the invocation on task1() works well while that on task3() not.* Answer

148 ***Public vs private***

Suppose a developer team is preparing a package that calculates an evolutionary tree given some allozymic frequencies. Some people decide to make the mathematical part while the other try out the interface with the user. Clearly, each sub team has its own Java problem. But the whole work must be united into a package. It is well known that pieces of code that work well separately fit no function when they work jointly because of mutual interference. So, modern programming languages provide us with the possibility to diminish the danger of interference. To this aim, a method can be declared as private else as public. A private method can be invoked only within the class that it belongs in. A public method can be invoked from the class it belongs in or from other classes in the same project or from other classes from other packages -if only the right addressing is furnished.

If one wants to avoid risks, one can declare as private every method with the exception of the main which must be declared public. If one needs maximal interconnectivity, one can proceed as follows: a method that depends just on the parameters of its definition (consigned in its declaration), can be declared as public. Otherwise, it realizes a work that is strongly tied to the rest of the neighboring code, so it must be private.

149 ***Divide with wisdom***

It is not at all clear when and how one must subdivide a task. Nevertheless, a simple criterion may help: if one perceives a task as a unity, it must be encoded as a unity. But if one perceives a task as a complex one, it is worth considering subdividing it in various methods, classes and projects.

150 ***Group naturally.*** *Many tasks are naturally related. One can posit them all in the same class to generate a toolbox. This is illustrated by the following class whose methods must be used from outside because the class itself has no main().*

```
//Program A150 ToolBox
//An ensemble of  mutations

package ejvol1;

public class ToolBox
{


//This method takes substring a
//and posits it at place Start inside c
public static  String insert(String a, int Start, String c)
{
    // System.out.println("original c " + c);
    // System.out.println("original a " + a);
    // System.out.println("Start " + Start);
    String  s1, s2;

    //We copy  the left part of c
    s1 = c.substring(0, Start);
    //We copy  the right part of c
    s2 = c.substring(Start);
    //We concatenate the left part of c
    //to a to the right part of c

    String z = s1 + a + s2;
    //System.out.println("Substring a was inserted " + z);
    return z;
}


//This method replaces  substring a by b inside c
public static String
      substitution(String a, String b, String c)
{
```

```
    /*System.out.println("original a " + a);
    System.out.println("original b " + b);
    System.out.println("original c " + c);*/
    String z = c.replace(a,b);
    /*System.out.println("Replacement done: " + z);*/
    return z;
}

//This method deletes the first
//occurrence of a from c.
public static  String delete(String a, String c)
{
    String  b = "";
    return substitution(a, b, c);
}


//This method inverts a
static public   String inversion(String a)
{
    int  la;
    String  d;

    //We initialize the output
    d = "";

    //We get the length of a
    la = a.length();

    //We copy the first character to the last place
    //and so on.
    for(int  i = 0; i < la; i++)
    {
        String chari = a.substring(i,i+1);
        d = chari + d;
    }
    return d;
}
```

```
//This method inverts   substring a  inside c
public  static String invert( String a,  String c)
{
    //We invert a properly
    String d =  inversion(a);
    int l = a.length();
    //System.out.println("a has " + l  + " characters");
    //We capture the first occurrence of a inside c.
    //default initialization for the
    //place n where a begins.

    int Start= -1;
    int i=0;
    while (  i<c.length()-3 )
    {
        String z = c.substring(i,i+l);
        System.out.println( "z " + z);
        if (a.matches(z))  Start = i;
        i = i+1;
    }
    if (Start>0)
    {
        System.out.println("a begins after position " + Start);
        //We copy  the first part of c
        String sl = c. substring(0, Start);
        //System.out.println(  "The left part of c is " + s1);

        //We copy  the rightmost part of c
        String sr = c. substring(Start+3);
        //We concatenate the left part of c
        //to d to the rightmost part of c
        return sl + d +sr;
    }
    else return c;
}

}//End of  class A150 ToolBox.
```

151 ***Exercise*** *Design a mean() for the previous toolbox as a separate class, whose purpose is just to test what the tools of the toolbox serve for.* Answer

11.3 Objects

In Java the word **Class** is a synonym of the word **Object**. Due to connectivity, all Java objects eventually can serve as toolboxes.

152 ***An object as toolbox***

We have seen that Java provides us with the possibility of linking diverse programs or classes to collaborate jointly into a great project. A program has its own constants, variables and methods, which could be private else public. We also have found that some classes need a head while other seem to need no one. Classes that need no head are entirely made for the use of other classes are instances of toolboxes. Now, we can understand an important point of the philosophy of ***OOP (Object Oriented programming)****: an* ***object*** *is a class that eventually serves as a toolbox which, apart from the elements that compound it, also comes with the methods that say how the elements of the toolbox function and how they interact with the environment.*

We have seen objects everywhere since the very beginning. Say, when we invoked the instruction

```
System.out.println("Inversion " + z);
```

we were invoked the object System, which has many services, i.e., methods, one of which is to report inner states to the world, an operation that can be done through many channels, one of which is to print to the console. In general, any time we encounter a point in an instruction, we are facing an object.

11.4 Interactive programming

Let us learn how to retrieve information from the keypad. This is a necessary skill to design **interactive programs**, which retrieve information from the user.

This type of programs is so important that a completely new technology has been created to make that operation nice for the User and that represents a gyro in programming: from programs composed of instructions to be executed in isolation, we pass to services to be offered and that are expectant to be contracted by the user. This indeed sounds like biology! An example early in the history of Java of this approach is presented by **applets**. We will learn about them somewhere below.

153 ***Let us look at our first interactive program, which reads data from the keyboard.***

```
//Program A153 KeyPad
//We read and process data from the keypad.


package ejvol1;



import java.io.*;
import java.util.Scanner;


public class KeyPad
{
    public  static void main(String[] args)
            throws IOException
    {
        System.out.println( "Click over the console." );
        System.out.println( "Write something " );
        System.out.println( "and close with enter!" );
        //Service to read
        Scanner scan= new Scanner(System.in);
        //Read a line from the console
        String text= scan.nextLine();
        System.out.println("This is what you typed: "
                + text);
        System.out.println( "Write a number " );
        //Read an int from the console
        int num1 = scan.nextInt();
        System.out.println( "Write another number " );
        int num2 = scan.nextInt();
        System.out.println(num2);
        int sum = num1 + num2;
        System.out.println(  num1 + " + "
                +  num2 + "  = " + sum);
    }
}
```

154 ***Exercise*** *Run the previous code and play with the code.*

155 ***Exceptions management.*** *What to do when a number is demanded from the User but he writes a string that is not a number?*

```
//Program A155 KeyPad2
//Dealing with exceptions.
//What to do when a number is demanded from the User
//but he writes a string that is not a number?

package ejvol1;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

class KeyPad2
{

    @SuppressWarnings("InfiniteRecursion")
    private static int readRadius()
    {
        //An int number will be inquired
        int r = 0;
        //Text to be captured
        String text;
        //Tool to read
        BufferedReader br = new BufferedReader(
                new InputStreamReader(System.in));
        //Prompt for the User
        System.out.println("Write  the radius of a circle. "
                + "It must be and integer");
        try
        {
            //Input is read
            text = br.readLine();
            //OK if text is an integer
            r = Integer.parseInt(text);

        }
        //If input is an integer, ignore this.
```

```
        //Otherwise, do this.
        catch(IOException | NumberFormatException e)
        {
           //Repeat request
            r = readRadius();
        }
        return r;
    }

    public static void main(String args[])
    {
        float PI = 3.1416f;
        System.out.println( "Click over the console." );
        int r= readRadius();
        System.out.println("Circle area is: " + PI*r*r );

    }
}
```

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

11.5 Disk management

157 ***Motivation:*** *Simulations can result in very large volumes of information. We have used the console to display results, but we have met many occasions in which that operation is already problematic and we as yet know nothing about great simulations. So, it is a good idea to learn to save information to a file in a hard disk and to retrieve it from in there.*

158 ***Example*** *Let us* ***create a file to save*** *there some data.*

```
/*
Program A158 HardDisc
This program creates and/or rewrites a file
and save data to it.
 */
package ejvol1;
```

```
//Needed tools
import java.util.Formatter;
import java.io.FileNotFoundException;
import java.util.FormatterClosedException;


public class HardDisc
{
    //We write   data  of different type
    static private Formatter output;

    // We open  file
    static  public void openFile()
    {
        try
        {
            //The file is created and a name is given.
            //A target directory also could be specified
            output = new Formatter( "Numbers.txt" );
        }
        catch ( SecurityException securityException )
        {
            System.err.println(
               "Permission denied"
                       + securityException.toString());
            System.exit( 1 );
        }
        catch( FileNotFoundException filesNotFoundException )
        {
            System.err.println( "Creation of file failed."
                + filesNotFoundException.getMessage());
            System.exit( 1 );
         }
    }


// We save data
static  public void writeDown()
{
    //try and catch pair
```

```
        try
        {
            // Data is fabricated and saved
            //Modify this at pleasure.
            for(int i = 0; i<10;i++)
            {
                int l = 2*i+1;
                double w = l;
                // Write  two data to file,
                // one integer and the other decimal (double)
                output.format( "%d   %.2f\n", l, w );
            }

        } // end try
        catch(FormatterClosedException formatterClosedException)
        {
            System.err.println( " Closing failure"
            + formatterClosedException.getMessage());
        } // end catch
}


    // We close file
    static  public void closeFile()
    {
        output.close();
    }


    public static void main(String[] args)
    {
        openFile();
        writeDown();
        closeFile();
    }
}//Program A158 HardDisc
```

159 ***Exercise*** *Run the code and after that verify with a file manager that in the same package of the program a file was created, whose name is Numbers.txt. Use any word processor to open and read the file. It must contain a double copy of the*

first 10 odd numbers, one as integer and the other as double.

160 ***Exercise*** *One must be very careful with the bookkeeping of the created files, because one may rewrite over important data and erase them. To verify rewriting, change the line that creates odd numbers by another that creates even numbers and run the program anew. Verify that odd numbers were erased and that odd ones appeared.* Answer

161 ***Example*** *We created and filled a file in diverse data in the previous program. Let us* ***open that very file*** *to read contained data that must next be written to the console.*

```
//Program A161 HardDisc2
//This program reads a  file
//and displays its content.

package ejvol1;


import java.io.File;
import java.io.FileNotFoundException;
import java.util.NoSuchElementException;
import java.util.Scanner;



public class  HardDisc2
{
    private static  Scanner input;

    // We open a file
    public  static  void openFile()
    {
        //try and catch pair
        try
        {
            input = new Scanner( new File( "Numbers.txt" ) );
        }
        catch ( FileNotFoundException fileNotFoundException )
```

```
        {
            System.err.println( "File does not exists."
            + fileNotFoundException.getMessage());
            System.exit( 1 );
        }
    }


    // We read  record from file
    public  static  void readData()
    {
        //try and catch pairs
        try
        {
            while ( input.hasNext() )
            {

                /* Data must be read in concordance with
                * the form as they were saved.
                * The writting procedure was:

                    {
                        //Modify this at pleasure
                        int l = 7*i+3;
                        double w = l;
                        // Write  new data to file,
                        // one integer and the other
                        //decimal (double)
                        output.format( "%d   %.2f\n", l, w );
                    }

                * The reading procedure follows:

                */
                for(int i = 0; i<10;i++)
                {
                    int l = input.nextInt();
                    double w = input.nextDouble();
                    System.out.printf("%d   %.2f\n", l, w );
                }
```

```
            }
        }
        catch ( NoSuchElementException elementException )
        {
            System.err.println( "File is corrupted."
            + elementException.getMessage());
            input.close();
            System.exit( 1 );
        }
        catch ( IllegalStateException stateException )
        {
            System.err.println( "Reading aborted."
            + stateException.toString());
            System.exit( 1 );
        }
     }


    // We close file and terminate application
    public static void closeFile()
    {
       if ( input != null )
          input.close(); //
    }

    public static void main(String[] args)
    {
       openFile();
       readData();
       closeFile();
    }
}//End of Program A161 HardDisc2
```

162 ***Exercise*** *Run the code and play with the code.*

163 ***Challenge*** *Devise a program that saves a matrix and other that reads and displays the created file.*

11.6 Communicating with Excel and Calc

164 ***Motivation.*** *From the very beginning we learned how to copy output data from a program of Java to make a graphic in Excel or Cal. This appeared to be a very useful communication. But there is more to it. Say, we also know how to input data into a Java program: just we declare them into a vector or matrix in the program or eventually one could read data from the keypad. By contrast, to input data through Excel or calc could be thousands of times easier and secure. Let us see how to do that.*

165 ***Challenging news.*** *Java and Open Office can work shoulder to shoulder. Learning how to manage that capability may become a sophisticated art. Our purpose is to show that we can make a huge profit adding a minimal knowledge: we use a disk as a very cozy interface between Excel or Calc and Java. This know-how allows one to input data to a Java program indirectly through Excel or Calc.*

166 ***Exercise*** *Witness that Excel and Calc have access to our saved file* Numbers.Text*. To that aim, just open that file with Excel or Calc. Remember that the saved file is in the same package that the program that created it (most possibly in package ejvol1 of a project with the same name and in the workspace of NetBeans 8).*

To use a word processor to input data for a Java program is more easily than using Excel or Calc.

167 ***Exercise*** *The following is the procedure to input data to a Java program through a word processor: data must be typed to a document text and saved in format .txt*

a) Copy-paste the following data to a text document. If you want to type them, use tab to separate.

```
1   0.9
3   2.9
5   4.9
7   6.9
11 10.9
13 12.9
17 16.9
19 18.9
23 22.9
27 26.9
```

else

```
1   0,9
3   2,9
5   4,9
7   6,9
11 10,9
13 12,9
17 16,9
19 18,9
23 22,9
27 26,9
```

b) Save the document: give to it the name *Numbers2.txt* and declare it as a file of type *text(delimited by tabs)* and save it. Next, save it to the same directory where your programs resides. c) Close the document.

d) To retrieve data use the following program. If the program outputs: "Corrupted file", the saving procedure must be amended and executed -reopen the . The code follows:

```
//Program A167 HardDisc3
//This program reads a  file in format .txt
//and displays its content.

package ejvol1;


import java.io.File;
import java.io.FileNotFoundException;
import java.util.NoSuchElementException;
import java.util.Scanner;



public class  HardDisc3
{
    private static  Scanner input;

    // We open a file
```

```
public  static  void openFile()
{
    //try and catch pair
    try
    {
        input = new Scanner( new File( "Numbers2.txt" ) );
    }
    catch ( FileNotFoundException fileNotFoundException )
    {
        System.err.println( "File does not exists."
        + fileNotFoundException.getMessage());
        System.exit( 1 );
    }
}


// We read  record from file
public  static  void readData()
{
    //try and catch pairs
    try
    {
        while ( input.hasNext() )
        {


            for(int i = 0; i< 1;i++)
            {
                String t = input.next();
                String s = input.next();
                System.out.println(t + "\t" +s);
            }
        }
    }

    catch ( NoSuchElementException elementException )
    {
        System.err.println( "File is corrupted."
        + elementException.getMessage());
        input.close();
```

```
            System.exit( 1 );
         }
         catch ( IllegalStateException stateException )
         {
            System.err.println( "Reading aborted."
            + stateException.toString());
            System.exit( 1 );
         }
      }


    // We close file and terminate application
    public static void closeFile()
    {
       if ( input != null )
          input.close(); //
    }

    public static void main(String[] args)
    {
       openFile();
       readData();
       closeFile();
    }
}//End of Program A167 HardDisc3
```

e) Copy-paste your data to an Excel or Calc sheet and make a graphic (whatever).

11.7 Multithreading

168 ***Motivation.*** *Nature works at many fronts at the same time. We have no problem in simulating this phenomenon of parallelism; actually, all operational systems can manage a word processor, a sheet of calculus, a platform for programming Java, a printer and a CD player at the same time. By the same token, parallel computing can be simulated in any programming language with the tools we know. Nevertheless, Java has a specially suited facility for that purpose:* ***multithreading****. It is not necessary at all, say, C++ lacks it, but it is the delikatessen that distinguishes the house. So, let us probe it. It is absolutely necessary to become acquainted with multithreading because Java uses it all the time -even to execute*

mathematical calculations that are mixed with string operations- and some phenomena cannot be understood without it.

169 ***Example*** *The next code implements two* ***threads****, one to print odd numbers and the other to print even numbers.*

Printing odd numbers or printing even numbers are very similar tasks, so we define a prototype *vector*, which enables the calculus and printing of the corresponding numbers. A marker, *k*, defines whether we deal with odd else even numbers. *Limit* is the variable that says how many numbers must be calculated and printed.

Once the prototype has been defined, we concoct a program that creates and prints the desired vectors. The code for the prototype or constructor of *vector* follows:

```
//Program A169 Vector
//Threading: part one of two.
//This class defines a prototype
//of a vector of odd numbers
//else even numbers.
package ejvol1;

//In threads, classes are called runnables.

public class Vector implements Runnable
{
    //Each instance has its name
    private final String vectorName;

    int l, limit;

    //Constructor
    //Prototype definition
    //k=0, even numbers
    //k =1, odd numbers
    public Vector( int k, int lim )
    {
        if (k==0)  vectorName = "evenNumbers";
         else vectorName = "oddNumbers";
```

```
        l=k;
        limit = lim;
    }

    // The vector is calculated and printed
    public void task(int k, int limit)
    {
        // Try and catch exception pair:

        int vec[] = new int[1000];
        try   // do this task
        {
            for(int j=0; j<=limit; j++)
            {
                vec[j]= 2*j+k;
                //Thread is put to sleep
                //during s milliseconds
                int s = 2;
                Thread.sleep( s );
            }
            System.out.println(vectorName);

            for(int j=0; j<=limit; j++)
            {
                System.out.println( vec[j]);

            }
        }
        catch ( InterruptedException exception )
        {
            System.out.println(   vectorName+
              "Killed from outside" );
        }
    }

    //This is the 'main method' of the
    //prototype constructor.
    @Override
    public void run()
    {
```

```
        task(l, limit);
        //l defines parity, limit is the number
        // of entries in the vector.
    }
}//End of Program A169 Vector
```

170 ***Using threads***

Threads are temporal phenomena. This means that one deals with time directly. In particular, one can put a thread to sleep, say, to allow the perception by a human being of a printing process. To that aim the 'Thread.sleep' method is used, whose parameter comes in units of milliseconds.

The code for the utilitarian program follows:

```
//Program A170 PrintVectors
//Threading: part two of two.
//This program creates instances of Vector
//and commands to run them in different threads.
package ejvol1;



import java.util.concurrent.Executors;
import java.util.concurrent.ExecutorService;

public class PrintVectors
{
   public static void main( String[] args )
   {
        // We create and name each instance or runnable
        Vector vec1 = new Vector(0, 20);
        Vector vec2 = new Vector(1, 5 );

        System.out.println( "Program started" );

        //Mandatory initialization
        ExecutorService threadExecutor =
            Executors.newCachedThreadPool();
```

```
        // threads are started
        threadExecutor.execute( vec1 );
        threadExecutor.execute( vec2 );


        //  threads are killed if tasks are completed.
        threadExecutor.shutdown();

    }
}
```

171 ***Exercise*** *Copy both programs to any project, save them and run the second one, in which the prototype is used to create instances and to run them.*

172 ***Exercise*** *Threads requires delicate care. To understand what this means, please, completely replace the constructor class by the next one. The only change that was made is that the delaying command is moved from the calculus of the vector to its printing process. Witness that the odd and even numbers appear mixed. The reason of this failure is that a special thread of low priority manages the printing process. This means, that two printing processes may compete for realization if they come from different threads. The just evidenced failure in threading management is remedied with* ***synchronization****, a theme that is left for research.*

```
//Alternate version of the class Vector
//Program A169 Vector
//Threading: part one of two.
//This class defines a prototype
//of a vector of odd numbers
//else even numbers.
package ejvol1;


//This class defines a prototype
//of a vector of odd numbers
// else even numbers.
public class Vector implements Runnable
{
    //Each instance has its name
    private final String vectorName;
```

```
int l, limit;

//Constructor
//Prototype definition
//k=0, even numbers
//k =1, odd numbers
public Vector( int k, int lim )
{
    if (k==0) vectorName = "evenNumbers";
    else vectorName = "oddNumbers";
    l=k;
    limit = lim;
}

// The vector is calculates and printed
public void task(int k, int limit)
{
    // Try and catch exception pair:

    int vec[] = new int[1000];
    try   // do this task
    {
        for(int j=0; j<=limit; j++)
        {
            vec[j]= 2*j+k;
        }
        System.out.println(   vectorName);
        for(int j=0; j<=limit; j++)
        {
            System.out.println( vec[j]);
            //Thread is put to sleep during k milliseconds
            int s = 2;
            Thread.sleep( s );
        }
    }
    catch ( InterruptedException exception )
    {
        System.out.println(   vectorName+
          "Killed from outside" );
```

```
        }
    }

    //This is the 'main method' of the
    //prototype constructor.
    public void run()
    {
        task(l, limit);
        //l defines parity,
        //limit is the number
        // of entries in the vector.
    }
}
```

11.8 Review

We have seen more on chaptering a project, on toolboxes and objects, on disk management and multithreading. Let us notice that these items are not necessary in strict sense but they exist because somehow they diminish the heavy load of software design, which is very difficult task. That is why the Reader is asked to have patience with him or herself and to endure all through the end of this work and further.

Chapter 12

Distributions

```
Systematic study of random events.
```

173 ***Motivation.*** *We have witnessed and will see it again that our simulations could touch very easily on important themes. And that the random generator is found at the root of every simulation. In consequence, random numbers, random variables and randomness in general are per se very important for our studies. So, it could be necessary to understand randomness a bit better.*

174 ***Purpose: In this chapter we theorize on the systematic study of random events.***

175 ***Some terminology***

One declares a sequence of events to be **random** when it is hard to predict which is the next event given the prior ones. That is what happens with a die. That is equivalent to say that the information given by the sequence is not redundant and that every event is important, so each event yields independent information. This also implies that the cheapest way to fully describe the sequence is giving the sequence itself.

In nature, random events have an origin or source. Mathematicians like to say that we have a **stochastic process** to denote the output of a source of random events. A stochastic process may be **stationary** or **non stationary**. In the first case, the source has a defined personality, which is depicted by a histogram of relative frequencies or probabilities, which is abstracted by a distribution. That is why we say: a stationary process is determined by its distribution function.

To prove that a process is stochastic and non stationary, it is necessary to prove that at each moment in time the output of the process is random and that it is

described by a distribution function and that this distribution may change from moment to moment.

It is widely believed that the fundamental laws of **matter** are correctly depicted by equations that render observables that are stochastic stationary processes: the fundamental laws of matter are knowledgeable with experiments that anyone can repeat because they are invariant over time and space.

Science is built over the belief that **nature**, in whatever form and complexity, is no more than an elaboration of the fundamental laws of matter. That includes you and me in all our integrity. We will not discuss that point; we will just make calculations and predictions in relation with biology and evolution that must be fulfilled if that scientific credo is taken for true. The next terrible point that is important for science is the **Occan's razor**: if a model explains some data, the model can be simplified but not burdened with more complexity. In other words, in regard with fundamentals, we need not worry about things that are redundant, unnecessary: fundamental science needs no earrings.

Randomness is fundamental for biology: it is believed in science that **mutations** in the DNA, the fuel of evolution, are depicted by a stochastic process: a mutation happens because the fundamental laws of quantum mechanics dictate that they may happen and, in absence of further information, all point mutations shall happen with equal probability. From this assertion, a number of predictions could be done and tested.

The purpose of the present chapter is to convert some features of this philosophical wording into operational procedures satisfying the modern approach to **model rejecting**. We say model rejecting because in some circles, no model can be accepted and the only purpose of science is to reject models. But in reality it is hard to find someone who is completely deprived of beliefs of the most varied type. So accepting and rejecting could be somehow taken as complementary options.

Our first rejection says that our Java random generator is not random.

12.1 The psudo-random generator

176 *Recursive functions*

We already have used the random generator that is inbuilt in Java. It is very easy to prove that our generator is a source of non random numbers: in fact, it is deterministic by construction: it is just a simulation of a random generator using recursive functions, i.e., functions for which each specific value depends on the previous one. Let notice that by construction, our random generator denies the essence of randomness but, anyway, it functions as a very good imitation.

We can understand what it is all about if we pay attention to the following recursive function

$x_o = 3.5$

$x_{j+1} = x_j(4 - x_j)$

This recursive scheme is expanded as follows:

$x_o = 3.5$

$x_1 = x_o(4 - xo) = 3.5(4 - 3.5) = 3.5(0.5) = 1.75$

$x_2 = x_1(4 - x2) = 1.75(4 - 1.75) = 1.75(2.25) = 3.9375$

and so on. Let us help ourselves with the next program to calculate the next 1000 terms:

```
//Program A176 RandGenerator
//A generator of pseudo-random numbers

package ejvol1;


public class RandGenerator
{
    public static void main(String[] args)
    {
        double x = 3.5;
        String s4, s5;
        for(int i = 0; i<1000; i++)
         {
            x = x*(4-x);
            // System.out.println(x);
            // conversion of number x
            // into a format appropriate for
            //some versions of Excel:
            //s4 = String.valueOf(x);
            //s5=s4.replace(".", ",");
            System.out.println(  i + "\t" + x);
         }
    }
}
```

177 ***Exercise*** *Run the previous code and change variously the initial value and the number of iterations. Make a scatter graphic of the output using Excel or Calc. What can we say about the behavior of the sequence?* Answer

178 *Pseudo-random numbers*

In the jargon, our numbers are called **pseudo-random**. What could be the price we must pay for using pseudo-random numbers instead of truly random numbers in our simulations?

The art of generating pseudo-random numbers is today well developed, so for many questions the difference could be rather unnoticeable. Anyway, we shall keep an eye on this problem.

Let pass now to study the personality of our source of random numbers.

179 *A bar diagram*

If one must analyze a short sequence of numbers, like 3, 5, 4, 6, 4, 7, 3, 5, 4, 6, 3, 8, one just takes a pocket calculator and calculates the mean, variance or deviation. But if one must analyze too large a sequence, one usually prefers to compress the information even if one lost fidelity. One simple and very useful form of compressing sequences of data is through histograms after a rounding procedure. For instance, a list of numbers like that just given can be compressed in a table of frequencies, as follows: The number 3 appears trice. Number 4 appears trice. Number 5 appears twice. Number 6 appears 2 times. Number 7 appears once. Number 8 appears once. We can write this in a table:

A frequency table			
Number	Absolute frequency	Relative frequency	Cumulative frequency
3	3	3/12	3/12
4	3	3/12	6/12
5	2	2/12	8/12
6	2	2/12	10/12
7	1	1/12	11/12
8	1	1/12	12/12

A **histogram** is a drawing that represents a frequency table. Since the correspondence among frequency tables and histograms is one to one, we use histograms and frequencies tables as equivalent terms. Observe that for this concrete example, the only information we lose is the relative order in which data appeared.

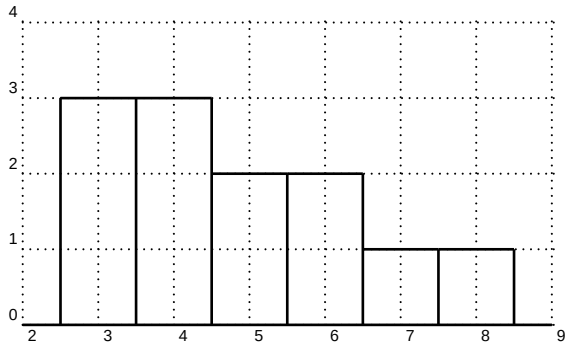


Figure 12.0. Histogram that represents the previous frequency table.

180 *The mean*

Histograms are done for the eyes. But when one takes decisions and must invest money, one gains a lot of satisfaction if one can compress the information of data to just two descriptors: the mean and the variance. To calculate the mean of a list x of n data, one adds all numbers and divides by the number of data.

181 ***Example*** *The mean of data 3, 5, 4, 6, 4, 7, 3, 5, 4, 6, 3, 8 is*

$$\bar{x} = \frac{3+5+4+6+4+7+3+5+4+6+3+8}{12} = \frac{3+3+3+4+4+4+5+5+6+6+7+8}{12}$$

$$= \frac{3(3)+3(4)+2(5)+2(6)+1(7)+1(8)}{12} = 4.8833$$

The last line says that to find the mean of a frequency table, with value x_i and absolute frequency n_i, one must multiply the x_i times n_i, row by row, and then add altogether. The result must be divided by the total number of events that is the sum of all the n_i. This is written as follows: let x_i be datum number i and n_i its absolute frequency. Then, the mean of all data is

$$\bar{x} = \frac{\sum n_i xi}{\sum n_i}$$

182 *The variance*

While the mean is the best representative of a given data, the variance measures the divergence of the population with respect to the mean. Let X be some data and V_x its variance. When the n data are listed, the variance is:

$V_X = \frac{\sum (x_i - \bar{x})^2}{n}$

When the data are compressed into a frequency table (x_i, n_i), the variance is

$V_X = \frac{\sum n_i (x_i - \bar{x})^2}{\sum n_i}$

183 ***Exercise*** *Take the data 3, 5, 4, 6, 4, 7, 3, 5, 4, 6, 3, 8, write the expression for its variance according to the definition and arrange that expression to recover the formula for the variance that corresponds to a table of frequencies.* Answer

184 ***The deviation***

The standard deviation is the positive square root of the variance. Therefore the standard deviation s_x of a frequency table (x_i, n_i) is

$$s_X = \sqrt{\frac{\sum n_i (x_i - \bar{x})^2}{\sum n_i}}$$

If the variance is 2.47, then the standard deviation is

$$s_X = \sqrt{2.47} = 1.57$$

185 ***All in one***

We already know what we mean by a frequency table and how to calculate its mean, variance and standard deviation. Let us devise a program that calculates the mean, variance and standard deviation of any given frequency table.

```
/*
 * To change this license header, choose License Headers in Project Pro
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package ejvol1;


// Program 185 FreqTable
// Mean and variance of a
// table of absolute frequencies.

public class FreqTable
{

   public  static void main( String args[] )
   {
        //Frequecy table: datum  and its frequency.
```

```
        int  Data[][] = {{3, 3},
              {4, 3},
              {5, 2},
              {6, 2},
              {7, 1},
              {8, 1}};


        // Report of input data
        System.out.println( "Datum and its frequency" );
        for ( int i = 0; i < Data.length; i++ )
        {
            System.out.println( Data[ i ][0]+
                           "\t "+ Data[ i ][1]);
        }
        // Let us calculate the mean
        double sum = 0;
        //Number of data
        int n = 0;
        for ( int i = 0; i < Data.length; i++ )
        {
            sum = sum + Data[i][0]*Data[i][1];
            n  = n + Data[i][1];
        }
        double mean = sum / n;
        System.out.println( "Sum \t "+ sum);
        System.out.println( "Mean \t "+ mean);
        // Let us calculate the variance
        double sum2 = 0;
        for ( int i = 0; i < Data.length; i++ )
        sum2 = sum2 + Data[i][1]*(Data[i][0]-mean)
                                      *(Data[i][0]-mean);
       double var = sum2 / n;
                   System.out.println( "Variance \t "+ var);
   }
}//End of Program 185 FreqTable
```

186 ***Exercise*** *Run out this code and study its design. Modify it to get the mean and variance of own data.*

12.2 The uniform distribution

187 ♣ ***Definition.*** *We say that a discrete random variable X observes a* ***uniform distribution*** *when all the possible elementary events have the same probability. For instance, if one plays with a die, one expects a uniform distribution over the set composed of integer numbers from one to six.*

188 ***Exercise*** *Tailor a definition of a uniform distribution for a continuous random variable, say, as one depicting the output of a roulette.* Answer

189 ***Example***

A source of pseudo-random numbers produced numbers in the interval (0,1). A sequence of 480 random numbers was generated and a histogram with 10 classes was made. To study the null hypotheses that the random numbers have a uniform distribution in within zero and one, a statistical test was run. The error is defined as $(Observed_i - Expected_i)^2/Expected_i$.

Random numbers			
Class	Observ.	Expect.	Error
0,1	44	48	0,333333333
0,2	49	48	0,020833333
0,3	40	48	1,333333333
0,4	42	48	0,75
0,5	40	48	1,333333333
0,6	54	48	0,75
0,7	59	48	2,520833333
0,8	48	48	0
0,9	56	48	1,333333333
1	48	48	0
Σ	480	480	8,375

The total error is compared with the theoretically critical error, which is given by the Chi-square distribution. We have 9 d.f. For $\alpha = 0.20$ the corresponding Chi-square is 12.242. For $\alpha = 0.10$ the corresponding Chi-square is 14.684. For $\alpha = 0.05$ the corresponding Chi-square is 16.919 and with $\alpha = 0.01$ the corresponding Chi-square is 21.666. Our experimental error is 8.375, which is less than all those values. Therefore, the discrepancy between our distribution and a uniform one is too small to be considered as significant. Hence, we conclude that the generator produces random numbers with a uniform distribution in the interval (0,1).

What we are claiming?

We are claiming that deterministic programs can mimic randomness somehow and that a chi-square test is unable to detect the fraud. This is the level of resolution that we will work with. We do not claim that other tests could not discover our lies.

190 ***Example***

Normally, random generators are designed to fit a uniform distribution. In the next program we generate 1000 numbers that range from 1 to 10, next, we build the corresponding histogram of absolute frequencies, and run a **chi-2 test** to know whether or no the histogram fits a uniform distribution.

```
//Program A190 RandGenerator2
//Generates a discrete uniform distribution
//from one to ten.

package ejvol1;



import java.util.Random;

public class RandGenerator2
{

    // Turn on of the random generator
    static Random r  = new Random();


    static int Frec[ ];//Predeclaration

    public static void  chisquare()
    {
        double chi = 0;
        for(int j = 1; j<= 10; j++)
         chi  = chi +  (Frec[j]-100)*(Frec[j]-100)/100;
        System.out.println("Calculated chi-square = " + chi);
    }
```

```
    public static void main(String[] args)
    {
        Frec= new int[11];
        for(int j = 1; j<= 10; j++)
                Frec[j] = 0;
        for(int j = 0; j< 1000; j++)
        {
           // We generate a random sequence
           int c = r.nextInt(10) +1;
           Frec[c] = Frec[c]+1;
        }
        System.out.println("j, observed and "
                + "expected frequencies");
        for(int j = 1; j<= 10; j++)
            System.out.println(j + "\t" + Frec[j] + "\t100");
        chisquare();
    }
}//End of Program A190 RandGenerator2
```

191 ***Exercise*** *Run the code 10 times and retrieve the chi-squares given by it. Compare those values with 16.9, the expected for a significance of 0.05. What proportion of runs yielded a uniform distribution?* Answer

192 ***Large sequences***

One is usually taught that for a small number of trials, one can obtain departures from the expected values but that when the number of trials increases, one gets closer to the expected frequency. This dream is precisely the reason that allows us to speak that a source of random events follows a, say, uniform distribution. Thus, we claim that a random stationary source acquires a personality when it is allowed to express itself in very long sequences. Is that true? Let us study this question with the next simulation. There, the uniformity of a sequence is measured using a chi-square test. Given a critical value, if the observed chi-square is greater than the corresponding critical value, we decide that the distribution fails to be uniform. Otherwise, we decide that data fits a uniform distribution. The chosen significance is 0.05.

```
// Program 192 RandGenerator3
// Generates a discrete uniform distribution.
```

```
// with range from 1 to 10.
// Uniformity is verified with a chi-2 test.
package ejvol1;


import java.util.Random;

public class RandGenerator3
{

    // One can vary the number of trials,
    // by changing the variable length.

    static int length;

    // Turn on of the random generator
    static private Random r  = new Random();


    static int Frec[ ];//Predeclaration

    public static void  chisquare()
    {
        double chi = 0;
        double Length = length;
        double expected = Length/10;
        for(int j = 1; j<= 10; j++)
            chi  = chi +  (Frec[j]-expected)
                      *(Frec[j]-expected)/expected;
        System.out.print("Number of trials = "+ length);
        System.out.print("  Observed chi-square = " + chi);
        if (chi<16.91 )System.out.println("  Uniform.");
        else System.out.println("  Non Uniform.");
    }

    public static void main(String[] args)
    {
        // factor is a scale parameter
        int factor = 10000;
        Frec= new int[11];
```

```
        for(int i= 1; i<10; i++)
        {
            length = i*factor;
            for(int j = 1; j<= 10; j++)
                    Frec[j] = 0; //Initialization
            // We generate a random sequence
            for(int j = 1; j<= length ; j++)
            {
                // We generate a random number
                //from 1 to 10
                int c = r.nextInt(10) +1;
                //book keeping
                Frec[c] = Frec[c]+1;
            }
            // for(int j = 1; j<= 10; j++)
                  // System.out.println(j + "\t" + Frec[j]);
            // We measure how uniform is the distribution.
            // Greater chisquare values mean
            // lesser uniformity
            chisquare();
        }
    }
}//Program 192 RandGenerator3
```

193 ***Exercise*** *Play with the code to decide whether or not the generated sequences fit a uniform distribution and if with larger sequences one fits better the expected distribution.*

194 ***Research*** *The previous exercise shows that it is false that a source with a planned uniform distribution tends asymptotically to a perfect uniform distribution. (Asymptotically means that when the number of trials increases to infinite then the approximation is all the better). To explain this failure, we have two options: to claim that our source of random numbers is rudimentary else that our source is good and the result is good but it is the belief in the asymptotic convergence that is false. This problem has been studied and solved by mathematicians and is an example of the type of questions that is best addressed with mathematics and not with simulations.*

12.3 The binomial distribution

195 ♣ ***Definition.*** *We say that we have a* ***binomial distribution*** *when we deal with events that belong in one of two classes, say, male and female, yes and no, to be or not to be. One declares which the probability of one of the events, p, is. The other probability is calculated as* $q = 1 - p$.

196 ***Simulation***

Beginning with a uniform distribution, one can simulate a binomial distribution: to simulate, say, white or black, one divides the range of the uniform distribution in two parts proportionally to p, the probability of white, and q, the probability of black. So, if a random number falls in within the first part, one says that one obtains a white otherwise one obtains a black.

With a binomial distribution one can simulate a **mutation rate**, p: if one throws a random number with a uniform distribution in the interval (0,1), then, if the number is less than p, one decides that a mutant arouse otherwise one produces no mutation.

197 ***Example***

Let us simulate a coin, whose faces are named m and f as in male and female.

In the program, we use the method that generates random numbers with a uniform distribution

```
r.nextDouble()
```

that simulates a random variable with uniform distribution in the interval (0,1). The code follows:

```
//Program 197 RandGenerator4
//Simulattion of a binomial distribution.

package ejvol1;


import java.util.Random;

public class RandGenerator4
{
    static Random r  = new Random();
```

```
    public static void main(String[] args)
    {
        // Probability of male
        double p = 0.3;

        for(int j = 0; j< 10; j++)
        {
            //We generate a random number
            //in within 0 and 1
            //with a uniform distribution.
            double  c = r.nextDouble();
            //We verify gender
            if (c<p)
                System.out.println("A male is born");
            else
            System.out.println("A female is born");
        }
    }
}//Program 197 RandGenerator4
```

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

199 ***Example*** *Let us simulate a population of 1000 individuals, each one* ***f*** *else* ***m****. Let us define the probability of* ***m*** *to be 0.2 and that of* ***f*** *to be 0.8.*

In our next program we use some special mathematical functions:

```
Math.pow(x,y); // number x to the power y.
Math.abs(z); //Absolute value of z.
```

The first instruction, **Math.pow(x,y)**, calculates x^y and the second, Math.abs(z), calculates the absolute value of a number z, which could be of the double type.

```
//Program 199 RandGenerator5
// Binomial dist with a z-test.

package ejvol1;
```

```
import java.util.Random;

public class RandGenerator5
{
static Random r  = new Random();

 public static void test(int Males, double p)
 {
//Statistical z-test to see
//the fairness of the simulation
//Square root of p * (1 - p) / 1000
double  sigmaP= Math.pow( (p * (1 - p)/1000), 0.5);
double M = Males;
double zObserved = (M/1000 - p) / sigmaP;
System.out.println("The z of the experiment is "
                + zObserved );
double  s = 1.96;
System.out.println("The critical z is   " + s);
String A ;
        //Absolute value of zObserved
if (Math.abs(zObserved ) < s)  A = "Yes";
  else A = "No";
System.out.println( "Is the generator fair?  " + A);
 }

public static void main(String[] args)
{
    // Probability of male
    double p = 0.2;
    int numberMales =0;
    int total = 1000;
    for(int j = 0; j< total; j++)
    {
        //We generate a random number
        //in within 0 and 1
        double  c = r.nextDouble();
        // We verify gender
        if (c<p) numberMales  = numberMales +1;
```

```
    }
    System.out.println("Males: " + numberMales );
    System.out.println("Females: "
            + (total - numberMales) );
    test(numberMales,p);
}
}//End of Program 199 RandGenerator5
```

200 ***Exercise*** *Run the code and play with the code.*

201 ***Exercise*** *Reuse the previous code to produce a population of random numbers of arbitrary size. Study the* ***asymptotic problem****: does the tendency of observed frequencies to fit the expected ones under a binomial distribution get better with the number of trials?* *Answer*

202 *Is* ***frequency*** *in a binomial distribution a robust concept?*

According to the previous exercise, there is an **asymptotic tendency** of the observed to fit the expected for the case of the frequencies of a binomial distribution. Thus, the concept of **frequency** in a binomial distribution is robust, with a possible asymptotic concordance between observed and expected. But in strict sense, we are cheating. There are two options to attempt a more rigorous approach: mathematical calculations (with an affirmative answer) else giving a try to super-computing. Today, a supercomputer can be 'easily' done by appropriately programming a local network, which could be connected through the web to others such assemblies. Such a super-computer is called a **grid**. Java has the potentiality to become the natural language of grids.

203 ♣ ***Definition.*** *Suppose that one throws n coins, with face one and face two and with probabilities p and q respectively, and counts the number of times that face one lands. Officially, this is the* ***binomial distribution*** *with parameters n and p. We note this as*

$X \sim Bi(n,p)$

and we read: X is the number of landed faces of type one (probability p) when one throws n coins. X is a random variable that has a binomial distribution $Bi(n,p)$.

204 ***Simulation*** *Let us count the number of boys in a set of families in which the probability of boy is p. Officially: let us make a population simulation of the binomial distribution* $X \sim Bi(n,p)$.

```
//Program A204 RandGenerator7
//Let us  count the number of boys in
//a set of families in which the
//probability of boy is p.
//Observed is compared to expected.

//This is a population study of
//Bi(n,p) the binomial distribution
//on n families and probability of boy equal to p.

package ejvol1;


import java.util.Random;

public class RandGenerator7
{
    static private int n;
    static private double p;

    // The number of studied families is
    // nFamilies

    static private double nFamilies;

    // Turn on of the random generator
    static private Random r  = new Random();

    // Observed frequencies
    static private double Obs[ ];//Predeclaration

    // Expected frequencies
    static private double Exp[ ];//Predeclaration


    private  static void  expectedFrec()
    {
    for(int k=0; k<n; k++)
    {
```

```
    double nk = 1;
    //Binomial coefficient
    for(int j=1; j<=k; j++)
             nk = nk* (n-k+j)/j;
    System.out.println("Coefficient " + k + " \t " + nk);
    Exp[k]=  nk*Math.pow(p, k)*Math.pow(1-p,n- k);
    Exp[k]= Exp[k]*nFamilies;
 }

}

private  static void  chisquare()
{
 double chi = 0;
 //length is int;
 for(int j = 0; j< n; j++)
  chi  = chi +  (Obs[j]-Exp[j])*(Obs[j]-Exp[j])/Exp[j];
 System.out.println("Number of families = "+ nFamilies);
 System.out.println("Number of children/family = "+ n);
 System.out.println("Observed chi-square = " + chi);
 int df = n-2;
 System.out.println("Compare with chi.2 critical "
        + "with " + df + " degrees of freedom.");
}

public static void main(String[] args)
{
    // number of children per family
    n= 6 ;
    // Probability of boy
    p = 0.5;
    //number of studied families
    nFamilies = Math.pow(2,n) * 10;
    // The observed distribution
    // is kept in Obs
    Obs= new double[(int) nFamilies]; //Declaration
    // The expected distribution
    // is kept in Exp
    Exp= new double[(int) nFamilies]; //Declaration
```

```
        for(int i = 0; i< n; i++)
                Obs[i] = 0; //Initialization

        for(int i = 0; i< n; i++)
                Exp[i] = 0; //Initialization

        // Total number of children


        // We generate nFamilies families
        for(int i = 0; i< nFamilies; i++)
        {
            // We generate a random family.
            // nboys counts the number of boys
            int nBoys = 0;
            for(int j = 0; j< n; j++)
            {
                //We generate a random number
                // in within 0 and 1
                double  c = r.nextDouble();
                // We count boys
                if (c<p) nBoys = nBoys +1;
            }
            Obs[nBoys] = Obs[nBoys]+1;
        }


        expectedFrec();
        System.out.println("Observed and Expected");
        for(int j = 0; j< n; j++)
             System.out.println(j + "\t" + Obs[j] + "\t" + Exp[j]);
        System.out.println();

        // We measure how good is the fitting
        // Greater chi-square values mean
        // worser fitting to a binomial distribution.
        chisquare();
        }
}//End of Program A204 RandGenerator7
```

205 ***Exercise*** *Play with the code. Modify the code to make it suitable for Excel and Calc and draw suitable graphics. Use results to study the asymptotic problem: does the tendency of observed frequencies to fit the expected ones under a binomial distribution get better with the number of trials? Answer*

206 ***General failure.*** *We have no reason to propose that one can read a binomial or a uniform distribution from nature. Instead, these distributions are theoretical constructs that frame the scientific method: science is not a part of nature, it is a product of the* ***mind****.*

12.4 The normal distribution

207 ♣ ***Definition.*** *The* ***normal distribution*** *is the continuous distribution that best fits a binomial one.*

208 ***Example*** *Let us learn to generate numbers with a normal distribution.*

To simulate a random variable with a normal distribution we use the method

```
r.nextGaussian()
```

which generates a random variable with mean 0 and standard deviation 1. We also use the instruction

```
int c = (int) Math.floor(5*d) + 25;
```

that contains the function **Math.floor(x)** that approximates a decimal number to the immediate inferior integer. Say, Math.floor(2.3)=Math.floor(2.8)=2. This function produces a number of type double but with the modifier **(int)** one forces a conversion from **double to int**. The code follows:

```
//Program A208 RandGenerator
//Simulation of
//a normal distribution with
//mean 25 and deviation 5

package ejvol1;

import java.util.Random;

public class RandGenerator8
```

```
{
    //Turn on of the random generator
    private static Random r  = new Random();

    private static int Frec[ ];//Predeclaration

    public static void main(String[] args)
    {
        //Size of the random sequence
        int size = 10000;
        // Frec contains the histogram.
        Frec= new int[50];
        // Initialization
        for(int j = 0; j< 50; j++)
                Frec[j] = 0;
        //We generate a random sequence
        for(int j = 0; j< size; j++)
        {
           // We choose a Gaussian distribution
           double d = r.nextGaussian() ;
           // System.out.println(d ) ;
           // Rescale numbers: mean 25, deviation 5-
           //Math.rint = approx. to the closest int
           int c = (int) Math.rint(5*d) + 25;
           //book keeping
           Frec[c] = Frec[c]+1;
        }
        // Report
        for(int j = 0; j< 50; j++)
            System.out.println(j + "\t" + Frec[j]);
    }
}//End of Program A208 RandGenerator
```

209 ***Exercise*** *Run the code and make a graphic of the histogram of absolute frequencies. Test the asymptotic tendency.* Answer

210 ***Exercise*** *Compose the code that converts a Gaussian of mean zero and deviation one into another of mean m and deviations s. Test your theories with suitable changes to the previous program and corresponding graphics.* Answer

211 ***Testing for normality***

One can prove according to the modern standards that a given histogram fits a normal distribution. For that task, a **Kolmogorov Smirnov test** is used. Let us see how it works.

Let us decide whether or not a distribution of height given by the histogram below fits a normal distribution. The mean height is 1.65 and the standard deviation is 9.5743. So the z associated to a height x is $z = \frac{x-1.65}{9.5743}$. Height is represented by x, the absolute frequency of x is $N(x)$, the cumulated absolute frequency is $F(x)$, the relative cumulated frequency of x is $\Phi(x)$, the distribution function of z is $\Phi(z)$ and the absolute value of the difference between the theoretical expected distribution and the experimental one is $|\Phi(x) - \Phi(z)|$. The null hypothesis in this test is that these differences are due to mere randomness. The test is calculated with the help of the next table:

Fitting of tallness to a normal distribution						
x	$N(x)$	$F(x)$	$\Phi(x)$	z	$\Phi(z)$	$\lvert\Phi(x) - \Phi(z)\rvert$
150	12	12	0.167	-1.57	0.059	0.108
160	24	36	0.5	-0.522	0.301	0.199
170	24	60	0.833	0.522	0.699	0.134
180	12	72	1	1.57	0.941	0.058

The Maximum value of $|\Phi(x) - \Phi(z)|$ is 0.199 but, according to the Kolmogorov Smirnov theory and if the null hypothesis is correct, the maximum allowed value would be 0.192. Since the observed difference is greater than the allowed one, we reject the null hypothesis: our data do not fit a normal distribution. The value 0.192 was calculated as $1.63/\sqrt{n} = 1.63/\sqrt{72} = 1.63/8.48 = 0.192$ an approximation that is valid when the number of events is greater than 35. The constant 1.63 is the value associated to the significance $\alpha = 0.01$ and n is the total number of data. For $\alpha = 0.05$, that constant is 1.36 and the corresponding maximum allowed difference would be $1.36/\sqrt{72} = 1.36/8.48 = 0.16$. For this value of α, the null hypothesis also would be rejected (of course).

212 ***Challenge*** *With the help of a Kolmogorov Smirnov test, prove beyond doubts that the previous program indeed produces a normal distribution. Prove else contradict that the fitting is better the larger is the size of the sequence. In other words, prove else contradict that for a normal distribution the asymptotic tendency is validated. Explain whether this is an effect of the poor programming of random numbers or of the real essence of nature.*

Java comes with uniform and normal distributions. And the others? Do it by yourself. Let us see.

12.5 Fitting anyone distribution

Let us learn how one can program a source of random numbers with a uniform distribution to fit **whatever distribution** one desires.

213 ***Fitting a discrete distribution.*** *To fix ideas, let us suppose that one has the next table that registers the relative frequencies of numbers 0,1,2 and 3. Our purpose is to program a random generator that follows exactly the distribution given by the table.*

A histogram of relative frequencies	
numbers	Frequency
0	3/10
1	2/10
2	0/10
3	5/10

The idea to solve the problem is the following: we generate numbers from 0 to 9 (included). Next, we divide the range zero to nine in four parts in proportion to the relative frequencies given by the table. Next, we rename the output of the source in agreement with the events given by the table. Say, our first sub-range covers 0,1,2. If the random generator happens to output any one of this numbers, we define our output as 0. The next sub-range covers 3 and 4. If anyone of this numbers is produced, we report 1.The next sub-range covers 5, 6, 7, 8 and 9. For any one of those cases, one answers 3. This is done in the next code:

```
//Program A213 RandGenerator9
// A source generates random integers
// with a given distribution.

package ejvol1;

//Program A213 RandGenerator9
// A source generates random integers
// with a given distribution.
```

```
import java.util.Random;

public class RandGenerator9
{
    // Absolute frequencies are defined:
    static int Table[] =  {3,2,0,5};

    // Counters are in count
    static int count[] = new int[4];

    // Turn on of the random generator
    static Random r  = new Random();

    public static void main(String[] args)
    {
        for (int i=0; i<Table.length;i++)
            count[i]=0;

        for (int i=0; i< 10000;i++)
        {
            //A random integer less than 10 is generated
            //with a uniform distribution
            int n = r.nextInt(10);
            //Outputs are renamed
            if (n<3) count[0]=count[0]+1;
            if ((3<= n )& (n<5)) count[1]=count[1]+1;
            if ((5<= n )& (n<10)) count[3]=count[3]+1;
        }
        for (int i=0; i<Table.length;i++)
                System.out.println(i + "\t" + count[i]);
    }
}//Program A213 RandGenerator9
```

214 ***Exercise*** *Run the code and assess the accuracy of the fitting.* *Answer*

215 ***Fitting a continuous distribution***

Let us suppose that we have a table of relative frequencies that describes a certain continuous distribution, such as in the next table:

A Table of relative frequencies	
Interval	Frequency
0-0.999	3/10
1-1.999	2/10
2-2.999	0/10
3-3.999	5/10

Our task is to device a source of decimal random numbers that takes values in within 0 and 4 to fit the distribution given by the proposed histogram of relative frequencies but with the condition that the decimal numbers must have a uniform distribution in within each one of the four intervals. We use the same idea of the previous program but with a tiny modification:

```
//Program A215 Randgeneratorr10
//Back-copied from
//Program E159 whateverCTest.

//A source generates random number of type double
//with a given distribution
//but with the condition that the   numbers must have
//a uniform distribution in within each one of
//given intervals.
//A test for the claim is included.

package ejvol1;




import java.util.Random;

public class RandGeneratorr10
{


//Absolute frequencies are defined:
//The frequency of 0 is 3, that of 1 is 2,
//that of 3 is zero and that of 4 is 5.
private static final int Dist[] =  {3,2,0,5};
```

```
private static int nClasses = Dist.length;
private static double sizeSample;
private static final int sizeSimulation = 100000;
private static final double Events[] =
                    new double[sizeSimulation];


//Counters of simulated events are in count
private static final double count[] = new double[nClasses];
//Predicted distribution
private static final double Predict[] = new double[nClasses];
//Discrepancies line per line
private static final double criticalChi2 = 7.8;
private static final double Discrep[] = new double[nClasses];
//Turn on of the random generator
private static final Random r  = new Random();
//A boolean variable has to values: true else false.
private static boolean printAll = false;

private static double min;
private static double max;
//The left border of intervals, less or equal to the
//minimum value of data.
private static double infLimit;
//Interval length
private static double intervalLength ;
//Max number of allowed classes
private static final int maxnClasses = 1000;
//Borders of the intervals
private static final double  Barriers[] =
                           new double[maxnClasses +1];
private static final double classMarkers[] =
                           new double[maxnClasses];

//Distribution of the discrepancies
private static final double    FreqTable[][] =
                          new double[maxnClasses][2];


//Prints data in vector
```

```
private static void printVector(double Vect[])
{
    for ( int i = 0; i < Vect.length; i++ )
        System.out.println( i + "    " + Vect[ i ]  );
}


//Returns the size of the sample
private static double sizeSample()
{
    double s= 0;
    for(int i= 0; i < nClasses; i++)
    {
        if (Dist[i]> 0 )
            s = s +  Dist[i];
    }
    return s;
}

//Predicts what must be observed
private static void prediction()
{
    for(int i= 0; i < nClasses; i++)
    {
        double probSubI =  Dist[i] / sizeSample;
        Predict[i] =  probSubI * sizeSimulation;
    }
}


//Runs a chi^2-test to check the accuracy
//of our source of random numbers
//that must fit the distribution given by Dist
private static double chiTest()
{
    double d= 0;
    for(int i= 0; i < nClasses; i++)
    {
        if (Predict[i]> 0 )
           Discrep[i]  =
```

```
                Math.pow((count[i] - Predict[i]),2)
                                              /Predict[i];
        d = d + Discrep[i];
    }
    return d;
}

//The range of discrepancies is evaluated
//and is used to calculate the length of each
//interval to group Discrepancies.
public static double intervalLength()
{
    min=100000;
    max = 0;
    int lim = sizeSimulation;
    for(int i = 0; i < lim; i++)
    {
        if (Events[i] < min ) min = Events[i];
        if (Events[i] > max ) max = Events[i];
    }
    System.out.println( "min = " + min );
    System.out.println( "max = " + max );
    double range = max - min;
    double length = range /nClasses;
    return length;
}


//Prints the frequency table
private static void printFreqTable(double FreqTable[][],
                                          int nClasses)
{
    System.out.println( "x and its absolute frequency" );
    for ( int i = 0; i <= nClasses; i++ )
        System.out.println( FreqTable[ i ][0]+
                           " "+ (int) (FreqTable[ i ][1]));
}


//Calculates the sum    of absolute frequencies
```

```
//of a frequency table
private static void
             sum(double FreqTable[][])
{
    double sum = 0;
    for ( int i = 0; i <= nClasses; i++ )
            sum = sum +  FreqTable[i][1];
    System.out.println( "Number of events = " + sum);
}

//The border of each interval is calculated
private static void getBorders(double Vect[])
{
    Barriers[0] = infLimit;
    classMarkers[0] = infLimit + intervalLength/2;
    //Calculate maximal value of data
    double max = Vect[0];
    for(int j=1; j < Vect.length; j++)
      if (Vect[j] > max)  max = Vect[j];
    System.out.println( "Max value = " + max);
    //Calculates number of classes
    nClasses =   (int) ((max- infLimit)/intervalLength) + 1;
    System.out.println( "Number of classes = " + nClasses);
    //Calculates interval borders and class markers
    for(int j=1; j <= nClasses; j++)
    {
        Barriers[j] = Barriers[j-1] + intervalLength;
        classMarkers[j] = classMarkers[j-1] + intervalLength;
    }
    /*System.out.println( "Borders are" );
    printVector(Barriers);
    System.out.println( "Class markers are" );
    printVector(classMarkers);*/
}

//Data are grouped into interval classes.
private static void group(double Vect[])
{
    intervalLength = intervalLength();
    infLimit = min - intervalLength/2;
```

```
    getBorders(Vect);
    int nData = sizeSimulation;
    //Class markers a re defined
    for(int j=0; j < nClasses; j++)
    {
        FreqTable[j][0] = classMarkers[j];
        FreqTable[j][1] = 0;
    }
    //Frequencies are calculated.
    for(int i=0; i< nData; i++)
    {
        for(int j=0; j < nClasses; j++)
        {
            if ((Vect[i] >= Barriers[j]) &
                (Vect[i] < Barriers[j+1]))
               FreqTable[j][1] =FreqTable[j][1] +1;
        }

    }
}


//The claim that there is a uniform distribution
//in within each inner barrier is tested
private static void  test()
{
    group(Events);
    System.out.println("Frequency table of events");
    printFreqTable(FreqTable, nClasses);
    //Minor test for correct grouping
    sum(FreqTable);
    System.out.println("Frequencies   of events (for Excel)");
    for(int i = 0; i <= nClasses ; i++)
            System.out.println(i + "\t" + FreqTable[i][1]);
}


public static void main(String[] args)
{
    for (int i=0; i< Dist.length;i++) count[i]=0;
```

```
for (int i = 0; i <  sizeSimulation;i++)
{
    //A random decimal number in (0,10) is generated
    //r.nextDouble() generates a random decimal number
    //with a uniform distribution in within 0 and 1.
    double e = 10*r.nextDouble();

    //Outputs are renamed
    if (e<3)
        {e = e/3; count[0] = count[0]+1;}
    if ((3<= e )& (e<5))
        {e = 1+ (e-3)/2; count[1]=count[1]+1;}
    if ((5<= e )& (e<10))
        { e = 3 + (e-5)/5; count[3]=count[3]+1;}
    Events[i] = e;
}
printAll = false;
if (printAll)
{
    System.out.println(" Simulated events");
    printVector(Events);
}
System.out.println("Observed in simulation values");
printVector(count);
sizeSample = sizeSample();
prediction();
System.out.println("Predicted values");
printVector(Predict);
double dExp = chiTest();
System.out.println("Ho: observed values fit " +
                        "predicted ones");
System.out.println("Discrepancies");
printVector(Discrep);
System.out.println("Experimental discrepancy = " + dExp);
System.out.println("Critical chi2 = " + criticalChi2);
if (dExp >  criticalChi2)
    System.out.println("The event is extreme "
        + " according to Ho: invent another theory.");
else
```

```
        System.out.println("Normal event according to Ho:"
            + "there is no reason to reject Ho");
    //We open 10 classes for each interval
    nClasses = 10* nClasses;
    //test for uniformity in within integer borders
    System.out.println(" \nTEST zone");
    test();
}//End of main()
}//End of class A215 RandGeneratorr10
```

12.6 Conclusions

It is amazing that randomness (the incapability to predict the next datum) could be simulated by very simple deterministic recursive functions. We learned to device random generators with a uniform, binomial or normal distributions. At last, we simulated a source of random numbers with anyone distribution. We also paid attention to the study of the intuitive belief that with more events a simulated distribution becomes more like the expected one. This resulted to be false, but the concept of relative frequency was found to converge to the p parameter of a binomial distribution although the convergence is a fuzzy one. Convergence for the normal distribution was better, a fact that forces one to think that one must try huge numbers to witness a better fitting to the expected distribution.

Chapter 13

The Pandora box

`Randomness as part of science.`

Here we learn the basics on how science deals with randomness.

13.1 General view

Modern science is built upon **materialism**, which is a philosophical school of thought that amounts to a **cosmovision** or **world view**, an ensemble of principles to interpret the world. According to it, mind is a byproduct of the dynamics of matter, which is compounded of elementary particles, time and space.

Materialism is not imposed by anybody. Instead, the natural and spectacular development of science has been framed by materialism and, moreover, those that consider that matter is enough to explain everything, life included, have largely influenced the development of science. Nevertheless, it has not been shown that matter is enough to explain life. Concretely and in spite of all efforts and promises, artificial life exists on computers, not in reality. Since we lack artificial life, we cannot claim that we have understood life. Because of this vacuum, materialism must be included in science as a regulative, dogmatic principle that demands that all explanations must be given in terms of matter such as it is known in labs.

Now, given materialism, the purpose of science is to describe matter, its fundamental parts, interactions and dynamics. Descriptions are given in terms of hypothesis, theories. The **null hypothesis** of science is randomness. This means that everything in this world can be explained by randomness. Say, we can deny the existence of gravitational fields that cause the falling of bodies towards the center of the Earth and claim, instead, that bodies fall by mere randomness. Nevertheless, such an explanation has a an insignificantly low probability. By contrast, modern gravitational theories produce quantitative predictions that allow the exact location

of an ensemble of satellites that enable GPS. Given this, no one on Earth beliefs that gravitation can be replaced by randomness. With this we are declaring that randomness is everything that cannot be explained in a simple way by deterministic models.

Deterministic theories, similar to gravitation, are grandiose but at last explain very few things. In fact, with rare exceptions all events that happen around you have exceedingly low probabilities.

So, insignificant probabilities happen all the time and science must accept them. In particular, insignificant probabilities that appear in a distribution of central tendency, such as in a normal distribution, are important in science when they represent an event that is an **outlier**, i.e., one that is far away from the mean. In a case like this, an outlier becomes a matter of further investigation to see whether or not it is explained by the same mechanism that generates the events in the center of the distribution.

Thus, outliers represent a place where new discoveries could be preferentially searched for.

Life is an outlier in comparison with most material systems. Say, a mammal can keep its corporeal temperature approximately constant during long periods of time but, by contrast, most systems tend to level their temperature to that of the milieu. This suffices to understand why life is a phenomenon of tremendous scientific importance. In other words, science concedes a great status to theories that pretend to explain life. Among them, the Evolutionary Theory of the origin of species has been singled out as the official explanation that is, furthermore, extremely elegant: it is the champion in the art of combining determinism and randomness. On one hand we have that mutation is declared to be at random while in the other we see that selection is an emerging deterministic phenomenon that is expected to produce better and better fitting of organisms to their environment. We have found this belief very interesting and worth being studied at depth. To begin with, let us try to understand randomness a bit better.

13.2 Competence among models

Randomness has many forms of being that are distinguished by their distributions, i.e., by the relative frequencies of events in long experiments. So, randomness has many channels and ensuing hypotheses or models to explain facts. Nevertheless, some models do it better than others. One family of channels refer to uniform distributions in which all events happen with equal probability. Allowed events form a set that is called the support of the distribution. We will consider that the

source of events corresponds with the landed faces of a die. Our dies can have 6 or whatever numbers of faces.

216 ***How many faces has the die?*** *Given a sequence of numbers that was generated with a die, we must decide how many faces the die has.*

Let us consider the sequence 2, 3, 3,4,5,4,2,1,2,3,2,4,3,5,1,2,3, 5, 4. One can imagine various theories to explain these data. To fix ideas, let us consider just two theories. First: the sequence is produced by a source that functions like a die with 5 faces that fall with a uniform distribution. Second: the sequence is produced by a source that functions as a die with 6 faces with a uniform distribution but 6 does not appear just because of randomness. It is clear that the first theory is sound. But we cannot reject the second one: it also explains data although with less probability.

So, let us device a program that a) simulates a die with 6 faces and b) asks how many trials are necessary to produce a sequence 15 numbers long that mimics a die with 5 faces.

```
//Program A216 Die
//A  source of random numbers simulates
//a face with 6 faces.
//Can it  simulate   a die with 5 faces?
//Yes, sometimes it does.
//We count  how many trials are
//necessary to produce a sequence 15 numbers long
//that mimicks a die with 5 faces.



package ejvol1;


import java.util.Random;

public class Die
{

    //Turn on of the random generator
    private static Random r  = new Random();
```

```
//size of the random sequence
private static int length= 15;

//Count counters the trials
private static int count;
//Absolute frequencies of faces 1 to 6
//that appeared in the sequence.
static int Observed[ ] = new int[7];
//Criticak chi
static private double criticalChi2 = 9.48;

public static double  Obschisquare(int length)
{
    double chi = 0;

    //The die with five faces is associated to a distribution
    //that assigns a probability 1/5 to numbers 1 to 5.
    //Hence, a sequence  of landed faces
    //must have 1/5*length instances of each face.

    double Expected = length/5;
    //Expected distribution is compared to observed one.
    //Discrepancy is approximately distributed
    //as a chi square:
    for(int j = 1; j<= 5; j++)
        chi  = chi + (Observed[j]-Expected)
            *(Observed[j]-Expected)/Expected;
    System.out.println("Observed chi-square = " + chi);
    return chi;
}

//Sequences are generated with a uiform distribution
//in within 1 and 6 included
public static void tryAndTry(int length)
{
    count = count +1;
    System.out.print(" Trial " + count + "  ");
    // Initialization
    for(int j = 1; j<= 6; j++)
```

```
            Observed[j] = 0;

        //We generate a random sequence
        for(int j = 0; j< length; j++)
        {
            // We choose a uniform distribution
            // from 1 to 6, as in a die.
            int c = r.nextInt(6)+1;
            System.out.print(c ) ;
            // bookkeeping
            Observed[c] = Observed[c]+1;
        }

        // Report
        // for(int j = 1; j<= 6; j++)
        // System.out.println(j + "\t" + Observed[j]);

        if (Observed[6] > 0)
        {
            //System.out.println("It landed 6");
            System.out.println(" Sequence with a 6");
            tryAndTry(length);
        }
        else
        {
            System.out.println(" Sequence WITHOUT a 6");
            //A uniform distribution
            //with support 1...5 is needed.
            if (Obschisquare(length) >= criticalChi2)
            {
                System.out.println(" NON UNIFORM distribution ");
                tryAndTry(length);

            }
            else
                System.out.println(" Uniform distribution ");
        }
    }
```

```
    public static void main(String[] args)
    {
        tryAndTry(length);
        System.out.println("We succeed at the trial " + count);
    }
}//End of Program A216 Die
```

217 ***Exercise*** *Run the program and play with the code. Play enough to agree else disagree with the Author: for the chosen length of sequences, a die with 5 faces can easily be mimicked by a die with 6 faces.*

218 ***Example***

Let us modify the previous code to learn in which form the number of trials to achieve the first success depends on the length of the sequence.

```
//Program A218 Die2
//Mimicking a die of 5 faces
//with a die of 6 faces.

package ejvol1;


import java.util.Random;


public class Die2
{
    //Turn on of the random generator
    static Random r  = new Random();
    //long is a very long integer
    static long nTrials = 0; // number of trials.
    static int Observed[ ] = new int[100];

    //size of the random sequence
    private static int length;

    private static double  Obschisquare(int length)
    {
        double chi = 0;
```

```
        //The die with five faces is associated to a distribution
        //that assigns a probability 1/5 to numbers 1 to 5.
        //Hence, a sequence  of landed faces
        //must have 1/5*length instances of each face.

        double Expected = length/5;
        //Expected distribution is compared to observed one.
        //Discrepancy is approximately distributed
        //as a chi square:
        for(int j = 1; j<= 5; j++)
            chi  = chi + (Observed[j]-Expected)
                *(Observed[j]-Expected)/Expected;
        //System.out.println("Observed chi-square = " + chi);
        return chi;
    }

    // While we have not succeed
    // try and try
    private static void tryAndTry(int length)
    {
        // done is a flag that says
        //when we have succeeded
        int done = 0;

        while (done==0 )
        {
            nTrials = nTrials +1;
            //System.out.println(" Trial " + nTrials);

            //Initialization
            for(int j = 1; j<= 6; j++)
                Observed[j] = 0;

            //We generate a random sequence
            for(int j = 0; j<length ; j++)
            {
                //We choose a uniform distribution
                //from 1 to 6, as in a die.
```

```
                int  c = r.nextInt(6)+1;
                //System.out.print(c );
                //book keeping
                Observed[c] = Observed[c]+1;
            }
            //Report
            //for(int j = 1; j<= 6; j++)
            //System.out.println(j + "\t" + Observed[j]);
            //We have succeed if 6 did not land
            // and if the distribution is uniform.
            if ((Observed[6] == 0)&& (Obschisquare(length) <= 9.48) )
                    done = 1;
        }// end of while.
    }

    public static void main(String[] args)
    {
        int rounds = 15;
        System.out.println( "length +  number of Trials");
        for(int j = 1; j< rounds; j++)
        {
            length = 10 * j;
            //Find a sequence that mimicks a not charged
            //die with 5 faces.
            tryAndTry(length);
            System.out.println( length + "\t " +nTrials);
        }
    }
}//End of Program A218 Die2
```

219 ***Exercise*** *Run the code and try to increase beyond imposed borders the length of the sequence. Show that the cost of explaining facts by randomness is an increasing function of the length of the input sequence. Can you say whether or not the cost is an exponential function in the length of the input sequence? Please, use a regression study.* *Answer*

220 ***Model selection***

We have illustrated a fact of science: randomness has infinitely many ways to explain facts. For applied purposes, it is wise to be ready to pay a cost in the

number of trials before success. Nevertheless, for pure science one cannot demand that a cost was paid by nature, i.e., one is allowed to believe that nature did it at the first trial or in any case in some of the trials that were made during the allowed interval of time. This is what means that randomness is the null hypothesis of science.

In practice, one has various models and one usually chooses the one that fits better. Let us consider again a random source that behaves like a die with five faces. Of course, we know that the model that better explains this source is a die with five faces. But, suppose we have other two models. The first model says that the data are generated by a die with six faces, while the second model proposes a die with seven faces. Instinctively, one would say that a die with six faces would do it better than a die with seven faces. And, what can we say after a simulation? Let us study this problem in the next program.

```
//Program A220 Die3
//A die with 6 (7) faces
//tries to mimic a die with 5 faces.

package ejvol1;


import java.util.Random;

public class Die3
{

//Turn on   the random generator
private static final Random r  = new Random();
// Counts the number of trials
static private long count ;

private static final int Observed[ ] = new int[50];

// The following is a chi2 table.
// Signifcance 0.05.
private static final double  Chi[]  = {
                        3.84,
                        5.99,
                        7.81,
```

```
                        9.48,
                        11.07,
                        12.59,
                        14.06,
                        15.5,
                        16.91,
                        18.30,
                        19.67,
                        21.02,
                        22.36,
                        23.68,
                        24.99,
                        26.29,
                        27.58,
                        28.86,
                        30.14,
                        31.41,
                        32.36,
                        33.92,
                        35.17,
                        36.41,
                        37.65,
                        38.88,
                        40.11,
                        41.33,
                        42.55,
                        43.77};
//This measures the error between
//expected and observed
//The distribution must be uniform on 1 to 5 inclusive.
private static double  observedChisquare(int length)
{
    double chi = 0;

    //The die with five faces is associated to a distribution
    //that assigns a probability 1/5 to numbers 1 to 5.
    //Hence, a sequence  of landed faces
    //must have 1/5*length instances of each face.

    double Expected = length/5;
```

```
    //Expected distribution is compared to observed one.
    //Discrepancy is approximately distributed
    //as a chi square:
    for(int j = 1; j<= 5; j++)
        chi  = chi + (Observed[j]-Expected)
            *(Observed[j]-Expected)/Expected;
    //System.out.println("Observed chi-square = " + chi);
    return chi;
}


// While we have not succeed
// try and try
private  static void tryAndTry(int length, int nfaces)
{
    // done = 1 is flag that says
    //that we have succeeded
    int done = 0;
    while (done == 0 )
    {
        count = count +1;
        //System.out.println(" Trial " + count);
        // Initialization
        // This instruction must be changed
        // if nfaces>7
        for(int j = 1; j<= 7; j++)
            Observed[j] = 0;

        // We generate a random sequence
        for(int j = 0; j< length; j++)
        {
            // We choose a uniform distribution
            // from 1 to nfaces, as in a die.
            int  c = r.nextInt(nfaces)+1;
            // System.out.println(c );
            //book keeping
            Observed[c] = Observed[c]+1;
        }
        // Report
        // for(int j = 1; j<= nfaces; j++)
```

```
        // System.out.println(j + "\t" + Observed[j]);


        //Degrees of freedom: nfaces -1;
        //but Java counts from zero, hence
        int df = nfaces -2;
        //System.out.println(observedChisquare(length)
        //+ "\t "+ Chi[df]);
        //If nfaces = 6:
        //We have succeed if 6   did not land
        // and if the distribution is uniform.
        //If nfaces = 7:
        //We have succeed if 6 and 7  did not land
        // and if the distribution is uniform.
        if ((Observed[6] +Observed[7] == 0)&&
            (observedChisquare(length) < Chi[df]) )
            done = 1;
    }
}

public static void main(String[] args)
{
    //Number of faces, nfaces, of the die
    //must be less than 30.
    int rounds = 4;
    for(int nfaces  = 6;nfaces <=7;nfaces ++)
    {
        System.out.println("Die with "
                + nfaces + " faces");
        System.out.println( "seq.length "
                + "+ number of trials until success");
        for(int j = 1; j< rounds; j++)
        {
            count = 0;
            //Observed contains the observed frequencies.
            //Let us make the simulation
            //length = size of the random sequence
            int length= nfaces*j;
            // For each die, we generate
            // sequences of increasing length,
```

```
            // which is defined by length,
            // which is a multiple of nfaces.
            tryAndTry(length,nfaces);
            //System.out.println("Die with "
            //      + nfaces + " faces" );
            //System.out.println("Sequences of length "
            //      + length );
            //System.out.println("We succeed at the trial "
            //      + count );
            //System.out.println("length and trials");
            System.out.println(length + "\t"  +   count);
        }
        System.out.println();
    }
}
}//Program A220 Die3
```

221 ***Exercise*** *Run the code and play with the code. Decide which model is the best to mimic a source that functions likes a die with 5 faces.* Answer

222 ***Science is possible.*** *Our previous exercise shows that the natural aim of science - to build, test and reject models- is a healthy task. We have the illusion that this process leads to the truth, i.e., an ultimate model that explains all in simple and elegant terms. Nevertheless, we cannot guarantee that the truth can be reached: all models of science can be called to a challenge and every* ***paradigm*** *or great model has its detractors.*

223 ***The origin of determinism.*** *We saw that randomness is the null hypothesis of science and the fundamental mechanism of physics (although physics also has deterministic features, say the charge of a proton is constant and is not random). But, nevertheless, our world is very consistent and stable: Where does such stability result from?*

We have reasons that are proper to quantum mechanics, which for certain phenomena and for large scales resembles classical mechanics. We also have reasons that belong in the realm of randomness. In the second case, determinism is expected to result from self-averaging. What does this mean?

Let us consider a family of sources, such that each one produces numbers with the same uniform distribution. If we add the output of one source to that of another, we can expect that, precisely because of their randomness, the two values would

fall apart and their discrepancies with respect to the mean annihilate one another. The net result is that a central tendency appears.

Is that true? Let us study that problem in the next simulation.

```
//Program A223 Die4
//We study sums of
//discrete uniforms distributions,
//each one  with range from 1 to 9
//and mean 5.

package ejvol1;



import java.util.Random;

public class Die4
{
    // One can vary the number of trials,
    // by changing the variable length.

    static private int length;
    static private double n;
    static private double p;

    //The number of studied families is
    //nFamilies

    //Observed frequencies
    static private double Obs[ ] = new double[9001];

    //Turn on of the random generator
    static private Random r  = new Random();

    public static void main(String[] args)
    {

        //nUniforms = number of random variables.
        // nUniforms can range from 1 to 1000
```

```
        int nUniforms = 10;
        int m = 9*nUniforms;

        for(int i = 1; i<= m; i++)
                Obs[i] = 0; //Initialization

        int factor = 5;
        length = m*factor;
        //Loop with many trials
        for(int j = 1; j<= length ; j++)
        {
            // We generate nUniform random numbers
            //from 1 to 9
            int sum = 0;
            for(int k = 0; k< nUniforms; k++)
            {
                int c = r.nextInt(9) +1;
                // and we add them:
                sum = sum + c;
            }
            //bookkeeping
            Obs[sum] = Obs[sum]+1;
        }
        System.out.println("Sums and their frequencies");
        for(int j = 1; j<= m; j++)
            System.out.println(j + "\t" + (int) Obs[j]);
    }
}//End of Program A223 Die4
```

224 ***Exercise*** *Run the code and verify that when one adds n uniform random variables with equal range, the discrepancies with respect to the mean annihilate one another. The net result is that their sum tends to concentrate itself around n times the mean. This is a valid slowly* ***asymptotic tendency****.*

225 ***Exercise*** *According to the output of the previous program, we predict that when one adds n uniform random variables with equal range, and the result is divided by n, the output tends to concentrate itself around the mean. Add one line to the previous code to check that this could be indeed so.* *Answer*

226 ***Challenge*** *The result of the previous exercise is known as* ***law of large numbers****. Verify that students of probability and statistics usually do not understand*

that law. Use previous simulations together with graphics in Excel to teach them the meaning of the law.

227 ***Exercise*** *Mathematicians like to make emphasis in the condition that the law of large numbers to be valid requires the random variables to be* ***independent****. This means that one cannot predict the value of the output of one of them knowing the output to the others. Explain why our previous programs comply with this condition. Design a program to average n* ***dependent*** *random distributions and verify that the law of large numbers fails.* Answer

13.3 Evolution creates determinism

Summing or averaging various independent random variables is one form of approaching determinism beginning from random variables. Another form as one can extract determinism from randomness is through a suitable process. One example of this is very nice: it is evolution. To prove this, let us arrive at a *determined* point in spite of and thanks to the randomness of mutation.

228 ***Example*** *Evolution solves an algebraic problem.*

Evolution is an artist that uses randomness as the fuel of its imagination to solve concrete problems that demand deterministic answers. To see this, let us solve the equation

$4x+3=0$

Its solution is $x=-3/4=-0.75$. We know that because we are intelligent, i.e., we can make and work with abstractions, including algebra. Now, to find a solution to that equation implies to select one point among the infinitude of points that belong in the real line. Let us verify that we can find that point with a simulation of evolution using mutation of strings. Technically, a simulation of evolution is called a **genetic algorithm**.

Our idea is to see a number as a string, as a chromosome. So, we produce a population of numbers and we subject the population to evolution but with an appropriate fitness function. To design that function, we notice that the solution a must fulfill the equation $4a+3=0$. So, if one has a number b, one calculates

$$f(b)=4b+3$$

Of course, $f(a)=0$. Thus, we declare the fitness of b to reward the nearness of $f(b)$ to zero. To this aim, we use a function that is positive and that has its maximum at zero:

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

Thus, we can define fitness by
$Fitness(b) = \frac{1}{1+(4b+3)^2}$

So, the solution -3/4 has fitness 1 and all other numbers has a lesser fitness. Chromosomes or numbers reproduce in proportion to fitness. We expect that the population converge to the solution. We will see in a program below whether or not that is true. But we need some previous work:

Let us notice that numbers can be added, multiplied while strings can be concatenated, replaced. Algebra deals with numbers while evolution deals with strings. How can we solve an algebraic problem with evolution? We must encode the algebraic problem in the language of evolution, i.e., in strings. Let us solve a first problem: to convert a digit into a char.

A **number from 0 to 9 can be represented as a char or letter**. To that aim, we use the method

```
//Integer number i is converted to char z
//when z is printed it looks just like i.
char z=  ( char ) ( i +48  );
```

where the index i goes from 0 to 9. How this works can be verified with the next program:

```
//Program A228 IntToChar
//Numbers as chars

package ejvol1;



public class IntToChar
{

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

```
            System.out.println("Int and its char "
                    + "representation");
            for(int i = 0; i <=9; i++)
            {
                //int i is converted to char z
                char z =  ( char ) ( i +48  );
                System.out.println(i + "\t"+ z);
            }

}
}
```

229 ***Exercise*** *Run the code and verify that it perfectly deals with chars that look like digits.*

230 ***Chars to numbers***

We also need to read the char at the $k-th$ position in a string, but we cannot forgive that Java begins counting with zero. The reading of the $k-th$ char of a string s is done by the method **s.charAt(k)** . Moreover, we need to convert a digit encoded as a char into a digit as a number. This is done thanks to the method **Character.getNumericValue(c)** that makes the conversion **char to digit**. The whole assemble looks like this:

```
    char c  =   s.charAt(k);
int i = Character.getNumericValue(c);
```

231 ***Encoding***

We encode a number as a string (that looks as a number) in the following form: the first character of the string could be zero or one. Zero means negative sign, and one means positive sign. The second character says in which place the decimal point appears. For instance, 3 means that the point is located *after* the third character. The other chars copy the number without the decimal point. Our numbers have 10 ciphers. In general, the code reads:

$$(sign)(place\ of\ point)(ten\ digits).$$

Thus, the string 151432345645 represents the positive number 14323.45645: it is positive because the first char is 1. The location of the point is given by

the second char which is 5: the point goes after the fifth char and the string 1432345645 with 10 chars represent all the digits in the number. By the same token, 025467846723 represents -54.67846723.

We know that the solution to our equation is -3/4= -0.75, so in our encoding it reads 0075. The reason is that the first zero indicates a negative sign, while the second zero means that the decimal point goes after the zero-th place.

Let us implement a program to decipher a string and that produces a number as output:

```
//Program A231 StringToNumber
//This program decodes strings into numbers.

package ejvol1;



public class StringToNumber
{

    private static double decoder(String s)
    {

        // We get the sign:
        char s1  =   s.charAt(0);
        int sign = Character.getNumericValue(s1);
        // We get the place of the point,
        // which implies multiples of ten.
        s1  =   s.charAt(1);
        int  point = Character.getNumericValue(s1);
        // We build the number with the other chars
        double number = 0;
        for(int j=2;j  <  11;j++ )
        {
            char s3  =   s.charAt(j);
            int  l = Character.getNumericValue(s3);
            int k = j-1;
            int power = point-k;
            //number1 = l * 10^power
```

```
            double number1 = l * Math.pow(10,power);
            number = number + number1;
        }
        if (sign == 0) return (-1) *number;
            else return  number;
    }

    public static void main(String[] args)
    {
        String s = "06123456789";
        double n = decoder(s);
        System.out.println("String = " + s + " Number = " + n);
    }

}//End of program A231 StringToNumber
```

232 ***Exercise*** *Run the code and test it with various inputs to verify that it correctly implements our conventions: sign in the first digit (one is plus, zero is minus); place of the point is in the second char: 3 means that the point is located after the third character. Then 10 digits follow.*

233 ***Evolution the mathematician.*** *Let us see in the next program how and how rapidly evolution solves the equation*

$4x+3=0$

```
//Program A233 Equation

//Evolution of a population
//of strings that finds a root
//of a polynomial.

//Mutation + selection.

package ejvol1;
```

```
import java.util.Random;

public class Equation
{
    // Strings are at the same time
    // genotype and phenotype.

    // Pre-declaration of global variables.
    // They are used all throughout the whole class.

    // Individuals are kept in the array
    // Individual[]. It is an array of strings.
    // The number of individuals must be
    // less than limit

    private static final int POPSIZE = 100;
    static private double Fitness[];
    static private String Individual[ ];//Predeclaration
    // Order  is an array of integers.
    static private int Order[];//pre-declaration
    static private int Generation;
    static private int  nGen;
    static private int ReportMin[], ReportMax[];
    private  static int numberOfClones ;
    private static final double MUTATIONRATE = 0.2;
    private  static int Champ;

    // Turn on of the random generator
    private static final  Random RAND = new Random();
    private static final boolean  PRINT = false;

    // This method generates random chars
    // that represent digits
    private  static  Character  randomChar()
    {
        return  ( char ) ( RAND.nextInt( 10 ) + 48  );
    }


    private static void Initialization( )
```

```
    {
        /* We generate POPSIZE individuals (strings)
        Sequences are completely random */

        //Formal declaration of our array.
        Individual= new String[POPSIZE];
        //System.out.PRINTln("ORIGINAL POPULATION");
        for(int i = 0; i< POPSIZE; i++)
        {
            //Initialization  in blank.
            Individual[i]="";

            //An individual is assembled char by char
            for(int j = 0; j<  9; j++)
            {
                // We generate a random numeric char
                Character c =  randomChar();
                //System.out.PRINTln( "char = " + c);
                // Char c is converted into a string s
                String s = c.toString();
                // s is concatenated to Individual[i]
                Individual[i] = Individual[i]+s;
            }
            // We add the place for a point.
            Character n =  randomChar();
            String s = n.toString();
            Individual[i] = s+Individual[i] ;

            //We add a sign: zero or one.
            n = (char) ( RAND.nextInt( 2 ) + 48  );
            s = n.toString();
            Individual[i] = s+Individual[i] ;
            //System.out.PRINT("Individual " +  i + "\t");
            // System.out.PRINTln( Individual[i]);
        }


        //Declarations and default initializations
// of other arrays.
```

```
        Order= new int[POPSIZE];
        ReportMin= new int[POPSIZE];
        ReportMax= new int[POPSIZE];
        Fitness= new double[POPSIZE ];

        for(int i = 0; i< POPSIZE; i++)
        {
            Order[i]=0;
            ReportMin[i]=0;
            ReportMax[i]=0;
        }
    }

    //The encoded number is withdrawn
    private static double decoder(String s)
    {
        // We get the sign:
        char s1  =   s.charAt(0);
        int sign = Character.getNumericValue(s1);
        // We get the place of the point,
        // which implies multiples of ten.
        s1  =   s.charAt(1);
        int  point = Character.getNumericValue(s1);
        // We build the number with the other chars
        double number = 0;
        for(int j=2;j  <  11;j++ )
        {
            char s3  =   s.charAt(j);
            int  l = Character.getNumericValue(s3);
            int k = j-1;
            int power = point-k;
            //number1 = l * 10^power
            double number1 =  l * Math.pow(10,power);
            number = number + number1;
        }
        if (sign == 0) return (-1) *number;
            else return  number;
    }

    //The method encodes the equation = 0 to solve.
```

```
//Perfect solution must ne a root and output zero.
//Other  numbers represents an error
private static double functionvalue(double b)
{
    //Root = 0.75;
    return 4*b+3;
    //Root = root = -4.71817809;
    //return b*b*b*b*b + 4*b*b*b*b - 3*b*b*b + 2*b*b +b +1;
    //Root = 1;
    //return b*b - 2*b + 1;
}


//The fitness of each individual is calculated
private static void fitness()
{
   for(int i = 0; i< POPSIZE; i++)
    {
        // Individual[i] is a string,
        // it is a genotype.
        // b is a number: it is the phenotype
        double b = decoder(Individual[i]);
        double  fV = functionvalue(b);
        //Fitness(root) = 1
        //Others have Fitness less than 1
        Fitness[i] =    1/(1+fV * fV);
    }
}


//Strings are sorted by fitness
private static void Sorting()
{

    fitness();

    for(int i = 0; i< POPSIZE;i++)
    {
        int champ = 0;
        for(int j = 0; j< POPSIZE; j++)
```

```
            if (Fitness[j] >= Fitness[champ])   champ = j;
        //The array Order classifies individuals by fitness.
        //The fittest, with less errors, go to the 0-th entry.
        Order[i] = champ;
        Fitness[champ] = 0;
    }


    String s = Individual[Order[0]];
    double b = decoder(s);
    double p = functionvalue(b);
    System.out.println("Gen = " + Generation + " best solution  = "
    + b + " Error = " + p );
}


//The top   fill the world
private  static void Cloning()
{
    //Array to host cloned individuals
String Individual2[ ] = new String[100];
int   i=0;
int   k=0;         //rank, 0 is the highest
//Cloning
while (i<POPSIZE)
{
    int counter = numberOfClones;
    while ((counter>0) & (i<POPSIZE))
    {
        Individual2[i] = Individual[Order[k]];
        i++;
        counter= counter-1;
    }
    k++;
}


//Cloned individuals become the official population
for(int j = 0; j < POPSIZE; j++)
    Individual[j] = Individual2[j];
```

```
    if (PRINT) printPopulation();
    }

    private static void Reproduction()
    {
        //Individuals are sorted by fitness
        Sorting();
        //for(int i = 0; i<  10;i++)
        //System.out.PRINTln(  Individual(Order(i))

        //Fitest individuals are cloned
        Cloning();
    }


private static  String change(String a)
{
    for(int l = 0; l < a.length(); l++)
    {
        double s = RAND.nextDouble();
        if (s < MUTATIONRATE)
        {
            if (a.length()==1) return ""+randomChar();
            else
            {
                //In case of a ghost
                if ( (l<0) || (l > a.length()-1))
                {

                    return a;
                }
                a = a.substring(0, l) + randomChar()
                    +  a.substring(l + 1);
            }
        }
    }
    return a;
}
```

```
    private static void Mutation()
    {
        for(int i=0; i < POPSIZE; i++)
       if (i != Champ) Individual[i] = change(Individual[i] );

    }



    private static void Dynamics()
    {
        //The top ten are allowed to  reproduce:
        //a child occupies the place of the shortest individual
        Reproduction();
        //The new population is subjected to mutation
        Mutation();
    }


    private static void printPopulation()
    {
        //Cloned individuals become the official population
    for(int j = 0; j < POPSIZE; j++)
    {

        double b = decoder(Individual[j]);
        double  fV = functionvalue(b);
        double    myFitness =    1/(1+fV*fV);
        System.out.println( j + "  ->  "  + Individual[j] + "  ->  " +
myFitness);

    }
    System.out.println();
    }


    public static void main(String[] args)
    {
        //To modify POPSIZE,
```

```
        // the method Copying()
        // must also be modified.
        Initialization( ) ;
        nGen = 400000;
        numberOfClones  = 10;
        //System.out.println("Root of 4*x + 3 = 0,  x = 0.75");
        for(int Gen = 0; Gen < nGen; Gen++)
        {
            Generation = Gen;
            Dynamics();
        }
    }
}//End of Program A233 Equation
```

234 ***Exercise*** *The previous program uses random mutation to find a solution to the equation* $4x+3=0$*. Verify that the error is of fifth order. Play with the code.*

235 ***Exercise*** *The previous program implemented mutation alone. Add recombination to get a full fledged simulation of evolution.* Answer

236 ***Exercise and research*** *Modify the code to solve the next equations and verify the answers:*

a) $7y+3=0$

b) $x^2+4x+4=0$

c) $x^5+4x^4-3x^3+2x^2+x+1=0$

d) Why is recombination important? Let us test the following theory: recombination is helpful to scape from local optima. The idea is that evolution is like finding the summit in within a landscape with many hills. One can climb a hill to find a ***local optimum****, a maximum that is not the global optimum. This becomes a point of* ***stagnation*** *of the genetic algorithm. How can we scape from this trap? In reality, the trap is created by selection that looks for high values to the expense of killing novelties. But the recurrent role of mutation can drive one outside visited hill into another. Now, recombination has two roles: it destroys what it has been achieved and it proposes a new solution built from partial ones. So, recombination can drive one to another hill and eventually to a high point. Is that true? Verify that without recombination one gets trapped in a local optimum in the third case but not in the first two. Run all 3 problems with recombination on. Does recombination help to find the solution faster? Or maybe you would like to create more clear cut examples.* Answer

237 ***Randomness as a spoiler***

We have seen that randomness can be invoked as the ultimate explanation of any fact: randomness contains a bunch of solutions to all problems. Moreover, we have seen that this is not a philosophy but that it is a crude reality. One can include randomness in the core of a process, such as evolution, that can be applied to solve algebraic and differential equations. Indeed, today evolution is used to solve the most varied type of problems.

Anyway, let us make it also clear that randomness is a great spoiler, so much that it denies the essence of owns existence. To see it, let us simulate a source of random numbers with a uniform distribution and let us verify that with certain frequency, the source fails, by mere randomness, to behave as it is due. This can be seen in the next program.

```
//Program A237 SelfSpoiler
//From time to time
//randomness betrays itself.
//We simulate a die with a uniform
//distribution, but in some cases,
//it generates data that
//do not fit a uniform distribution.
//The program reports:
//demanded length of a random sequence
//that does not fit a uniform distribution
//and
//the number of trials that were
//necessary to get such a sequence.

package ejvol1;



import java.util.Random;


public class SelfSpoiler
{
    // Turn on of the random generator
    static Random r  = new Random();
    // long is a very long integer
    static long trials ; // Number of trials.
```

```
    static int Observed[ ];//Predeclaration

    // The following is a chi2 table.
    // Signifcance 0.05.
    static double  Chi[]  = {
                            3.84,
                            5.99,
                            7.81,
                            9.48,
                            11.07,
                            12.59,
                            14.06,
                            15.5,
                            16.91,
                            18.30,
                            19.67,
                            21.02,
                            22.36,
                            23.68,
                            24.99,
                            26.29,
                            27.58,
                            28.86,
                            30.14,
                            31.41,
                            32.36,
                            33.92,
                            35.17,
                            36.41,
                            37.65,
                            38.88,
                            40.11,
                            41.33,
                            42.55,
                            43.77};

private static double  observedChisquare(int length, int nFaces)
{
    double chi = 0;
```

```
    //The die with five faces is associated to a distribution
    //that assigns a probability 1/5 to numbers 1 to 5.
    //Hence, a sequence  of landed faces
    //must have 1/5*length instances of each face.

    double Expected = length/nFaces;
    //Expected distribution is compared to observed one.
    //Discrepancy is approximately distributed
    //as a chi square:
    for(int j = 1; j<= nFaces; j++)
        chi  = chi + (Observed[j]-Expected)
            *(Observed[j]-Expected)/Expected;
    //System.out.println("Observed chi-square = " + chi);
    return chi;
}


public static void tryAndTry(int length, int nFaces)
 {
    // We count the number of trials
    //necessary to produce for the first
    // time a sequence that does not fit
    // a uniform distribution.
    trials = 0;

    // done = 1 is a flag that says
    //when we have succeeded
    int done = 0;

    while (done==0 )
    {
        trials = trials +1;
        //System.out.println(" Trial " + count);

        // Initialization
        for(int j = 1; j<= nFaces; j++)
            Observed[j] = 0;

        // We generate a random sequence
        for(int j = 1; j<= length; j++)
```

```
        {
            // We choose a uniform distribution
            // from 1 to nFaces, as in a die.
            int  c = r.nextInt(nFaces)+1;
            //System.out.println(c ) ;
            //book keeping
            Observed[c] =Observed[c]+1;
        }

        // Report
        // for(int j = 1; j<= nFaces; j++)
        //System.out.println(j + "\t" +Obs[j]);
        //We have succeed if
        // the distribution is not uniform.
        //Degrees of freedom: nFaces -1;
        //but Java counts from zero, hence
        int df = nFaces -2;
        /*System.out.println(chisquare(length,nFaces)
          + "\t "+ Chi[df]); */
        if (observedChisquare(length, nFaces) >= Chi[df])
        done = 1;
    }
 }

public static void main(String[] args)
{
    //Number of faces, nFaces, of the die
    //must be less than 30.
    int nFaces = 6;
    // For each die, we generate
    // sequences of increasing length,
    // which is defined by length,
    // which is a multiple of nFaces.
    int rounds = 3000;

    System.out.println("Length and trials to fail to be uniform");
    for(int j = 1; j<= rounds; j++)
    {
        //length =size of the random sequence
        int length= nFaces*j;
```

```
        //Obs contains the frequency table.
        Observed = new int[51];
        //Let us make the simulation
        tryAndTry(length,nFaces);
        //System.out.println("Die with " + nFaces + " faces");
        //System.out.println("Sequences of length " + length);
        /* System.out.println("We succeed at the trial "
           + trials ); */

        System.out.println(  length + "\t " + trials);
    }
}
}//End of Program A237 SelfSpoiler
```

238 ***Exercise*** *Decipher the code and play with it. Make a graphic of the relation of length of the sequence vs. the necessary trials to produce a sample that fails to fit a uniform distribution. Also change the number of faces of the simulated die. Draw your conclusions about the form as a source with a uniform distribution fails to generate a sequence that fits a uniform distribution.* Answer

239 ***The duality***

We already have reasons to believe that randomness can be the natural explanation of everything but at the same time we have reasons to believe that randomness can spoil everything even its own essence. Concretely: a sample of a source with a given distribution may frequently fail to fit the mother distribution. Between these two faces of randomness, which is more important? In nature both aspects are important as well as in applications.

Technical jargon: the overall lesson of our applications is that one can see at a **random mutation as noise**, as in information theory, and as a **computing operation**, as in computing science. Both aspects play an important role in biology: most mutations are deleterious, pointing to their capability to create noise and errors, but mutations reside at the adaptive capability of microorganisms to battle antibiotics. In hindsight, one can say that evolution calculates the form to defeat chemical weapons. So, we can say in technical terms that the genome is a parallel distributed computing system.

13.4 Conclusions

Randomness is at the core of modern science and a theme that is always controversial. Randomness is so controversial that it also can cope with determinism,

such as in the law of large numbers. Randomness is the null hypothesis of science and its power resides on one hand in the fact that predictions of ordinary quantum mechanics are probabilistic and on the other that deterministic dynamical systems can behave erratically and are best described by statistical descriptors. Randomness happens in nature in a typified form that is represented by a distribution. This fact allows the existence of science that aims at choosing the best model among various proposals that use randomness as a fundamental ingredient of its quantitative explicative mechanisms. Randomness is also present in mutation. Information theory sees mutation as noise as in deleterious changes but computing science sees mutation as computing operations to solve problems as in adaptive mutations. We have shown, in particular, that mutation can be framed into a computing machinery to solve mathematical problems.

Chapter 14

Markov chains

`A first probabilistic model.`

240 ***Motivation****: We have claimed that randomness comes typified and that this characteristic allows us to make science, i.e. to formulate quantitative measurements, models and predictions. But we as yet do not know how. In this chapter we see a first example: we simulate a Markov Chain, a probabilistic model that is mathematically well understood, so we have the means to mathematically test our results.*

241 ***Purpose***

When the complexity of a studied system is too high, a deterministic approach could be very expensive, and so a **statical description** of its behavior and a **probabilistic formulation** of its models could do it much better. Our purpose in this chapter is to get acquainted with the simulation of Markov chains, a very popular probabilistic model that has various applications to biology.

14.1 Implementation

242 ♣ ***Definition.*** *A **Markov chain** is a model about nature in which one studies a system, which may occupy one among a finite number of states and that the transitions among states is both random and with a constant probability.*

243 ***Example and exercise***

Let us consider a system that may occupy one of two states. State one is A, and state two is B. The transitions probabilities are

$p(A \to B) = 0.2$
$p(A \to A) = 0.8$
$p(B \to A) = 0.3$
$p(B \to B) = 0.7$

Our aim is to understand the behavior of the system in the long run for a given initial condition.

Let us consider a total population of 3000 individuals. The initial distribution is given by $A_o = 1000$ and $B_o = 2000$. In probabilistic terms, this corresponds to an initial probability of A equal to 1/3 and that of B equal to 2/3. Let us simulate this Markov chain in the next program, where we use the following conventions

p_{aa}= transition probability from state A to state A in one generation to the next.
p_{ab}= transition probability from state A to state B in one generation to the next.
p_{ba}= transition probability from state B to state A in one generation to the next.
p_{bb}= transition probability from state B to state B in one generation to the next.
p_a initial probability for state a.
p_b initial probability for state b.
The program that simulates that chain follows and must be run:

```
//Program A243 Markov
//A Markov Chain

package ejvol1;



import java.util.Random;

public class Markov
{

    static int X[] = new int[3000];
    //Probabilities
    static double  pa,paa,pb,pbb;

    //Turn on of the random generator
    static Random r  = new Random();

//Define probabilities of transition
```

```
public static void initialization()
{
    //Independent probabilties
    paa = 0.8;
    pbb = 0.7;
    // initial conditions
    pa = 0.25;
    pb = 0.75;
}

 public static void main(String[] args)
 {
    initialization();
    // Generation of 3000 individuals
    // some of type a other of type b
    int na = 0;
    int  nb = 0;
    for(int i = 0; i< 3000; i++)
    {
        double s = r.nextDouble();
        if (s < pa)  na = na + 1;
         else nb = nb + 1;
    }


    //Report the number of individuals in each state
    System.out.println("Initial condition");
    System.out.println("The number of individuals in " +
                       "state a is  " + na);
    System.out.println("The number of individuals in " +
                       "state b is  " + nb);
    //Execute 1000 transitions
    for(int i = 0; i< 1000;i++)
    {
        //Decide whether or not an individual
        //in state a must make a transition
        for(int j= 1; j<=na;j++)
        {
            double s = r.nextInt();
            if (s < paa)
```

```
            {
                na = na - 1;
                nb = nb + 1;
            }
        }

        //Decide whether or not an individual
        // in state b must make a transition
        for(int j= 1; j<=nb;j++)
        {
            double s = r.nextDouble();
            if (s < pbb)
            {
                nb = nb - 1;
                na = na + 1;
            }
        }
    }
    System.out.println("Final condition");
    System.out.println("The number of individuals in " +
                        "state a is  " + na);
    System.out.println("The number of individuals in " +
                        "state b is  " + nb);
    }
}//End pf Program A243 Markov
```

244 ***Analysis***

The code was run with diverse initial probabilities, p_a of state A. The number of individuals in state A with the corresponding relative frequency f_a are given in the next table.

Markov Chains		
p_a	n_a	f_a
0.10	1780	0.593
0.20	1772	0.591
0.30	1815	0.605
0.40	1800	0.6
0.50	1831	0.610
0.60	1784	0.595
0.70	1790	0.597
0.80	1828	0.609
0.90	1800	0.6

The mean value for n_a is 1800, which corresponds to a proportion of 1800/3000= 0.6.

The table seems to predict that the limit does not depend on the initial conditions. This implies that a regression analysis of the limit against the initial condition would render a slope that is zero for all purposes. The Anova to test whether or not the slope of the relation (p_a, f_a) is different than zero was calculated and given in the following table:

Anova for regression					
Variation	DF	SS	Mean Square	Fisher	Critical F
Regression	1	0.09437037	0.09437037	1.306475242	5.59
Error	7	0.50562963	0.07223280		
Total	8	0.6			

This test detects no significant departure from slope zero: observed frequencies can be taken as estimators of the same constant value.

245 ***There is a limit***

We clearly see that the initial condition does not influence the final state of our given Markov chain. We may postulate that a Markov chain has a particular asymptotic state to which the system tends no matter from which initial condition it is set on.

246 ***Complaint.*** *How do we know that after 1000 generations we already achieved the final state?*

247 ***Exercise*** *Modify the previous program to make a report each 100 generations. Verify that by generation 100 we already have attained the asymptotic state. Make a graphic of the dynamics.* Answer

248 ***Exercise*** *Since we know that the system tends to the equilibrium before the 100th generation, instruct Java to report the first 100 generations. Make the corresponding graphic. Verify that for this particular Markov chain the equilibrium has been attained by generation 20. Try various runs to get a complete scenario for the evolution of the system beginning from different initial conditions.* Answer

249 *Equilibrium else forces from outside*

Our simulation allows us to postulate that any Markov chain tends toward an equilibrium and that the system arrives near the equilibrium in the very first generations. This implies that if a Markov chain describes a natural system, it is near equilibrium with a high probability. Otherwise, there shall exist forces that maintain the system far from equilibrium.

250 *Complaint*

We have reasons to think that a Markov chain attends its equilibrium rather soon. But, how can we exclude out the possibility of false equilibriums that possibly are similar to false solutions of an equation, as it was previously found?

Questions of this sort are best dealt with mathematics. So, let us investigate the general form of equilibrium for our particular Markov chain, if any.

Let us use the following conventions

p_{aa}= transition probability from state A to state A in one generation to the next.
p_{ab}= transition probability from state A to state B in one generation to the next.
p_{ba}= transition probability from state B to state A in one generation to the next.
p_{bb}= transition probability from state B to state B in one generation to the next.

This implies that the proportions evolve according to the following equations:

$$x_a(t+1) = p_{aa}x_a(t) + p_{ba}x_b(t)$$
$$x_b(t+1) = p_{ab}x_a(t) + p_{bb}x_b(t)$$

For our example, $p_{aa} = 0.8$ so, $p_{ab} = 0.2$ and $p_{bb} = 0.7$ so $p_{ba} = 0.3$ Then, we get

$$x_a(t+1) = 0.8x_a(t) + 0.3x_b(t)$$
$$x_b(t+1) = 0.2x_a(t) + 0.7x_b(t)$$

In equilibrium, the occupation proportion of any state at $t+1$ coincides with that at t. Hence, the proportions at equilibrium obey the equations

$$x_a = 0.8x_a + 0.3x_b$$

$x_b = 0.2x_a + 0.7x_b$

Replacing $x_b = 1 - x_a$ in the first equation, we get

$x_a = 0.8x_a + 0.3(1 - x_a) = 0.8x_a + 0.3 - 0.3x_a$
$x_a = 0.5x_a + 0.3$
$0.5x_a = 0.3$
$x_a = 0.3/0.5 = 3/5 = 0.6$

251 ***Is the limit unique?*** *This shows that our particular Markov chain tends to equilibrium, which is unique.*

Thus, if we have 3000 individuals, the expected number of individuals in state A when the system is in equilibrium shall be 9000/5= 1800. Our experimental results render values with mean precisely 1800. So, we have witnessed that the random generator and the numeric precision of Java are really good!

252 ***Exercise*** *Give one example in which a Markov chain has too many equilibria. Is the limit unique?* Answer

253 ***Exercise*** *Shows that when the equilibrium state of a Markov chain is unique, it is stable, i.e., any perturbation is balanced throughout the dynamics. Show that this does not happen for the Markov chain whose transition matrix is the identity.* Answer

254 ***Exercise*** *In certain simulation of a Markov chain, it was obtained a proportion of individuals in state A equal to 0.688 when the expected proportion was 0.6. Is that acceptable or is a witness against the quality of the used random generator?* Answer

255 ***Challenge*** *Simulate a Markov Chain with 3 states and study its evolution. Work out the mathematical predictions and compare them with your simulations.*

256 ***Improving the style***

No one is able to produce at once a program that makes what it must. Every developer tries many times before arriving to a correct solution. Experienced developers write some 60 lines of debugged code per day. That is too much. Usually, one tends to refine the style of the software after one sees that it works. But experience has shown that a good style of programming is very useful to diminishing

the suffering of software development. Experts write advices of very diverse flavor, a discrepancy pointing to the extreme hardness of software design. In fact, the modern programming languages come with environments specially suited to ease the debugging process.

A good style is always said to include encapsulating separate simple tasks into different methods. For the last program, the ensuing structuring might be as follows:

```
//Program A256 Markov4
//Markov Chains
//There is a rapid convergence to the equilibrium state.
//Encapsulated code.

package ejvol1;

import java.util.Random;


public class Markov4
{

    static private int X[] = new int[3000];
    //Probabilities
    static private double  pa,paa, pbb;
    static private int na = 0;
    static private int nb = 0;

    //Turn on the random generator
    static Random r  = new Random();

    //Define probabilities of transition.
    private static void setParameters()
    {
        //Independent probabilties
        paa = 0.8;
        pbb = 0.7;
        // initial conditions
        pa = 0.25; //pb = 1-pa;
    }
```

```
private static void initialization()
{
    // Generation of 3000 individuals
    // some of type a other of type b

    for(int i = 0; i< 3000; i++)
    {
        double s = r.nextDouble();
        if (s < pa)  na = na + 1;
         else nb = nb + 1;
    }

    //Report the number of individuals in each state
    System.out.println("Initial condition");
    System.out.println("The number of individuals in " +
                       "state a is  " + na);
    System.out.println("The number of individuals in " +
                       "state b is  " + nb);

    System.out.println("Generation,    populations in "
                                    +"states A and    B  " );
}

private static void dynamics()
{
//Execute 1000 transitions
    for(int i = 0; i< 100;i++)
    {
        //Decide whether or not an individual
        //in state a must make a transition
        for(int j= 1; j<=na;j++)
        {
            double s = r.nextInt();
            if (s < paa)
            {
                na = na - 1;
                nb = nb + 1;
            }
```

```
        }

        //Decide whether or not an individual
        // in state b must make a transition
        for(int j= 1; j<=nb;j++)
        {
            double s = r.nextDouble();
            if (s < pbb)
            {
                nb = nb - 1;
                na = na + 1;
            }
        }
        System.out.println( i + "\t"    + na + "\t"   + nb);
    }
}

private static void report()
{
    System.out.println("Final condition");
    System.out.println("The number of individuals in " +
                         "state a is  " + na);
    System.out.println("The number of individuals in " +
                       "state b is  " + nb);
}


public static void main(String[] args)
{
    setParameters();
    initialization();
    dynamics();
    report();
}
}//Program A256 Markov4
```

257 ***Intrigue***

Observe the changes that were needed to just achieve encapsulation. Encapsulation creates many troubles for the developer of software. Why do developers prefer to

face up these problems? It is because this generates a style that easies debugging, the unavoidable process of correcting errors in the design of software. The same problem is seemingly faced by the genome, the natural epitome of software: Why?

14.2 Conclusion

We have simulated the simplest case of a Markov Chain, a well-known mathematical object. Our results were confronted with mathematical predictions, and no discrepancy was evidenced. In a sense, we have made science: a mathematical model correctly describes a digital world.

Chapter 15

Lotka Volterra

`Wolfs and rabbits.`

258 ***Motivation*** *One of our general purposes is to understand evolution and to learn to assess its possibilities to explain life in all its beauty, splendor and glory. In this regard, one delicate problem is to explain* ***why we are so perfect****.*

The scholastic explanation says that competence among the members of the same group is the fuel that boosts the escalate towards perfection. Nevertheless, other proposals also exist. One of them resorts to a **prey-predator system**: if you are under danger to be caught and killed, you must invent something to escape. At the same time, if it happens that you are the owner of a genetic variation that appeared by mere randomness and that is good for escaping, then you could enjoy a longer life. Additionally, if you can reproduce and if your genetic characteristics can be passed on to your offspring, then your children will also live and fill the earth.

Something similar will happen to the predator. After many generations of this struggle for live and let die, both species will be highly competitive.

Thus far, so good. Our task is to convert this nice story into a scientific project, whose first level of inquiry is the stability of the populations: since killing is at the very heart of the process, can we guarantee that life will remain? Can we claim that life will exist long enough to allow for **evolution of competiveness, of perfection, of complexity**?

259 ***Our purpose***

We simulate a **Lotka-Volterra** prey-predator system, which is a very good stage to exhibit the problem of stability of the populations. We begin with the deterministic model to pass next to consider an stochastic version. We close with a logistic correction.

15.1 Deterministic model

260 ***Example*** *The fundamentals of Lotka Volterra.*

The **deterministic Lotka Volterra model** is as follows: we have a prey, say rabbits, whose number is x, and a predator, say, wolfs, whose number is y. The prey feeds on grass, which is always abundant. Thus, the number of offspring of rabbits that is born per unit time is independent of food but it is proportional to the number of rabbits (all rabbits can reproduce and sex is forgiven). The constant of proportionality is m.

Predation is the only cause of death among rabbits. The number of dead rabbits per unit time is proportional to the number of encounters among rabbits and wolfs, which in its turn is proportional to the population numbers of both species. The constant of proportionality is p. Encounters represent the only possible source of food for the predator, from which reproduction ensues. The constant of proportionality of birth of wolfs per unit time is l.

On the other hand, wolfs die at natural death. The number of dead wolfs per unit time is proportional to their censal number with constant of proportionality k.

From what we have said, the deterministic model obeys the following system of differential equations, in which all constants are positive:

$$\frac{dx}{dt} = mx - pxy$$
$$\frac{dy}{dt} = lxy - ky$$

Animals are best counted with integer numbers but our differential system assumes that fractions are also valid, so we need to round results.

This system of differential equations is solved in the next program:

```
// Program 260 LotkaVolterra
// Study  of a prey-predator system.
// Deterministic model.
// Lotka-Volerra equations.

package ejvol1;



public class LotkaVolterra
{
    static double nr, nw, e, m,p,l,  k;
```

```
public static void initialization()
{
    //Generates 500 rabbits,  50 wolfs
    nr = 500;
    nw = 50;
    // Rate of rabbit's reproduction
    m = 0.1;
    // Rate of predation
    p = 0.01;
    // Rate of wolf's death
    k = 0.05;
    // Rate of wolf reproduction
    l= 0.1 * p ;
}

public static void dynamics()
{
    //We run N generations
    int n = 1000;
    for(int  i = 0; i< n;i++)
    {
        //Rabbits reproduce
        nr = nr + m* nr;
        //Rabbits are preyed on
        nr = nr - p * nr * nw;
        //Wolfs reproduce
        nw = nw + l* nr * nw;
        //Wolfs die
        nw = nw - k * nw;

        System.out.println(i + "\t"
                +  nr + "\t " +  nw);
    }
}

public static void main(String[] args)
{
    initialization();
    System.out.println("time, rabits and wolfs");
```

```
        dynamics();
    }
}//End opf Program 260 LotkaVolterra
```

261 ***Exercise*** *Run the code for diverse initial conditions and parameters. Notice and explain the oscillations of the populations, when they exist.*

262 ***Exercise*** *Use Excel or Calc to graphic the dynamics of the output. Consider the possibility that our notation of decimal numbers is not compatible with some versions of Excel. A adjust parameters leading to the stability of the system. Hint: the graphic of preys vs predators will form a close curve.* Answer

263 ***Exercise*** *How easy is to get an stable system? Are stable setting highly probably?* Answer

15.2 A stochastic version

Previous model is deterministic. Let us face to a stochastic version.

264 ***Example*** ***A discrete sthocastic Lotka Volterra model.***

The system of equations that describes the prey-predator system could be given a stochastic interpretation if only the fundamental process is given a random interpretation. But beware, if a wolf exists it is not a random entity. It is absolutely sure that it exists. The same happens with the rabbits. By contrast, the processes of birth, death or hunting may or may not produce results: they are best modeled by random variables. A program that depicts these characteristics is below. It includes the next piece of code that presents a terrible complication:

```
for(int j = 1; j<= nr; j++)
{
// The process of reproduction
// is ruled by a binomial
// random variable
double s = r.nextDouble();
if (s < m){ nr = nr + 1;}
}
```

Let us observe that the parameter *nr* enters both as a control statement, in the first line of the for-loop, and as a modifiable property in within the code. This causes a self reference looping that in general is very difficult to manage. To avoid problems, this possibility is ruled out in other programming languages. Anyway, Java allows one to use it, so we must use that right with responsibility. How do we solve this problem? We use small values of *r* and so born rabbits per generation are not too many and the ensuing perturbation is tolerable.

Good news: numbers of rabbits and wolfs are given the type integer, such as it is due. The complete code follows:

```
// Program A264 LotkaVolterra3
// Study  of a prey-predator system.
// Lotka-Volterra equations.
// Stochastic model.

package ejvol1;

import java.util.Random;


public class LotkaVolterra3
{
    static int  nr, nw;
    static double e,rr,p,wr,wd;

    //Turn on of the random generator
    static Random r  = new Random();

    public static void initialization()
    {
        //Generates 500 rabbits,  50 wolfs
        nr = 500;
        nw = 50;
        //Rate of rabbit's reproduction
        rr = 0.1;
        //Rate of predation
        p = 0.01;
        //Rate of wolf's death
        wd = 0.05;
```

```
        // Rate of wolf's reproduction
        wr = 0.7 * p ;
        //Titles
    }

    public static void dynamics()
    {
        //We run n generations
        int  n = 2000;
        for(int i = 0; i < n; i++)
        {
            //Rabbits reproduce
            for(int j = 0; j < nr; j++)
            {
                // The process of reproduction
                // is ruled by a binomial
                // random variable m
                double s = r.nextDouble();
                if (s < rr) nr = nr + 1;
            }
            //Rabbits are preyed, wolfs reproduce
            //Binomial random variables:
            //an encounter may or may not
            //produce a chase.
            for(int l = 0; l < nw; l++)
            {
                for(int j = 1; j < nr; j++)
                {
                    double s = r.nextDouble();
                   // a rabbit is predated
                    if (s < p) nr = nr - 1;
                    // a wolf is born
                    if (s < wr)  nw = nw + 1;
                }
            }
            //Wolfs die
            for(int j = 0; j < nw; j++)
            {
                double s = r.nextDouble();
                if (s < wd)  nw = nw - 1;
```

```
            }
            System.out.println(i + "\t" + nr + "\t " + nw);
        }
    }

    public static void main(String[] args)
    {
        initialization();
        System.out.println("time, rabits and wolfs");
        dynamics();
    }
}//End of Program A264 LotkaVolterra3
```

265 ***Exercise*** *Run the code and make your best to understand it.*

266 ***Challenge*** *Make an experimental program of investigation of this virtual world. Pay special attention to the extinction problem. Compare the deterministic and the stochastic models. Hint: to study this world, such as it has been programmed, is a long time consuming activity. It would be more intelligent to reprogram that world with the inclusion of automatic registration of suitable descriptors.*

15.3 A logistic correction

Our stochastic model seems to be unstable. How can we help?

267 ***Example*** *Resources are limited.*

One observes that the virtual world that we have built is unstable: a spur of rabbits always appears and after that wolfs reproduce too much and then rabbits disappear. To make that world stable, we could cause rabbits to reproduce only in the measure that reproduction is allowed by finite resources. Our new setting is the **sthocastic logistic Lotka Volterra model**.

```
// Program A267 LotkaVolterra4
// Study  of a prey-predator system.
// Logistic Lotka-Volterra equations.
// Stochastic model.
```

```
package ejvol1;

import java.util.Random;


public class LotkaVolterra4
{
    static int  nr, nw, max;
    static double e,rr,p,wr,wd;

    //Turn on of the random generator
    static Random r  = new Random();

    public static void initialization()
    {
        //Generates 500 rabbits,  50 wolfs
        nr = 500;
        nw = 50;
        //Rate of rabbit's reproduction
        rr = 0.1;
        //Maximum number of Rabbits
        max = 2000;
        //Rate of predation
        p = 0.01;
        //Rate of wolf's death
        wd = 0.05;
        // Rate of wolf's reproduction
        wr = 0.9 * p ;
        //Titles
    }

    public static void dynamics()
    {
        //We run n generations
        int  n = 2000;
        for(int i = 0; i < n; i++)
        {
            //Rabbits reproduce
            for(int j = 0; j < nr; j++)
```

```
            {
                // The process of reproduction
                // is ruled by a binomial
                // random variable m
                double s = r.nextDouble();
                if (s < rr * (1 - (nr / max))   ) nr = nr + 1;
            }
            //Rabbits are preyed, wolfs reproduce
            //Binomial random variables:
            //an encounter may or may not
            //produce a chase.
            for(int l = 0; l < nw; l++)
            {
                for(int j = 1; j < nr; j++)
                {
                    double s = r.nextDouble();
                   // a rabbit is predated
                    if (s < p) nr = nr - 1;
                    // a wolf is born
                    if (s < wr)  nw = nw + 1;
                }
            }
            //Wolfs die
            for(int j = 0; j < nw; j++)
            {
                double s = r.nextDouble();
                if (s < wd)  nw = nw - 1;
            }
            System.out.println(i + "\t" + nr + "\t " + nw);
        }
    }

    public static void main(String[] args)
    {
        initialization();
        System.out.println("time, rabits and wolfs");
        dynamics();
    }
}//End of Program A267 LotkaVolterra4
```

268 ***Exercise and Challenge*** *Run the code and verify that it gets stable, in the sense that it allows for the existence of an ecosystem. Is it stable against any perturbation caused by the variation of parameters? Would we need to purge wolfs when they get too many? How would we do that?*

15.4 Conclusion

We have simulated various flavors of the Lotka-Volterra prey-predator model. This model is important in ecology, but its importance in evolution is also outstanding: the model allows the possibility of evolution of perfection and of complexity. The first requisite that this could happen is that populations could exist during long time. In this regard, we have proved that this family of models produces stable systems. The door is now open to study more sophisticated questions such as the evolution of perfection as a subsidiary effect.

Chapter 16

Moths

`Testing evolution thanks to moths.`

269 *Motivation*

A moderately wild world hardens our heart, making us more recursive, more intelligent and more happy. Can we aspire to explain some folds of the complexity of life, of its overwhelming complexity, as a response to **environmental challenges**? Our hope is that a more varied environment could favor the variability of possible responses and the ensuing **evolution of complexity**. But there is a great separation between our hope and a scientific treatise of this theme. So, let us give a first lap: let us make a simulation in which some genetic responses may be favored by the environment when it has some properties while other genetic responses may be favored under other properties of the environment. This is called **adaptation**.

270 *Our purpose*

In this chapter we will understand **evolution as the differential surviving of the members of a population in response to environment pressures**. Everything gets clear with a single example. To that purpose, let us reproduce a fact that played an important historical role in the supporting of the evolutionary theory: white moths survive predation by birds better than black ones in a clean and pale environment but worse in a blacked one.

Curiously enough, this case, that is in principle the most clean of all those that could support evolution, has resulted to be a source of tremendous discussions that have given the Peppered moth *Biston betularia* a great fame all around (Majerus [8], 2007).

16.1 A model without mutation

The fundamental ingredients of a real biological phenomenon are captured by our next model.

271 *Minimal setting*

We consider a population of white and brown moths. We suppose that the number of encounters among birds and moths is proportional to their population numbers. Next, the constant of proportionality is color dependent: in a clean environment, the constant of proportionality is greater for black moths than for white ones and in a contaminated environment, things happen the other way around.

We suppose that both types of moths reproduce in equal proportions and the reproduction observes a logistic model, which takes into account the role of ecological restrictions. We fix the maximum number of moths in 1000, counting both types. Birds are predators and we will suppose that they feed on moths only. For stability reasons, we also add a logistic grow for the number of birds. Our initial setting is a clean environment with white moths abounding and black ones depleted.

```
package ejvol1;


//Program A271 Moths

/*
 Study of predation of moths by birds.
 Moths could be white or brown.
 Reproduction obeys a logistic model
 Two environments: clean and dirty.
 Discrete stochastic logistic model.
 */

import java.util.Random;

public class Moths
{

    static private int nwm, nbrm, nb ;
```

```
static private double  mr;
static private double  maxM, maxB, pwm, pbrm,bd;
static private int max;
static private int n;
// Turn on of the random generator
static private Random r  = new Random();

public static void initialization()
{
    //Generates   moths,
    //white moths
    nwm = 450;
    // brown moths
    nbrm = 450;
    //Generates 10 birds
    nb = 10;
    //Probability of moth's reproduction
    mr = 0.1;
    //Maximum number of Moths
    maxM = 2000;
    //Maximum number of Birds
    maxB = 20;
    //probability of predation on white moths
    pwm = 0.001;
    //probability of predation on brown moths
    pbrm = 0.001;
    //Probability of bird's death
    bd = 0.05;
}


public static void dynamics()
{
    for(int i = 0; i < n; i++)
    {
        // White moths reproduce
        for(int j = 0; j< nwm; j++)
        {
            double s = r.nextDouble();
            if (s < mr * (1 - (nwm + nbrm) / maxM) )
```

```
        nwm = nwm + 1;
}

//Brown moths reproduce
for(int j = 0; j < nbrm; j++)
{
    double s = r.nextDouble();
    if (s < mr * (1 - (nwm + nbrm) / maxM) )
        nbrm = nbrm + 1;
}

//Moths are preyed, birds reproduce
for(int l = 0; l < nb; l++)
{
    //White moths
    for(int j = 0; j < nwm; j++)
    {
        double s = r.nextDouble();
        // a white moth is predated
        if (s < pwm) nwm = nwm - 1;
        //birds reproduce
        if ( s < 0.1 * pwm * (1 - nb / maxB))
          nb = nb + 1;
    }
    //brown moths
    for(int j = 0; j < nbrm; j++)
    {
        double s = r.nextDouble();
        // a brown moth is predated
        if (s < pbrm) nbrm = nbrm - 1;
        //birds reproduce
        if ( s < 0.1 * pbrm * (1 - nb / maxB))
            nb = nb + 1;
    }
}


//Birds die
for(int j = 0; j < nb; j++)
{
```

```
                double s = r.nextDouble();
                if (s < bd) nb = nb - 1;
            }

            System.out.println(i + "\t " +  nwm + "\t "
                + nbrm + "\t " + nb);
        }//end i for
    }


    public static void main(String[] args)
    {
        //We run n generations
        n = 1000;
        System.out.println("time, white moths, brown moths and birds");
        initialization();
        dynamics();
    }
}//End of Program A271 Moths
```

272 ***Exercise*** *Play with parameters to prevent the extinction of the disfavored type of moths. What does happen to oscillations in the population numbers? Prove that the program favors white moths just because white moths are the first to be reproduced. This is a positional bug.* Answer

16.2 A model with mutation

Field observation shows that both type of moths survive in spite of environmental pressures. How is that possible?

273 ***Mutation is necessary.*** *In principle, any tiny difference between the predation parameters should cause the disappearance of the disfavored type of moths. But, it is known that the disfavored type of moths always have representatives in the population. The only explanation is* ***mutation****: the offspring of a type of moth not necessarily belongs to the same type.*

The implementation follows, where we see for the first time the appearance of nested ifs.

```
//Program A273 Moths2
/*
    Study of predation of moths by birds.
    Moths could be white or brown.
    Reproduction obeys a logistic model
    Two environments: clean and dirty.
    Discrete stochastic logistic model.
    Mutation is included.
*/

package ejvol1;



import java.util.Random;


public class Moths2
{
    static private int nwm, nbrm, nb ;
    static private double  mr;
    static private double  maxM, maxB, pwm, pbrm,bd;
    static private int n;
    static private double mut;


    //Turn on of the random generator
    static private Random r  = new Random();

    public static void initialization()
    {
        //Number of generations
        n = 1000;
        //Generates   moths,
        //white moths
        nwm = 450;
        // brown moths
        nbrm = 450;
        //Generates birds
        nb = 10;
```

```
        //Probability of moth's reproduction
        mr = 0.1;
        //Mutation rate of moths
        mut = 0.01;
        //Maximum number of Moths
        maxM = 2000;
        //Maximum number of Birds
        maxB = 20;
        //probability of predation on white moths
        pwm = 0.001;
        //probability of predation on brown moths
        pbrm = 0.002;
        //Probability of bird's death
        bd = 0.05;
    }

    public static void dynamics()
    {
        for(int i = 0; i < n; i++)
        {
            // White moths reproduce and mutate
            for(int j = 0; j < nwm; j++)
            {
                double s = r.nextDouble();
                if (s < mr * (1 - (nwm + nbrm) / maxM) )
                {
                    //Reproduction
                    nwm = nwm + 1;
                    //Mutation
                    if (s < mut)
                    {
                        nbrm = nbrm + 1 ;
                        nwm = nwm-1;
                    }
                }
            }

            //Brown moths reproduce and mutate
            for(int j = 0; j < nbrm; j++)
            {
```

```
        double s = r.nextDouble();
        if (s < mr * (1 - (nwm + nbrm) / maxM) )
        {
            nbrm = nbrm + 1;
            if (s < mut)
            {
                nbrm = nbrm - 1 ;
                nwm = nwm+1;
            }
        }
    }// end j for


    //Moths are preyed, birds reproduce
    for(int l = 0; l < nb; l++)
    {
        //White moths
        for(int j = 0; j < nwm; j++)
        {
            double s = r.nextDouble();
            // a white moth is preyed
            if (s < pwm){ nwm = nwm - 1;}
            //birds reproduce
            if ( s < 0.1 * pwm * (1 - nb / maxB))
                nb = nb + 1;
        }
        //brown moths
        for(int j = 0; j < nbrm; j++)
        {
            double s = r.nextDouble();
            // a brown moth is preyed
            if (s < pbrm){ nbrm = nbrm - 1;}
            //birds reproduce
            if ( s < 0.1 * pbrm * (1 - nb / maxB))
                nb = nb + 1;
        }
    }
```

```
            //Birds die
            for(int j = 0; j < nb; j++)
            {
                double s = r.nextDouble();
                if (s < bd)
                    nb = nb - 1;
            }
            System.out.println(i + "\t " +  nwm + "\t "
                + nbrm + "\t " + nb);
        }//end i for
    }


    public static void main(String[] args)
    {
        initialization();
        System.out.println("time, white moths, brown moths and birds");
        dynamics();
    }
}//End of Program A273 Moths2
```

274 ***Exercise*** *Verify that with the included mutation, brown moths never disappear.*

275 ***Exercise*** *Develop a code to reproduce the following field observation. Initial conditions: clean environment, prevalence of white moths. After equilibrium has been achieved, external contamination causes a flip-flop of the selection parameters with the ensuing prevalence of brown moths.* *Answer*

276 ***Research*** *We discovered a positional bug in the first version of the simulation of moths. The addition of mutation to our processes alleviates to a very good extent the effects of that bug. Anyway, remake the program else design one on your own, which kills the bug once and forever.*

16.3 In hindsight

277 ***At the edge of chaos.*** *One knows by experience that it is easily to commit many errors in developing software. But the most important thing is to understand how can we know that we commit an error, a bug. In first place, it is because we*

have an aim that we follow. In second place, it is because any tiny change in the code causes a possible abrupt change in the resulting dynamics. We can say, using the technical jargon, that software executing is ***at the edge of chaos****: this means that we still know what we do and what we want, but if we deviate a bit, we pay an exorbitant price in the results.*

The same applies to genetic software: it is very good to produce mutations, if they are viable. In effect, minor changes in the genetic messages could cause exorbitant changes in the morphology or physiology of the carrier organism. But in nature, mutation has no aim, so the word bug has no meaning in nature. It seems that the only point of contact among human developers and the blind developer is that both have problems with viability: it is not easy to concatenate symbols that could produce a code that could be executed, be it in software development or in the genome.

278 ***Research*** *Investigate the rate of abortion in mammals and humans. How could we relate those rates with our problem of evolving viable code?*

16.4 Conclusion

Java has allowed us to nicely simulate various scenarios which appear in higher mathematics. In particular, we have simulated the evolution of the genetic composition of a system of white-brown moths that look how to survive in an environment that favors one color against predation by birds. We have proven that evolution is responsible for explaining the primacy of one color and for keeping the variability of the system. In spite of all the propaganda, nothing else can be claimed.

Chapter 17

A Java interface

`Attracting the user with a nice interface.`

279 ***Motivation and purpose.*** *We have seen that many of our programs eventually might be converted into a pedagogical laboratory for ecology or genetics or molecular biology. The problem is that the user must modify the code to change parameters. As a rule, this operation causes bugs that may kill the program. Thus, it is very good to know and to remember that Java offers various possibilities for isolating the code from the user and, more to the point, to achieve a friendly interface that could be visually very attractive.*

We learn in this chapter how to use an ***Applet*** *as an* ***interface*** *to capture data from the user. As a plus, we learn another style of programming.*

Warning: *The problem of designing a graphic interface with the User has been solved variously in Java by AWT, Applets, Swing and JavaFX. We will learn all of them to get a flavor of the evolution of software for a difficult problem. The order is as follows: the rest of this volume comes in AWT and Applets. Most of the material of next volumes comes in Swing, which is nice and reliable. But if someone is interested in just in the last and more powerful solution, he or she can go directly to the point and meet volume XII Fossils, in our series. The overlapping of Swing and JavaFX is very high so, after learning Swing one can very rapidly understand JavaFX. Nevertheless, one can learn JAVAFX directly using aforementioned Vol XII. The form to know what system is in use is to look at the imported resources, say,* `import javax.swing.JOptionPane;` *means that we deal with Swing.*

17.1 Dialogs

280 ***Purpose.*** *The next program shows how one can capture data using dialogs.*

```
//Program A280 Dialog
//Swing interface to capture data from the user.

package ejvol1;



import javax.swing.JOptionPane;
//Swing based code.

public class Dialog
{

public static void main(String[] args)
{
    // first string entered by user
    String string1;
    // second string entered by user
    String string2;

    // first parameter
    double parameter1;
    // second parameter
    double parameter2;

    // We open a dialog box;
    // We obtain   parameters from user;
    // parameters are read as   Strings.

    string1 = JOptionPane.showInputDialog(
       "Enter first decimal number. Use a Point." );

    // We obtain second parameter from user
    string2 = JOptionPane.showInputDialog(
       "Enter second decimal number. Use a Point." );

    // We convert   String to   double
    parameter1  = Double.parseDouble( string1 );
    parameter2  = Double.parseDouble( string2 );
```

```
    System.out.println("parameter1 = " + parameter1);
    System.out.println("parameter1 = " + parameter2);
}

}//End of Program A280 Dialog
```

281 ***Exercise*** *Run the code and verbalize its functioning.*

17.2 My first Applet

282 ***Drawing over an Applet.*** *The previous program offers a good solution to the problem of preventing unintentional damage to the code by the user whose only purpose could be to play with the already designed code. But, a simulation may require 10 or more parameters to be specified. One of course can adapt this method to deal with that situation. But nevertheless, the modern visual facilities of Java allow us to write all parameters over the same panel and to see them all together at the same time. Additionally, one is allowed to correct values at will. The corresponding technology is known as Applets.*

This is very good but, of course, we must pay a price in complexity. Let us do it bit by bit. The first peculiarity is that we do not write in an Applet, instead, anything must be drawn. That is why we must work with a paint toolbox.

```
//Program A282 DrawString
//My first Applet
//This applet opens a window
//and paints a message.

package ejvol1;



//Tools to draw:
import java.awt.Graphics;
import javax.swing.JApplet;

public class DrawString extends JApplet
{
```

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

    public void paint( Graphics g )
    {
       super.paint( g ); // intention of drawing
       // draw a message at coordinates (10,25)
       g.drawString( "This is my first Applet.", 10, 25 );
    }
}
```

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

284 ***Exercise*** *Modify the code to investigate where the origin of coordinates is and where the positive directions point to.* Answer

17.3 A new style of programming

It is very nice when one listens: How can I help you? This spirit is also in Java.

285 ***The AWT suystem.*** *One could marvel at the fact that our first Applet has not head, i.e., has no mean method. Additionally, its execution must be stopped by hand. Why? The reason is that we have here another style of programming, which is called event based programming. What does it consist in?*

In our previous style of programming, instructions were executed serially, one after another, by explicit invocation in within the code. We have no **serial execution** in the **event based programming** . Instead, we have a set of services that are ready to answer to the stimuli prompted by the user. He or she must click some button or fill in some field in a pane that is drawn on the display of the desktop.

Let notice that the **genome** is guided by a event based programming: different stimuli from the exterior or from inside fire events that are the brand of life.

Applets has no head, but they must have an initiation method, which is called init(). Let us get the next step into the development of our interface. The AWT API (abstract Window TooKit Application Programming Interface) was released in 1995. By that time, an applet might had looked like this:

```
package ejvol1;

//Program A285 Dialog2
```

```
//Event based programming.
//A dialog with two inputs.
//AWT + Applet based code.


//Silence this line to know wich items come from awt:
//they will appear in red.
import java.awt.*;
//Silence this line to know wich items come from Applet
import java.applet.Applet;


//We declare that the class must
//be armed with Applet's tools.
public class Dialog2 extends Applet
{
    private static final long serialVersionUID = 1L;

    //A label contains a message to the User
    private static Label resize;
    private static Label invitation1; //message to the user
    //A textField reads the input from the User
    private static TextField input1;  //input from the user
    private static Label invitation2;
    private static TextField input2;
    private static double num1, num2; //working variables
    private static Label sum ;
    private static Double result;


@Override
public  void   init()
{
    resize = new Label("Resize the applet for a better view!" );
    //Elements must be added to the Applet
    add(resize);
    //Input machinary
    //Declaration
    // A Label displays an invitation to operate
    invitation1 = new Label("Input first parameter ");
```

```
    //A TextField is a place for writing.
    input1 = new TextField( 10);
    invitation2 = new Label("Input second parameter + enter ");
    sum = new Label("The sum is " );
    input2 = new TextField( 10);
    //Activation
    add(invitation1);
    add(input1);
    add(invitation2);
    add(input2);
    add(sum);
}


//Report to the console
public void report()
{
    // Here we report results.
    // We use the console
    // because we expect large volumes of data
    System.out.println("Number 1 " + "\t " + num1 );
    System.out.println("Number 2" + "\t " + num2 );
    result = num1 + num2;
    System.out.println("The sum is" + "\t " + result );
}

//What to do if the User demands attention
//This code is outmoded and deprecated but is simple.
@Override
@SuppressWarnings( "deprecation" )
public boolean action(Event event, Object o)
{
    // If the User presses enter into the second field
    if (event.target == input2)
    {
        //Read textField1 and textField2
        //Read as String and translated to Double
        num1 = Double.parseDouble(input1.getText() );
        num2 = Double.parseDouble(input2.getText() );
        result = num1+num2;
```

```
        //The text of label sum is updated
        sum.setText("The sum is "  + result.toString());
        //Make a report to the console.
        report();
    }
    //We indicate to the system that the action
    //of the User has been processed.
    return true;
}
}//End of Program A285 Dialog2
```

286 ***Exercise*** *Run the code and verbalize it. Play with it. Clearly describe the methods that build inner structure, that listen to the User and that responds with actions. Explain why this style of programming is called event based. Verify that one can call into execution the previous program various times. Officially, it is said that a call initializes a* ***thread****. So, Java is a multi-thread platform. That is why various Applets can run at the same time and may overburden the CPU. So, one must take care of the awaken Applets and turn they off as soon as they are anymore necessary. To that purpose use the square in red.* Answer

287 ***Swing*** *The next proposal for Java in relation with interfaces was Swing. It was released by the end of 1996. Swing was designed to seem a soft change to the AWT system. For instance, an AWT Applet had JApplet as Swing equivalent. And so on. An example of code produced with Swing is the following:*

```
//Program A287 Dialog3
//Event based programming.
//A dialog with two inputs.
//AWT + JApplet + Swing based code.
package ejvol1;


import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JApplet;
import javax.swing.JLabel;
import javax.swing.JTextField;
```

```
//We declare that the class must
//be armed with JApplet's tools
//and that must include a facility to
//listened at the prompts of the User.
public class Dialog3 extends JApplet implements ActionListener
{
    private static final long serialVersionUID = 1L;

    //A label contains a message to the User
    private JLabel resize;
    private  JLabel invitation1; //message to the user
    //A textField reads the input from the User
    private  JTextField input1;  //input from the user
    private  JLabel invitation2;
    private JTextField  input2;
    private JLabel sum ;



@Override
public void init()
{
    this.resize = new JLabel("Resize the applet for a better view!" );

    //Input machinary
    //Declaration
    // A Label displays an invitation to operate
    this.invitation1 = new JLabel("Input first parameter ");
    //A TextField is a place for writing.
    this.input1 = new JTextField( 10);
    this.invitation2 = new JLabel("Input second parameter + enter ");

    this.sum = new JLabel("The sum is " );
    this.input2 = new JTextField( 10);
    this.input2.addActionListener( this);

    //Activation
    this.add(resize);
```

```
    this.add(invitation1);
    this.add(input1);
    this.add(invitation2);
    this.add(input2);
    this.add(sum);
    // Let Swing manage the layout.
    this.setLayout(new FlowLayout());

}

@Override
public void actionPerformed(ActionEvent event)
{
    // If the User presses enter into the second field
    //Read textField1 and textField2
    //Read as String and translated to Double
    Double num1 = Double.parseDouble(this.input1.getText() );
    Double num2 = Double.parseDouble(this.input2.getText() );
    Double result = num1 + num2;
    //The text of label sum is updated
    this.sum.setText("The sum is "  + result.toString());
    //Make a report to the console.
    // Here we report results.
    // We use the console
    // because we expect large volumes of data
    System.out.println("Number 1 " + "\t " + num1 );
    System.out.println("Number 2" + "\t " + num2 );

    System.out.println("The sum is " + "\t " + result );
    System.out.println();
}
}//End of Program A287 Dialog3
```

288 ***Exercise*** *Run the program and play with the code. Modify it to capture, operate and report three parameters.* Answer

17.4 Conclusion

Java is proud of the visual facilities offered to delight users of software even through Internet. We get acquainted with the very first example of how that could

be achieved: we programmed an Applet to retrieve information from the user. While this new facility could ease pedagogical projects, what is more important for a researcher is that at last programming science produces something similar to the natural software: it contains services that are ready to satisfy exterior calls. So, the genome is an example of a very advanced style of programming. Is that style necessary for surviving? Yes? No? Why?

Chapter 18

A virtual lab

```
Moths and birds + a nice interface.
```

289 ***Motivation***

We have learned how to make good use of the visual facilities of Java to built a friendly interface but we would want to see them right now on our programs of interest.

290 ***Purpose.*** *We will connect our simulation of moths vs birds to a cozy interface to capture and modify parameters.*

18.1 Implementation

Our virtual lab comes in two programs. The first contains a simulation and the second the interface.

291 ***Part one: moths vs birds***

We have two programs, one for the interface and the other for the simulation properly. The next is a new version of the simulation of moths vs birds: this program is an improvement over our previous ones because the positional bug has been fixed. Moreover, it has been adapted to be a satellite program, which orbits around the interface, part two, which functions as the head of the ensemble.

```
//Program A291 Interface

/*
```

```
Study of predation of moths by birds.
Moths could be white or brown.
Two environments: clean and dirty.
Discrete  stochastic logistic model.
Mutation is included.
Positional bug has been corrected.
AWT + Applet interface.
*/

package ejvol1;


import java.util.Random;
public class Interface
{
    static int n, nw, nbr, nb;
    static double   mr, mut;
    static double  maxM, maxB, pw, pbr,bd;


    //Turn on of the random generator
    static Random r  = new Random();


    private static void report()
    {
        //We report results to the console.
        System.out.println("n" + "\t " + n );
        System.out.println("nw" + "\t " + nw );
        System.out.println("nbr" + "\t " + nbr );
        System.out.println("nb" + "\t " + nb );
        System.out.println("mr" + "\t " + mr );
        System.out.println("mut" + "\t " + mut );
        System.out.println("maxM" + "\t " + maxM );
        System.out.println("maxB" + "\t " + maxB );
        System.out.println("pw" + "\t " + pw );
        System.out.println("pbr" + "\t " + pbr );
        System.out.println("pd" + "\t " + bd );
    }
```

```
private static void reinitialization()
{
    //probability of predation on white moths
    double tr = pw;
    pw = pbr;
    //probability of predation on brown moths
    pbr = tr;
    //Probability of bird's death
}

private static void whiteReproduce()
{
    //White moths reproduce and mutate
    double s = r.nextDouble();
    if (s < mr * (1 - (nw + nbr) / maxM) )
    {
        nw = nw + 1;
        if (s < mut)
        {
            nbr = nbr + 1 ;
            nw = nw-1;
        }
    }
}

private static void brownReproduce( )
{
    //Brown moths reproduce and mutate

    double s = r.nextDouble();
    if (s < mr * (1 - (nw + nbr) / maxM) )
    {
        nbr = nbr + 1;
        if (s < mut)
        {
            nbr = nbr - 1 ;
            nw = nw+1;
        }
    }
}
```

```
private static void dynamics( )
{
    for(int i = 0; i < n; i++)
    {
        // Reproduction is assigned
        // by a binomial random variable ( a coin)
        // to white else brown moths
        // in proportion to their numbers

        double totalMoths = nw + nbr;
        double probWhite = nw/totalMoths;
        for(int k = 0; k < totalMoths; k++)
        {
            double s = r.nextDouble();
            if (s < probWhite ) whiteReproduce();
                else brownReproduce();
        }

        //Moths are preyed, birds reproduce
        for(int l = 0; l < nb; l++)
        {
            //White moths
            for(int j = 0; j < nw; j++)
            {
                double  s = r.nextDouble();
                // a white moth is predated
                if (s < pw){ nw = nw - 1;}
                //birds reproduce
                if ( s < 0.1 * pw * (1 - nb / maxB))
                nb = nb + 1;
            }
            //brown moths
            for(int j = 0; j < nbr; j++)
            {
                double  s = r.nextDouble();
                // a brown moth is predated
                if (s < pbr) nbr = nbr - 1;
```

```
                //birds reproduce
                if ( s < 0.1 * pbr * (1 - nb / maxB))
                nb = nb + 1;
            }
        }


        //Birds die
        for(int j = 0; j < nb; j++)
        {
            double   s = r.nextDouble();
            if (s < bd)  nb = nb - 1;
        }



        System.out.println(nw + "\t " + nbr + "\t " + nb);
    }//end i for
}

public static void master(
    int nwa, int nbra,
    double mra, double muta,
     double  maxMa, double maxBa,  int nba, double pwa,
     double pbra, double bda,   final int na)
{

    //Observe the new form as we initialize
    //Generates   moths,
    //white moths
    nw = nwa;
    // brown moths
    nbr = nbra;
    //Generates 10 birds
    nb = nba;
    //Probability of moth's reproduction
    mr = mra;
    // Mutation rate of moths
    mut = muta;
    //Maximum number of Moths
```

```
        maxM = maxMa;
        //Maximum number of Birds
        maxB = maxBa;
        //probability of predation on white moths
        pw = pwa;
        //probability of predation on brown moths
        pbr = pbra;
        //Probability of bird's death
        bd = bda;
        //Number of generations
        n = na;

        report();
        dynamics();
        // Here a change of environment is simulated:
        // the predation rates are interchanged:
        // white moths that before were invisible
        // now had become visible because
        // of blackening of their skin.
        // Now, brown moths are conspicuous.
        reinitialization();
        dynamics();
    }
}//Program A291 Interface
```

292 ***Exercise*** *Observe the code and verbalize it. Pay attention to the new adaptations made to the program in order to serve as a tool for the interface.*

293 ***Part two: the interface properly.*** *In the next program we see an Applet that serves as interface within the user and the simulation contained in the first program.*

```
//Program A293 Interface2
//Interface for birds vs moths.
//AWT based code.

package ejvol1;

import java.awt.*;
import java.applet.Applet;
```

```
public class Interface2 extends Applet
{

    private static final long serialVersionUID = 1L;

    Label invitation1; //message to the user
    TextField input1;  //input from the user
    Label invitation2;
    TextField input2;
    Label invitation3;
    TextField input3;
    Label invitation4; //message to the user
    TextField input4;  //input from the user
    Label invitation5;
    TextField input5;
    Label invitation6;
    TextField input6;
    Label invitation7; //message to the user
    TextField input7;  //input from the user
    Label invitation8;
    TextField input8;
    Label invitation9;
    TextField input9;
    Label invitation10;
    TextField input10;
    Label invitation11;
    TextField input11;
    static int n,nw,nbr, nb; //working variables
    static double     mr, mut;
    static double  maxM, maxB, pw, pbr,bd;



    @Override
    public  void   init()
{

    //Input machinery
```

```
//Declaration
// A Label displays an invitation to operate
invitation1 = new Label("Number of generations");
//A TextField is a place for writing.
input1 = new TextField( "1000");
invitation2 = new Label("Initial number of white moths");
input2 = new TextField( "450");
invitation3 = new Label("Initial number of brown moths");
input3 = new TextField( "450");
invitation4 = new Label("Initial number of birds");
//A TextField is a place for writing.
input4 = new TextField( "10");
invitation5 = new Label("Reproduction rate  of moths");
input5 = new TextField( "0.1");
invitation6 = new Label("Moth's mutation rate");
input6 = new TextField( "0.01");
invitation7 = new Label("Max number of moths");
//A TextField is a place for writing.
input7 = new TextField( "2000");
invitation8 = new Label("Max number of birds");
input8 = new TextField( "20");
invitation9 = new Label("Predation rate on white moths");
input9 = new TextField( "0.001");
       // A Label displays an invitation to operate
invitation10 = new Label("Predation rate on brown moths");
//A TextField is a place for writing.
input10 = new TextField( "0.002");
invitation11 = new Label("Bird's rate of death  + ENTER");
input11 = new TextField( "0.05");

// activation
add(invitation1);
add(input1);
add(invitation2);
add(input2);
add(invitation3);
add(input3);
add(invitation4);
add(input4);
add(invitation5);
```

```
    add(input5);
    add(invitation6);
    add(input6);
    add(invitation7);
    add(input7);
    add(invitation8);
    add(input8);
    add(invitation9);
    add(input9);
    add(invitation10);
    add(input10);
    add(invitation11);
    add(input11);
  }

//This is the listening instrument of the interface:
//The system reads parameters
//if the user presses enter into the ELEVENTH field.
//This method is deprecated because a better technology
//was developed since 1996 and comes in three versions:
//1) AWT + Applet improved, simple and effective:
//Internet Programming with Java Course.Applets
//http://www.nakov.com/inetjava/lectures/part-2-applets/InetJava-2.1-2.2-Intro
lets.html
//2) Swing, very good and robust: keep working with this series.
//3) JavaFX, released in 1914. Modern and very powerfull
//but  a bit inmature as yet (2015):.
//If you aim at great things wih Java programming,
//JavaFX is your must.
//Find a soft introduction in Vol XII Fossils, in this series.

@SuppressWarnings( "deprecation" )
    @Override
public boolean action(Event event, Object o)
  {
    if (event.target == input11)
    {
        //We read textFields
        n = Integer.parseInt(input1.getText() );
        nw = Integer.parseInt(input2.getText() );
```

```
        nbr=Integer.parseInt(input3.getText() );
        nb = Integer.parseInt(input4.getText() );
        mr = Double.parseDouble(input5.getText() );
        mut= Double.parseDouble(input6.getText() );
        maxM = Double.parseDouble(input7.getText() );
        maxB = Double.parseDouble(input8.getText() );
        pw= Double.parseDouble(input9.getText() );
        pbr = Double.parseDouble(input10.getText() );
        bd = Double.parseDouble(input11.getText() );
        //We call the simulation.
        Interface.master(nw, nbr, mr, mut, maxM, maxB,
        nb, pw, pbr, bd, n);
    }
    //Indicate to the system that the action
    //of the user has been processed.
    return true;
  }
}//End of Program A293 Interface2
```

294 ***Exercise*** *To run the code: 1) Add to any project the class A291, where the simulation has been programmed. Compile it. 2) Add the class A293 to the same project. Compile and run it . Verify that it correctly displays the values of the parameters. Verify that there is a perfect correspondence among values and names of the parameters. The window can be resized both horizontally and vertically: do that if some fields are not seen but choose a size that causes no possibility of error of association among prompts and textFields. Make a graphic of the output.*

295 ***Challenge*** *Learn to use this program as a tool for your ecological studies and/or for pedagogical work. As a first study, program experiments to disentangle some of the following questions: the system oscillates under certain parameters: which and in which range? The system is stable under other parameters: which and in which range? Some oscillations may lead to the annihilation of the species: which and in which range? Does we need the division of the prey species in two subtypes, white and brown, to assure the stability of the prey-predator system? Elaborate your laboratory inform as a 5 pages article.*

296 ***Intrigue:*** *The perfectionism displayed in the last code is justified here because the program could be considered as a specific virtual laboratory of ecology. How do we justify perfectionism in nature? Actually, are we indeed perfect or are we retarded to visit hospitals and grandfathers? If we use an economic directive,*

we could say: perfectionism pays when the fruits it produces overcome the cost of its implementation. Is that correct? Does perfectionism have a cost in nature? Which are the components of that cost? Would that balance pose a limit to the development of perfectionism?

297 ***Challenge*** *We have learned that under some setting of parameters, a prey-predator stochastic system necessarily oscillates and that an outburst of the prey is followed by an outburst of predators that kills the preys and that causes predators to die at starvation. In short: a stochastic prey - predator system cannot survive for ever. The remedy was to pose an artificially maximum number of predators, which governs the predator reproduction through a logistic model. When this is considered, the whole system becomes stable for appropriate values of the parameters. Please, make an estimation of the range of stability of the system.*

298 ***Challenge*** *Notice that in nature, our artificially imposed maximum number of predators corresponds with the charge of the system, the tolerated amount of individuals. In predators this is also related to territoriality. We have proved that territoriality is more necessary than convenient for the stability of a prey-predator system. Equivalently: without territoriality there cannot exist prey-predator systems. Now, develop the simplest model that proves that a system of the form grass-game must be enhanced into a system of the form grass-game-predator - high-level predator with territoriality. This is certainly true because there are fluctuations of the low level links of the feeding chain and some of them favor the multiplication of high-level populations. This would cause a complete exhaustion of low-level components. A consequence is the starvation to death of high-level members. Net result: the system disappears. Would you explain why territoriality could be tied to a single sex or even to the alpha member of the hierarchy? What about feminine participation in the control of picks of birth rate?*

299 ***Challenge*** *Develop a virtual laboratory to study a food chain. Keep a record of all your attempts and try to understand the evolution of your style and of the complexity and realism of your codes.*

18.2 Conclusion

We have begun to probe sophistication of programming in Java. Nice but very complex. At the same time, we are getting some expertise in formulating terrible questions of biological importance: this is the first fruit of simulation as a formative tool.

Chapter 19

Simulation in statistics

```
Random numbers is at the base of statistics.
```

300 ***Motivation.*** *Usual statistical analysis demands data that fit special properties, say, normality. Otherwise one shall use non-parametric statistical analysis, which is very inefficient. What can we do to achieve efficiency to cover abnormal data?*

301 ***Purpose***

In this chapter we will show how one can use simulation to run a test for statistics . We begin with running a test for a one factor anova. Next, we run an anova with blocks. Because these tests are well known (Walpole, [14] 2011), we could calibrate our method. Next, we pass to study a block design with incomplete data. In that way, we open a new world of possibilities.

302 ***The normal distribution***

Let us review the normal distribution. We study the inbuilt in Java simulation of a normally distributed random variable. From a previous exercise we know that the generator is O.K. in regard with normality but now we want to calibrate the mean and variance of that simulation. This is done in the next program.

```
// Program 302 StatTest
// Mean and variance of
// the Java inbuilt normal random generator.

package ejvol1;
```

```
import java.util.Random;


public class StatTest
{
    //Turn on  the random generator
    static Random r  = new Random();

    static double Data[] = new double[10000];
    static int n;




public static void main( String args[])
{

    int n = 10000;
    //We generate n random numbers
    for(int i = 0; i < n; i++)
    {
        double s = r.nextGaussian() ;
        Data[i]= s;
        //System.out.println( Data[ i ]);
    }

    //Let us calculate the mean
    double sum = 0;

    for(int i = 0; i < n; i++ )
        sum = sum + Data[i];
    double mean = sum / n;
    //System.out.println( "Sum \t "+ sum);
    System.out.println( "Mean \t "+ mean);

    // Let us calculate the variance
    double sum2 = 0;
```

```
    for(int i = 0; i < n; i++ )
        sum2 = sum2 + (Data[i]-mean)*(Data[i]-mean);
    double var = sum2 / n;
    //System.out.println( "Variance \t "+ var);
    System.out.println( "Deviation \t "+ Math.pow( var, 0.5));
}
}//End of Program 302 StatTest
```

303 ***Exercise*** *Run the previous code various times to convince yourself that the normal distribution that is inbuilt with Java has mean zero and standard deviation one. Else, carry a based on the central limit theorem z-test for the mean; and for the variance, a chi2-test.* *Answer*

304 ***Challenge*** *A simulation to be really good needs to fit normal data as well as outliers. Verify for the previous simulation that the outliers that lie farther than 2 standard deviations form a subpopulation of around 5 %.*

305 ***Challenge*** *Prove that the mean and variance of the normal distribution generated in the previous program converge to zero and one respectively. Prove that the cost of the approximation is exponential in the number of significant ciphers of the approximation.*

306 ***Any mean, any deviation***

Let us learn to simulate a normally distributed random variable with mean μ and standard deviation σ. The next program contains the corresponding code:

```
// Program 306 StatTest2
// Simulation of a
// normally distributed random variable
// with arbitrary mean and deviation.

package ejvol1;


import java.util.Random;

public class StatTest2
{
    //Turn on  the random generator
```

```
    static Random r  = new Random();


    public static void main( String args[])
    {
        // Definition of projected values
        double meanProj = 2;
        double deviationProj = 3;

        int n = 1000;
        double Data[] = new double[10001];
        //We generate n random numbers
        for(int i = 0; i < n;i++)
        {
            double s = r.nextGaussian() ;
            Data[i]= deviationProj* s +meanProj ;
            //System.out.println( Data[ i ]);
        }

        //Let us calculate the mean
        double sum = 0;

        for(int i = 0; i < n; i++ )
          sum = sum + Data[i];
        double mean = sum / n;
        // System.out.println( "Sum \t "+ sum);
        System.out.println( "Mean \t "+ mean);

        // Let us calculate the variance
        double sum2 = 0;
        for(int i = 0; i < n; i++ )
            sum2 = sum2 + (Data[i]-mean)*(Data[i]-mean);
        double var = sum2 / n;
        // System.out.println( "Variance \t "+ var);
        System.out.println( "Deviation \t "+ Math.pow( var, 0.5));
    }
}//End of Program 306 StatTest2
```

307 ***Exercise*** *Run the previous code various times to convince yourself that the normal distribution that is generated with Java has the projected stadigraphs.*

Change these values various times.

19.1 One factor anova

We are born with statistical facilities that allow us to say, for instance: Joe likes Mary because when she passes by, he gets red. We are describing in this way two treatments: Mary is present vs. Mary is absent. And we declare that the system, Joe, changes from treatment to treatment. A nice formalization for quantitative variables follows.

308 ♣ ***Definition.*** *A **one factor anova** is the analysis of variance with the purpose of studying differences in means among diverse random samples that come from diverse treatments. The idea is that if a factor influences a system, then when treatments vary, the mean response of the system also must vary. The anova studies this variation to unveil whether or not the effect of the changes from treatment to treatment is stronger than the changes caused by the fluctuations of the system.*

309 ***Example*** *We study strawberries under three diverse temperatures: we measure the crop in kg per square meter. If temperature affects the yield of this plant, then one must observe a variation of the average yield when varying the temperature of the culture. Results and the analysis by Excel follow:*

9^o C	18^o C	25^oC
1	3	1
2	4	1
1	5	2

The One Factor anova analysis carried out by Excel renders the following table:

Anova for three temperatures						
Origin of variation	SS	DF	Mean Square	F-Exp	Probab	Critical F
Among groups	14,22222222	2	7,111111111	12,8	0,0068453	5,143249382
Within groups	3,333333333	6	0,555555556			
Total	17,55555556	8				

We see that the anova test authorizes us to claim that some differences in temperature are matched by a difference in average production. In familiar jargon:

strawberries prefer a fresh climate. Anyway, it is not excluded that our decision could be wrong: randomness can mimic the same effect with a probability of 0.006.

Can we run this test on our own without Excel?

310 ***Simulation***

Let us learn now how to carry the same anova test with the help of a simulation. The central idea is simple and is the same for all problems no matter how complex they could be. Let us see it:

1) One examines exactly what one did in an experiment.

2) One declares exactly what is the null hypothesis that one wants to test.

3) One builds an artificial, virtual world, with the same characteristics as those proposed by the null hypothesis.

4) One runs the virtual world under the same conditions as those met by the experiment.

5) For each run in the virtual world one measures exactly the same stadigraph as in the real experiment.

6) One compares the experimental result found in the real world with the distribution of results found in the virtual world.

7) If the stadigraph associated to the experimental result gets a value that is an outlier with respect to the distribution of the virtual world, one declares that the experiment does not support the null hypotheses. Otherwise, one declares that the null hypothesis explains our experiment or, more rigorously, one should say: our experiment was unable to reject the null hypothesis.

The implementation of this directive for our temperature experiment is as follows.

1) We arranged three levels of temperature with three replicas per temperature.

2) The null hypothesis is that a change in temperature plays no special role in the amount of crop of the strawberry. Instead, all observed differences are due to uncontrolled variables, such as the intrinsic genetic variability that causes that different plants produce different amounts of fruit.

3) Let us build an artificial, virtual world, with the same characteristics as those proposed by the null hypothesis.

We must, in first place, refine our null hypothesis. We said that all observed differences are effect of uncontrolled variables. But we have the observed differences in front of our eyes, so we are allowed to characterize them. The first supposition is that the fluctuations were created by many independent factors, so they follow a normal distribution: our simplest model is that all observed values dance around

its average with deviations that fluctuate with a normal distribution. Hence, we must find unbiased estimators of the mean and of the standard deviation of the fluctuations.

Now, if the null hypothesis is correct, the division of data into three columns is unjustified: all data belong to the same sample. In that case, the unbiased estimator $\widehat{\mu}$ of the mean is the ordinary mean:

$$\widehat{\mu} = \frac{1+2+1+3+4+5+1+1+2}{9} = 20/9$$

If the null hypothesis is correct, the unbiased estimator $\widehat{\sigma}$ of the standard deviation is

$$\begin{aligned}
&\widehat{\sigma} = \sqrt{\frac{\sum(x_i-\widehat{\mu})^2}{n-1}} = \\
&= \sqrt{\frac{\sum x^2}{n-1} - \frac{n\widehat{\mu}^2}{n-1}} \\
&= \sqrt{\frac{1^2+2^2+1^2+3^2+4^2+5^2+1^2+1^2+2^2}{8} - \frac{9(20/9)^2}{8}} \\
&= \sqrt{\frac{62}{8} - \frac{400/9}{8}} \\
&= \sqrt{7.75-5.55} = \sqrt{2.2} = 1.483
\end{aligned}$$

Thus, an observed value is, under the null hypothesis, the sum of the mean value, $20/9 = 2.222$, plus a fluctuation that follows a normal distribution with mean zero and standard deviation 1.483. These conditions define our virtual world: random numbers with a normal distribution with mean 2,2222 and standard deviation 1.483.

4) Let us run now the virtual world under the same conditions as those met by the experiment.

The program just below produces all we need. The module randMatrix produces a 3×3 matrix that is filled in random numbers with a normal distribution with mean 2,2222 and standard deviation 1.483. A recurrent call to randMatrix produces a sequence of matrices. To each matrix a F-stadigraph is associated and all results are recorded to study their distribution. We can use that distribution to classify an event as normal or outlier.

In the next program, we use the new instruction

```
int d = (int)  Math.ceil(  k);
```

that assigns to the integer variable d the minimum integer that is larger than the double variable k. The program follows:

```
package ejvol1;


//Program A310 StatTest3

//  Analysis of variance
//  by simulation.
//  Random variables
//  are normally distributed.

// A 3x3 matrix is filled in  random numbers
// with a normal distribution
// with mean 2.222
// and standard deviation 1.483
// The distribution of
// anova stadigraphs is found.

import java.util.Random;


public class StatTest3
{


    //Turn on  the random generator
    static Random r  = new Random();

    static int freqTable[] = new int[1000];
    static int n;
    static  double meanProj ;
    static  double deviationProj ;
    static double M[][] = new double[3][3];
    static double means[] = new double[3];

    //Experimental data (invented)
    static double mExp[][] = {{1,3,1},{2,4,1},{1,5,2}};

    //Initialization
    public static void freqTableToZero()
```

```
    {
       for(int i = 0; i < 1000; i++) freqTable[i]= 0;
    }

    //Print to console
    public static void reportM(double M[][])
    {
        for(int i = 0; i < 3; i++)
        {
            for(int j = 0; j < 3; j++)
                System.out.print(  M[i][j] + " \t");

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

//Fills a 3x3 matrix with random numbers
//with a normal distribution
//with mean 2.222
//and standard deviation 1.483
public static void randMatrix()
{
    double s;
    for(int i = 0; i < 3; i++)
    {
        for(int j = 0; j < 3; j++)
        {
            s = deviationProj * r.nextGaussian()+meanProj;
            M[i][j]= s;
            //System.out.print(  M[i][j] + " \t");
        }
        //System.out.println();
     }
//System.out.println();
}

//Calculation of the F stadigraph
public static double F(double M[][])
{
```

```
// reportM(mExp);
double ssw = 0;
double meanxi ;
for(int i = 0; i < 3; i++)
{
    meanxi = 0;
    for(int j = 0; j < 3; j++)
        meanxi= meanxi + M[j][i];
    //Matrix[row][column]
    meanxi = meanxi/3;
    means[i]=meanxi;
    //System.out.println("meani " + means[i]);

    for(int j = 0; j < 3; j++)
    ssw = ssw + Math.pow((M[j][i]-meanxi),2);
}

double  mean = 0;
for(int i = 0 ; i < 3; i++)
    for(int j = 0; j < 3; j++)
        mean = mean + M[j][i];
mean = mean /9;

double ssa = 0;
for(int i = 0; i < 3; i++)
    ssa = ssa + 3*Math.pow((means[i]-mean),2);
double F = (ssa/2)/(ssw/6);
return F;
}

//Print table
private static void reportFreqTable(int k)
{
    int m = 0;
    int i = 1;
    System.out.println( "FreqTable of F values"  );
    System.out.println( "F" + "\t " + "frec"
          + "\t " + "Cumulated frec");
    while (m<k)
    {
```

```
            m = m + freqTable[i];
            System.out.println(i+ "\t " + freqTable[i] + "\t " + m);
            i = i+1;
        }
    }


    // Report of means and F value
    private static void FExp( )
    {
        System.out.println("F exp = " + F(mExp));
        // for(int i = 0; i < 3; i++)
        //System.out.println("meani " + means[i]);
    }

    public static void main( String args[])
    {
        // Definition of projected values
        //According to the null hipothesis
        meanProj = 2.222;
        deviationProj = 1.483;
        freqTableToZero();
        int n = 1000 ;
        //We generate n random matrixes
        // filled in numbers that
        // follow a normal distribution
        // with mean meanProj and
        // deviation deviationProj
        for(int i = 0; i < n;i++)
        {
            randMatrix();
            // reportM(M);
            // System.out.println( i + "\t " + "F = \t " + F());
            int d = (int)  Math.ceil(  F(M));
            freqTable[d]=freqTable[d]+1;
        }
        int k = n;
        reportFreqTable(k);
        // We calculate the F for the experiment
        FExp();
```

```
        /* What does ciel do?
         int d = (int)  Math.ceil(  1.2);
         System.out.println( d);
         d = (int)  Math.ceil(  1.7);
         System.out.println( d); */
    }
}//End of Program A310 StatTest3
```

5) For each run in the virtual world one measures exactly the same stadigraph as in the real experiment.

For a given matrix of data with K columns and N entries = number of rows times number of columns, the stadigraph F is calculated as follows:

$$SSW = \sum_{i=1}^{k} \sum_{j=1}^{n_i} (X_{ij} - \bar{X}_i)^2$$

$$SSA = \sum_{i=1}^{k} 3(\bar{X}_i - \bar{X})^2$$

Then

$$F = \frac{SSA/(k-1)}{SSW/(N-k)}$$

This stadigraph is calculated in the sub that is called F. The code was fed with the matrix of data that we are analyzing and it produced the same F as Excel: the F for our experimental data was 12.8. This testing was necessary to see whether or not our program was well designed.

6) One compares the experimental result found in the real world with the distribution of results found in the virtual world.

To do this, we must repeat the experiment in the virtual world a significant number of times. We choose 1000 times, but feel free to prefer 10000 times.

The program outputs 1000 values of the stadigraph F. They were reported as a histogram and then drawn and the distribution resulted to be a typical F. The histogram is reported in the next table:

Class	Frequency	Cumulated Freq.
0	562	562
1	208	770
2	98	868
3	43	911
4	29	940
5	14	954
6	13	967
7	9	976
8	10	986
9	3	989
10	4	993
11	0	993
12	2	995
13	3	998
14	0	998
15	0	998
≥ 15	2	1000

Now, we are ready to take a verdict.

7) If the F-value associated to the experimental result is an outlier with respect to the distribution of the virtual world, one declares that the experiment does not support the null hypotheses. Otherwise, one declares that the null hypothesis explains our experiment, or, if one prefers, that we were unable to reject the null hypothesis.

The experimental stadigraph takes the value 12.8. There were 993 data among 1000 below that value. This means that our experimental result renders a value that classifies as outlier with a probability that is less than 0.01. (The corresponding value predicted by Excel, and calculated on theoretical grounds, was 0.007. This close correspondence is necessary to have confidence in the exactitude of our simulation methods to test statistical hypothesis.) As a consequence, we declare that, with significance 0.01, temperature is a factor that influences the average production of strawberries.

311 ***Challenge*** *Work out the simulation of a one factor anova test for a matrix of data with 4 treatments, 3 data in the first treatment, 4 in the second, 5 in the third, and 6 in the fourth. Hint: try to selectively adapt the previous program to your needs: some pieces are modifiable, but other are best designed ab initio.*

19.2 A block design

A fundamental principle of science is that the more one takes control of an experiment, the sharper are the conclusions. How does one apply this principle in a real situation?

312 ***The power of control.***

To fix ideas, let us consider mitochondria. We want to know whether or not temperature is a factor that influences its ATP production. This means that if one varies temperature, then the average production of ATP also varies.

But there are many other factors that also influences mitochondria, say, mineral or acid activities. This is clear. This means that the variation in the output has many sources of origin that could smear out the effect of temperature, because we do not know exactly whether or not the variation in ATP production comes from the variation in temperature. Thus, we keep control of the most significant factors and in that way we can increase the sharpness of our conclusions. This is proved in an anova with blockage: we control pH as a probably important background factor.

313 ***Calculations***

The receipt to calculate a Randomized block anova is a slight modification of the one way anova, which is exemplified with our mitochondria case study. We have raw data as follows:

Mitochondria			
ATP production			
	pH 4	pH 6	pH 8
30^o	5	8	6
36^o	7	10	8
42^o	4	7	3

The statistical analysis is as follows:
We have $k = 3$ treatments and $l = 3$ levels of blockade, $N = kl = 9$.
GS_i= Great Square under treatment i = $(\sum_{j=1}^{n_i} X_{ij})^2$
$GS_1 = (5+7+4)^2 = 16^2 = 256$
$GS_2 = (8+10+7)^2 = 25^2 = 625$
$GS_3 = (6+8+3)^2 = 17^2 = 289$
$T = \sum_{i=1}^{k} \sum_{j=1}^{n_i} X_{ij} = 5+7+4+8+10+7+6+8+3 = 58$
GS= Great Square = $(\sum_{i=1}^{k} \sum_{j=1}^{n_i} X_{ij})^2$

$GS = T^2 = (5+7+4+8+10+7+6+8+3)^2 = 58^2 = 3364$

Among Columns SS = $\sum_{i=1}^{k} \frac{GS_i}{l} - \frac{GS}{N}$ =(256+ 625+ 289)/3 - 3364/9= 390 -373.7 = 16.3

Among columns DF= Among columns degrees of freedom= k-1= 3-1=2.

Among columns Mean square= $\frac{AmongColumnsSS}{AmongcolumnsDF} = 16.3/2 = 8.1$

We repeat the procedure with blocks or rows or levels of blockade:

$BGS_1 = (5+8+6)^2 = 19^2 = 361$

$BGS_2 = (7+10+8)^2 = 25^2 = 625$

$BGS_3 = (4+7+3)^2 = 14^2 = 196$

$T = \sum_{i=1}^{k} \sum_{j=1}^{n_i} X_{ij} = 5+7+4+8+10+7+6+8+3 = 58$

GS= Great Square = $(\sum_{i=1}^{k} \sum_{j=1}^{n_i} X_{ij})^2$

$GS = T^2 = (5+7+4+8+10+7+6+8+3)^2 = 58^2 = 3364$

Among Rows SS = $\sum_{i=1}^{k} \frac{GS_i}{k} - \frac{GS}{N}$ =(361+ 625+ 196)/3 - 3364/9= 394 -373.7 = 20.3

Among rows DF= Among rows degrees of freedom= l-1= 3-1=2.

Among rows Mean square= $\frac{AmongRowsSS}{AmongrowsDF} = 20.3/2 = 10.1$

Now we calculate the square of each entry.

X_{ij}^2		
pH4	pH6	pH8
25	64	36
49	100	64
16	49	9

Total SS= Total Sum of Squares= $\sum_{i=1}^{k} \sum_{j=1}^{n_i} (X_{ij}^2) - \frac{GS}{N}$

25 + 64 +36 +49 + 100 +64+16 + 49+9

$=(25+64+36+49+100+64+16+49+9) - \frac{3364}{9} = 412 - 373.7 = 38.3$

Error SS=error sum of squares= Total SS - Among columns SS -Among row SS= 38.3-16.3-20.3= 1.7

Error DF= Total DF - Among groups DF - Among rows DF= $8-2-2=4$.

Error Mean square= $\frac{ErrorSS}{errorDF} = 1.7/4 = 0.43$

$F - experimental\ for\ columns = \frac{Among\ columns\ mean\ square}{Error\ Mean\ square} = \frac{8.1}{0.43} = 18.8$

$F - experimental\ for\ rows = \frac{Among\ rows\ mean\ square}{Error\ Mean\ square} = \frac{10.1}{0.43} = 23.4$

Next, we compare this experimental stadigraphs with the critical ones given by a table or by Excel. For columns and rows, that value was the same: 6.94. Because

the experimental values of F are larger than the critical ones, we conclude that pH and Temperature are factors that influence the average production of ATP by the mitochondria.

314 *Simulation*

Let us see how we can run the same test but based on a simulation. Let us pay attention to the fact that our methodology is exactly the same for every design.

The following code makes the complete task and apes on the example above.

```
// Program 314 StatTest4
// Study of an anova design with blockage.
// Normally distributed random variables
// with given mean and deviation.



package ejvol1;



import java.util.Random;


public class StatTest4
{
    // Turn on  the random generator
    static Random r  = new Random();

    static int freqTableC[] = new int[10001];
    static int freqTableR[] = new int[10001];
    static  double meanProj ;
    static  double deviationProj ;
    static double M[][] = new double[4][4];
    static double means[] = new double[4];
    static double Fcolumns  ;
    static double Frows ;
```

```
    public static void freqTableToZero()
    {
        for(int i = 0; i<=1000; i++)
        {
            freqTableC[i]= 0;
            freqTableR[i]= 0;
        }
    }


    public static void reportM( )
    {
        for(int i = 1; i<=3; i++)
        {
            for(int j = 1; j<=3; j++)
             System.out.print(  M[i][j] + " \t");
            System.out.println();
        }
        System.out.println();
    }

// Fills a 3x3 with random numbers
// with a normal distribution
// with mean 2.222
// and standard deviation 1.483
public static void randMatrix()
{
    double s;
    for(int i = 1; i<=3; i++)
    {
        for(int j = 1; j<=3; j++)
        {
            s=  deviationProj * r.nextGaussian()+meanProj;
            M[i][j]= s;
            //System.out.print(  M[i][j] + " \t");
        }
        //System.out.println();
    }
    //System.out.println();
}
```

```
// reportM(mExp);
public static void FF()
{

/*GSi Great  Square under treatment i =
(\sum^{n_i}_{j=1}X_{ij})^2 */
double GS1 = Math.pow(M[1][ 1] + M[2][ 1] + M[3][ 1], 2);
double GS2 = Math.pow(M[1][ 2] + M[2][ 2]+ M[3][ 2], 2 );
double GS3 = Math.pow(M[1][ 3] + M[2][ 3] + M[3][ 3], 2);
//System.out.println("GS1 "+ GS1);
//T= \sum^k_{i=1} \sum^{n_i}_{j=1} X_{ij}

/*GS= Great  Square =
(\sum^k_{i=1} \sum^{n_i}_{j=1} X_{ij})^2 */

double GS = Math.pow(M[1][1]+M[2][1]+M[3][1]+M[1][2]
            +M[2][2]+M[3][2]+M[1][3]+M[2][3]+M[3][ 3],2);

int n = 8;
//Among Columns SS =   \sum^k_{i=1}\frac{GS_i}{ l_i}
//                                          -\frac{GS}{N}.
double ACSS = (GS1 / 3 + GS2 / 3 + GS3 / 3) - GS /9;

//Among columns DF= Among columns degrees
//of freedom= k-1= 3-1=2.
//int ACDF = 2;

/*Among columns Mean square=
\frac{Among Columns SS}{Among columns DF}*/

double ACMS = ACSS / 2;


//The same but for rows or blocks


double BGS1 = Math.pow(M[1][ 1] + M[1][ 2] + M[1][ 3],2) ;
double BGS2 = Math.pow(M[2][ 1] + + M[2][ 2]+ M[2][ 3],2);
double BGS3 = Math.pow(M[3][ 1] + M[3][ 2] + M[3][ 3],2) ;
```

```
// System.out.println( "  BGS1 "+ BGS1);
//Among Rows SS = $\sum^k_{i=1}\frac{GS_i}
//{ k_i}-\frac{GS}{N}$

double ARSS = (BGS1 / 3 + BGS2 / 3 + BGS3 / 3) - GS / 9;
// System.out.println( "ARSS " + ARSS);
//Among rows DF= Among rows degrees of freedom= l-1= 3-1=2.

//Among rows Mean square=\frac{Among Rows SS}{Among rows DF}

double ARMS = ARSS / 2;
// System.out.println( "ARMS " +ARMS);

//Calculates the square of each entry and sums.


double UIK = Math.pow(M[1][ 1] ,2)+ Math.pow(M[2][ 1] ,2)
       + Math.pow(M[3][ 1] ,2)+ Math.pow(M[1][ 2],2)
       + Math.pow(M[2][ 2],2)
       + Math.pow(M[3][ 2],2) + Math.pow(M[1][ 3] ,2)
       +Math.pow( M[2][ 3] ,2) + Math.pow(M[3][ 3] ,2);


//Total SS=Total Sum of Squares=
//$\sum^k_{i=1} \sum^{n_i}_{j=1}[X_{ij}^2]-\frac{GS}{N}$

double TotalSS = UIK - GS / 9;
// System.out.println( "TotalSS " + TotalSS); //OK
//Error SS=error sum of squares=
//Total SS - Among columns SS -Among row SS

double ERRORSS = TotalSS - ACSS - ARSS;

//System.out.println( "ERRORSS " + ERRORSS);

//Error DF= Total DF - Among groups DF - Among rows DF
//=    8-2-2=4 .
//Error Mean square= \frac{Error SS}{error DF}

double EMS = ERRORSS / 4;
```

```
//System.out.println( "EMS " + EMS);
Fcolumns = ACMS / EMS;
Frows = ARMS / EMS;
//  System.out.println( Fcolumns  ); //OK
// System.out.println( Frows  ); //OK

}

private static void reportFreqTable(int k)
{
    System.out.println( " F + freqTableColumns "
            + "+ freqTableRows " );
    int mc = 0;
    int mr = 0;
    int i = 1;
    while ((mc<k)| (mc < k))
    {
        // System.out.println( "b "  );
        mc = mc + freqTableC[i];
        mr = mr + freqTableR[i];
        System.out.print(i+ "\t " + freqTableC[i]
                + "\t " + freqTableR[i]);
        //System.out.println(i+"\t "+freqTablec[i]
        //+"\t"+mc+"\t"+mr);
        System.out.println() ;
        i = i+1;
    }
}

private static void FExp( )
{
    // Only the non zero blocks matters

    final double mExp[][] = {{0,0,0,0},
                             {0,5,8,6},
                             {0,7,10,8},
                             {0,4,7,3},
                             };
    for(int i= 1; i<=3; i++)
        for(int j= 1; j<=3; j++)
```

```
            M[i][j] = mExp[i][j];
    FF();
    System.out.println( "Experimental matrix"  );
    reportM();
    System.out.println( "Fcolumns " + Fcolumns  );
    System.out.println("Frows " +   Frows  );

}

// Definition of projected values
public static void main( String args[])
{
    meanProj = 6.444;
    deviationProj = 2.1858;
    freqTableToZero();
    int n = 1000 ;
   //We generate n random matrixes
    // filled in numbers that
    // follow a normal distribution
    // with mean meanProj and
    // deviation deviationProj
    for(int i=1; i<=n;i++)
    {
        randMatrix();
        //reportM(M);
        //System.out.println(i+"\t " + "F = \t " + F());
        FF();
        int d = (int)  Math.ceil(  Fcolumns);
        freqTableC[d]=freqTableC[d]+1;
        d = (int)  Math.ceil(  Frows);
        freqTableR[d]=freqTableR[d]+1;
    }
    int k =n;
    reportFreqTable(k);
    // We calculate the F's for the experiment
    FExp();
    /* What does Math.ciel do?
    int d = (int)  Math.ceil(  1.2);
    System.out.println( d);
    d = (int)  Math.ceil(  1.7);
```

```
    System.out.println( d); */
 }
}//End of Program A 314 StatTest4
```

Let us explain now in which form this code answers the questions posed by our directives on the scientific method:

1) One examines exactly what one did in an experiment: we arranged an experiment in which diverse levels of two factors were combined as input to a mitochondria system. The measured output was the production of ATP.

2) One declares exactly what is the null hypothesis that one wants to test: in our experiment, the null hypothesis is that the two factors do not influence the average response of our system and that all observed differences were due to the effect of uncontrolled variables or to the interaction of the two factors.

3) One builds an artificial, virtual world, with the same characteristics as those proposed by the null hypothesis. So, let us characterize the answers of our system. To that effect we use all the information available and organize it with a theoretical idea: random fluctuations around the mean value appeared as a result of many independent factors, so a normal distribution is expected. Let us evaluate the mean and the standard deviation of our experimental data using unbiased estimators. The information of the experimental data was consigned in the data matrix, but according to the null hypotheses, the matrix is an artificial unimportant arrangement because the null hypothesis says that all data come from the same population.

Thus, we have just data, which are 5, 8, 6, 7, 10, 8, 4, 7,3. The mean of these numbers is 58/9=6.444. The standard deviation is:

$$\widehat{\sigma} = \sqrt{\frac{\sum(x_i-\widehat{\mu})^2}{n-1}} =$$
$$= \sqrt{\frac{\sum x^2}{n-1} - \frac{n\widehat{\mu}^2}{n-1}}$$
$$= \sqrt{\frac{5^2+8^2+6^2+7^2+10^2+8^2+4^2+7^2+3^2}{8} - \frac{9(58/9)^2}{8}}$$
$$= \sqrt{\frac{412}{8} - \frac{400/9}{8}}$$
$$= \sqrt{51.5-46.722} = \sqrt{4.778} = 2.1858$$

4) One runs the virtual world under the same conditions as those met by the experiment. Thus, we filled a 3x3 matrix with random numbers with a normal distribution with mean 6.444 and standard deviation 2.1858.

5) For each run in the virtual world one measures exactly the same stadigraph as in the real experiment. We measure two F stadigraphs, one for rows and the other for columns, which were calculated by the standard procedures. The result was tabulated and the output was:

Block design		
Class	Columns 2	Rows
0	561	559
1	170	184
2	104	88
3	60	49
4	27	35
5	27	18
6	11	13
7	9	12
8	7	7
9	2	4
10	3	6
11	2	3
12	5	4
13	0	3
14	0	2
15	1	1
16	1	1
17	0	2
18	1	0
19	0	0
20	1	0
21	0	1
22	0	1
23	1	1
24	2	0
25 or more	6	3

6) One compares the experimental result found in the real world with the distribution of results found in the virtual world. The F stadigraphs of the experiment, approximated up to two significant figures, were 18,25 for columns and 22,75 for rows. There were 11 values among 1000 that were higher than 18.8 and 4 that were higher than 22.75.

7) If the stadigraph associated to the experimental result gets a value that is an outlier with respect to the distribution of the virtual world, one declares that the experiment does not support the null hypothesis. Otherwise, one declares that the null hypothesis explains our experiment. This directive reads in our case as follows: with a significance of 0.012, (a proportion of less than 12 in 1000) we

may conclude that temperature and pH influence the average production of ATP by mitochondria.

315 ***Exercise*** *Run the code of the previous program and verify that it supports all done claims.*

316 ***Challenge*** *Make a simulation study of a* 3×4*-block design.*

317 ***Challenge*** *We have produced some frequency tables of the distribution of certain stadigraphs. These histograms are estimations of well-known theoretical distributions. Use an appropriate statistical test to compare our estimations with theoretical ones. The use of Mathematica to generate the theoretical distributions might be of help.*

19.3 An incomplete block design

318 ***Motivation*** *In an experiment one varies the form of the input to the system and records the output. Then, one submits the data to a statistical analysis. The problem is that the mathematics associated to real circumstances is in general very complex or even unknown. That is why some few designs have been worked out completely and the experimenter tries to box science in within them.*

But it may happen that under the best wishes, the final result does not fit the statistical box planned for the experiment. For instance, one may lose a datum in a block design and an entry in the corresponding rectangular data matrix gets empty. We say that we have an ***incomplete design****. Mathematicians have solved the problem for an incomplete block design and they know what must be done. But we cannot expect the same for every design.*

319 ***Objective*** *We will produce a methodology that is universally applicable to any incomplete design - of whatever nature- and that is not available without simulation.*

320 ***The duplex methodology***

To fix ideas, let us analyze a block design with one lost datum. Our matrix is the following:

Mitochondria			
ATP production			
	pH 4	pH 6	pH 8
30^o	5	8	6
36^o	7		8
42^o	4	7	3

Were the matrix complete, we would calculate an F stadigraph, which is expected to have an F distribution. But the design is not complete, so one could try to adapt the calculation of the F stadigraph to our circumstances but causing the minimum modification to the algorithm to calculate the F. One can see that this option leads to nowhere: negative values of the modified F would appear plus a great uncertainty in the interpretation about what one really does.

Let us try now another strategy: let us perturb data and keep the methodology intact. In this method we compare two distributions and that is why our method is called duplex. Let us see how.

The first distribution begins with inventing a datum to fill in the missing one with a random number generated according to a null hypothesis, say from a normal distribution inferred from the other real data. Next, we calculate the desired stadigraphs on this refilled matrix. Varying the random number produces varying stadigraphs, which define a distribution. This is our first distribution. The second one comes from matrices completely filled in random numbers all around with the same normal distribution taken in the first case. We get in that way pairs of distributions to be compared. If they agree to the desired degree, we may support the null hypothesis otherwise we could reject it.

321 ***Null hypothesis: our system is completely insensible in average to the external row and column factors. Thus, under this null hypothesis, data are completely characterized by a global mean and the corresponding variance around it.***

322 ***The simulation follows:***

We use in the next program the new instruction:

```
if (!(i==2)) |(!(j==2))
```

which means:

```
if [ (it is false that (i =2))
  or (it is false that (j =2))]
  then do this and that
```

The program follows:

```
// Program 322 StatTest5
// An incomplete block design.
```

```
// Duplex method (see text).



package ejvol1;



import java.util.Random;


public class StatTest5
{
    //Study of an incomplete block design


    //Turn on  the random generator
    static Random r  = new Random();

    static int freqTableC[] = new int[10001];
    static int freqTableR[] = new int[10001];
    static double help[] = new double[25];
    static  double meanProj;
    static  double deviationProj;
    static double M[][] = new double[4][4];
    static double means[] = new double[4];
    static double Fcolumns;
    static double Frows;


    public static void freqTableToZero()
    {
        for(int i = 0; i<=1000; i++)
        {
            freqTableC[i]= 0;
            freqTableR[i]= 0;
        }
    }
```

```
    public static void reportM( )
    {
        for(int i = 1; i<=3; i++)
        {
            for(int j = 1; j<=3; j++)
                System.out.print(  M[i][j] + " \t");
            System.out.println();
        }
        System.out.println();
    }


    //Fills a 3x3 with random numbers
    //with a normal distribution
    //with mean 2.222
    //and standard deviation 1.483
    public static void randMatrix()
    {
        double s;
        for(int i = 1; i<=3; i++)
        {
            for(int j = 1; j<=3; j++)
            {
                s =  deviationProj * r.nextGaussian()
                        + meanProj;
                M[i][j]= s;
                //System.out.print(  M[i][j] + " \t");
            }
            //System.out.println();
        }
        //System.out.println();
    }


public static void FF()
{
// reportM(mExp);
/*GSi Great  Square under treatment i =
 (\sum^{n_i}_{j=1}X_{ij})^2 */
double GS1 = Math.pow(M[1][1] + M[2][1]+M[3][1],2);
```

```
double GS2 = Math.pow(M[1][2] + M[2][2]+M[3][2],2);
double GS3 = Math.pow(M[1][3] + M[2][3]+M[3][3],2);
//System.out.println("GS1 "+ GS1);
//T= \sum^k_{i=1} \sum^{n_i}_{j=1} X_{ij}

/*GS= Great  Square =
(\sum^k_{i=1} \sum^{n_i}_{j=1} X_{ij})^2*/

double GS = Math.pow(M[1][1]+M[2][1]+M[3][1]+
M[1][2] +M[2][2] +M[3][2]+M[1][3]+M[2][3]
+M[3][ 3],2);


/*Among Columns SS =
 \sum^k_{i=1}\frac{GS_i}{ l_i}-\frac{GS}{N}.*/
double ACSS = (GS1 / 3 + GS2 / 3 + GS3 / 3) - GS /9;

/*Among columns DF =
Among columns degrees of freedom= k-1= 3-1=2.*/


/*Among columns Mean square=
 \frac{Among Columns SS}{Among columns DF}*/

double ACMS = ACSS / 2;


//The same but for rows or blocks


double BGS1 = Math.pow(M[1][1] + M[1][2]+M[1][3],2);
double BGS2 = Math.pow(M[2][1] + M[2][2]+M[2][3],2);
double BGS3 = Math.pow(M[3][1] + M[3][2]+M[3][3],2);

// System.out.println( "  BGS1 "+ BGS1);
/*Among Rows SS =
$\sum^k_{i=1}\frac{GS_i}{ k_i}-\frac{GS}{N}$ */

double ARSS = (BGS1/3 + BGS2/3 + BGS3/3) - GS/9;
// System.out.println( "ARSS " + ARSS);
```

```
/*Among rows DF=
Among rows degrees of freedom= l-1= 3-1=2.*/

/*Among rows Mean square=
  \frac{Among Rows SS}{Among rows DF}*/

double ARMS = ARSS / 2;
// System.out.println( "ARMS " +ARMS);

//Calculates the sum of the squares of   entries.


double UIK = Math.pow(M[1][ 1] ,2)
+ Math.pow(M[2][ 1] ,2)
+ Math.pow(M[3][ 1] ,2)+ Math.pow(M[1][ 2],2)
+ Math.pow(M[2][ 2],2)
+ Math.pow(M[3][ 2],2) + Math.pow(M[1][ 3] ,2)
+Math.pow( M[2][ 3] ,2) + Math.pow(M[3][ 3] ,2);


//Total SS=Total Sum of Squares=
//$\sum^k_{i=1}\sum^{n_i}_{j=1}[X_{ij}^2]-\frac{GS}{N}$

double TotalSS = UIK - GS / 9;
// System.out.println( "TotalSS " + TotalSS); //OK
//Error SS=error sum of squares=
//Total SS - Among columns SS -Among row SS

double ERRORSS = TotalSS - ACSS - ARSS;

//System.out.println( "ERRORSS " + ERRORSS);

//Error DF= Total DF - Among groups DF - Among rows DF
//=    8-2-2=4 .

//Error Mean square= \frac{Error SS}{error DF}

double EMS = ERRORSS / 4;
//System.out.println( "EMS " + EMS);
  Fcolumns = ACMS / EMS;
```

```
  Frows = ARMS / EMS;
//  System.out.println( Fcolumns  ); //OK
// System.out.println( Frows  ); //OK

}

private static void reportFreqTable(int k)
{
    System.out.println( " i + Columns + Rows " );
    int mc = 0;
    int mr = 0;
    int i = 1;
    while ((mc<k)| (mc < k))
    {
     // System.out.println( "b "  );
     mc = mc + freqTableC[i];
                              mr = mr + freqTableR[i];
    System.out.print (i+ "\t " + freqTableC[i]
            + "\t " + freqTableR[i]);


    /*System.out.println(i+ "\t " + freqTablec[i] +
     "\t " + mc + "\t "+ mr ); */
    System.out.println() ;
    i = i+1;
    }
}


// We copy the matrix into a vector
// The central entry is ignored.
// The code
// if (!(i==2)) |(!(j==2))
// means:
// if ( (it is false that (i =2))
// or (it is false that (j =2)))
// then do this and that
private static void meanVariance()
{
    int  n =0;
```

```
    for(int i = 1; i<=3; i++)
    {
            for(int j = 1; j<=3; j++)
            {
                    if ((!(i==2)) |(!(j==2)) )
                    {
                            n = n +1;
                            help[n]= M[i][j];

                    }
            }
    }
    //System.out.println( "count \t " + count);
    /*for(int i = 1; i<=count; i++)
        System.out.println( help[i]); */
    /* We calculate mean and deviation of the vector:
        Let us calculate the mean */
    double sum = 0;

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

    for ( int i = 1; i <= n; i++ )
        sum = sum + help[i];
    double mean = sum / n;
        System.out.println( "Sum \t "+ sum);

    System.out.println( "Mean \t "+ mean);
    // Let us calculate the variance
    double sum2 = 0;

    for ( int i = 1; i <= n; i++ )
        sum2 = sum2 + (help[i]-mean)*(help[i]-mean);
    double var = sum2 / n;
    System.out.println( "Variance \t "+ var);
    meanProj = mean;
    deviationProj = Math.pow(var, 0.5);
    System.out.println( "Deviation \t "+ deviationProj);
    System.out.println();
}
```

```
//Only the nonzero blocks matters
// This is an incomplete design:
// The central 0 is a provisional value:
// it stands in the place of a hole and
// it will be replaced by a random number.
// To that aim, we calculate the mean
// and variance of other data
// to design the normal distribution
// from which the random number will
// be sampled.
private static void initialization( )
{
    //The central 0 represents a hole and
    //will be is replaced by
    //a random number.
    final double mExp[][] = {{0,0,0,0},
                             {0,5,8,6},
                             {0,7,0,8},
                             {0,4,7,3},
                             };


    for(int i= 1; i<=3; i++)
            for(int j= 1; j<=3; j++)
                    M[i][j] = mExp[i][j];
    System.out.println( "Experimental matrix"  );
    System.out.println( "The central 0 is a hole"  );
    reportM();
    meanVariance();
}

private static void completing( )
{
    M[2][2] = deviationProj *
         r.nextGaussian()+meanProj ;
}

public static void main( String args[])
{
    // Definition of the exp matrix
```

```
// calculation of projected values
initialization();
int n = 1000 ;
freqTableToZero();
// We build n matrixes, in which
// the void entry is filled
// with a random number
// and the statistics are calculated.
System.out.println( "Distribution of F values ");
System.out.println( "for the refilled matrix");
for(int k=1; k<=n;k++)
{
    completing();
    FF();
    int d = (int)  Math.ceil(  Fcolumns);
    freqTableC[d]=freqTableC[d]+1;
    d = (int)  Math.ceil(  Frows);
    freqTableR[d]=freqTableR[d]+1;
}

int k =n;
reportFreqTable(k);

freqTableToZero();

//We generate n random matrixes
// filled in numbers that
// follow a normal distribution
// with mean meanProj and
// deviation deviationProj
System.out.println( "Distribution of F values ");
System.out.println( "for the random  matrix");
for(int i=1; i<=n;i++)
{
    randMatrix();
    // reportM(M);
    //System.out.println( i + "\t " + "F = \t " + F());
    FF();
    int d = (int)  Math.ceil(  Fcolumns);
    freqTableC[d]=freqTableC[d]+1;
```

```
            d = (int)  Math.ceil(  Frows);
            freqTableR[d]=freqTableR[d]+1;
        }
        k =n;
        reportFreqTable(k);
        // We calculate the F's for the experiment

        /* What does Math.ciel do?
         int d = (int)  Math.ceil(  1.2);
         System.out.println( d);
         d = (int)  Math.ceil(  1.7);
         System.out.println( d); */
       }
}//End of Program 322 StatTest5
```

323 ***Exercise*** *Run the code. It produces 4 distributions. In a first instance, it presents the distributions of F stadigraphs for Columns and Rows of the original matrix that has been filled at its empty central cell with a random number with a normal distribution. In a second instance, the program presents the same stadigraphs but for a matrix that has been completely filled in random numbers. Draw the 4 distributions. Make a K-S test to compare the distributions for columns under both treatments and then do the same but for rows.* Answer

324 ***Exercise*** *Digest the code. Help yourself activating the tracer instructions.*

325 ***Challenge*** *Our previous simulation considered the model:*
datum = average value + fluctuation.
Study the model
datum = great average + average differential effect of row + average differential effect of column + fluctuation.
Implement this new model under the duplex method.

19.4 Conclusion

Ordinary statistics can be understood as a result of gedanken (mental, imagined, thought) simulations made through mathematical calculations. They are possible when problems are not very complex. But when one faces complexity one has two options: to use very general but inefficient tools, as non-parametric analysis, else to make a simulation, which is very efficient but completely devoid of generality. Nevertheless, a simulation for statistics has a very clear scheme that may

go through adaptations to specific purposes, such as we have seen in the duplex method for analysis of experiments with incomplete data.

Chapter 20

Genetic drift

`Evolution without mutation but due to sampling effects.`

326 ***Problem and purpose*** *Since mutation is always at work, many variants or* ***alleles*** *of the same enzyme may exists. They could have different fitness but, according to the neutral theory of evolution proposed by Kimura ([7], 1997), most variants are equivalent in practice, so that short term evolution is just the random change of the frequencies of the diverse alleles in the population. More explicitly, the change in allele frequencies is not due to selection pressure but to the sampling effects implied in reproduction plus the ecological limitations over the size of the population.*

Sampling effects cause genetic frequencies to change as a ship that is erratically dragged by the wind from one place to another. That is why this phenomenon is known among population geneticists as ***drift****. Our purpose is to simulate a world in which drift is implemented ab initio.*

20.1 One population

We proceed from very simple models to more complex ones.

327 ***Implementation*** *In our first example, we will consider one asexual population with constant size, one locus or enzyme, two alleles and no mutation. We will follow the dynamics of the allele frequencies and we will have the opportunity to witness that always one allele disappears and the other gets fixed in the population. Next we pass to a statistical study of the probability of* ***fixation****.*

328 ***Example*** *The next code models a population with constant size, one locus or enzyme, two alleles and no mutation. The frequency of one of the alleles is reported generation after generation.*

```
package ejvol1;


// Program 328 Drift

// In this program we study drift
// in one asexual  population with constant size,
// one locus or enzyme, two alleles and no mutation.

//One allele is eroded and the other is fixed.
//Reported: Repetition + final frequency of allele1
//and the probability of fixation of allele1 for a given
//initial probability of allele1

import java.util.Random;

public class Drift
{


    //Turn on  the random generator
    static Random r  = new Random();


    static int size; //population size.
    static double po; // initial probability of allele one
    static int nGen; //number of generations
    static double counterAllele1;
    static double counterFixations;
    static double nRepetitions;
    static double probFixation;
    //Individuals have one of two alleles
    //True: allele1; false = allele2
    static boolean individual1[],individual2[];
```

```
public static void initialization()
{
 size = 100;
 po = 0.8;
 nGen = 200;
 counterAllele1 = 0;
 counterFixations=0;
 nRepetitions=1;
 individual1 =new boolean[1001];
 individual2 =new boolean[1001];
 System.out.println("Initial probability of allele1 = "
         + po);

}

//One population during nGen generations.
//The new generation is generated by random sampling.
public static double dynamics()
{
    System.out.println("Generation "
            + "+ relative frequency of allele1");
    //Local probability of fixation
    double f = 0;
    for(int i=0;  i < nGen; i++)
    {
        counterAllele1=0;
        //Sampling procedure
        for(int j=0; j < size; j++)
        {
            int  m = r.nextInt(size) ;
            individual2[j]= individual1[m];

            if (individual2[j] == true)
                counterAllele1 = counterAllele1 +1;
        }
        f=counterAllele1/size;
        System.out.println(i + "\t" + f);

        /* String s4 = String.valueOf(p);
```

```
            String  s5=s4.replace(".", ",");
             System.out.println( s5);  */


            for(int j= 0; j <  size; j++)
                individual1[j]= individual2[j];
        }
        return f;
    }

    //For each repetition,
    public static void repetitions()
    {
        //System.out.println("Repetition + final frequency of allele1"
        for(int j=0; j < nRepetitions; j++)
        {

            //initial population:
            //allele1=true;
            //allele2=false;
            //All have allele2
            for(int i=0; i< size; i++)
                    individual1[i]=false;
            //Some get allele1
            int n = (int) Math.ceil( po*size);
            for(int i = 0; i < n ; i++)
            {
                individual1[i]=true;
            }
            //frequency of fixation after dynamics
            double f = dynamics();
            System.out.println(j + "\t" + f);
                if(f==1)counterFixations=counterFixations +1;
        }
        probFixation = counterFixations/nRepetitions;
        System.out.println("probFixation = " + probFixation);
    }

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

```
        initialization();
        repetitions();
    }
}//End of Program 328 Drift
```

329 **Exercise** *In the method 'initialization', set the parameter nRepetitions = 1, in 'dynamics' turn on the instruction that prints the actual value of the relative frequency of allele one, f, and run the code. Make a graphic of the dynamics.*

330 **Exercise** *In the method 'initialization', set the parameter nRepetitions = 1000, in 'dynamics' turn off the instruction that prints the actual value of the relative frequency of allele one, f, and run the code to see the probability of fixation of allele one. Repeat this task 10 times for initial $p_o = 0.1, ..., 0.9$. Pursue a regression analysis to predict the slope of the straight line that depicts the relation of initial p_o and the probability of fixation. Show that if in a pure line of allele2 appears by mutation or by immigration an individual with allele1, the probability of fixation is $1/N$ where N is the size of the population.* Answer

20.2 Many populations

If we run many populations in parallel and isolation, we can make statistics.

331 ***Purpose.*** *In the previous exercise, we witnessed that when the population size is constant, any initial distribution of two alleles is faded away to give rise to* ***monomorphism****, in which one of the two alleles prevails. We also arrived to the conclusion that the probability of fixation of allele one is equal to its initial frequency. But we did not determine the form as the relative frequency evolves in time. Nevertheless, we prove in this section that the distribution of relative frequencies in a population of populations is a well defined entity, whose evolution can be clearly depicted.*

332 ***Implementation***

We take 1000 populations, each one with 100 individuals. We study the distribution of the proportion of allele1 in the populations. This program reports a table which displays for generations 0, 20, 40,.. ..100 (vertically) the quantity of populations in which allele1 has frequency i/100 for i = 0, 1, 2,.. .., 100 (horizontally).The code follows:

```
/*
```

```
 * To change this license header, choose License Headers in Project Pro
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package ejvol1;

// Program 332 Drift2

// In this program we study drift
// in 1000 asexual  populations with constant size,
// one locus or enzyme, two alleles and no mutation.

import java.util.Random;

public class Drift2
{

    //Turn on  the random generator
    static Random r  = new Random();


    // number of populations
    static int nPopulations = 1000;
    // population size, the same for all populations.
    static int size;
    static double po;
    // initial frequency of allele one,
    // the same for all populations.

    // actual frequency of allele one
    static int nOne[];
    // actual proportion of allele one
    static double p[];
    // number of generations
    static int nGen;
    // counts the number of allele1
    static double counterAllele1;
    // counts the number of times that allele1
    // was fixed.
    static double counterFixations;
```

```
static double probFixation;
// Record of the type of individuals
// initial population:
// allele1=true;
// allele2=false;
// The first entry is the pop subindex,
// the second entry is the individual subindex
static boolean population1[][],population2[][];
// The distribution of p in the population
static int distribp[];
// We do not report every generation
// but only when the generation is multiple of genSpace.
static int genSpace ;

public static void constructors()
{
    p = new double[1000];
    nOne = new int[1000];
    distribp = new int[1000];
    population1 =new boolean[1000][1000];
    population2 =new boolean[1000][1000];
}


public static void report( )
{
    for(int i=0; i<size; i++)
        System.out.print(distribp[i] + "\t" );
    System.out.println();
}

public static void initialization()
{
    size = 100;
    po = 0.3;
    nGen = 500 ;
    genSpace = 5;
    counterAllele1 = 0;
    counterFixations=0;
```

```
    //All populations are initialized
    int n;
    n = (int) Math.ceil( po*size);
    for(int pop=0; pop<nPopulations; pop++)
    {
        //initial population:
        //allele1=true;
        //allele2=false;


        for(int ind=1; ind<=n ; ind++)
                population1[pop][ind]=true;

        for(int ind=n+1; ind<=size ; ind++)
                population1[pop][ind]=false;
    }

    for(int j=0; j < size; j++)
            distribp[j]=0;
    distribp[n]=nPopulations;
    report( );
}


public static void iteration(int pop)
{
    //Sampling procedure
    for(int j=0; j<size; j++)
    {
        int  m = r.nextInt(size)+1 ;
        population2[pop][j]= population1[pop][m];

        if (population2[pop][j]==true)
            counterAllele1 = counterAllele1 +1;
    }
   p[pop]=counterAllele1/size;
   nOne[pop]= (int) counterAllele1;

   // System.out.println( nOne[pop] + p[pop]);
```

```
        /* String s4 = String.valueOf(p[pop]);
        String  s5=s4.replace(".", ",");
         System.out.println( s5);  */

        //updating
        counterAllele1=0;
        for(int ind=0; ind<size; ind++)
                 population1[pop][ind]=  population2[pop][ind];
    }

    public static void reInitialization( )
    {
        for(int i=0; i<size; i++)
            distribp[i] = 0;
        for(int pop=0; pop<nPopulations; pop++)
        {
            nOne[pop] = 0;
            p[pop]=0;
        }
    }

    public static void thousandPopulations(int gen)
    {
        for(int pop=0; pop<nPopulations; pop++)
        {
            iteration(pop);
            /* System.out.println(pop +"\t"+
                  nOne[pop] + "\t"+ p[pop]); */

            distribp[nOne[pop]] = distribp[nOne[pop]]+1;
        }
        //System.out.println(gen);
        if (gen % genSpace  == 0 )  report( );
        reInitialization();
    }

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

```
        for(int gen=0; gen<nGen; gen++)
        thousandPopulations(gen);
    }
}//Program 332 Drift2
```

333 ***Exercise*** *Run the code and observe the drift of the frequencies of allele1 toward monomorphism, which is evidenced because most populations tend to occupy the same extreme column: the most left means relative frequency zero, the most right means relative frequency one. All intermediate frequencies tend to vanish.*

334 ***Exercise*** *Play with different values of po (initial frequency of allel1) and nGen (number of generations) to verify that the initial frequency and the probability of fixation are statistically equal. You are invited to pursue a regression analysis.*

335 ***Exercise*** *Try to make an understandable three-dimensional graphic of the output of the previous program. Most probably you will feel disappointed with the result. The reason is that data is too dispersed. So, help yourself: instead of calculating the distribution of frequencies using a subdivision in 100 subintervals, round data just to a precision of 10 subintervals, i.e., to the first significant cipher.* *Answer*

20.3 Conclusion

We have witnessed that any initial variability is washed away by the sampling procedure associated with reproduction when mutation is turned off. Thus, monomorphism is the ultimate state of evolution without mutation. The path to monomorphism is similar to a path followed by a ship under drift in the sea but nevertheless the final mode of allele fixation can be statistically predicted: the probability of fixation of a given allele is equal to its initial frequency.

Chapter 21

Graphics and movement

`Drawings that move.`

336 ***Motivation*** *Humans are incredibly good at visual analysis. Thus a significant portion of the stimuli coming from the external world enters throughout the eyes. This is certain for an adult as for a child. In consequence, it is good idea to explore the visual possibilities of Java.*

337 ***Objective*** *We learn in this chapter the basics of making graphics and movement.*

338 ***Beware*** *In this chapter we work with Applets. Actually, we decided to work with Aplets because they allow to display graphics very simply. Java can run many Applets at the same time. Thus, it may happen that one clicks on the red terminator icon of Eclipse to terminate a running Applet, to note some time later that a lot of Applets are running at the same time. So, make sure that there are no applets running behind the scene.*

21.1 Graphics

339 ***My first drawing***

We see in the next program the most simple drawing, that of a **line**. In a drawing program, the main method is called paint, as follows:

```
public void  paint (Graphics g)
{ Code here }
```

A line that goes from (10,10) to (230, 95) is drawn with the instruction:

```
g.drawLine(10,10,230,95);
```

The complete code follows:

```
//Program A339 Drawing
//This class draws a line.

package ejvol1;

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

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

    @Override
    public void paint (Graphics g)
    {
        g.drawLine(10,10,230,95);
    }
}
```

340 ***Exercise*** *Run the program and study its syntax.*

341 ***Exercise*** *Play with the numerical entries in the method g.drawLine of the previous program to make it clear that the first two numbers refer to the x and y coordinates of the initial point of the line and the last two refer to the coordinates of its final point. The origin of coordinates sits at the left superior corner and the positive senses are rightwards for the x and downwards for the y.*

342 ***Example*** *Drawing a rectangle.*

A **Rectangle** is defined by the two coordinates of the superior right corner, width and height.

```
g.drawRect(10,15,100,100);
```

An **ovale** or ellipse is defined by the rect that encloses it.

```
g.fillOval(150,20,20,20);
```

The **color** is defined as follows:

```
g.setColor(Color.red);
```

The next program draws two rectangles, one void and the other filled, and a line:

```
// Program 342 Drawing2
// Two rectangles  and a line.

package ejvol1;

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

public class Drawing2 extends Applet
{
    private static final long serialVersionUID = 1L;

    @Override
    public void paint (Graphics g)
    {
        g.setColor(Color.blue);
        //Initial and final points.
        g.drawLine(10,200,270,150);
        // A Rectangle is defined by
        // the coordinates of the
        // superior right corner,
        // width and hight.
        g.drawRect(10,15,100,100);
        g.fillRect(150,15,100,100);
        g.setColor(Color.red);
        // An ovale is defined by the
        // rect that encloses it.
        g.fillOval(150,20,20,20);
```

```
    }
}//End of Program 342 Drawing2
```

343 ***Exercise*** *Draw a face using ovales, rects and lines. Choose colors among pink, yellow, orange, blue, red, black, cyan, gray, green, lightGray, darkGray, magenta and white.* *Answer*

It is worth mentioning that, apart from many facilities, Java also can draw polygons of the most irregular form and fill in with the appropriate color. This makes of drawing a true art.

21.2 Movement

How can we simulate movement if we can just draw static drawings?

344 ***Example*** *Animation.*

We can produce animations using what we know, but is it good to keep in mind that animation is in Java a refined art. Animations are created by the recurrent composition of two operations: erasing of the object in the old position and drawing of the object in the new position. Let us produce a living face. The next program draws a face with blinking eyes. The program includes delays to make possible the observation of movements by a human being.

```
//Program A344 Drawing4
//This class  draws a face with blinking eyes.

package ejvol1;


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

public class Drawing4 extends Applet
{
    private static final long serialVersionUID = 1L;
```

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

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

public void openEyes (Graphics g)
{
    // small eyes are erased
    // We draw smaller eyes
    g.setColor(Color.pink);
    g.fillOval(30,65,30,5);
```

```
        g.fillOval(70,65,30,5);
        // We redraw eyes:
        // eyes are drawing in blue
        // and normal size
        g.setColor(Color.blue);
        g.fillOval(30,50,30,20);
        g.fillOval(70,50,30,20);
    }

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

    @Override
    public void  paint (Graphics g)
    {
        drawFace(g);
        // This loop control the   blinking number
        for(int i=0; i<4; i++)
        {
            // A delay must be introduced to
            // make movement visible for a human being.
            System.out.println(i);
            long  l1 = System.nanoTime();

            int l = 500;
            for(int j=0; j<4000000; j++)
             delay(l1,l);

            closeEyes(g);
            // The next loop creates a delay,
            // the duration of the delay is regulated
            // by the upper limit of j.

            l1 = System.nanoTime();
            l = l*40;
            for(int j=0; j<2000000; j++)
                delay(l1,l);
            openEyes(g);
```

```
        }
        System.out.println("Job done.");
    }
}//End of Program A344 Drawing4
```

345 ***Exercise*** *Run the program and verbalize its functioning. Make suitable modifications to test your comprehension, say, enlarge the mouth, change the color of the skin, and make a more rounded face.*

346 ***Movement*** *Let us pass now to programming a moving face with blinking eyes. To that aim, we proceed just as before: we erase the face in the old place and redraw it in the new place.*

```
//Program A346 Drawing5
//This class  draws a moving  face
//with blinking eyes.

package ejvol1;


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

public class Drawing5 extends Applet
{

private static final long serialVersionUID = 1L;

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

```
        g.setColor(Color.red);
        g.fillOval(50+k,120,30,20);
        //eyes
        g.setColor(Color.blue);
        g.fillOval(30+k,50,30,20);
        g.fillOval(70+k,50,30,20);
        //nose
        g.setColor(Color.red);
        g.drawRect(55+k,70,20,40);
    }

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


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

```
        g.setColor(Color.blue);
        g.fillOval(30+k,65,30,5);
        g.fillOval(70+k,65,30,5);
    }

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

    }

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

    @Override
    public void  paint (Graphics g)
    {
        drawFace(g,0);
        // This loop controls the numbers
        // of cycles of the graphic
        for(int kl = 0; kl<5; kl++)
        {
            // k is the horizontal jump
            int k = 10*kl;
            //We erase everything
            g.clearRect(0,0,200,200);
            // We draw face in the new place
            drawFace(g,k);
```

```
            // This loop controls the   blinking number
            for(int i=0; i<2; i++)
            {
                // A delay must be introduced to make
                //  movement visible for a human being.
                System.out.print("Cycle = " + kl);
                System.out.println(". Blinking = " + i );


                long  l1 = System.nanoTime();

                int l = 500;
                for(int j=0; j<4000000; j++)
                    delay(l1,l);

                closeEyes(g, k);

                //A delay is inserted
                l1 = System.nanoTime();
                l = l*40;
                for(int j=0; j<2000000; j++)
                         delay(l1,l);

                openEyes(g,k);
            }
        }
        System.out.println("Job done.");
    }
}//End of Program A346 Drawing5
```

347 ***Exercise*** *Run the previous code to see a face that moves horizontally.*

348 ***Exercise*** *Modify the program to display a vertical movement. Adjust the velocity of blinking, of movement, and the length of the jump.* Answer

21.3 Conclusion

Managing drawing and animation in Java is a quite simple task. This enables a graphic display of outputs of programs.

Chapter 22

Prototypes in OOP(Object Oriented Programming)

`From models to realization.`

349 ***Purpose***

We get acquainted with the concept of **prototype**, which is one of the most powerful aspects of **OOP** (object Oriented programming). A prototype is an idea, it is an ideal model of what one want to implement. It is a plan, it is a design. So, one has the opportunity to fabricate instances of the prototype, as many as desired. We learn to define prototypes and how to nest them. Used names -prototype, model, design- does not exist in Java: they serve to connect us with common sense. Instead, Java has the equivalent and reserved word **object** that is synonym of `class`.

22.1 Philosophical directives

350 ***From experience to planning***

Experience has shown that the design of software is a very difficult enterprise. So, software engineering has aimed at the construction of platforms of programming to ease the work. Thus, it could be now the right occasion to introduce more aspects of **OOP (Object oriented programming)** that helps to deal with the complexities associated with design, debugging and interpretation problems. The philosophical directive underneath these advances could be expressed as follows:

1) Developing code is very difficult. So, one shall not expect to develop code the right way on the first trial. To be sure, a code 40 lines long could be corrected some 50 times before one could say that one has what one wanted.

2) Divide, conquer and name. This means that complex problems become easier if one naturally divides them in sub problems. This requires giving a name to each sub problem in such a way that one can at once imagine the nature of the sub problem by just listening at its name. Or, if one is studying a system, that task becomes easier if one divides the system into subsystems. So, programs become structured. We already know this aspect quite well.

3) Related data must be kept united and related structures must form a hierarchy. Tasks must be divided and data must be organized. Organized but not disrupted. Say, if one looks at a vertebrate, one automatically organizes it into head, trunk and limbs. But in the mind all those entities remain united and observe a hierarchy so, the heart is naturally associated with the trunk. This implies that the process of data generation must not cause disruption or distortion of the relations in within the observed world.

4) If any conflict among tasks and data arises, decide in favor of data: data are the natural OBJECT, they form the world. Your code must serve the data and not the other way around.

6) Your programming language and style of encoding shall reflect all this: there shall be a transparent correspondence among observed world, data, algorithm of analysis, simulations and displayed results.

6) Tools used to create code must not propitiate the creation of bugs, errors of programming. So, to allow working with uninitialized variables has proven to be a very effective source of bugs. This explains why Java is so jealous with non initialization. Additionally, every new user type of data must include its own methods for acquisition of data that allows correct and effective initialization. Those distinguished methods are called **constructors**.

7) Another source of bugs is to allow variables to take values on forbidden regions, whose exact nature depends on the problem being solved. This is similar to having a car and driving at 250 miles per hour at the downtown. To keep values in within non dangerous ranges is a responsibility of the developer but Java also offers facilities to do this directly in the constructors.

22.2 Implementation

A good example must suffice to understand the idea.

351 *The hand-arm-body super-structure*

Let us learn how our directives are implemented in Java. We aim at the following purposes: 1) related data must be kept united, 2) related structures must form a

hierarchy 3) Unity and relationships must be naturally reflected in the programming language.

We can illustrate our task with the notion of a hand: it is composite but it is a unit. Thus, we must have the possibility to device a data structure, a **type** that has, say, five fingers. But a hand forms part of an arm, which in its turn forms part of the body. So, we must have the possibility of nesting a hand into an arm into a body. In that way we could have expressions that contain **nested types** as the following:

```
The hand has a thumb 7 cm long.
The body   has an arm with an elbow 45cm and
a hand that has a forefinger with 9 cm.
```

Java has inherited from C++ a formalism in which the two aforementioned expressions read as follows:

```
hand.thumb = 7;
arm.elbow = 45;
arm.hand.forefinger=9;
```

352 ***Beware*** *Initializations of nested types always present troubles. To avoid them, we will try to initialize all variables as locally as possible, i.e. as close as possible to the method that use them. In general, invocation of a non-initialized variable implies the inclusion of garbage. That is why Java is so jealous with non-initialization. In that way, many bugs are prevented.*

353 ***Types and idealism*** *Our formalism includes a strong differentiation between ideas and reality. So, when one defines a type, it is understood as an idea. It is nothing real as yet. The only thing that one can do with an idea is to realize it. So, we have the possibility to make real entities in the image and likeness of the already defined idea. With those* ***instantiations or implementations****, one can work in the desired forms. All this is best understood with the comments includes in the next pieces of code.*

354 ***Lab*** *Please, add the next three classes to any project. Each class must go in its own file.*

This is the first class: it is a Hand, it is a new type of data.

```
// Program A354a Hand
// We define Hand as a prototype, as an object.
// It functions as a new  Java type.

package ejvol1;



public class Hand
{


    /*The object Hand keeps the information
    of the lengths of
    the forefingerLength and the thumbLength */

    int forefingerLength;
    int thumbLength;

    // Constructor:
    // method that specifies how
    // a specific object acquires information.

    // A method is overloaded when its function
    // is implemented in various ways.

    // Hand has an overloaded constructor:


    //Void initialization to be filled in later
    public   Hand( )
    {

    }

    // Information is acquired
    // by an ordinary list
    public   Hand( int a, int b)
    {
```

```
        this.forefingerLength = a;
        thumbLength = b;
    }

    //Information is acquired
    //from structured data:
    public   Hand(Hand hand1)
    {
        forefingerLength = hand1.forefingerLength;
        thumbLength = hand1.thumbLength;
    }

    // An object may contain methods
    // that are specific to it.
    public void initialization()
    {
        // fake, symbolic initialization.
        forefingerLength = -5;
        thumbLength = -3;
    }
}//End of Program A354a Hand
```

The next class defines the type Arm, in which we include a Hand. Please, add this new class to that

project, where Hand was added.

```
//Program A354b Arm
//The object Arm is composite:
//it has a hand and an elbowLength

package ejvol1;




public class Arm  extends Hand
{
```

```
    //It is assumed that hand is in the same package
    Hand hand;
    int elbowLength;

    //Void initialziation
    public Arm ()
    {

    }

    //Initialization from and ordinary list
    public Arm (int a, int b, int c)
    {
       this.hand = new Hand();
       hand.forefingerLength =  a;
       hand.thumbLength = b;
       this.elbowLength = c;
    }
}//End of Program A354b Arm
```

Now we present the definition of the type Body, in which we invoke specific hands and arms:

```
package ejvol1;

// Program A354c Body
//A body is composed of an arm and a neckLength

public class Body  extends Arm
{



    private Arm arm;
    int neckLength;
```

```
    // How does Body acquires data?
    // Many  forms are possible, some follow.


    //Void instantiation  to be filled in later
    Body(  )
    {

    }

    //From an arm and a neckLength
    Body( Arm arm,  int d )
    {
this.arm = arm;
this.neckLength = d;
    }

    //From a hand, an elbowLength and a neckLength
    Body( Hand h,  int c, int d )
    {
this.arm = new Arm();
this.arm.hand = h;
        this.arm.elbowLength = c;
this.neckLength = d;
    }


    //From  forefingerLength, thumbLength,
    //elbowLength and neckLength
    Body( int a, int b, int c, int d )
    {
        this.arm = new Arm(a,b,c);
        this.neckLength = d;
    }




//Object is reported to console
```

```
private static void report(String name, Body body )
{
System.out.println("==== " + name + " =====");
System.out.println("Arm:");
System.out.println("     Hand:");
System.out.print("          ForfingerLength = ");

System.out.println(body.arm.hand.forefingerLength);
System.out.print("          ThumbLength     = ");
System.out.println(body.arm.hand.thumbLength);
System.out.print("          ElbowLength     = ");
System.out.println(body.arm.elbowLength);
System.out.print("     NeckLength           = ");
System.out.println(body.neckLength);
}


//This is how to deal with our objects
public static void main(String[] args)
{
    // We provide information in  a list
    final Body bodyJohn = new Body(11,12,13,14) ;
    String name = "John";
    report( name, bodyJohn );

    // Construction and
    //symbolic initialization
    Body bodyJoseph = new Body(-1,-2,-3,-4);
    name = "Joseph";
    //Real initialization
    //Substructured input
    final Hand  hand1 = new Hand(7,5);
    //True partial initialization
    bodyJoseph.arm.hand = hand1;
    report( name, bodyJoseph );

    //Structured initialization
    final Body bodyPeter = new Body(hand1, 40, 25);
    report( name, bodyPeter );
```

```
    //Cloning
    final Body bodyLukas = bodyJoseph;
    report( name, bodyLukas );

    //Void initialization
    final Body bodySteven = new Body();
    //Activate next line and add a point
    //to see the editor hints.
    //Select a hint with arrows + return.
    //bodySteven
    //report( name, bodySteven );
}
}//End of Program A354c Body
```

355 ***Exercise*** *Save all three classes and run the class Body. Relate its output with the code.*

356 ***Exercise*** *The next pieces of code present the same last three classes but without comments. Add to them all necessary comments that they may be easily interpreted. Can you imagine a better way to name different items than as it was done by the Author? Is that important?*

The hand:

```
// Program A354a Hand


package ejvol1;



public class Hand
{


    int forefingerLength;
    int thumbLength;


    public   Hand( )
```

```
    {

    }


    public   Hand( int a, int b)
    {
        this.forefingerLength = a;
        thumbLength = b;
    }


    public   Hand(Hand hand1)
    {
        forefingerLength = hand1.forefingerLength;
        thumbLength = hand1.thumbLength;
    }


    public void initialization()
    {
        // fake, symbolic initialization.
        forefingerLength = -5;
        thumbLength = -3;
    }
}//End of Program A354a Hand
```

The arm:

```
//Program A354b Arm


package ejvol1;




public class Arm  extends Hand
```

```
{


    Hand hand;
    int elbowLength;


    public Arm ()
    {

    }


    public Arm (int a, int b, int c)
    {
       this.hand = new Hand();
       hand.forefingerLength =  a;
       hand.thumbLength = b;
       this.elbowLength = c;
    }
}//End of Program A354b Arm
```

Now we present the definition of the type Body, in which we invoke specific hands and arms:

```
package ejvol1;

// Program A354c Body


public class Body  extends Arm
{



    private Arm arm;
    int neckLength;
```

```
    Body(  )
    {

    }


    Body( Arm arm,  int d )
    {
this.arm = arm;
this.neckLength = d;
    }


    Body( Hand h,  int c, int d )
    {
this.arm = new Arm();
this.arm.hand = h;
        this.arm.elbowLength = c;
this.neckLength = d;
    }



    Body( int a, int b, int c, int d )
    {
        this.arm = new Arm(a,b,c);
        this.neckLength = d;
    }




private static void report(String name, Body body )
{
System.out.println("==== " + name + " =====");
```

```
System.out.println("Arm:");
System.out.println("     Hand:");
System.out.print("          ForfingerLength = ");

System.out.println(body.arm.hand.forefingerLength);
System.out.print("          ThumbLength     = ");
System.out.println(body.arm.hand.thumbLength);
System.out.print("          ElbowLength     = ");
System.out.println(body.arm.elbowLength);
System.out.print("     NeckLength           = ");
System.out.println(body.neckLength);
}



public static void main(String[] args)
{

    final Body bodyJohn = new Body(11,12,13,14) ;
    String name = "John";
    report( name, bodyJohn );


    Body bodyJoseph = new Body(-1,-2,-3,-4);
    name = "Joseph";

    final Hand  hand1 = new Hand(7,5);

    bodyJoseph.arm.hand = hand1;
    report( name, bodyJoseph );


    final Body bodyPeter = new Body(hand1, 40, 25);
    report( name, bodyPeter );


    final Body bodyLukas = bodyJoseph;
    report( name, bodyLukas );
```

```
    final Body bodySteven = new Body();

    //bodySteven
    //report( name, bodySteven );
}
}//End of Program A354c Body
```

357 ***Variation*** *Let us enable the possibility to speak of wife-husband couples and their bodies:*

```
//Program A357 Couple
//Keeps information in reference to
//the bodies of  a wife-husband couple
package ejvol1;


public class Couple
{
    Body wife;
    Body husband;



    Couple(Body b1, Body b2)
    {
        wife = b1;
        husband = b2;
    }


    public static void main(String[] args)
    {
        Body bodyMary = new Body(1,2,3,4);
        Body bodyAlf= new Body(-1,-2,-3,-4);
        Couple c = new Couple(bodyMary, bodyAlf);

        Body.report("Mary", c.wife);
        Body.report("Alf", c.husband);
    }
```

```
}//End of Program A357 Couple
```

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

359 ***Exercise*** *Create the type Family that keeps information about the bodies of a couple and a child.* *Answer*

22.3 Conclusion

Java allows the possibility to define new types, which we have called prototypes, models, patterns, archetypes and that Java calls Objects or, simple, Classes. A prototype functions as a type of variable. Types usually come structured and nested. A type or prototype is like an idea while an instance is like a real object made on the image and likeness of the idea. Instances of a type may be realized and operated with. We have proved that Java is a language that the User can naturally extend to fit in his or her particular needs.

Chapter 23

Bugs

```
Using OOP to simulate of world of bugs.
```

360 ***Purpose.*** *To exercise ourselves with the prototypes of OOP, we model a two dimensional surface inhabited by bugs that move around but that cannot occupy the same place. We use the new concept of* ***inner class*** *that is a class that is defined inside another.*

23.1 Inner classes

361 ***Prototypes and instances.*** *As we have seen, one can pack each class or program in its own file. But a class might represent a prototype and these usually come nested. So, Java allows the possibility to define a class as inner class of a bigger one.*

A word in regard with style: we have seen that one can partition a **large program** in diverse parts, which can be posited in corresponding files. Java is in fact designed to place every one class in each own file. This is especially suited for solving very complex problems that demand huge amounts of code, say with one million lines. But given the modern possibilities of technology, one thousand lines could be a short program and Java offers the possibility to place and manage it in one single file.

To posit a complex program in one file, one must learn about inner classes, i.e., classes that are included into another and which in general come in nested form. The next program below is a good place to learn how they operate.

```
//Program A361 Bugs
```

```
//We learn about inner classes,
//classes defined in within another one.

//This program defines the prototype
//Bug
//and many of its instances.
//Other inner classes are also defined
//to facilitate programming.

//WARNING:
//To stop the program use the red square or
//the menu Run -> Stop run.

package ejvol1;

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

public class Bugs   extends Applet
{

    //Class bugs  is an archetype.

    //Archetype means old-type, prototype.
    //The word archetype does not belong
    //in evolutionary biology,
    //but it belongs in paleontology.
    //We use here the word archetype as
    //a synonym of a plan or design,
    //of an idea.

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

    static  Bugs  bugs   = new Bugs();
```

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

    // Definition of needed prototypes.
    // Prototypes come in UpperCase,
    // instances in lowerCase.

    // A bug has an address over the screen;

    public  class Address
    {
        int x; // x  coordinate
        int y; // y coordinate

        // The next is a method, a function
        // that  declares
        // how information is acquired.
        // It is a constructor.
        // It is overload by two possibilities.
        // a) The input is a list
        Address(int a, int b)
        {
            x= a;
            y= b;
        }

        // b) The input is a structure
        Address(Address ad  )
        {
            x= ad.x;
            y= ad.y;
        }

    }


    // Bugs can move in any of 9 directions:
    // 8 cardinal points + nothing.
    // A direction is implemented by  a move
    public  class Move
    {
```

```
    int x; // x coordinate
    int y; // y coordinate

    // The next is a method, a function
    // that  declares
    // how information is acquired.
    // It is a constructor.
    // It is overload by two possibilities.
    // a) The input is a list
    Move(int a, int b)
    {
        x= a;
        y= b;
    }

    // b) The input is a structure
    Move(Move move)
    {
        x= move.x;
        y= move.y;
    }

}

//BodyMeasure is defined
int   width =4;


//Constructor of the Bug class
//Prototype definition:
//How information is acquired


public class Bug
{
    //  Each bug has its number, address and color
    int number;
    Address address;
    int color [];
```

```
        // We use various methods to acquire information
        Bug()
        {
            color = new int[4];
        }

        Bug(int a, Address ad, int c[])
        {
            number = a; //default
            address =   ad ;
            color = c;
        }

        Bug(Bug b)
        {
            number = b.number;
            address = b.address;
            color = b.color;
        }
    }

    //Turn on  the random generator
    Random r  = new Random();


    //Each bug lives over the screen,
    //with a given occupation matrix:
    boolean occupied[][];


    //This method is fired automatically
    // at the beginning
    @Override
    public void init()
    {

        //Measures of the window
        int   xWin = 700;
        int   yWin = 800;
        setSize(xWin,yWin);
```

```
}


// Possibilities of movement
static Move  vectMoves[ ];
static Bug[] bug;
static int nBugs;
static int nRounds;

public  void initializer( )
{
    vectMoves = new Move[9];
    occupied = new boolean[200][100];
    bug = new Bug[10000];
}

//To avoid overwriting of bugs,
//an occupation record is kept:
//The screen is divided in cells 4x4.
//At the beginning every cell is empty=false.
//Occupancy is recorded in occupied[][],
//which is a boolean matrix.
//A boolean  variable can be
//true= occupied,  else false= not occupied.
public void  occupationMatrix()
{
    //System.out.println("occupationMatrix");
    for(int i = 0; i < 200; i++)
    {
        for(int j = 0; j < 100; j++)
            occupied[i][j]= false;
    }
 }


// Here we define 9 cardinal points
public void moveDef( )
{
    // System.out.println("MoveDef   ");
    final Move move1 =  new Move(0,1);
```

```
        vectMoves[0]   = move1 ;
       final Move move2 =  new Move(1,1);
        vectMoves[1]   = move2 ;
       final Move move3 =  new Move(1,0);
        vectMoves[2]   = move3 ;
       final Move move4 =  new Move(1,-1);
        vectMoves[3]   = move4 ;
       final Move move5 =  new Move(0,-1);
        vectMoves[4]   = move5 ;
       final Move move6 =  new Move(-1,-1);
        vectMoves[5]   = move6 ;
       final Move move7 =  new Move(-1,0);
        vectMoves[6]   = move7 ;
       final Move move8 =  new Move(-1,1);
        vectMoves[7]   = move8 ;
       final Move move9 =  new Move(0,0);
        vectMoves[8]   = move9 ;
       /* System.out.println(vectMoves[6].x);
        System.out.println(vectMoves[6].y);
        System.out.println(vectMoves[7].x);
        System.out.println(vectMoves[7].y); */
   }

   //An address is generated at random
   public Address  generateAddress( )
   {
       Address   ad = bugs.new  Address (1,1);
       int m = r.nextInt(100);
       int n = r.nextInt(100);
       ad.x =m;
       ad.y = n;
       if (occupied[m][n] == true) generateAddress( );
       return ad;
   }

   //nBugs bugs are initialized
   public void initialization(Graphics g)
   {
   System.out.println("initialization");
   for( int i = 0; i < nBugs;i++)
```

```
{
    // an auxiliary bug is initialized
    Bug  bug1 = bugs.new Bug();
    bug1.number = i;
    // An initial color is chosen
    int c1,c2,c3;


    // a color is assigned to each bug,
    // but different than white
    c1 = r.nextInt(256);
    c2 = r.nextInt(256);
    c3 = r.nextInt(256);
    if (c1+c2+c3==765) c3=1;
    //colors are memorized.
    bug1.color [1]=c1;
    bug1.color [2]=c2;
    bug1.color [3]=c3;


    //The color is prepared
    g.setColor( new Color(c1,c2,c3));


    // An initial position is chosen:
    // symbolic initialization
    Address   ad  = generateAddress();
    bug1.address = ad;
    int m=ad.x;
    int n=ad.y;
    //The bug is  painted.
    //A Rectangle is defined by
    // the coordinates of the
    // superior right corner,
    // width and hight.
    // The cell is painted
    g.fillRect(width * m,width* n,width, width);
    // System.out.println("occupied = " + m + "  " + n );
    //Occupation is recorded
    occupied[m][n]= true;
```

```
        //The bug is memorized
        bug[i]= bug1;
     }
    }



    //Bugs move
    public void movement(Graphics g)
    {
        int number=-1;
        //Try and catch exception pair:

        try   // move
        {
            int  mo,no, m,n;

            for( int i = 0; i < nBugs;i++)
            {
                number = i;
                Bug  bug1 = bug[i];

                mo = bug1.address.x;
                no = bug1.address.y;

                //System.out.println("moveOn");
                //A move is generated  at random

                //Select  which direction to move on
                //Throw a number from 0 to 9
                Move  move   = vectMoves[r.nextInt(9)];


                m = mo +  move.x;
                n = no -  move.y;
                //Barriers
                if (m<0)m=0;
                if (n<0)n=0;
                if (m>199)m=199;
                if (n>99)n=99;
```

```
//is the target cell vacant?
if(occupied[m][n] == false)
{
    //To move:
    //the cell occupied by  the bug is cleared,
    //the cell to be occupied is painted.

    //The occupied cell is cleared=
    //painted in white

    // the white color is prepared
    g.setColor(Color.white);
    // The cell is cleared
    g.fillRect(width*mo,width*no
            ,width, width);
    //Clearing is reported
    occupied[mo][no] = false;



    //Movement=repainting in the new place;
    occupied[m][n] = true;
    g.setColor( new Color(
            bug1.color[1]
            , bug1.color[2]
            , bug1.color[3] ));
    g.fillRect(width*m,width*n,width, width);

    //Address is updated
    bug[i].address.x = m;
    bug[i].address.y = n;

    //If movement is too fast,
    //nothing can be seen.
    //To attune the velocity of movement,
    //augment or diminish
    //the delay as   pleased

    // Execution thread is put to sleep
    // during a delay in milliseconds
```

```
                    int delay =  2;
                    Thread.sleep( delay );
                }
            }//end of for
        }//end of try

        catch ( InterruptedException exception )
        {
            System.out.println(   number+
                "Killed from outside" );
        }
     }


    public void dynamics(Graphics g)
    {
        for(int round= 0; round < nRounds;round++)
        {
            movement(g);
            System.out.println( "Round = " + round);
        }
    }


    @Override
    public void paint(Graphics g)
    {
        System.out.println( "Applet started");
        nBugs = 1000; //
        nRounds = 10000;
        initializer( );
        occupationMatrix();
        moveDef( );
        initialization(g);
        dynamics(g);
    }
}//End of Program A361 Bugs
```

362 ***Exercise*** *Run the program and play with the code: enlarge bugs, change the number of bugs, resize the field where they live, define and implement some*

measurement processes.

363 ***Challenge*** *Previous program serves to track the movement of bugs and to see all places that have been occupied by some bug. Modify the program to erase the marks over occupied places.*

23.2 Conclusion

Types and instances can be placed in the same piece of code using the facility of inner classes. The Java structures that we have learned about - Objects, prototypes, inner classes - were in ancient times something outstanding. Anymore: very complex programming is the ordinary world where modern computing struggles for life. All this is now at the disposition of Biology researchers. In plain words: for the first time in the history of science, biology researchers have now as a group the very heavy responsibility of testing their ideas and beliefs using all the power of Java.

Chapter 24

Conclusions

```
The most important conclusion of our work is that the simulation
of evolution is so full in challenging problems that it
can be chosen as a profession for a life.  Other conclusions
follow.
```

1. A simulation consists in the building in the memory of a computer of a world with the specific properties we want, with specific instruments of measurement and with appropriate procedures of reporting what is happening there.

   Some very important simulations are rather simple while the corresponding mathematical models include hyper complex items, say, a stochastic differential equations, as the stochastic prey predator model. This means that simulation has a great future as a pedagogical tool.

   Nevertheless, simulation could become very complex and very difficult. This must not come as a surprise because simulation is not intended to destroy the complexity of nature. As a consequence, the study of a simulated world could become so difficult that we do not know if we are making any advance in regard with the understanding of the real world. This means that simulations without wisdom may look as silly exercises.

2. The applications of simulation to biology seem to be restricted only by the imagination. Among those made by us, very intriguing is the duplex method to analyze experiments with the use of models for which we have incomplete data. And, of course, the simulation of evolution is an essential elementary right, which is more difficult to exercise than one imagines.

3. Evolution is rooted on mutations. Those that occur in nature in asexual beings correspond exactly to the operations of a word processor. These mu-

tations are readily implemented in any modern programming language. We have simulated many popular mutations, i.e., we have created a virtual world populated by strings that can evolve thanks to mutations of the same sort as those happening in nature.

4. We have simulated evolution to mimic nature and to solve academic problems, such as to find the root of a fifth degree polynomial. These examples demonstrate the general power of evolution: if a computer can solve a problem, then evolution also can. Concretely: any problem that can be formulated as an optimization problem can be solved by simulated evolution. But the solution found is not, in general, a true solution to the given problem but an approximation given by a local optimum. The degree of approximation may depend on the accuracy of the encoding of the problem. We have devised problems for which the solution found by evolution is far from the true solution as much as desired. These facts prompt a question: Which are the limits of natural evolution, if any?

5. Our programs allow us to suspect that evolutionless life could in principle exits. Moreover, the genome of those living beings would be orders of magnitude simpler than those that we see in nature, where evolution exists. Then, we ask: why shall life be so complex? Or, why did not evolution disappear long ago? Or, how does evolution have survived in nature until today? A possible answer is that living beings change the milieu in such unexpected ways that there is no provision for that: just imagine the chemical world. So a vivid, malleable process shall produce adaptation: Is there something better than evolution? Nevertheless, evolution takes its time: infinitely many populations and hundreds of species have disappeared from the Earth because there was not enough time that evolution could find a functional exit to its problems.

6. Simulation generates its own culture. It is so strong that it has the right to dare you: How can you say that you understand evolution if you have not simulated it? This complaint is very actual: with the popularization of genetic engineering, many people seem to imagine that directed mutations correspond to the modern way of making natural evolution. No one, of course, would recognize that, but the fact is that it is very difficult for a beginner in programming to distinguish directed from undirected mutations.

7. Simulation generates its own ethics. The ethics of simulation is based on the historical responsibility that we have: any modern desktop is thousands of times better than those primitive super-computers that at their time were

worth one million dollars and, moreover, present computers are enlivened by software whose development amounts to a billion dollar. Thus, in the light of history, the opportunities are so huge that it is anymore allowed to pretend that science can be made on **gedanken experiments** alone. Instead, simulation is the historical duty of our days.

8. Random numbers are found at the root of any simulation. So, it is important to make clear some important points. The first is that randomness is not synonym of zero-knowledge. It is certain that in a sequence of random numbers it is impossible in practical terms to predict which will be the next element. But, one can find the distribution of the corresponding random variable along with important stadigraphs, in particular its mean and variance. That is why simulations are important in science: they allow making quantitative predictions.

9. In strict sense, the predictions made by a simulation are solid only after an infinite number of events. But for any finite number of events, one must recognize that everything is possible for randomness, especially when there is no possibility of repetition. This is why we say that randomness is the null hypothesis in science: it explains any physical thing. Now we have a question: if everything is explained by randomness, which is the role of science? The work of science is to produce explanative models that, in comparison with randomness, suggest more certainty, need less iterations, are more simple, are more elegant and are more predictive encompassing a great number of phenomena.

10. When one allows randomness to do the same task over and over, there always appears a typification of the needed number of iterations required to do the task. Curiously enough, all experiments made in this course observed an exponential number of iterations as a function of some input parameter that one can take as an indicator of the complexity of the task.

11. Everything is possible for randomness, but in general, it arrives at the solution of any specific problem rather late. Why? The reason is this: in the hands of a human or in those of nature, randomness has too few channels to express itself. So, it creates some objects rather quickly, at once, but others appear late in its future and if on needs one of those that are retarded, one is in risk of general failure. That is why one uses to say in evolutionary biology that mutation is orthogonal to function.

12. Our experience gathered along this course clearly shows that the work of one person is insufficient to give a serious revision to any important problem.

The solution is to form teams of works all across the world. Java and modern development environments include the possibility to work in teams. Besides, the grid possibilities of Java allow us to unite million of computers into a single computing frame.

Glossary

Algorithm An ordered set of instructions for solving a problem.

Allele A variant form of the same gene.

Amino acid A building block of a protein. The genetic code deals with 20 amino acids but there are more than 150 amino acids in the biochemistry of plants. In the proteins of humans there are 25 amino acids, the 20 of the genetic code plus other five that result as modifications of the firsts.

Applet A type of program in Java specially suited to deal with graphics. They are the enlivening actor of the web.

Application program A computer program to fulfill a set of functions.

AWT (Abstract Window Toolkit) The initial facility of Java to make GUIs (graphic user interfaces). It has been superseded by Swing and recently by JavaFX.

Base pair A building block of DNA: A (adenine) , T (thymine) , C (cytosine) and G (Guanine). In RNA the T is changed by U (Uracil).

Binary The quality of having only two possibilities.

Binary code A message written in an alphabet consistent of 0 and 1. The rules to decipher such a message.

Biocomputation Computation done by biological processes.

Bioinformatics The study of how information is represented and transmitted in biological systems.

Bit A quantity and measure of information that corresponds to a yes else not answer.

Boolean operators The mathematical operators AND, OR, and NOT.

Bug Misbehavior of a program.

Cell The building block of any living organism.

Chromosome The structure formed by DNA and the proteins that cover it.

Chip Encapsulated electronic circuit.

Clone A population of genetically identical beings. Quantum mechanics forbids the existence of exact clones.

Cloning The formation of clones or exact replicas.

Coding region A portion of DNA that encodes for a protein.

Code A portion of a program.

Codon A sequence of three adjacent nucleotides that encodes for an amino acid or a signal to begin or stop transcription.

Complex The quality of an object that cannot be described with few words.

Compiler A program that translates a portion of code written in Java to a machine-language program, which can be understood by the CPU.

Computer program A set of instructions that tells a computer which operations to perform.

Computer A machine that with the appropriate instructions can run programs.

Conditional probability The probability of an event that depends on the occurrence of another event.

Convergence The quality of approaching a final state.

CPU (Central processing unit) The brain of the computer.

Datum The record of an observation.

Debugger A program that helps the developer to find bugs.

Debugging The process of cleaning bugs from a program.

Deletion The operation of cutting some substrings from a string.

Design Developing of a program or an artifact.

Designer Any entity (rational or not) that is committed to design.

Distributed The quality of being partitioned in pieces.

DNA (deoxyribonucleic acid) The molecule that contains the genetic information in most living beings.

eJGrid The Grid of the evolJava community. To be done.

Environment The world outside the studied system.

Eukaryote A cell whose nucleus is bounded by a membrane and that has organelles.

EvolJava community The community of all persons that currently use Java as a precious tool to study own and foreign ideas about evolution. Since our task surpasses the possibilities of one single person, our community must soon get organized. Please, visit http://www.evoljava.com or write to the Author: jose@evoljava.com

Evolution In epistemology, it is the scientific project whose aim is to prove that variations along living beings can be explained by a natural mechanism from just some few origins or maybe from one or maybe from no origin at all. In evolutionary biology, it is the change along lineages that add together to give rise to the origin of new species and phyla. In the genome project, evolution is the union of all sort of factors that affect genomes of individuals or of populations, be it mutations from parents to offspring or changes from a germ cell to a somatic one or

transfer of genetic information from an organism to another. In sociobiology, it is the fundamental force that programs instincts.

Evolutionism The philosophical vision that evolution is the correct and final explanations to our existence. This vision rests upon materialism, which by definition cannot be questioned.

Execute To perform instructions.

Exon A region of DNA that is transcribed to an amino acid substring.

Failure An inability to perform the assigned function.

Gedanken A German word that means thought, imagined, say, a gedanken experiment. This was the only form of attacking the fundamentals of science in ancient times. The new one is simulation. Simulation all along the wide world with Java is possibly the future of simulation. Warning: simulation alone is insufficient to account with all questions of biology, so we need mathematical models and very clever gedanken experiments.

Gene A segment of DNA that encodes a protein and all the signals that regulate its transcription.

Genetic code The correspondence among codons and amino acids + start and stop signal for transcription.

Genome The genetic material of an organism.

Genotype The specific genetic information of an individual.

Google An Internet tool to search documents on the web. It does not demand fees but invades the world of the user with advertisements and moreover it sells to others his or her electronic address and usual searched motifs.

Grid The union of many desktop computers in a single structure to exploit the potentialities of parallel computation.

GUI (graphic user interface) The nice face of programs with windows, menus, buttons and the like.

Haploid An organism whose cell contain only one set of chromosomes.

Hardware The wiring of a computer.

Information theory The science of communication.

Interface A piece make to connect two other parts of a system.

Intron A part of DNA that is non-coding.

JavaFX The modern proposal of Java to make graphics, animations and sound. It is still young and therefore a bit unstable but anyway it is a great promise. Internet compatible.

Machine language The set of instructions in binary code that are understood and executed by a computer processor.

Markov chain A probabilistic model in which a system can occupy some states and that has constant probabilities to pass from one state into another.

Memory Chips that are used to store programs and data.

Model An abstraction of some part of the world.

Modeling The art of making models.

Mutation A change in the genetic information.

Neural network A computing devise that imitates nervous system.

OOP (Object-Oriented Programming) The style of programming in which data, operators and protocols of communication are encapsulated into a single entity.

Occam's razor The advise to prefer simplicity when it explains all things.

OS (Operating system) The software that enlivens a computer.

Operator A machine that transforms some input.

Output The result produced by a process or program.

Parameter Variable of a model whose value can be changed by the user.

Phenotype Any observable feature of an organism whose existence can be traced to some few genes.

Point mutation A change of a single nucleotide in a DNA sequence.

Programming language A set of reserved words and rules to specify how to command a computer.

Prokaryote An organism whose cells lacks a membrane-bounded nucleus.

Prototype A model of something to be built.

RAM (Random-access memory) A portion of electronic memory that can be used to write and retrieve information.

Recombinant The product resulting from the interchange of different portions of two initial items.

Recombination A string operation that receives as input two strings and produces one or more strings by interchanging their parts.

Recursion The repeated call to the same procedure.

Simulation A program that represents a model.

Software Computer programs.

Structured programming The style of programming in which modular approach and reuse is preferred.

Swing A package to generate GUIs in Java. It is a very modern and robust toolkit for making a Java Interface.

Thread A specific sequencing of execution. Java can run many threads at the same time.

Transcription The operation of making mRNA, which is a RNA photocopy of some DNA information.

t-RNA (Transfer RNA) A small RNA molecule that serves to the cell to decipher a genetic sequence thanks to the genetic code: in one extreme it has a codon and in the other it has the corresponding amino acid.

Translation The process of converting mRNA to protein.

Weasel concourse The concourse to produce the most proficient program to guess a word using evolution.

Solutions

Problems of Chapter 1

8, page 8. Complete instructions are finished with ";". Commentaries are initialized with // and can be ended with any sign. A line that declares the name of the class or of a method does not have ";", instead it has a key pair: {}

11, page 9.

```
//Program   A11  PrintMyName
//This program  prints my name and birthday
//to the console.

//Package declaration
package ejvol1;


//Class declaration
public class PrintMyName
{
    //The next is the main method
    public static void main(String[] args)
    {
        System.out.println("I am JB, the Intelligent! ");
        System.out.println("My birthday: March the 5th! ");
    }
}
```

Problems of Chapter 2

16, page 12.

```
//Program A16 SumOfIntegers2
//This program introduces integers
//and their addition.

package ejvol1;

public class SumOfIntegers2
{
    public static void main(String[] args)
    {
        //Declaration and initialization
        //of variables as integers.
        int x = 2;
        int y = 3;
        //Computation
        int result  = x+y; //result is declared as integer
        // Report
        System.out.print(" The sum of x =" + x + " and ");
        System.out.println( " y = " + y + " is " + result );
    }
}
```

17, page 12.

```
/*
 * Program A17 ArithmeticsWithInt
 * Integer arithmetic operations.
 */
package ejvol1;


public class ArithmeticsWithInt
{
 public static void main(String[] args)
  {
//Declaration and initialization
// of variables
int x = 7;
int y = 2;
```

```
//Computation
int result1  = x*y;
int result2  = x/y;
int result3  = x-y;
// Report
System.out.print(" The multiplication of x =" + x );
System.out.println( " and  y = " + y + " is " + result1);
System.out.print(" The integer division of x =" + x );
System.out.println( " by  y = " + y + " is " +  result2);
System.out.print("  x =" + x );
System.out.println( " minus  y = " + y + " is " + result3);
  }
}
```

19, page 14.

```
  /*
 * Program A19 StructuringProgramming2
 * Some code is encapsulated into methods
 */
package ejvol1;


public class StructuredProgramming2
{

    //The next is  a method.
    //It encodes for a function with input
    //two integers and output their sum.
    //The output is also integer and must be declared
    //explicitly before the name of the method.

    //The output of the method appears after
    //the reserved word "return".
    //The declaration of a method or a class
    //does not terminate with semicolon ";"
    public static int sum(int x, int y)
    {
        return  x+y;
```

```
    }


    public static int multiplication(int x, int y)
    {
        return  x*y;
    }

    public static int subtraction(int x, int y)
    {
        return  x-y;
    }

    public static int division(int x, int y)
    {
        return  x/y;
    }

    //This is the main method.
    //This is the head of the class.
    public static void main(String[] args)
    {
        //Declaration and initialization
        //of variables
        int x = 2;
        int y = 3;
        //Invocation of a method
        int z1 = sum(x,y);
        int z2 = multiplication(x,y);
        int z3 = subtraction(x,y);
        int z4 = division(x,y);
        // Report to the console
        System.out.print(" The sum of x =" + x + " and ");
        System.out.println( " y = " + y + " is " + z1);
        System.out.print(" The multiplication of x =" + x );
System.out.println( " and  y = " + y + " is " + z2);
System.out.print(" The integer division of x =" + x );
System.out.println( " by  y = " + y + " is " +  z3);
System.out.print("  x =" + x );
System.out.println( " minus  y = " + y + " is " + z4);
```

```
    }
}//End of program A19 StructuringProgramming2
```

20, page 14. The order does not matter, the program runs softly no matter the relative position of the methods along the text.

25, page 18.

```
//Program A25 DivisionByZero
//Division by zero

package ejvol1;

public class DivisionByZero
{
    public static void main(String[] args)
    {
        //Declaration and initialization
        // of variables
        int x = 6;
        int y = 0;
        System.out.print(" The division of x =" + x
                + " by  y = " + y  + " is ");
        //Computation
        int result  = x/y;
        // Report
        System.out.println(result);
    }
}
```

When Java is commanded to run this program, it answers: 'Exception in thread "main" java.lang.ArithmeticException: / by zero'.

A division by zero leads to a contradiction:

$0 = 3 \times 0 = 5 \times 0 = 0$

Dividing by zero, we get:

$3 = 5$

which is a contradiction.

27, page 19. The code is robust against division by zero but the division of 7 by 2 renders 3 instead of 3.5.

Problems of Chapter 3

36, page 25. The new program prints numbers from 1 to 7 and their squares. The code follows:

```
/*
 * Program A36 ForLoop2
 */
//Introduction of the for-control structure.
//The "for" instruction causes a loop
//to deal with repetitive tasks.
package ejvol1;


public class ForLoop2
{

    public static void main(String[] args)
    {
        String tab = "\t";

        for(int i = 1; i<=7; i++)
        {
            System.out.println(i + tab  + i*i);
        }
    }
}
```

37, page 25.

```
/*
 * Program A37 ForLoop3
 */
// Variation of the for statement.

package ejvol1;


public class ForLoop3
{
```

```
    public static void main(String[] args)
    {
        String tab = "\t";
        for(int i = 0; i<=10; i++)
            {
                System.out.println(i + tab + i*i);
            }
    }
}
```

38, page 25.

```
/*
 * Program A38 ForLoop4
 */
// Variation of the for statement.

package ejvol1;


public class ForLoop4
{

    public static void main(String[] args)
    {
        String tab = "\t";
        for(int i = 0; i<=20; i++)
            {
                System.out.println(i + tab + i*i*1);
            }
    }
}
```

41, page 26. JAVA accepts 100000000 as an integer. Anyway, if one doubts that an integer will not suffice, one can use a long integer, which is more generous. Or one can resort to use a disk.

Problems of Chapter 4

44, page 30. Possible problem for some versions of Excel and Calc: they may fail to make a graphic because JAVA uses a period in decimals numbers while Excel and Calc use a comma. This is a bug of intercommunication of developer teams. This is typical. We will tailor our own remedy to this problem in the next section.

45, page 30. The console supports more than 7000 lines of text with two decimal numbers per line. So it is more than enough for a good part of our course.

47, page 31. In previous programs , we used the expression $i++$ to mean that the index be incremented by one. The new instruction

```
for(int i = 0; i<=100 ; i= i + 7 )
```

says that the index must be incremented by 7 beginning from 0 and ended with 100 included.

49, page 31.

```
//Program A49 ForLoop8
//The for statement
//with fractional increments

package ejvol1;


public class ForLoop8
{
    public static void main(String[] args)
    {
        String tab = "\t";
        // The index is declared as "double".
        for(double i = 1; i<=2.05  ; i= i + 0.05 )
        {
            System.out.println(i + tab + i*i );
        }
    }
}
```

Problems of Chapter 5

52, page 34.

```
//Program A52 Concatenation2
//Concatenation of strings
package ejvol1;


public class Concatenation2
{

    public static void main(String[] args)
    {
        //Declaration of type of variables
String s0, s1, s2, s3, s4;
        //Assignation
        s0 = "My aim is to become ";
        s1 = "the best ";
        s2 = "simulator ";
        s3 = "of evolution ";
        s4 = "in the history of the world";
        //Declaration and assignment
        String s5 = s0 + s1+s2+s3 +s4;
        System.out.println(s5 + "!"); //action
    }
}
```

54, page 35.

```
//Program A54 Replace2
//Replacement of a substring with another one.

package ejvol1;


public class Replace2
{
```

```
    public static void main(String[] args)
    {
        //Declaration of type of variables
        String s1, s2;
        //Assignation
        s1 = "No limits to my imagination.";
        System.out.println(s1); //action
        s2 = s1.replace("limit", "barrier");
        System.out.println(s2); //action
    }
}
```

57, page 36. As a rule, the first versions of a program are redundant. So, a process of simplification could follow. Let us remember this: viable programs are easier to design if they are redundant. Does this apply to evolution? The answer is affirmative and at the end of our course we will produce material in which this question could be studied. The next is a simplified version of program A55.

```
//Program A57 Replace4
//Simplified version of program A55

package ejvol1;



public class Replace4
{
    public static void main(String[] args)
    {
        String tab = "\t";
        for(double i = 1; i<=2.05 ; i= i + 0.05 )
        {
            String s1 = String.valueOf(i);
            String s2=s1.replace(".", ",");
            String s3,s4;
            //We calculate the square of i:
            s3 = String.valueOf(i*i);
            s4=s3.replace(".", ",");
```

```
            System.out.println(s2 + tab + s4);
        }
    }
}
```

58, page 36. Translators face to an immediate problem with declinations, derived and compound words. But the most terrible of all problems is that a word can have many diverse meanings and it is difficult to single out the correct one according to the context. The code for an elementary German to English translator follows:

```
//Program A58 GermanToEnglish
//Translator model.
package ejvol1;


public class GermanToEnglish
{
    public static void main(String[] args)
    {
        //Targets
        String s1 = "Er hast ein Auto.";
        String s2 = "Er ist gut Fahrer.";
        String s3 = "Ich bin schlecht Fahrer.";
        String s4 = "Ich müsse fahren.";
        //Output
        String t = s1 + " " + s2 + " " + s3 + " " + s4;
        System.out.println("German : " + t);
        t = t.replace("Auto", "car");
        t = t.replace("bin", "am");
        t = t.replace("Er", "He");
        t = t.replace("ein", "a");
        t = t.replace("Fahrer", "driver");
        t = t.replace("fahren", "drive");
        t = t.replace("gut", "good");
        t = t.replace("hast", "has");
        t = t.replace("Ich", "I");
        t = t.replace("ist", "is");
```

```
        t = t.replace("müsse", "must");
        t = t.replace("schlecht", "bad");
        t = t.replace("teuer", "expensive");
        System.out.println("English : " + t);
    }
}
```

63, page 39.

```
//Program A63 CopyPaste2
// This program copies a substring
// from sOrigin and pastes it into sTarget.

package ejvol1;


public class CopyPaste2
{
    public static void main(String[] args)
    {
        String sOrigin, sTarget, sMiddle, sInitial, sFinal;
        sOrigin = "How does function appear in evolution?";
        sTarget = "appears as a byproduct "
                + "of reproduction under selection";
        System.out.println("Origin = " + sOrigin);
        System.out.println("Target = " + sTarget);
        // Let us copy into sMiddle
        // 4 characters of sOrigin:
        // initial point included: 9,
        // terminal point excluded: 18.
        sMiddle = sOrigin.substring(9, 18);
        System.out.println("sMiddle = " + sMiddle);
        // We copy into sFinal the last part of sTarget
        // beginning  with the zeroth char
        sFinal = sTarget.substring(0);
        System.out.println("sFinal = " + sFinal);
        // We concatenate sMiddle and sFinal.
        String  sOutput = sMiddle + sFinal;
        System.out.println("output = " + sOutput);
```

```
    }
}
```

67, page 41. To delete the last 3 characters from a string s, we report a substring that contains the first $n-3$ characters of s.

68, page 41.

```
//Program A68 Length2
//This program deletes from s1
//3 characters beginning with the (l-4)th one,
//where l is the length of s.

package ejvol1;



public class Length2
{
    public static void main(String[] args)
    {
        String s1, s2, s3, s4;
        s1 = "Evolution = mutation + selection";
        System.out.println("Original sentence: " + s1);
        int l = s1.length();

        //We copy into s2 the first part of s1
        s2 = s1.substring(0,l-4);
        System.out.println("Left tail: " + s2);
        //We copy into s3 the last part of s1
        s3 = s1.substring(l-1);

        //We concatenate s2 and s3.
        s4 = s2+s3;
        System.out.println("Deleted sentence: " + s4);
    }
}
```

72, page 43.

```
//Program A72 Invsersion2
//This program inverts substring a
//in within string c.
package ejvol1;


public class Inversion2
{
public static void main(String[] args)
  {
    String a = "important";
    String c = "Inversion is important for sub-speciation";
    System.out.println("Original sequence =" +  c);
    System.out.println("Subsequence to be inverted =" +  a);
    int Start;
    String charI;
    //We get the length of a
    int la = a.length();
    //We initialize the output
    String  d = "";
    //To invert a, we copy each character of a,
    //one by one, and paste them together
    //in reversed order:

    for(int i = 0; i<la ; i++ )
    {
        Start = i;
        charI = a.substring(Start,Start + 1);
        d = charI + d;
    }
    System.out.println(" Inverted sub-sequence =" +  d);

    // The first part of c:
    String cLeft = c.substring(0,13);
    // The last part of c:
    String cRight = c.substring(22);
    // Concatenation
    String Inversion = cLeft + d +cRight;
    System.out.println(" Partially inverted sequence ="
```

```
                                                    +  Inversion);
  }
}
```

74, page 44. A solution with a new method glue(a,b,c) follows:

```
//Program A74 Concatenation4
//Concatenation of 3 strings
//Structured programming.

package ejvol1;


 public class Concatenation4
{
    // This method concatenates a,  b and c
    public static String  glue(String a, String b, String c)

    {
        // The reserved word "return"
        // points to the output of the method glue.
        return a+b+c;
    }

    public static void main(String[] args)
    // This is the head of the program.
    {
        String a = "pqrst";
        String b = "123";
        String c = "ATCG";
        // the output of glue is placed in string c
        String sOutput = glue(a,b,c);
        //Report
        System.out.println("a =   " + a);
        System.out.println("b =   " + b);
        System.out.println("c =   " + c);
        System.out.println("a + b +c =   " + sOutput);
    }
}
```

77, page 46. Structuring with three methods.

```
//Program A77 Chain
//Playing with methods.

package ejvol1;


public class Chain
{
    static String e;

    // This method concatenates a and b.
    //Its output is a String,
    // So it is declared as a String
    public static String  glue(String a, String b)
    {
        return a+b;
    }

    // This method replaces string e by f
    // inside g. Its output is a String
    // so the method is declared as String.
    public static String
          replacement(String e, String f, String g)
    {
        return g.replace(e,f);
    }

    //This methods reports results.
    //This method makes an action,
    //so its output is declared void.
    public static void  Report(String a,String b,
                        String c,String d, String sOutput)
    {
        System.out.println("a =   " + a);
        System.out.println("b =   " + b);
        System.out.println("a+b =   " + e);
        System.out.println("c =   " + c);
```

```
        System.out.println("d =    " + d);
        System.out.println("In a+b, c is replaced by d =  " + sOutput);
    }

    // This is the head of the program.
    public static void main(String[] args)
    {
        String a = "123";
        String b = "def";
        String c= "23";
        String d = "bc";
        // We concatenate a to b
        e = glue(a,b);
        // We replace c by d inside e
        String sOutput = replacement(c,d,e);
        //Report
        Report(a,b,c,d,sOutput);
    }
}//End of program  A77 Chain
```

Problems of Chapter 6
82, page 48.

```
// Program 82 Arrays2
// Array to calculate and print cubes.
package ejvol1;


public class Arrays2
{
   public static void main( String args[] )
   {
    // Declaration of an array with integer entries
    int data[];  // data is the name.

    // Assignation of the capacity of the array:
    // ten entries, beginning from zero.
    // data[i] can be defined for i from zero to 9
```

```
    // data[10] is not defined
      data = new int[ 10 ];

    // Assignation of each entry
    for(int i = 0; i<=9; i=i+1)
    {
        data[i] = i*i*i;
    }

    // The array is written to the console.
    for(int i = 0; i < 10; i++)
        System.out.println(  i + " " + data[i] );
   }
}
```

84, page 50. To make sure that all data are included, one can force the for loop to calculate $i <= n$ instead of $i < n$. The result is an error because of a boundary violation.

```
//program A84 Arrays4
//Mean and variance of decimal data.
package ejvol1;


public class Arrays4
{
   public  static void main( String args[] )
   {
        //Declaration and assignation of the data array
        double  Data[] = {1.1,5.2,2.3,4.4,7.5,8.6,9.7,5.8};

        // Report of data.
        System.out.println( "Data are" );
        // To know the length of the array : Data.length
        for ( int i = 0; i < Data.length; i++ )
        {
            System.out.print( Data[ i ]);
            System.out.println(); //pass to the next datum
        }
        //   Let us calculate the mean
```

```
        double sum = 0;
        int n = Data.length;
        for ( int i = 0; i < n; i++ )
            sum = sum + Data[i];
        double mean = sum / n;
          System.out.println( "Sum \t "+ sum);
          System.out.println( "Mean \t "+ mean);
        //    Let us calculate the variance
        double sum2 = 0;
        for ( int i = 0; i < n; i++ )
         sum2 = sum2 + (Data[i]-mean)*(Data[i]-mean);
        double var = sum2 / n;
        System.out.println( "Variance \t "+ var);
        System.out.println( "Deviation \t "+ Math.sqrt(var));
   }
}
```

85, page 50. Ile-Leu-Leu-Tyr-Arg-Tyr-Arg-Tyr-Arg-Tyr-Ile-Tyr-Arg-Tyr-Ile-Term

Problems of Chapter 7

98, page 64.

```
//Program A98 RandomGenerator2
//Random chars

package ejvol1;

import java.util.Random;
public class RandomGenerator2
{
    public static void main( String args[] )
    {
        //We turn on the random generator:
        Random r  = new Random();

        //Upper case chars
        for(int i= 1; i<=20; i++)
        {
```

```
            char ch =   ( char ) ( r.nextInt( 26 ) + 65 );
            System.out.print (  ch + "  " );
        }
        //Change of car
        System.out.println();

        //Lower case chars
        for(int i= 1; i<=20; i++)
        {
            char ch =   ( char ) ( r.nextInt( 26 ) + 97 );
            System.out.print (  ch + "  " );
        }
        System.out.println();
    }

}//End of Program A98 RandomGenerator2
```

101, page 65.

```
//Program A101 RandomRNA
//Random  RNA string.
package ejvol1;


import java.util.Random;

public class RandomRNA
{
   public static void main( String args[] )
   {
        //We turn on the random generator:
        Random r  = new Random();
        // Definition of the alphabet
        char Alphabet[] = {'U','T','C','G'};

        // Generation of a RNA string 25 bases long
        for(int i= 0; i< 25; i++)
        {
```

```
            char ch =   Alphabet[ r.nextInt( 4 )];
            System.out.print (  ch);
        }
        System.out.println();
    }
}
```

Problems of Chapter 8

110, page 75. To get a run of 10 units of a distinguished face one needs around 10000 trials, although in some occasions 10 trials could be enough. So, one cannot deny that runs 10 units long may happen by mere randomness.

112, page 77. The required number of trials to get a run k units long is an exponential function in the input k. We say that the cost of making a run is exponential in the length of the run. This is readily seen if one makes a graphic of the log of the cost against the input k. In fact, let k be the input, the required length of the run. And let y be the necessary number of trials to get a run that long. So, a graphic of y against k renders a non linear curve, whose nature is difficult to disentangle. So, we study the dependence of $log(y)$ against k. We obtain a linear equation of the form $log(y) = ck$ with $c \sim 4/15$, where the log is taken in basis 10. This means that

$log(y) = (4/15)k$
$10^{logy} = 10^{(4/15)k}$
$y = 10^{(4/15)k}$

So, suppose that we need a run 15000 long to make what could be equivalent to build the first living being that could exist. The number of necessary trials to get such a run would be:

$y = 10^{(4/15)15000} = 10^{4000}$

an expression that is valid if we suppose that the probability of success in the run is 0.5. Now, the estimated number of hydrogen atoms in the universe is around 10^{80}, while the age of the universe is considered to be less than 10^{10} years or 10^{18} seconds or 10^{33} phentoseconds. This means then that 10^{4000} is a very large number at cosmic scale.

From these considerations, the most important for us conclusion is that the origin of complexity and of life are problems that are worth a careful study. Nevertheless, our results cause a predisposition to reject the proposal that randomness could be a natural explanation for complexity of large holistic units (units that admit no functional subparts). Even large richnesses cannot be explained in that form: if

you want to get rich, you must acquire wisdom for detecting advantageous trades that recurrently produce high incomes and that can disappear in a second.

114, page 82. Results clearly shows that there is a marked tendency of the population to visit regions of high energy, which increases with time. Thus, one has a reason to pursue a quantum mechanical study of the mechanism, whose conclusion would not contradict our classical results: recurrent collisions of charged particles with photons can lead to particles of very high kinetic energy. Eventually, one might make an estimation of the age of the universe given the present distribution of energy of cosmic rays and compare it with the age estimated by other means, some $15x10^9$ years.

117, page 88. . The model is a failure. Seemingly, recombination is not useful to enlarge strings, instead, it possibly could be very useful to interchange substrings and thus to pack high levels of information in short strings.

Problems of Chapter 9

122, page 92. The proportion of randomly generated words 4 chars long that have a meaning in English was estimated by the Author as 6 in 2000. An orthographic corrector rendered 10 in 2000. We use an estimation of that proportion by a simulation instead of an exact inquiry because that would demand hours of work instead of some few minutes.

127, page 94. Six mutations in tandem may be enough to cause a lost of sense of the word 'meaning':

meaeing- meceing- meceina- mecmina- mecyina- mecgina- mecgipa.

128, page 94. The first result is that meanings are grouped in small islands. To see this, let us consider the phrase:

HAVE THE GATES OF DEATH BEEN REVEALED TO YOU?

According to Wicken (1987, Evolution, Thermodynamics and Information. Oxford U. Press. N.Y. Oxford.) this is, in the version of Job, the challenge given to modern biology and the one that has not been answered as yet. If you mutate, one place at a time, each one of the words that compound that phrase, then the resulting strings that have a meaning, and that we imagine as indwelling one and the same island, are:

HAVE, cave, gave, lave, nave, pave, rave, save, wave, hive, hove, hake, hale, hare, hate, haze.

THE, she, tee, tie, toe.

GATES, bates, dates, fates, hates, mates, pates, rates, sates, gages, gales, games, gapes, gases, gazes, gated.

OF, if, oh, ok, on, op, or, ox, oz.

DEATH, heath, depth.

BEEN, keen, peen, seen, teen, bean, beef, beep, beer, bees, beet.

REVEALED, repealed.

TO, do, go, no, so, tn.

YOU, yon.

Thus, the string "you" is in the same island with "yon". In these cases the island has some few members. In other cases, the island has just one member: try the word "mistake", to which I have not found one-place meaningful mutations. It is easy to walk around an island and mutation can do this readily, but to jump to another island you will need happy coincidences that are by far less probably.

If the same mutational process is exercised on our phrase, the only mutating phrases that have a meaning are:

Have the gates of heath been revealed to you? Have the dates of death been revealed to you? Have the fates of death been revealed to you? Have the mates of death been revealed to you? Have the games of death been revealed to you?

These strings delimit the island where our phrase lives. These examples suffice to show us that mutation can produce nice and unexpected results at the level of words, phrases and sentences. Please, notice now that most words have more allowed mutations than the whole studied phrase. This implies that the **sea of nonsense** that surround any meaningful island is steeper the higher the level of recursion. This is illustrated in the next branch of a mutational process:

0: SOME CHANGES ARE MISTAKES OTHERS ARE SUCCESSES
1:snme changes are mistakes others are successes
2:snme changes are mistakes otherslare successes
3:snme cjanges are mistakes otherslare successes
4:snme cjanges are mistakes otyerslare successes
5:snme cjanges are mistakes otyerslare succesdes
6:snme cjanges are mistakes otyerslarm succesdes
7:xnme cjanges are mistakes otyerslarm succesdes
8:xnme cjanges are mistakes otyprslarm succesdes
9:xnme ijanges are mistakes otyprslarm succesdes
10:xnmeuijanges are mistakes otyprslarm succesdes
11:xnmeuijanges are mistakes otyprylarm succesdes
12:xkmeuijanges are mistakes otyprylarm succesdes

13:xkmeuijanges are mistakes otyprylarm succesdej
14:xkmeuijanges afe mistakes otyprylarm succesdej
15:xkmeuijanges afe mistakes otyprylarm juccesdej
16:xkmeuijanges afe mistakes otyprylarmwjuccesdej
17:xkmeuijanges afe mistakes otyprylarmwjuccysdej
18:xkmeuijanges afe mistakes otybrylarmwjuccysdej
19:xkmeuijanges afe mistakes otybrylarqwjuccysdej
20:xkmeuijanges afe mistakes otybrylarqwjuccwsdej
21:jkmeuijanges afe mistakes otybrylarqwjuccwsdej
22:jkmeuijanges ade mistakes otybrylarqwjuccwsdej
23:jkmeuijangeo ade mistakes otybrylarqwjuccwsdej
24:jkmeuijangeo ade mistakos otybrylarqwjuccwsdej
25:jkmeuijangeo ade mistakos otywrylarqwjuccwsdej
26:jomeuijangeo ade mistakos otywrylarqwjuccwsdej
27:jomeuijangeo ade mistakos otywrylarqwjuccwbdej
28:jomeuijangeo ade mistakos otywrylarqkjuccwbdej
29:jomeuijangeo ade mistakos jtywrylarqkjuccwbdej
30:jomeuijangeo ade mistakos jtywrylarqkjuccwbjej

So, meanings are grouped in small islands, which are surrounded by an infinite sea of nonsense, which is the deeper the higher the level of recursion. The reader is invited to verify this law in any human language. One can consider all languages at the same time as a single one and the conclusion is the same.

Nonsense seas are ubiquitous. Let us consider two cases, one in relation with genetics and the other with human education.

A nonsense sea is found in genetics: most mutations are deleterious if not lethal. Is that surprising? Yes, and indeed too much. Let us see why: we can say that a genetic string is a message transmitted by means of biochemical machinery to nature: we say that nature understands the message when it allows the transmitting carrier cell to live and reproduce further. But, why does the versatility of the universe seems so limited as to being unable of understand most mutations? That is very surprising.

The official formulation of our claim of small islands surrounded by a nonsense sea is known in population genetics as the **neutral theory** of molecular evolution formulated by Motoo Kimura: a DNA function is filled by some few strings whose performance is roughly equivalent. Strings are few, because of the sea of nonsense surrounding small islands of meaning. Nonetheless, our explanations do not suffice to account for that nonsense sea. One must go to the mechanism of enzymatic catalysis to understand why.

When Kimura published his theory (around 1968), the meaningful strings were imagined to be some few in numbers, in a literal sense. Now there is at least one studied case in which the equivalent strings are not few: it is t-RNA . The reason of a huge of allowed neutral mutation is that the object of selection is the phenotype and not the genotype, and we have that to many genotypes corresponds one single phenotype. For t-RNA, the **genotype** is the sequence of nucleotides, but the **phenotype** is represented very well by its secondary structure. Notice that this is another case in which one can have a nonsense sea coexisting with rapid evolution:

"Call two sequences connected if they differ by one or at most two point mutations. A **neutral network**, then, is a set of sequences with identical structure so that each sequence is connected to at least one other sequence. The crucial point for our discussion comes from a recent study of the standard secondary structure prediction algorithm, which showed that such networks percolate though sequence space. For example, starting at a sequence that folds into a tRNA structure, it is possible to traverse sequence space along a connected path, thus changing every nucleotide position without ever changing the structure. Moreover, due to the high-dimensionality of sequence space, networks of frequent structures penetrate each other so that each frequent structure is almost realized within a small distance of any random sequence. These features seem to be intrinsic to RNA folding, since they are insensitive to whether the folding algorithm is thermodynamic, kinetic, or maximum matching (...) or whether one considers one minimum free energy structure or the entire Boltzmann ensemble." (Huynen et al, 1996).

From our discussion we learn that we have a theme that merits a deep study.

130, page 95. There is a very strong tradition of enmity in Papua, so fierce that there are many people that never have gone further than 200 meters away from his or her home (after the aforementioned book by Nettle). At the same time, all children make mistakes against the language norm but because of family ties, mistakes are celebrated and so a variation arises. This variation is faded away by the interaction with other people alien to the family, but when those other people do not exist, day after day a new language can arise. Thus, one can have both things: a nonsense sea and rapid evolution: the semantics of the language evolves and therefore the nonsense sea also evolves and rapid evolution can coexist with a deep sea of nonsense.

133, page 96. One can say that words have a function: to have a meaning. By the same token, one can say that a DNA string has a meaning when it encodes for something functional for the cell or the carrier organism. Since a sea of nonsense surrounds words with a meaning, we can expect that a sea of nonsense also sur-

rounds meaningful DNA strings. The reason is that a language must be adapted to the reality as well as DNA strings to life and in both realms we have constraints of memory resources and computing capabilities. So, both of them are answers to the problem of understanding and controlling reality. Hence, we must expect similar properties in both realms.

Problems of Chapter 10

140, page 113. Recombination is included in the following program:

```
//Program A140 Letterzymes2
//Evolution of a population
//of strings.
//Recombination is added to
//selection and mutation.

//Strings are at the same time
//genotype and phenotype.
//They are like ribozymes.


package ejvol1;



import java.util.Random;

public class Letterzymes3
{

//Pre-declaration of global variables.
//They are used all throughout the whole class.

//Individuals are kept in the array
//Individual[]. It is an array of strings.
//The number of individuals must be
//less than limit

private static final int limit = 1000;
```

```
//How many generation to run
private static final int  nGen = 200;
//Define here when a string is long enough
//to be purged
private static final int largeSize = 1;
private static final String Individual[ ] = new String[100];
private static final int NIndiv = 15;
//Rank  keeps the rank of each individual.
private static final int Rank[] = new int[limit];
static private String b;

static private int ReportMin[], ReportMax[];


//Turn on of the random generator
private static final Random r  = new Random();

//This method generates random chars
private  static  Character  randomChar()
{
    return  ( char ) ( r.nextInt( 26 ) + 65  );
}

/* We generate NIndiv individuals (strings)
 ten characters long.
 Sequences are completely random */
private static void Initialization( )
{
    System.out.println("ORIGINAL POPULATION");
    for(int i = 0; i< NIndiv; i++)
    {
        //Initialization  in blank.
        Individual[i]="";

        //An individual is assembled char by char
        for(int j = 0; j< 10; j++)
        {
            //We generate a random char
            Character c =  randomChar();
            //System.out.println( "char = " + c);
```

```
            //Char c is converted into a string s
            String s = ""+c;
            //s is concatenated to Individual[i]
            Individual[i] = Individual[i]+s;
        }
        System.out.println("Individual " +  i);
        System.out.println( Individual[i]);
    }



    for(int i = 0; i< NIndiv; i++)
        Rank[i]=0;

    ReportMin= new int[nGen];
    ReportMax= new int[nGen];
    for(int gen = 0; gen < nGen; gen++)
    {
        ReportMax[gen]=0;
        ReportMin[gen]=0;
    }



}


//This method takes substring a
//and posits it at place Start inside c
private  static  String
      Insert(String a, int Start, String c)
{

    String  s1, s2;
    // We copy  the left part of c
    s1 = c.substring(0, Start);
    // We copy  the right part of c
    s2 = c.substring(Start);
    // We concatenate the left part of c
    //to a to the right part of c
```

```
    String z = s1 + a + s2;
    /*System.out.println("Original c " + c);
    System.out.println("Original a " + a);
    System.out.println("Start " + Start);
    System.out.println("PosInsertion c = " + z);
    */
    return z;
}

//The letters of "ALGAE" are inserted
//into the string of individual i
private static  void feeding(int i)
{
    String a;
    // We measure the length of Individual[i]
    int l = Individual[i].length();
    //Null strings cannot feed anyhow
    if (l == 0) ;
    else //Non null strings  feed on ALGAE
    {
        a = "A";
        int Start = r.nextInt(l);
        Individual[i] = Insert(a, Start, Individual[i]);
        a = "L";
        l = Individual[i].length();
        Start = r.nextInt(l);
        Individual[i] = Insert(a, Start, Individual[i]);
        a = "G";
        l = Individual[i].length();
        Start = r.nextInt(l);
        Individual[i] = Insert(a, Start, Individual[i]);
        a = "A";
        l = Individual[i].length();
        Start = r.nextInt(l);
        Individual[i] = Insert(a, Start, Individual[i]);
        a = "E";
        l = Individual[i].length();
        Start = r.nextInt(l);
        Individual[i] = Insert(a, Start, Individual[i]);
    }
```

```
}


//This Function replaces all occurrences of
//substring a by b inside c
private static String
Substitution(String a, String b, String c)
{
    String z = c.replace(a,b);
    /*System.out.println("original a " + a);
    System.out.println("Original b " + b);
    System.out.println("Original c " + c);
    System.out.println("replaced c " + z);
    */
    return z;
}


//This function or method deletes all occurrences
//of a from c.
static private String Delete(String a, String c)
{
    b = "";
    return Substitution(a, b, c);
}


//Here we see what a PARA-SITE  does to individuals.
private  static void parasitation(int i)
{
    int l  =  Individual[i].length();
    //The parasite deletes from the
    //beginning of the individual
    //any substring   matching "PARA".
    //We copy to Beginning the first
    //4 chars of Individual[i]
    if ( l > 4)
    {
        String Beginning =  Individual[i].substring(0,4);
        String s = "PARA";
```

```
        // We compare Beginning and s
        if (Beginning.equals(s))
            Individual[i] = Individual[i].substring(4);

        //The parasite deletes from the tail
        //of the individual
        //any substring   matching "SITE".
        String Ending =  Individual[i].substring(4);

        String t = "SITE";
        // We compare Ending and t
        if (Ending.equals(t))  Individual[i] =
                    Individual[i].substring( 0,l - 4);
    }
 }


//In this method individuals are mutilated
//as a measure to hinder overgrowth.
//We delete from the original string
//a substring of a random length beginning with
//a random Start position.
private static void Purge(int i)
{
    String c, s1, s2;
    int lc, Start, n;
    c = Individual[i];
    lc = c.length();
    if (lc > largeSize)
    {
        Start = r.nextInt(lc);
        //We copy  the first part of c
        s1 = c.substring(0, Start);
        // We throw a random number
        n = r.nextInt(lc - Start+1);
        //We copy  the right part of c
        // after Start + n places
        s2 = c.substring( Start+ n);
        //We concatenate the left part of c
        //to  the right part of c
```

```
        Individual[i] = s1 + s2;
    }
}


//Individuals are sorted by fitness.
//Fitness = length, as an indicator of complexity.
private static void Sorting()
{
    int   Champ;
    //Necessary default initialization
    int Fitness[] = new int[100];
    for(int i = 0; i< NIndiv; i++)
        Fitness[i] =0;
    //We define a fitness function equal to the length
    //of the string
    for(int i = 0; i< NIndiv; i++)
    {
        String s = Individual[i];
        Fitness[i] = s.length();
        //System.out.println(Individual[i] + Fitness[i]);
    }
    //We sort individuals by fitness
    for(int i = 0; i< NIndiv;i++)
    {
        Champ = 0;
        for(int j = 0; j< NIndiv;j++)
            if (Fitness[j] > Fitness[Champ])   Champ = j;

        //The array Rank keeps a record of fitness
        //by decreasing order.
        Rank[i] = Champ;
        //The champ leaves the game
        Fitness[Champ] = 0;

    }
    /*
    //Report in decreasing order
    System.out.println("\nSorting :");
    for(int i = 0; i< NIndiv;i++)
```

```
        System.out.println(Individual[Rank[i]]);
    */
}


//The top ten fill the world
    private  static void Copying()
    {
        int   k;
        for(int i = 0; i< 10; i++)
        {
            int  l = 10*(10-i);
            for(int j = 0; j< 10; j++)
            {
                k = l -j-1;
                Individual[k] = Individual[Rank[i]];
                /*System.out.println("Ind " + k
                        + " replaced by top" + i);*/

            }
        }

        //System.out.println(  "THE NEW POPULATION IS");

        // for(int i = 0; i <  NIndiv;i++)
        //System.out.println(  Individual[i]);
    }

private static void Reproduction()
{
    //Individuals are sorted by length
    Sorting();
    //for(int i = 0; i< 10;i++)
    //System.out.println(  Individual(Rank(i))
    //The top ten produce a copy that
    //substitutes the bottom ten.
    Copying();
}

// From individual i
```

```
// the fist n  chars are deleted
private static   void Deletion(int i)
{
    int l, n;
    l = Individual[i].length();
    n = 0;
    if (l > 0)  n = r.nextInt(l);
    Individual[i] = Individual[i].substring(n);
 }

//This method inverts a
static private   String Inversion(String a)
{
    int  la = a.length();
    //Output
    String  d  = "";

    //We copy the first character of a to the last place
    //and so on.
    for(int  i = 0; i < la; i++)
     {
        char charI = a.charAt(i);
        d = charI + d;
     }
    return d;
}

//Local inversion of a substring inside individual i.
private static void innerInversion(int i)
{
    String sLeft, sRight;
    int start;
    int n;
    String c = Individual[i];
    int lc = c.length();
    if (lc > 1)
    {
        start = r.nextInt(lc);
        //We copy  the first part of c
        sLeft = c.substring(0, start);
```

```
        // We throw a random number
        n = r.nextInt(lc - start+1);
        //Inner substring
        String a = c.substring(start, start + n);
        String d = Inversion(a);
        //We copy  the right part of c
        // after Start + n places
        sRight = c.substring( start+ n);
        //We concatenate the left part of c
        //to  the right part of c
        Individual[i] = sLeft + d + sRight;
    }
    /*System.out.println("Input  =   " + c + "  " + n);
    System.out.println("Output  =  " + Individual[i]);
    */
}

//We consider only two types of mutation:
//deletion of the beginning
//and inversion of an inner part.
private static void Mutation()
{
    //Which mutation affects which individual
    //is a matter of gambling.
    //We throw a 0 else a 1
    for(int i = 0; i< NIndiv; i++)
    {
        int Luck = r.nextInt(2);
        if ( Luck == 1) Deletion(i);
        else  innerInversion(i);
    }
}

private static String recombinant(int i)
{

    int n = Individual.length;
    //A partner for i is found
    int j = r.nextInt(n);
```

```
    //A place for recombination
    int lengthI =  Individual[i].length();
    int lengthJ =  Individual[j].length();
    int minLength = Math.min(lengthI,lengthJ);
    int place = r.nextInt(minLength);
    String recombinant = Individual[i].substring(0, place)
            + Individual[j].substring(place);
    /*
    System.out.println( "Inputs = " + Individual[i]
            + "    " + Individual[j]
    + " Recombinant = " + recombinant);
    */
    return recombinant;
}


//Two strings are taken as input
//and one recombinant is produced
private static void Recombination()
{


    //Array with recombinant individuals
    String Individual2[ ] = new String[100];
    for(int i = 0; i< NIndiv; i++)
    {
        Individual2[i] = recombinant(i);
    }
    //Recombinants become the official population
    for(int i = 0; i< NIndiv; i++)
    {
        Individual[i] = Individual2[i];
    }

}



//This is the heart of the process
private static void Dynamics()
```

```
{
    //All individuals feed
    for(int i = 0; i< NIndiv; i++)
    {
        //The string Algae is digested and inserted
        //into the individual[i] char by char
        /*System.out.println( "The hungry individual "
                    + i + " is " + Individual[i]); */
        feeding(i);
        /*System.out.println(  "The full individual "
             + i + " is " + Individual(i)); */
        //All individuals are tested by the parasite,
        //which corrodes any  large substring
        //similar to it located at any end.
        parasitation(i);
        /*System.out.println(  "Pos-parasitation individual "
             + i + " is " + Individual(i));*/
        //Very long chains are attacked recurrently,
        //the same as very large fishes in the sea.
        Purge(i);
        /*System.out.println(  "The purged  individual "
             + i + " is " + Individual(i)); */
    }
    //The top ten are allowed to  reproduce:
    //a child occupies the place of the shortest individual
    Reproduction();
    //The new population is subjected to mutation
    Mutation();
    Recombination();
}


private static void report(int gen )
{
    System.out.println("\nPopulation at gen = " + gen );
    for(int j = 0; j< NIndiv; j++)
    {
        System.out.println("Gen " + gen + " Ind "
                    + j +" " +  Individual[j]);
    }
```

```
}

//Min and Max lengths are detected
private static void Observation(int gen)
{
    int m, Min, Max;
    Max = 0;
    Min = 32000000;
    for(int j = 0; j< NIndiv; j++)
    {
        m =  Individual[j].length();
        /*System.out.println(  "The Individual  "
           + Individual(j) + " has length " + m);*/
        if (m > Max)  Max = m;
        if (m < Min)   Min = m;
    }

    /*System.out.println(  "Generation "  + Gen
    + " MinLength = " + Min + " MaxLength = " + Max);*/
    ReportMin[gen] = Min;
    ReportMax[gen] = Max;
}

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

    for(int gen = 0; gen < nGen; gen++)
    {
        //System.out.println( "Beginning3");
        Dynamics();
        report(gen);
        //The aim of observation is to record
        //the minimum and maximum lengths
        //of the individuals in each generation.
        Observation(gen);
    }
    //Report of the dynamics of min and max indicators.
    System.out.println( "GENERATION MIN AND MAX VALUES");
    for(int Gen = 0; Gen < nGen; Gen++)
```

```
        System.out.println( Gen + "\t" + ReportMin[Gen]
        + "\t" + ReportMax[Gen]);

}

}//End of Program A140 Letterzymes3
```

Problems of Chapter 11

147, page 118. Public methods can be invoked from outside. Private not.

151, page 121. The next program tests the tools contained in ToolBox.

```
//Program A151 ToolBox2
//This is a test for the methods in ToolBox

package ejvol1;


public class ToolBox2
{
    public  static void main(String[] args)
    {
        //A method of class TopDown is invoked
        System.out.println("Invokation of methods from ToolBox:");
        String a = "aaaa";
        String b = "aaaaaaaaabbbbb";
        String c = "cdfghhjk";
        System.out.println("a = " + a);
        System.out.println("b = " + b);
        System.out.println("c = " + c);
        String d = ToolBox.delete(a, b);
        System.out.println("Delete output = " + d);

        String s = ToolBox.insert(a, 5, c);
        System.out.println("Insert output = " + s);
```

```
        String i = ToolBox.inversion(c);
        System.out.println("Inversion output= " + i);

        String j = ToolBox.invert("fgh", c);
        System.out.println("Invert output = " + j);

        String k = ToolBox.substitution(a, c, b);
        System.out.println("Substitution output = " + k);
    }
}
```

160, page 128. The next code replaces the corresponding code in program 158 where data is fabricated and saved:

```
    // Data is fabricated and saved
    //Modify this at pleasure.
    for(int i = 0; i<10;i++)
    {
int l = 2*i;
double w = l;
// Write  two data to file,
// one integer and the other decimal (double)
output.format( "%d   %.2f\n", l, w );
    }
```

Problems of Chapter 12

177, page 143. For the chosen initial condition, the sequence tries to fill in all available space in a unorganized fashion. That is why this function is known as a **chaos generator**. This type of deterministic behavior has been used to model the dynamics of some populations of birds whose censal numbers change almost unpredictably but nevertheless the population is stable and does not disappear.

183, page 146.

$V_X = (1/12)((3-4.8833)^2 + (5-4.8833)^2 + (4-4.8833)^2 + (6-4.8833)^2$
$+(4-4.8833)^2 + (7-4.8833)^2 + (3-4.8833)^2 + (5-4.8833)^2 + (4-4.8833)^2$
$+(6-4.8833)^2 + (3-4.8833)^2 + (8-4.8833)^2) =$

$$\frac{3(3-4.8833)^2+3(5-4.8833)^2+3(4-4.8833)^2+2(6-4.8833)^2+(7-4.8833)^2+(8-4.8833)^2}{12}$$

$V = 2.47$

188, page 148. The probability density function is constant over its range. Say, if a roulette is a source of numbers from 0 to 360, then, the density function is constant and equal to 1/360.

191, page 150. All ten runs rendered a chi-square that was less than the critical value 16.91 with 9 d.f and a significance of 0.05. So, we have as yet no reason to suspect that our generator is biased, i.e., that it does not generate a uniform distribution.

201, page 156. There is an **asymptotic tendency** of fitting better the expected distribution when the number of trials increases. Nevertheless, the tendency is fuzzy: the distribution of the error (measured by a z-test) is more concentrated near zero as the number of trials increases but eventually the error can be large.

205, page 159. The Author did not verified the **asymptotic tendency** for a binomial distribution. So, it will not be verified for a normal distribution, since it is the continuous fitting to a binomial one.

209, page 161. For a sequence of 10000 random numbers the distribution looks almost like a perfect Gaussian bell with deviation 5 and mean 25 but not for a sequence of size 100. The asymptotic tendency is verified.

210, page 161. A tiny change in the code suffices to redefine mean and deviation:

```
// We choose a Gaussian distribution
double d = r.nextGaussian() ;
// System.out.println(d ) ;
// Rescale numbers: mean m and deviation s
int c = (int) Math.floor(s*d) + m;
```

214, page 164. The accuracy is low for 10, 10 or 1000 trials. But for 10000 trials it is fairly good.

Problems of Chapter 13

219, page 180. If y is the number of necessary trials to find a sequence x numbers long without a six, then $y = Ae^{mx}$ where $m \sim 0.06$.

221, page 185. A source that is like a die with five faces admits for not very long sequences the possibility to be modeled by a die with 6 faces. But to do the same with a die of seven faces is impossible, even for moderate lengths. The reason is that a correct matching may need infinite time (say, more than half an hour).

225, page 187.

```
//Program A225 Die5
//We study the average of
//discrete uniforms distributions,
//each one  with range from 1 to 9
//and mean 5.

package ejvol1;



import java.util.Random;

public class Die5
{
    // One can vary the number of trials,
    // by changing the variable length.

    static private int length;


    //The number of studied families is
    //nFamilies

    //Observed frequencies
    private static final double Obs[ ] = new double[9001];

    //Turn on of the random generator
    private static final Random r  = new Random();
```

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

        //nUniforms = number of random variables.
        // nUniforms can range from 1 to 1000
        int nUniforms = 10;
        int m = 9*nUniforms;

        for(int i = 1; i<= m; i++)
                Obs[i] = 0; //Initialization

        int factor = 5;
        length = m*factor;
        //Loop with many trials
        for(int j = 1; j<= length ; j++)
        {
            // We generate nUniform random numbers
            //from 1 to 9
            double sum = 0;
            for(int k = 0; k< nUniforms; k++)
            {
                int c = r.nextInt(9) +1;
                // and we add them:
                sum = sum + c;
            }
            double average = sum/nUniforms;
            int i = (int) average;

            //bookkeeping
            Obs[i] = Obs[i]+1;
        }
        System.out.println("Sums and their frequencies");
        for(int j = 1; j<= m; j++)
            System.out.println(j + "\t" + (int) Obs[j]);
    }
}//End of Program A225 Die5
```

227, page 188.

```
//Program A227 Die6
```

```
//We study the average of
//discrete dependent uniforms distributions,
//each one  with range from 1 to 9
//and mean 5.

package ejvol1;



import java.util.Random;

public class Die6
{
    // One can vary the number of trials,
    // by changing the variable length.

    static private int length;


    //The number of studied families is
    //nFamilies

    //Observed frequencies
    private static final double Obs[ ] = new double[9001];

    //Turn on of the random generator
    private static final Random r  = new Random();

    public static void main(String[] args)
    {

        //nUniforms = number of random variables.
        // nUniforms can range from 1 to 1000
        int nUniforms = 10;
        int m = 9*nUniforms;

        for(int i = 1; i<= m; i++)
                Obs[i] = 0; //Initialization

        int factor = 5;
```

```
        length = m*factor;
        //Loop with many trials
        for(int j = 1; j<= length ; j++)
        {

            double sum = 0;
            // We generate one random numbers
            //from 1 to 9
            int c = r.nextInt(9) +1;
            // and  make with it a sum with nUniforms terms:
            for(int k = 0; k< nUniforms; k++)
            {
                sum = sum + c;
            }
            double average = sum/nUniforms;
            int i = (int) average;

            //bookkeeping
            Obs[i] = Obs[i]+1;
        }
        System.out.println("Sums and their frequencies");
        for(int j = 1; j<= m; j++)
            System.out.println(j + "\t" + (int) Obs[j]);
    }
}//End of Program A227 Die6
```

235, page 200. Recombination has been added:

```
//Program A235 Equation2

//Evolution of a population
//of strings that finds a root
//of a polynomial.

//Mutation + recombination + selection.

package ejvol1;
```

```
import java.util.Random;

public class Equation2
{
    // Strings are at the same time
    // genotype and phenotype.

    // Pre-declaration of global variables.
    // They are used all throughout the whole class.

    // Individuals are kept in the array
    // Individual[]. It is an array of strings.
    // The number of individuals must be
    // less than limit

    private static final int POPSIZE = 100;
    static private double Fitness[];
    static private String Individual[ ];//Predeclaration
    // Order  is an array of integers.
    static private int Order[];//pre-declaration
    static private int Generation;
    static private int  nGen;
    static private int ReportMin[], ReportMax[];
    private  static int numberOfClones ;
    private static final double MUTATIONRATE = 0.2;
    private  static int Champ;

    // Turn on of the random generator
    private static final  Random RAND = new Random();
    private static final boolean  PRINT = false;

    // This method generates random chars
    // that represent digits
    private  static  Character  randomChar()
    {
        return  ( char ) ( RAND.nextInt( 10 ) + 48  );
    }
```

```
    private static void Initialization( )
    {
        /* We generate POPSIZE individuals (strings)
        Sequences are completely random */

        //Formal declaration of our array.
        Individual= new String[POPSIZE];
        //System.out.PRINTln("ORIGINAL POPULATION");
        for(int i = 0; i< POPSIZE; i++)
        {
            //Initialization  in blank.
            Individual[i]="";

            //An individual is assembled char by char
            for(int j = 0; j<  9; j++)
            {
                // We generate a random numeric char
                Character c =  randomChar();
                //System.out.PRINTln( "char = " + c);
                // Char c is converted into a string s
                String s = c.toString();
                // s is concatenated to Individual[i]
                Individual[i] = Individual[i]+s;
            }
            // We add the place for a point.
            Character n =  randomChar();
            String s = n.toString();
            Individual[i] = s+Individual[i] ;

            //We add a sign: zero or one.
            n = (char) ( RAND.nextInt( 2 ) + 48  );
            s = n.toString();
            Individual[i] = s+Individual[i] ;
            //System.out.PRINT("Individual " +  i + "\t");
            // System.out.PRINTln( Individual[i]);
        }


        //Declarations and default initializations
// of other arrays.
```

```
    Order= new int[POPSIZE];
    ReportMin= new int[POPSIZE];
    ReportMax= new int[POPSIZE];
    Fitness= new double[POPSIZE ];

    for(int i = 0; i< POPSIZE; i++)
    {
        Order[i]=0;
        ReportMin[i]=0;
        ReportMax[i]=0;
    }
}

//The encoded number is withdrawn
private static double decoder(String s)
{
    // We get the sign:
    char s1  =   s.charAt(0);
    int sign = Character.getNumericValue(s1);
    // We get the place of the point,
    // which implies multiples of ten.
    s1  =   s.charAt(1);
    int  point = Character.getNumericValue(s1);
    // We build the number with the other chars
    double number = 0;
    for(int j=2;j  <  11;j++ )
    {
        char s3  =   s.charAt(j);
        int  l = Character.getNumericValue(s3);
        int k = j-1;
        int power = point-k;
        //number1 = l * 10^power
        double number1 =  l * Math.pow(10,power);
        number = number + number1;
    }
    if (sign == 0) return (-1) *number;
        else return  number;
}
```

```
//The method encodes the equation = 0 to solve.
//Perfect solution must ne a root and output zero.
//Other  numbers represents an error
private static double functionvalue(double b)
{
    //Root = 0.75;
    return 4*b+3;
    //Root = root = -4.71817809;
    //return b*b*b*b*b + 4*b*b*b*b - 3*b*b*b + 2*b*b +b +1;
    //Root = 1;
    //return b*b - 2*b + 1;
}


//The fitness of each individual is calculated
private static void fitness()
{
   for(int i = 0; i< POPSIZE; i++)
    {
        // Individual[i] is a string,
        // it is a genotype.
        // b is a number: it is the phenotype
        double b = decoder(Individual[i]);
        double  fV = functionvalue(b);
        //Fitness(root) = 1
        //Others have Fitness less than 1
        Fitness[i] =    1/(1+fV * fV);
    }
}


//Strings are sorted by fitness
private static void Sorting()
{

    fitness();

    for(int i = 0; i< POPSIZE;i++)
    {
        int champ = 0;
```

```
            for(int j = 0; j< POPSIZE; j++)
                if (Fitness[j] >= Fitness[champ])   champ = j;
            //The array Order classifies individuals by fitness.
            //The fittest, with less errors, go to the 0-th entry.
            Order[i] = champ;
            Fitness[champ] = 0;
        }


        String s = Individual[Order[0]];
        double b = decoder(s);
        double p = functionvalue(b);
        System.out.println("Gen = " + Generation + " best solution  = "
        + b + " Error = " + p );
    }




    //The top   fill the world
    private  static void Cloning()
    {
        //Array to host cloned individuals
    String Individual2[ ] = new String[100];
    int   i=0;
    int   k=0;         //rank, 0 is the highest
    //Cloning
    while (i<POPSIZE)
    {
        int counter = numberOfClones;
        while ((counter>0) & (i<POPSIZE))
        {
            Individual2[i] = Individual[Order[k]];
            i++;
            counter= counter-1;
        }
        k++;
    }
```

```
    //Cloned individuals become the official population
    for(int j = 0; j < POPSIZE; j++)
        Individual[j] = Individual2[j];
    if (PRINT) printPopulation();
    }

    private static void Reproduction()
    {
        //Individuals are sorted by fitness
        Sorting();
        //for(int i = 0; i<  10;i++)
        //System.out.PRINTln(  Individual(Order(i))

        //Fitest individuals are cloned
        Cloning();
    }


private static  String change(String a)
{
    for(int l = 0; l < a.length(); l++)
    {
        double s = RAND.nextDouble();
        if (s < MUTATIONRATE)
        {
            if (a.length()==1) return ""+randomChar();
            else
            {
                //In case of a ghost
                if ( (l<0) || (l > a.length()-1))
                {

                    return a;
                }
                a = a.substring(0, l) + randomChar()
                    +  a.substring(l + 1);
            }
        }
    }
```

```
    return a;
}



    private static void Mutation()
    {
        for(int i=0; i < POPSIZE; i++)
       if (i != Champ) Individual[i] = change(Individual[i] );

    }




private static String recombinant(int i)
{
    //A partner for i is found
    int j = RAND.nextInt(POPSIZE);
    //A place for recombination
    int lengthI =  Individual[i].length();
    int lengthJ =  Individual[j].length();
    int minLength = Math.min(lengthI,lengthJ);
    int place = RAND.nextInt(minLength);
    String recombinant = Individual[i].substring(0, place)
            + Individual[j].substring(place);
    /*
    System.out.PRINTln( "Inputs = " + Individual[i]
            + "    " + Individual[j]
    + " Recombinant = " + recombinant);
    */
    return recombinant;
}


//Two strings are taken as input
//and one recombinant is produced
private static void Recombination()
{
```

```
    //Array with recombinant individuals
    String Individual2[ ] = new String[100];
    for(int i = 0; i< POPSIZE; i++)
    {
        Individual2[i] = recombinant(i);
    }
    //Recombinants become the official population
    for(int i = 0; i< POPSIZE; i++)
    {
        Individual[i] = Individual2[i];
    }

}


    private static void Dynamics()
    {
        //The top ten are allowed to  reproduce:
        //a child occupies the place of the shortest individual
        Reproduction();
        //The new population is subjected to mutation
        Mutation();
        //Recombination();
    }


    private static void printPopulation()
    {
        //Cloned individuals become the official population
    for(int j = 0; j < POPSIZE; j++)
    {

        double b = decoder(Individual[j]);
        double  fV = functionvalue(b);
        double    myFitness =    1/(1+fV*fV);
        System.out.println( j + "  ->  "  + Individual[j] + "  ->  " +
myFitness);
```

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


    public static void main(String[] args)
    {
        //To modify POPSIZE,
        // the method Copying()
        // must also be modified.
        Initialization( ) ;
        nGen = 400000;
        numberOfClones  = 10;
        //System.out.println("Root of 4*x + 3 = 0,  x = 0.75");
        for(int Gen = 0; Gen < nGen; Gen++)
        {
            Generation = Gen;
            Dynamics();
        }
    }
}//End of Program A235 Equation2
```

236, page 200. In the method preFitness(), one must update the return value to

```
a) 4*b +3;
b) b*b+4*b+4;
c)  b*b*b*b*b + 4*b*b*b*b - 3*b*b*b + 2*b*b +b +1; root = -4.71817809;
```

d) A polynomial is a curve in the plane. A root is a point where the curve intersects the horizontal axis. A root is seen in the fitness function as a maximum. The curve may have a local minimum that is close to zero but different than zero. This local minimum looks in the fitness function as a local maximum that is close to the global and true one. The algorithm may get trapped at a local maximum, which is converted into a false solution. Why? Because our mechanisms to create variability do not cope with the complexity of the problem. Variability maybe created by allowing mutation rate to be increased, or population size to augment or letting time to stretch. One can choose very long times, say, 400000 generations and one would arrive at a true solution. If these remedies do not function, it could

be because of a deficiency of the encoding: our chromosomes can encode numbers with 10 digits, but 20 could be needed to solve a given problem.

238, page 205. The necessary trials to produce a sample that fails to fit a uniform distribution is almost independent on the length of the generated random sequence although as the length increases the dispersion of the histogram grows a bit. The number of faces plays no major role.

Problems of Chapter 14

247, page 211.

```
//Program A247 Markov2
//Markov Chains
//There is a rapid convergence to the equilibrium state.

package ejvol1;



import java.util.Random;

public class Markov2
{

    static int X[] = new int[3000];
    //Probabilities
    static double  pa,paa,pb,pbb;

    //Turn on of the random generator
    static Random r  = new Random();

//Define probabilities of transition
public static void initialization()
{
    //Independent probabilties
    paa = 0.8;
    pbb = 0.7;
```

```
    // initial conditions
    pa = 0.25;
    pb = 0.75;
}

 public static void main(String[] args)
 {
    initialization();
    // Generation of 3000 individuals
    // some of type a other of type b
    int na = 0;
    int  nb = 0;
    for(int i = 0; i< 3000; i++)
    {
        double s = r.nextDouble();
        if (s < pa)  na = na + 1;
         else nb = nb + 1;
    }


    //Report the number of individuals in each state
    System.out.println("Initial condition");
    System.out.println("The number of individuals in " +
                       "state a is  " + na);
    System.out.println("The number of individuals in " +
                       "state b is  " + nb);

    System.out.println("\tGeneration,    populations in "
                                  +"states A and    B  " );

    //Execute 1000 transitions
    for(int i = 0; i< 1000;i++)
    {
        //Decide whether or not an individual
        //in state a must make a transition
        for(int j= 1; j<=na;j++)
        {
            double s = r.nextInt();
            if (s < paa)
            {
```

```
                na = na - 1;
                nb = nb + 1;
            }
        }

        //Decide whether or not an individual
        // in state b must make a transition
        for(int j= 1; j<=nb;j++)
        {
            double s = r.nextDouble();
            if (s < pbb)
            {
                nb = nb - 1;
                na = na + 1;
            }
        }

        int  l = i / 100;
        if( i == l * 100)
            System.out.println( i + "\t"    + na + "\t"   + nb);

    }
    System.out.println("Final condition");
    System.out.println("The number of individuals in " +
                        "state a is  " + na);
    System.out.println("The number of individuals in " +
                       "state b is  " + nb);

}
}//End of Program A247 Markov2
```

248, page 212.

```
//Program A248 Markov3
//Markov Chains
//There is a rapid convergence to the equilibrium state.

package ejvol1;
```

```
import java.util.Random;

/**
 *
 * @author jose
 */
public class Markov3
{

    static int X[] = new int[3000];
    //Probabilities
    static double  pa,paa,pb,pbb;

    //Turn on of the random generator
    static Random r  = new Random();

//Define probabilities of transition
public static void initialization()
{
    //Independent probabilties
    paa = 0.8;
    pbb = 0.7;
    // initial conditions
    pa = 0.25;
    pb = 0.75;
}

 public static void main(String[] args)
 {
    initialization();
    // Generation of 3000 individuals
    // some of type a other of type b
    int na = 0;
    int  nb = 0;
    for(int i = 0; i< 3000; i++)
    {
        double s = r.nextDouble();
        if (s < pa)  na = na + 1;
         else nb = nb + 1;
```

```
}


//Report the number of individuals in each state
System.out.println("Initial condition");
System.out.println("The number of individuals in " +
                   "state a is  " + na);
System.out.println("The number of individuals in " +
                   "state b is  " + nb);

System.out.println("Generation,    populations in "
                              +"states A and    B  " );

//Execute 1000 transitions
for(int i = 0; i< 100;i++)
{
    //Decide whether or not an individual
    //in state a must make a transition
    for(int j= 1; j<=na;j++)
    {
        double s = r.nextInt();
        if (s < paa)
        {
            na = na - 1;
            nb = nb + 1;
        }
    }

    //Decide whether or not an individual
    // in state b must make a transition
    for(int j= 1; j<=nb;j++)
    {
        double s = r.nextDouble();
        if (s < pbb)
        {
            nb = nb - 1;
            na = na + 1;
        }
    }
    System.out.println( i + "\t"    + na + "\t"   + nb);
```

```
    }
    System.out.println("Final condition");
    System.out.println("The number of individuals in " +
                        "state a is  " + na);
    System.out.println("The number of individuals in " +
                       "state b is  " + nb);

}
}//Program A248 Markov3
```

252, page 213. If the Identity Matrix is the transition matrix (when nodes are unconnected), the system remains forever in the initial state. So, the corresponding Markov chain has an equilibrium state for every initial condition and so it has infinitely many equilibria and the limit is not unique.

253, page 213. If the equilibrium state is unique then it is attained from every other state. Therefore, if a perturbation causes the system to leave its equilibrium without impairing its transition dynamics, the system will evolve from the new state to that of equilibrium.

254, page 213. Let use a z test to compare the experimental and the expected frequencies. We have a binomial distribution with n=3000, p=0.6, 1-p=0.4, so
$\sigma_p = \sqrt{p(1-p)/n} = \sqrt{0.6(0.4)/3000} = 0.009$:
Hence $z = (0.6 - 0.688)/0.009 = -9.7$ which is very improbable by mere randomness. How can we explain the discrepancy? We must remember that everything is possible for randomness in its range of action. At the same time, it would be good to make many runs to obtain a distribution of the final states. In that way, we could judge fairly.

Problems of Chapter 15

262, page 222. Everything is right except that very large populations must be considered, because in our units, a size close to zero is a possible value.

```
// Program 262 LotkaVolterra2
// Study  of a prey-predator system.
// Deterministic model.
// Lotka-Volerra equations.
// Periodicity observed.
```

```
// Compatible with all versions of Excel and Cal.

package ejvol1;



public class LotkaVolterra2
{
    static double nr, nw, e, m,p,l,  k;

    public static void initialization()
    {
        //Generates 500 rabbits,  50 wolfs
        nr = 500;
        nw = 50;
        // Rate of rabbit's reproduction
        m = 0.1;
        // Rate of predation
        p = 0.01;
        // Rate of wolf's death
        k = 0.05;
        // Rate of wolf reproduction
        l= 0.1 * p ;
    }

    public static void dynamics()
    {
        String s1,s2,s3,s4;
        //We run N generations
        int n = 1000;
        for(int  i = 0; i< n;i++)
        {
            //Rabbits reproduce
            nr = nr + m* nr;
            //Rabbits are preyed on
            nr = nr - p * nr * nw;
            //Wolfs reproduce
            nw = nw + l* nr * nw;
            //Wolfs die
            nw = nw - k * nw;
```

```
            //Decimal with point.
            //System.out.println(i + "\t"
            //        +  nr + "\t " +  nw);


            //Decimal numbers with comma.
            //Conversion of number nr, nw
            //into a format appropriate for Excel:

            s1 = String.valueOf(nr);
            s2 = s1.replace(".", ",");
            s3 = String.valueOf(nw);
            s4 = s3.replace(".", ",");

            System.out.println(i + "\t"
                    +  s2 + "\t " +  s4);
        }
    }


    public static void main(String[] args)
    {
        initialization();
        System.out.println("time, rabits and wolfs");
        dynamics();
    }
}//End opf Program 262 LotkaVolterra2
```

263, page 222. . It is rather difficult to find an adjustment of parameters that are compatible with life continuation. For instance, the next setting in program 262 produces an stable system but for larger values of the number of wolfs, *nw*, the system is driven into extinction.

```
   //Generates 500 rabbits,  50 wolfs
 nr = 500;
 nw = 100;
 //Rate of rabbit's reproduction
```

```
 m = 0.1;
 //Rate of predation
 p = 0.01;
 //Rate of wolf's death
 wd = 0.05;
 // Rate of wolf reproduction
 l= 0.1 * p ;
```

Problems of Chapter 16

272, page 233. The order of reproduction of white moths along the program can be interchanged with that of reproduction of brown moths to prove that the winner is the first type to reproduce. Or, if one runs with a given initialization and next reinitializes with reverse rates of predation, the first to reproduce is the most favored.

275, page 237.

```
//Program A275 Moths3
/*
    Study of predation of moths by birds.
    Moths could be white or brown.
    Reproduction obeys a logistic model
    Two environments: clean and dirty.
    Discrete stochastic logistic model.
    Mutation is included.
*/

package ejvol1;



import java.util.Random;


public class Moths3
{
    static private int nwm, nbrm, nb ;
```

```
static private double  mr;
static private double  maxM, maxB, pwm, pbrm,bd;
static private int n;
static private double mut;


//Turn on of the random generator
static private Random r  = new Random();

public static void initialization()
{
    //Number of generations
    n = 1000;
    //Generates   moths,
    //white moths
    nwm = 450;
    // brown moths
    nbrm = 450;
    //Generates birds
    nb = 10;
    //Probability of moth's reproduction
    mr = 0.1;
    //Mutation rate of moths
    mut = 0.01;
    //Maximum number of Moths
    maxM = 2000;
    //Maximum number of Birds
    maxB = 20;
    //probability of predation on white moths
    pwm = 0.001;
    //probability of predation on brown moths
    pbrm = 0.002;
    //Probability of bird's death
    bd = 0.05;
}

public static void reinitialization()
{
    //probability of predation on white moths
    pwm = 0.002;
```

```
    //probability of predation on brown moths
    pbrm = 0.001;
    //Probability of bird's death
}

public static void dynamics()
{
    for(int i = 0; i < n; i++)
    {
        // White moths reproduce and mutate
        for(int j = 0; j < nwm; j++)
        {
            double s = r.nextDouble();
            if (s < mr * (1 - (nwm + nbrm) / maxM) )
            {
                //Reproduction
                nwm = nwm + 1;
                //Mutation
                if (s < mut)
                {
                    nbrm = nbrm + 1 ;
                    nwm = nwm-1;
                }
            }
        }

        //Brown moths reproduce and mutate
        for(int j = 0; j < nbrm; j++)
        {
            double s = r.nextDouble();
            if (s < mr * (1 - (nwm + nbrm) / maxM) )
            {
                nbrm = nbrm + 1;
                if (s < mut)
                {
                    nbrm = nbrm - 1 ;
                    nwm = nwm+1;
                }
            }
        }// end j for
```

```
            //Moths are preyed, birds reproduce
            for(int l = 0; l < nb; l++)
            {
                //White moths
                for(int j = 0; j < nwm; j++)
                {
                    double s = r.nextDouble();
                    // a white moth is preyed
                    if (s < pwm){ nwm = nwm - 1;}
                    //birds reproduce
                    if ( s < 0.1 * pwm * (1 - nb / maxB))
                        nb = nb + 1;
                }
                //brown moths
                for(int j = 0; j < nbrm; j++)
                {
                    double s = r.nextDouble();
                    // a brown moth is preyed
                    if (s < pbrm){ nbrm = nbrm - 1;}
                    //birds reproduce
                    if ( s < 0.1 * pbrm * (1 - nb / maxB))
                        nb = nb + 1;
                }
            }



            //Birds die
            for(int j = 0; j < nb; j++)
            {
                double s = r.nextDouble();
                if (s < bd)
                    nb = nb - 1;
            }
            System.out.println(i + "\t " +  nwm + "\t "
                + nbrm + "\t " + nb);
        }//end i for
    }
```

```
    public static void main(String[] args)
    {
        initialization();
        System.out.println("time, white moths, brown moths and birds");
        dynamics();
        System.out.println("Contamination.");
        System.out.println("time, white moths, brown moths and birds");
        reinitialization();
        dynamics();
    }
}//End of Program A275 Moths3
```

Problems of Chapter 17

284, page 242. (0,0) is at the uppermost left corner. The first coordinate runs in the horizontal direction, the second in the vertical direction. Positive directions are rightwards and downwards.

286, page 245. The actions of the system are executed only in response to the events produced by the user. The services offered by the program always are alive until explicit termination by the user.

288, page 247. .

```
//Program A288
//Event based programming
// We learn to capture data from the user.

import java.awt.*;
import java.applet.Applet;

public class A288 extends Applet
{
private static final long serialVersionUID = 1L;

Label invitation1; //message to the user
TextField input1;  //input from the user
```

```
Label invitation2;
TextField input2;
Label invitation3;
TextField input3;
double num1, num2, num3; //working variables


public  void   init()
  {
 //Input machinary
 //Declaration
 // A Label displays an invitation to operate
   invitation1 = new Label("Input first parameter ");
   //A TextField is a place for writing.
   input1 = new TextField( 10);
   invitation2 = new Label("Input second parameter  ");
   input2 = new TextField( 10);
   invitation3 = new Label("Input second parameter + enter ");
   input3 = new TextField( 10);
   // activation
   add(invitation1);
   add(input1);
   add(invitation2);
   add(input2);
   add(invitation3);
   add(input3);
  }

public void report()
  {
 //Here we report results.
 //We use the console
   //because we expect large volumnes of data
   System.out.println("num1" + "\t " + num1 );
   System.out.println("num2" + "\t " + num2 );
   System.out.println("num1" + "\t " + num3 );
  }

public boolean action(Event event, Object o)
  {
```

```
 //This is the interface:
 // If the user presses enter into the second field
 if (event.target == input3)
 {
//read textField1 and textField2
    num1 = Double.parseDouble(input1.getText() );
num2 = Double.parseDouble(input2.getText() );
num3 = Double.parseDouble(input3.getText() );
//make the report of what was read.
report();
}
//Indicate to the system that the action
// of the user has been processed.
return true;
  }
}
```

Problems of Chapter 19

303, page 263. A simulation that produced 1000 data yielded
Mean = 0.18276133155390503 Deviation = 1.0150733498928597
A z-test for the mean renders:
$z = \frac{0.18276133155390503-0}{\sqrt{1/1000}} = 0.031$ that is normal for all interesting significances.
An chi2 -test for the variance renders
$\chi = \frac{(999*1.015^2}{1^2} = 1029.$
The theoretical distribution of this stadigraph follows an approximately normal distribution with mean = 999 and standard deviation $\sqrt{2(999)} = 44.7$, so we are allowed to believe that the mean is zero and that the variance is 1.

323, page 294. The necessary procedures and adaptations were added to the previous program to enable a Kolmogorov-Smirnov test. In first place, we need to know the cumulative frequencies, which are next normalized to a maximum value of one. Next, to compare two normalized vectors of cumulative frequencies, we calculate for each subindex of the two vectors, the absolute value of the subtraction of the two corresponding entries at each vector. Next: we pick the maximum absolute difference. This number is compared to a critical value given by a table. The new program detected a significant difference between columns and rows. The code follows:

```
// Program 323 StatTest6
// An incomplete block design.
// Duplex method

// Fills a 3x3 with random numbers
// with a normal distribution
// with mean 2.222
// and standard deviation 1.483

//A Kolmogorov Smirnov test is included

package ejvol1;

import java.util.Random;


public class StatTest6
{
    //Turn on  the random generator
    static Random r  = new Random();

    static int freqtableColsRefilledMatrix[] = new int[10001];
    static int freqtableRowsRefilledMatrix[] = new int[10001];
    static int freqtableColRandMatrix[] = new int[10001];
    static int freqtableRowsRandMatrix[] = new int[10001];
    static double cumColsRefilledMatrix[] = new double[10001];
    static double cumRowsRefilledMatrix[] = new double[10001];
    static double cumColRandMatrix[] = new double[10001];
    static double cumRowsRandMatrix[] = new double[10001];

    static double help[] = new double[25];
    static int nLim;
    static  double meanProj;
    static  double deviationProj;
    static double M[][] = new double[4][4];
    static double means[] = new double[4];
    static double Fcolumns;
    static double Frows;
```

```
    private static void freqtableToZero()
    {
        for(int i = 0; i<=1000; i++)
        {
            freqtableColsRefilledMatrix[i]=0;
            freqtableRowsRefilledMatrix[i]=0;
            freqtableColRandMatrix[i]=0;
            freqtableRowsRandMatrix[i]=0;
        }
    }

    private static void reportM( )
    {
        for(int i = 1; i<=3; i++)
        {
            for(int j = 1; j<=3; j++)
                System.out.print(  M[i][j] + " \t");

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

//Fills a 3x3 with random numbers
//with a normal distribution
//with mean 2.222
//and standard deviation 1.483
private static void randMatrix()
{
    double s;
    for(int i = 1; i<=3; i++)
    {
        for(int j = 1; j<=3; j++)
        {
            s = deviationProj * r.nextGaussian()
                + meanProj;
            M[i][j]= s;
            //System.out.print(  M[i][j] + " \t");
        }
        //System.out.println();
```

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


private static void FF()
{
    // reportM(mExp);
    /*GSi Great  Square under treatment i =
     (\sum^{n_i}_{j=1}X_{ij})^2 */
    double GS1 = Math.pow(M[1][1] + M[2][1]+M[3][1],2);
    double GS2 = Math.pow(M[1][2] + M[2][2]+M[3][2],2);
    double GS3 = Math.pow(M[1][3] + M[2][3]+M[3][3],2);
    //System.out.println("GS1 "+ GS1);
    //T= \sum^k_{i=1} \sum^{n_i}_{j=1} X_{ij}

    /*GS= Great  Square =
    (\sum^k_{i=1} \sum^{n_i}_{j=1} X_{ij})^2*/

    double GS = Math.pow(M[1][1]+M[2][1]+M[3][1]+
    M[1][2] +M[2][2] +M[3][2]+M[1][3]+M[2][3]
    +M[3][ 3],2);


    /*Among Columns SS =
     \sum^k_{i=1}\frac{GS_i}{ l_i}-\frac{GS}{N}.*/
    double ACSS = (GS1 / 3 + GS2 / 3 + GS3 / 3) - GS /9;

    /*Among columns DF =
    Among columns degrees of freedom= k-1= 3-1=2.*/


    /*Among columns Mean square=
     \frac{Among Columns SS}{Among columns DF}*/

    double ACMS = ACSS / 2;


    //The same but for rows or blocks
```

```
double BGS1 = Math.pow(M[1][1] + M[1][2]+M[1][3],2);
double BGS2 = Math.pow(M[2][1] + M[2][2]+M[2][3],2);
double BGS3 = Math.pow(M[3][1] + M[3][2]+M[3][3],2);

// System.out.println( "  BGS1 "+ BGS1);
/*Among Rows SS =
$\sum^k_{i=1}\frac{GS_i}{ k_i}-\frac{GS}{N}$ */

double ARSS = (BGS1/3 + BGS2/3 + BGS3/3) - GS/9;
// System.out.println( "ARSS " + ARSS);
/*Among rows DF=
Among rows degrees of freedom= l-1= 3-1=2.*/

/*Among rows Mean square=
  \frac{Among Rows SS}{Among rows DF}*/

double ARMS = ARSS / 2;
// System.out.println( "ARMS " +ARMS);

//Calculates the sum of the squares of   entries.


double UIK = Math.pow(M[1][ 1] ,2)
+ Math.pow(M[2][ 1] ,2)
+ Math.pow(M[3][ 1] ,2)+ Math.pow(M[1][ 2],2)
+ Math.pow(M[2][ 2],2)
+ Math.pow(M[3][ 2],2) + Math.pow(M[1][ 3] ,2)
+Math.pow( M[2][ 3] ,2) + Math.pow(M[3][ 3] ,2);


//Total SS=Total Sum of Squares=
//$\sum^k_{i=1}\sum^{n_i}_{j=1}[X_{ij}^2]-\frac{GS}{N}$

double TotalSS = UIK - GS / 9;
// System.out.println( "TotalSS " + TotalSS); //OK
//Error SS=error sum of squares=
//Total SS - Among columns SS -Among row SS

double ERRORSS = TotalSS - ACSS - ARSS;
```

```
    //System.out.println( "ERRORSS " + ERRORSS);

    //Error DF= Total DF - Among groups DF - Among rows DF
    //=    8-2-2=4 .

    //Error Mean square= \frac{Error SS}{error DF}

    double EMS = ERRORSS / 4;
    //System.out.println( "EMS " + EMS);
      Fcolumns = ACMS / EMS;
      Frows = ARMS / EMS;
    //  System.out.println( Fcolumns  ); //OK
    // System.out.println( Frows  ); //OK

}

private static void cum(int vect[], double cumvect[])
{

    int sum = 0;
    for(int i = 0; i<=nLim; i++)
    {
        sum = sum + vect[i];
        cumvect[i] = sum ;
    }

    for(int i = 0; i<=nLim; i++)
           cumvect[i] = cumvect[i] /sum;
}


private static void cumulatives()
{
    cum(freqtableColsRefilledMatrix,cumColsRefilledMatrix);

    cum(freqtableRowsRefilledMatrix,cumRowsRefilledMatrix );

    cum(freqtableColRandMatrix,cumColRandMatrix);
```

```
    cum(freqtableRowsRandMatrix,cumRowsRandMatrix);
}

private static void ksTestCols()
{
    System.out.println("Cumulative " +
                  " distributions for Columns");
    System.out.println(" i + RefilledMatrix+ " +
                  " RandomMatrix + diff" );

    double dMax = 0;
    double c;
    for(int i = 1; i<=nLim; i++)
    {
        c= Math.abs(cumColsRefilledMatrix[i]
             -cumColRandMatrix[i]);
        System.out.print (i+ "\t " + cumColsRefilledMatrix[i] +
                "\t " + cumColRandMatrix[i] + "\t " + c);

        System.out.println() ;
        if (c>dMax) dMax = c;
    }
    System.out.print ("Max difference = " + dMax);
    System.out.println();
    //  dCritic for alpha = 0.05
    double h = 1.36/Math.sqrt(1000);
    System.out.print ("dCritic = " + h);
    System.out.println() ;
    if (dMax<=h) System.out.print("Cols: " +
                  "The two distributions are the same. ");
     else   System.out.print("Cols: " +
                  "The two distributions are different.");

}

private static void ksTestRows()
{
    System.out.println("Cumulative " +
        "distributions for Rows");
    System.out.println(" i + " +
```

```
        "RefilledMatrix+ RandomMatrix + dif" );

    double dMax = 0;
    double c;
    for(int i = 1; i<=nLim; i++)
    {

        c= Math.abs(cumRowsRefilledMatrix[i]
             -cumRowsRandMatrix[i]);
        System.out.print (i+ "\t " +
            cumRowsRefilledMatrix[i] +
            "\t " + cumRowsRandMatrix[i] +"\t " + c);

        System.out.println() ;
        if (c>dMax) dMax = c;
    }
   System.out.print ("Max difference = " + dMax);
   System.out.println();
   //dCritic for alpha = 0.05
   double h = 1.36/Math.sqrt(1000);
   System.out.print ("dCritic = " + h);
   System.out.println() ;
   if (dMax<=h) System.out.print("Rows:" +
        "The two distributions are the same. ");
    else   System.out.print("Rows:" +
        " The two distributions are different.");

}

// We copy the matrix into a vector
// The central entry is ignored.
private static void meanVariance()
{
    int  n =0;
    for(int i = 1; i<=3; i++)
    {
        for(int j = 1; j<=3; j++)
        {
            if ((!(i==2)) |(!(j==2)) )
            {
```

```
                    n = n +1;
                    help[n]= M[i][j];

            }
        }
    }
    //System.out.println( "count \t " + count);
/*for(int i = 1; i<=count; i++)
                System.out.println( help[i]); */
/* We calculate mean and deviation of the vector:
          Let us calculate the mean */
    double sum = 0;

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

    for ( int i = 1; i <= n; i++ )
      sum = sum + help[i];
    double mean = sum / n;
      System.out.println( "Sum \t "+ sum);

    System.out.println( "Mean \t "+ mean);
    // Let us calculate the variance
    double sum2 = 0;

    for ( int i = 1; i <= n; i++ )
        sum2 = sum2 + (help[i]-mean)*(help[i]-mean);
    double var = sum2 / n;
    System.out.println( "Variance \t "+ var);
    meanProj = mean;
    deviationProj = Math.pow(var, 0.5);
    System.out.println("Deviation \t "+ deviationProj);
    System.out.println();
}

//Only the nonzero block matters
// This is an incomplete design:
// The central 0 is a provisional value:
// it stands in the place of a hole and
// it will be replaced by a random number.
// To that aim, we calculate the mean
```

```
// and variance of other data
// to design the normal distribution
// from which the random number will
// be sampled.
private static void initialization( )
{
    final double mExp[][] = {{0,0,0,0},
                                        {0,5,8,6},
                                        {0,7,0,8},
                                        {0,4,7,3},
                                        };


    for(int i= 1; i<=3; i++)
            for(int j= 1; j<=3; j++)
                    M[i][j] = mExp[i][j];
    System.out.println( "Experimental matrix"  );
    System.out.println( "The central 0 is a hole"  );
    reportM();
    meanVariance();
    // The central 10 is replaced by
    // a random number.

}

private static void completing( )
{
    M[2][2]=  deviationProj *
       r.nextGaussian()+meanProj ;
}




public static void main( String args[])
{
    // Definition of the exp matrix
    // calculation of projected values
    initialization();
     nLim = 1000 ;
```

```
freqtableToZero();
// We build n matrixes, in which
// the void entry is filled
// with a random number
// and the statistics are calculated.
for(int k=1; k<=nLim;k++)
{
    completing();
    FF();
    int d = (int)  Math.ceil(  Fcolumns);
    freqtableColsRefilledMatrix[d]=
             freqtableColsRefilledMatrix[d]+1;
    d = (int)  Math.ceil(  Frows);
    freqtableRowsRefilledMatrix[d]=
    freqtableRowsRefilledMatrix[d]+1;
}



//We generate n random matrixes
// filled in numbers that
// follow a normal distribution
// with mean meanProj and
// deviation deviationProj
for(int i=1; i<=nLim;i++)
{
    randMatrix();
    // reportM(M);
    //System.out.println( i + "\t " + "F = \t " + F());
    FF();
    int d = (int)  Math.ceil(  Fcolumns);
    freqtableColRandMatrix[d]=freqtableColRandMatrix[d]+1;
    d = (int)  Math.ceil(  Frows);
    freqtableRowsRandMatrix[d]=freqtableRowsRandMatrix[d]+1;
}
cumulatives();

ksTestCols();
ksTestRows();
}
```

```
}//End of Program A323 StatTest6
```

Problems of Chapter 20

330, page 301. The probability of fixation of an allele is equal to its initial frequency. In consequence, if in the population there is just one individual with allele1, the probability of fixation is $1/N$.

335, page 306. The next code reports the first 25 generations. They can be painted in Excel or in Calc in a three dimensional figure that clearly shows how an initial pick fades away and begins to slide towards a side. To draw in Calc: click on the pie chart, choose 3D look and Columns. To draw in a probably old version of Excel: choose insert graphic > lines > 3d lines > series in rows . In calc: copy data to the clipboard > paste special > unformatted text > Separarted by tabs > OK > Insert chart > Next > Data series in rows + 3d lines > OK to all prompts. You shall see that one wall grows and grows indicating fixation while other walls decay towards extinction. If one changes the parameters to study the first 1000 generations and to make a report once every 100 generations, then one can see that the initial pick squeezes itself against one of the the borders, i.e., monomorphism is the ultimate state of any population deprived of mutation. Or, mutation is the only source of variability. Fortunately, mutation is predicted by quantum mechanics and cannot be halted. But moreover, the cell has specially suited machinery to make mutations, say, as in the immune system.

```
// Program 335 Drift3

// In this program we study drift
// in 1000 asexual  populations with constant size,
// one locus or enzyme, two alleles and no mutation.

package ejvol1;



import java.util.Random;

public class Drift3
{
```

```
//Turn on  the random generator
static Random r  = new Random();


// number of populations
static int nPopulations = 1000;
// population size, the same for all populations.
static int size;
static double po;
// initial frequency of allele one,
// the same for all populations.

// actual frequency of allele one
static int nOne[];
// actual proportion of allele one
static double p[];
// number of generations
static int nGen;
// counts the number of allele1
static double counterAllele1;
// counts the number of times that allele1
// was fixed.
static double counterFixations;
static double probFixation;
// Record of the type of individuals
// initial population:
// allele1=true;
// allele2=false;
// The first entry is the pop subindex,
// the second entry is the individual subindex
static boolean population1[][],population2[][];
// The distribution of p in the population
static int distribp[];
// We do not report every generation
// but only when the generation is multiple of genSpace.
static int genSpace ;

public static void constructors()
{
    p = new double[1000];
```

```
        nOne = new int[1000];
        distribp = new int[1000];
        population1 =new boolean[1000][1000];
        population2 =new boolean[1000][1000];
    }


    public static void report( )
    {
        int distribution[] = new int[11];
        for(int i= 0; i <= 10; i++)
        {
            distribution[i]=0;
            for(int j=0; j<10; j++)
                distribution[i]=  distribution[i]+  distribp[10*i +j];
            System.out.print(distribution[i] + "\t" );
        }
        System.out.println();
    }

    public static void initialization()
    {
        size = 100;
        po = 0.3;
        nGen = 500 ;
        genSpace = 5;
        counterAllele1 = 0;
        counterFixations=0;

        //All populations are initialized
        int n;
        n = (int) Math.ceil( po*size);
        for(int pop=0; pop<nPopulations; pop++)
        {
            //initial population:
            //allele1=true;
            //allele2=false;


            for(int ind=1; ind<=n ; ind++)
```

```
                population1[pop][ind]=true;

        for(int ind=n+1; ind<=size ; ind++)
                population1[pop][ind]=false;
    }

    for(int j=0; j < size; j++)
            distribp[j]=0;
    distribp[n]=nPopulations;
    report( );
}


public static void iteration(int pop)
{
    //Sampling procedure
    for(int j=0; j<size; j++)
    {
        int  m = r.nextInt(size)+1 ;
        population2[pop][j]= population1[pop][m];

        if (population2[pop][j]==true)
            counterAllele1 = counterAllele1 +1;
    }
   p[pop]=counterAllele1/size;
   nOne[pop]= (int) counterAllele1;

   // System.out.println( nOne[pop] + p[pop]);

   /* String s4 = String.valueOf(p[pop]);
   String  s5=s4.replace(".", ",");
    System.out.println( s5);  */

   //updating
   counterAllele1=0;
   for(int ind=0; ind<size; ind++)
            population1[pop][ind]=  population2[pop][ind];
}

public static void reInitialization( )
```

```
    {
        for(int i=0; i<size; i++)
            distribp[i] = 0;
        for(int pop=0; pop<nPopulations; pop++)
        {
            nOne[pop] = 0;
            p[pop]=0;
        }
    }

    public static void thousandPopulations(int gen)
    {
        for(int pop=0; pop<nPopulations; pop++)
        {
            iteration(pop);
            /* System.out.println(pop +"\t"+
                  nOne[pop] + "\t"+ p[pop]); */

            distribp[nOne[pop]] = distribp[nOne[pop]]+1;
        }
        //System.out.println(gen);
        if (gen % genSpace  == 0 )  report( );
        reInitialization();
    }

    public static void main(String[] args)
    {
        constructors();
        initialization();
        for(int gen = 0; gen < nGen; gen++)
            thousandPopulations(gen);
    }
}//Program 335 Drift3
```

Problems of Chapter 21

343, page 310.

```
//Program A343 Drawing3
//This class draws a face.

package ejvol1;

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

public class Drawing3 extends Applet
{
    private static final long serialVersionUID = 1L;

    public void paint (Graphics g)
    {
        //The face.
        g.setColor(Color.pink);
        g.fillOval(15,15,100,150);
        //The hair.
        g.setColor(Color.black);
        g.fillRect(3,3,130,35);
        // The mouth
        g.setColor(Color.red);
        g.fillOval(50,120,30,20);
        // The eyes
        g.setColor(Color.blue);
        g.fillOval(30,50,30,20);
        g.fillOval(70,50,30,20);
        // The nose
        g.setColor(Color.red);
        g.drawRect(55,70,20,40);
     }
}//End of Program A343 Drawing3
```

348, page 316.

```
package ejvol1;
```

```
//Program A348 Drawing6
//This class  draws a moving face with blinking eyes.
//Vertical movement.
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;

public class Drawing6 extends Applet
{

private static final long serialVersionUID = 1L;

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

public void  eraseface(Graphics g,int k)
{
    // face
    g.setColor(Color.white);
    g.fillOval(15,15+k,100,150);
    //hair
    //g.setColor(Color.black);
```

```
    g.fillRect(3,3+k,130,35);
    //mouth
    //g.setColor(Color.red);
    g.fillOval(50,120+k,30,20);
    //eyes
    //g.setColor(Color.blue);
    g.fillOval(30,50+k,30,20);
    g.fillOval(70,50+k,30,20);
    //nose
    //g.setColor(Color.red);
    g.drawRect(55,70+k,20,40);
}

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

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

```
}

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

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

            long  l1 = System.nanoTime();

            int l = 500;
            for(int j=0; j<4000000; j++)
                delay(l1,l);

            closeEyes(g, k);

            //A delay is inserted
            l1 = System.nanoTime();
            l = l*40;
```

```
            for(int j=0; j<2000000; j++)
                delay(l1,l);

            openEyes(g,k);
        }
    }
    System.out.println("Job done.");
}
}//End of Program A348 Drawing6
```

Problems of Chapter 22

359, page 331.

```
//Program A359 Family
//Keeps information about the bodies of
//the members of a family: wife, husband and child.
package ejvol1;

public class Family
{

    Couple couple;
    Body child;

    Family(Couple c, Body b)
    {
        couple  = c;
        child = b;
    }

    public static void main(String[] args)
    {
        Body bodyMary = new Body(1,2,3,4);
        Body bodyAlf= new Body(-1,-2,-3,-4);
        Couple c = new Couple(bodyMary, bodyAlf);

        Body bodyJr = new Body ( 11,12,13,14);
```

```
        Family f = new Family(c, bodyJr);
        Body.report("Mary", f.couple.wife);
        Body.report("Alf", f.couple.husband);
        Body.report("Jr", f.child);
    }

}//End of Prgram A359 Family
```

Bibliography

[1] EvolJava (2015) Java for the study of Evolution.
http://www.evoljava.com/
Verified 27/VII/2015 5

[2] dextercowley (2015) Eclipse and Java Video Tutorials. Video tutorials for learning Java OOP programming with Eclipse.
http://sourceforge.net/projects/eclipsetutorial/
Verified 23/VII/2015 5

[3] Allen B. Downey (2012) Think Java. How to Think Like a Computer Scientist
http://greenteapress.com/thinkapjava/thinkapjava.pdf
Verified 22/VII/2015 vi

[4] Eclipse (2015) Downloads
https://www.eclipse.org/
Verified 23/VII/2015 4

[5] filmaffinity (2011) Culo de vaso
http://www.filmaffinity.com/es/evideos.php?movie_id=333845
Verified 23/VII/2015 ii

[6] IDR Solutions (2014) Tutorial : How To Fix Catastrophic Bugs with NetBeans Debugger
https://blog.idrsolutions.com/2014/07/use-netbeans-debugger/
Verified 29/VII/2015 51

[7] Limura M (1977) Preponderance of synonymous changes as evidence for the neutral theory of molecular evolution. Nature 267, 275 - 276 (19 May 1977); doi:10.1038/267275a0
http://www.nature.com/nature/journal/v267/n5608/abs/267275a0.html
Verified 13/VIII/2015 297

[8] Majerus E. N. (2007) The Peppered Moth: The Proof of Darwinian Evolution
http://www.gen.cam.ac.uk/Research/Majerus/Swedentalk220807.pdf
Verified 10/VIII/2015 229

[9] NetBeans Developing Applications with NetBeans IDE. (2014)
http://docs.oracle.com/cd/E50453_01/doc.80/e50452/whatsnew.htm
Verified 27/VII/2015 4

[10] NetBeans IDE Java Quick Start Tutorial (2015)
https://netbeans.org/kb/docs/java/quickstart.html
Verified 23/VII/2015 5

[11] Oracle (2015)
http://www.oracle.com/index.html
Verified 23/VII/2015 4

[12] Jean-Claude Rozec (2010) Cul de Bouteille (Specky four eyes)
http://www.vivement-lundi.com/vivement-lundi/Cul_de_Bouteille.html
https://www.youtube.com/watch?v=IkShVOFNMlA
Verified 23/VII/2015 ii

[13] Rothamsted Research (2001) A Beginners' Guide to Molecular Biology
http://www.rothamsted.ac.uk/notebook/gene.htm
Verified 22/VII/2015 2

[14] R Walpole, R Myers, S Myers, K Ye (2011) Probability & Statistics for Engineers & Scientists NINTH EDITION
http://folk.ntnu.no/jenswerg/40CEFd01.pdf
Verified 12/VIII/2015 261