

JAVA FOR THE STUDY OF EVOLUTION
VOLUME VIII
MUSIC AS A PARABLE
version 2

José del Carmen Rodríguez Santamaría
The EvolJava Community

July 2018

Contents

1	Getting started	1
1.1	Let us learn music	1
1.2	Plain text notations	2
1.3	ABC NOTATION	3
1.4	Two free applications	5
1.5	Music as a parable	6
1.6	Installing NETBEANS and Java	7
1.7	Purposes and projects	8
1.8	How to import a project	11
1.9	Conclusion	11
2	The Java Sound API	13
3	ABC4J	15
3.1	Starting up with ABC4J	15
3.2	Basic tasks at once	16
3.3	The ABC4J MusiText Class	24
3.4	A graphical User Interface	31
3.5	Conclusion	39
4	The first theory of music	41
4.1	Formulation and predictions	41
4.2	The sound of randomness	42
4.3	DNA as a composer	49
4.4	Recognizing pitches	56
4.5	The statistical version of the first theory	91
4.6	Musical analysis	92
4.7	Updating our GUI	92
4.8	Conclusion	145

5 Evolutionary Darwinian music	147
5.1 Putting evolution to help	147
5.2 Conclusion	175
6 Evolutionary Lamarckian music	177
6.1 Lamarckian evolution	177
6.2 Conclusion	212
7 Summary	213
Answers to exercises	217

Preface

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

1. To endow the community of **researchers in biology and evolution** with *high level programming*, enabling an accurate study of models and simulations of the most diverse nature.
2. To clearly show how this tool is used to study the fundamental questions of evolution.
3. To suggest that the study of Java could be *very fruitful* for **undergraduates** in biological sciences even more than calculus alone.

This is the eighth volume: a musiText is a text in plain text format that encodes for music and that can be played by computers. This results into the following parable: in the same way as DNA information encodes for an organism, a musiText encodes for a musical piece. To unfold the power of this parable we need an appropriate lab. This is what this volume is all about. The text was made more readable for version 2.

Bogotá, Colombia,

José Rodríguez
July 2018

Introduction

Life is exceedingly complicated, more than any human being can tolerate. A usual mean to tame complexity is to abstract some features of that complex reality and to enclose then into models or parables. In that regard, we propose here to study music, a purpose that we find perfectly realizable even for very busy people because, as we see in chapter I, computers help a lot. Since they process strings of symbols, it is necessary to encode musical pieces as **musiTexts**, which are texts in plain format that represent music and that can be played by computers. Many forms have been proposed to implement this idea and we invite the Reader to consider in first place the ABC NOTATION.

Our parable reads now: as a DNA string encodes for an organism, a musi-Text encodes for a musical piece. Therefore, music is a model of life in which the genetic aspect of information appears in first plane and so we will have the opportunity to deal in that world with evolution to understand and to apply it.

The Java community has taken music very seriously and have produced various related APIs, packages that deal with a concrete problem. One very friendly is ABC4J, that is directed to support work with ABC NOTATION. So, chapter II is dedicated to show how one can use it. In chapter III we present our ABCEditor, a GUI that uses ABC4J and that shows how one can tackle the problems that arise when dealing with musiTexts.

Chapter IV proposes a methodology for the study of music. We define in first place that a theory about music is good if it allows one to compose good music. In second place, we long for simple theories that are as universal as possible. Thus, we choose to begin with the simplest theory of music to next add complications but only in the measure that they are necessary. Our first theory reads: all musical notes are pleasant and therefore music is just a string of notes. We find this theory to be very primitive but nevertheless it is kind enough to allow for an important pedagogic application together with an implementation of Darwinian evolution as a helper for composing good music. This is discussed in chapter V, where evolution

is also armed with a hint that comes from the human nature: what matters in music is not notes in themselves but the interval in semitones between neighboring notes. We also implement the evolution that runs on top of this hint that is not Darwinian, it is Lamarckian.

We conclude in chapter VI with our conclusions, the first of which is that studied Lamarckian version of evolution is found to be fruitful but not enough to think that our programs could be of appeal for musicians. The great conclusion is that music is a promising metaphor of life, which is rich and complex but tractable. We expect that its study will teach us a lot about both evolution and the human nature.

We used NetBeans 8.2 and Java 8. A zip file is provided with all programs and a needed *.jar* file. By July/2018, Java JDK 8u171 with NetBeans 8.2 can be downloaded from

<http://www.oracle.com/technetwork/java/javase/downloads/jdk-netbeans-jsp-142931.html>

<http://www.oracle.com/technetwork/java/javase/downloads/jdk-netbeans-jsp-142931.html>

For those that prefer other editors, we have provided a *.txt* file with all programs.

Chapter 1

Getting started

Prerequisites

1 Purpose. *Some notions of music and of its encoding as a string are needed. We indicate how to acquire them. We also need to install Java, our programs and a package.*

1.1 Let us learn music

Basically, one needs to understand how music is encoded into a stave and how sounds are represented by notes in there.

2 Learning the fundamentals.

One can learn the basics of theoretical music from one of the various free online courses of music, say, from that offered at the web page:

<https://www.musictheory.net/lessons>

(Verified 13/VI/2018)

These lessons are animated and must be played as if they were musical peaces.

The very first lessons are enough to understand how to encode music as an ordinary text, and one can advance by studying both themes at the same time. Everything is highly structured so, one can dedicate some 10 to 15 minutes every day to see nice fruits very soon.

The Author also has found very useful to have an authoritative and free pdf. The following one contains an excellent exposition:

Understanding Basic Music Theory
Collection edited by:

Catherine Schmidt-Jones

By:

Catherine Schmidt-Jones

Russell Jones

(Authors listed in the order their modules appear)

Online:

<http://cnx.org/content/col110363/1.3/>

<http://cnx.org/content/col110363/1.3/>

(Verified 13/VI/2018)

1.2 Plain text notations

While the ordinary representation of music by means of a staff has proven to be excellent for musicians, another representation is needed for those people that want to use computers for musical work.

3 MusiTexts. *Computers work with strings of symbols, so, we need a form to encode musical pieces as **musiTexts**, which are strings of symbols that encode for musical pieces.*

4 Popular proposals. *Currently, we have various proposals to encode music in plain text format. Let us cite some of them, whose names span in the Internet a lot of answers:*

1. MIDI: Musical Instrument Musical Interface is a notation that was developed for the need to interconnect electronic synthesizers and computers. It is simple and direct but too technical for human beings.
2. GUIDO: an historical hit into the world of musiText technology (Guido is the name of a pioneer of ordinary musical notation based on staves). It promotes a compromise between economy and transparency.
3. JMUSIC: This is a very good Java project that teaches everything about digital music. We hope that our material here enables the Reader to better enjoy that work.
4. ABC NOTATION: its name derives from the fact that A, B and C represent musical notes in English and German. It is the prototype of what most humans like: something simple, transparent, intuitive and smart.

5. LILYPOND: shows how one can climb from simplicity to perfection. It is good for those that want and are mature enough to strive at musical expertise.
6. JFUGUE: shows how strongly a well thought idea of a Java developer can impact the community.
7. MusicXML: is an extension of XML for music. It is rather cumbersome but anyway broadly used.
8. MUSIC21: shows how one can flip from simplicity into studies for CAMC (computer aided musical composition).
9. MUESCORE and TUXGUITAR exemplify applications for musical work with a very friendly graphical interface. One can use them without knowing that they use MusicXML, but that becomes necessary if one aims at using them in connection with personally developed programs.

All cited options and others are very good and the election of one of them for personal use can be just circumstantial. Nevertheless, MIDI notation is needed for everyone that wants to understand how musical electronic synthesizers connect with computers. At the same time, one needs a human friendly language. Our recommendation for this volume is to begin with ABC NOTATION and then one can diversify to one of the aforementioned options. Actually, every person in the deal recognizes various formats or languages and ordinary applications may offer translation facilities.

1.3 ABC NOTATION

One can rapidly learn the basics of the ABC language from the `abcnotation` site:

<http://abcnotation.com/blog/2010/01/31/how-to-understand-abc-the-basics/>

(Verified 13/XVI/2018)

Visiting the home of this website allows one to get acquainted with may free offers as, say, thousand of tunes encoded in ABC NOTATION.

5 Example

The most encouraging point of ABC NOTATION is that it is simple and transparent. One can see that at once if one pays attention to the next tune encoded in ABC NOTATION:

We will also need a one letter notation that will appear in the corresponding programs.

For most of this work, this is all one needs to know. Nevertheless, it is better to aspire for more specially if help is offered:

1.4 Two free applications

Our purpose is to devise our own application to use evolution for music composition under the ABC NOTATION. This is a complex goal. A simpler one is just to make an app to play music formatted with ABC NOTATION. In this regard, there are many free applications that allows us to enjoy the ABC language. Two examples follow, the first for Windows and the second for Linux:

6 *ABCExplorer*

This is a very friendly application for ABC NOTATION. It can be downloaded from:

<http://stalikez.info/abc/abce2.php?clc=-1&zc=1zzuA>

(Verified 13/VI/2018)

Let us notice that this application was developed for Windows XP and therefore one needs the compatibility tool for advanced versions of that system. The application also offers the possibility to change the interface language to some 7 European languages.

7 *ABCJ*

One can use ABCJ as an application to play and edit musiTexts. It can be downloaded from

<http://abcj.ganderband.com/>

(Verified 13/VI/2018)

It comes in the form of a *.jar file* which is self-installing and that needs no platform as Eclipse or Netbeans to run. It works on both Windows and Linux. The Java source is also available.

8 *Playing a tune*

The GUI of both aforementioned applications are built over the same model: one window for editing ABC-tunes, a second window for the score and a third one

for the control of files. To listen at a tune, one paste it in the editor window and then looks for the play command in the menu bar. If nothing happens in ABCExplorer, click F5 and try again.

9 *ABC for Beginners*

With the hope of facilitating the learning of both music and ABC NOTATION, we propose a form of using anyone of those two aforementioned applications to make a lab as follows:

1. ABC NOTATION uses plain text format, so one can use aforementioned applications or any word processor to encode a musical piece and save it with the *.abc* suffix. It can be retrieved for further use by both applications.
2. ABC allows to organize tunes in books. A book contains many tunes and the corresponding file is in plain text format with all tunes, one after other.
3. One can use the book *A first course in Music* that appears in our site

www.evoljava.com

(Verified 14/VI/2108)

This work comes in Spanish. An accompanying file with many ABC tunes also can be retrieved.

1.5 Music as a parable

We will need possibly millions of interconnected computers in order to cope for realistic models of artificial life to see exactly what is the function of a given genetic feature or the effect of a proposed change. Nevertheless, humans beings also need simple models that can be used to transmit the knowledge of some folds of a very complex reality. In this regard, we are committed to show that music is and for ever will be a wonderful parable for genetics:

10 *The musiText parable:*

A musiText encodes for a musical piece much as DNA encodes for an organism.

A first fact about musicTexts is that the possible forms of encoding music could be infinite. If we make use of our parable, we will immediately ask: why, then, we have only one genetic code in nature? Actually, we have some few genetic

codes but the accepted explanation for their existence is that they appeared by modification of descent of an ancient original single genetic code (this assertion is worth serious simulations by our community).

We see here that the immediate application of our parable is to illustrate simple, direct, attractive and challenging problems.

As a second example, let us consider the next question: how can one say to the computer that a given substring encodes for a musical piece while other one encodes for a command to play that string very slowly? Thus, we arrive to the conclusion that when encoding music as a string, one needs two types of substrings: the first is structural and encodes for musical events and the second is regulative and encodes for commands that the computer must obey when playing the music. A regulative substring in ABC NOTATION is called a **field** and is distinguished by a letter and a semicolon. Let us turn now to our parable: Is there any need to include regulative substrings in the DNA information? Is that need absolute or just optional? Why? How? Is there a unique form of doing that?

Third example: music and music composition go hand in hand. For instance, children five years old may get engaged in composing variations. As a result we have many diverse genres of music. But, what is a genre? How many possible genres can exist? How can one invent a new and charming genre? Now, would the Reader use our parable to formulate analogous problems in biology?

If one already have a problem and a lab, one rapidly will pick up many data that will push one to formulate possible answers and this is precisely what this volume is all about. Indeed, we will see how to build a lab in such a way that the possibility to set towards perfection might be triggered.

1.6 Installing NETBEANS and Java

Our material is addressed to persons that are committed to clearly understand why the genome is software and how evolution is a software developer. Since that can hardly be done without knowing what a computer program is, the need of learning a programming language arises. The Java fundamentals can be learned with either Volumes I or V of this series, *Java for the study of evolution*, at our site. Nevertheless, making music could fire the desire for programming even more than our official proposals. If that is your case, you can learn how to run our programs by reading this section.

Sun Microsystems composed Java as a great symphony and distributed it without fees. Sun Microsystems was acquired by Oracle in 2010, which continues its

free distribution. Use Google to contact this company and to load and install Java on your desktop. To that aim, you can prompt Google with `java netbeans bundle`: By July/2018 you can get there through a direct link:

<http://www.oracle.com/technetwork/java/javase/downloads/jdk-netbeans>

This bundle comes with a lot of tools, which together form a run time environment.

Under Windows one can install the bundle with one click. Over Linux Ubuntu one can try the instructions given here:

<https://www.youtube.com/watch?v=TawU0xIWe3o>

How to install Netbeans and the Java JDK on Ubuntu (Linux)

kodemonkeez

Published on Sep 30, 2013

Verified 3/VII/2018

Alternatively, you can try:

<https://websiteforstudents.com/how-to-install-netbeans-on-ubuntu-16>

Website for Students

How to Install NetBeans on Ubuntu 16.04 / 17.10 / 18.04 with Oracle JDK 8

Verified 3/VII/2018

1.7 Purposes and projects

In Java, before copying or developing a program, one must make clear what the purpose one has in mind is. Let us imagine that the present purpose could be just to begin. This intention defines a project: `starUp`. Then, we must communicate to Java our intention.

11 *Opening a project*

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

12 *Appending a program*

The first thing we must do is to acquire the right to append a program into a project. In Java a program is called class. So, let us click on *File* → *new* → *class* and a window will pop up. Let us fill in the name of the source project: *javaEvolV8*. Let us give a name to our new class: *Welcome*. If desired, we could activate the *Public static void main* property and then we may click on *Finish*. Next, one shall see a new window with the name *Welcome.java* in its upper bar. We can edit our new program in that window.

13 *Copying a program*

Programs are not typed except when they are created. Otherwise, they are transcribed by a copy-paste procedure, for which you have various options

1. To copy each program from the present pdf document. In this case, you must resolve a little problem: the copy operation also copies navigating signals of the document, headers or page numbers, so they must be identified and deleted. The identification is done automatically by Eclipse after the paste operation is complete: extraneous lines of text appear with a red mark at the right margin. They must be deleted.
2. To get the \LaTeX source, which can be opened with any text processor. You can copy any program from it without suffer anyhow. Additionally, you are invited to use our \LaTeX source to devise your own material for yourself or for your students. In general, feel free to do anything good for forming a healthy and strong **evolJava** community.
3. To get a file, in plain text format, with all programs from which you can copy the desired program immediately. To get the file, go to

<http://www.evoljava.com>

The copy-paste procedure might consists of the following steps:

1. Clear from the editor or main window of Eclipse the code that was automatically written by Eclipse for you, if any.
2. Highlight the text that you want to copy.
3. Copy the highlighted text to the clipboard.

4. Verify that the active window corresponds to the chosen program. To activate a window with a program, click on its name in the upper bar of the editor window.
5. Paste the enclosure of the clipboard to the blanked document. One also can try commands. The command for **delete** is control + x, for **copy** is control + c, for **paste** is control + v, for **select all** is control + a.
6. Save the new program: click on the corresponding icon.

14 *Running our first program*

A Java program must obey a certain pattern or structure that is complexity dependent. The simplest is:

```
//Program H14 Welcome
//Every class or program has a name to identify it

public class Welcome {
    //Every program has a main method or procedure

    public static void main(String[] args) {
        //Instruction to print a text to the copnsole
        System.out.println("Welcome to the java world!");
    }
}
```

When Java runs a program, it looks in first instance for the main method and begins its execution.

We have here a true program whose purpose is to write a message in the console. The **console** is a special window where NetBeans reports results and errors. Officially, this window is called **Output**.

Warning: closing the Output does not imply to end the running program. So, to end a program you must click the red button on the left bar of the Output. If the Output is already closed, to end the program you must go to the menu *Run* → *StopBuild/Run*.

If whatever cause hides the console, one can restore its visibility in any one of the two following options:

- a) In the main menu follow the path: *Windows* → *Output*.
- b) Click *Ctrl + 4*

If one uses Eclipse, the procedure to restore the console is as follows:

- a) In the main menu follow the path: *Windows* → *ShowView* → *Console*.
- b) Click *Alt+Shift + Q*, then click *C*.

1.8 How to import a project

Instead of copy-pasting programs into NetBeans, one at a time, one can import a whole project with all its programs at once. All one needs is to turn NetBeans on with its already created workspace and follow the next procedure:

15 Importing the *javaEvolV8* project

Let us import into the workspace the *javaEvolV8* project with all its ready to be run files. To that aim:

1. Download the *.zip* file associated to Vol 8 v2 and keep it in a suitable folder.
2. Over NetBeans follow the menus: *File* → *Import Project* → *From zip*. In the new dialog, browse for the folder where you put the file. Click over the file and next on *Finish*. This operations create a new project, with all the source files of Vol VIII.
3. To see your recently created project, go to the menu *Window* → *Projects*. On the corresponding window, look for your project and expand its tree to find source files. After that, all Java files will appear below *src*.
4. To run a specific program, select it and next pulse *Shift +F6*. Also: left click on the editor window and with right clicking follow the *run file* menu.

16 Guarantee. *If after some two hours of work, you cannot follow the instructions of the present chapter to install Java and NetBeans and to run the first program, it is mandatory that you write me because you are guaranteed to have a happy start. Express your discomfort to*

jose@evoljava.com

1.9 Conclusion

A creative enterprise is more fruitful when one enriches it with appropriate ingredients. We have proposed to enrich the study of evolution with the study of music plus suitable ways of encoding it in the form of *musiTexts*, strings of symbols that

encode for music that it could be played by computers. To enrich own life with the study of music is in modern times perfectly realizable even for very busy people because we have many online courses of music and many applications that help in the study of encoding of music as musiTexts. With this we get ready to see in music a parable of genetics: A musiText encodes for a musical piece much as DNA encodes for an organism. To convert this parable into a source of biological knowledge, we need a lab. To explain how to build one and how to make it to grow is the main purpose of this volume.

Chapter 2

The Java Sound API

Enabling sound in Java

17 Introduction

Java is enabled with support for musical projects thanks to an appropriate package: The `Java Sound API`. It contains elementary instructions and so, it is too technical for our work. Hence, we will use high level APIs that will allow us to practice music rather directly. Nevertheless, one must keep in mind that high level packages or interfaces solve many problems but the price is a lost of freedom. If such a situation appears, it is advisable to return to the Java Sound API to look for elementary building blocks. So, this chapter is for reference but not necessary for going further.

18 A demo

To learn about the potentialities of the Java Sound API, one can download a specially developed demo from

<http://www.oracle.com/technetwork/java/index-139508.html>

(Verified 14/VI/2018)

After checking that sound is enabled in our computer, one can run the demo as follows:

Under Windows, it can be run with one click. Under Linux, one opens a console at the folder where the `.jar` application was placed, say, `/home/jose/JavaSoundDemo`, and types the command:

```
java -jar JavaSoundDemo.jar
```

In response, a GUI with various functionalities will appear, in which one might appreciate some very nice features.

Chapter 3

ABC4J

Music at once

19 Introduction and purpose. *The magic of software consists in encapsulate hard task in boxes to be used with simple calls. So, let us learn how to use ABC4J, a high level Java sound API that has been specially developed to work with the ABC NOTATION.*

3.1 Starting up with ABC4J

If you created your project with the `.zip` file provided by our site, you can skip this section except to get informed: we use a powerful API that can also be installed manually.

Let us learn how to install ABC4J and how to connect it to a program to play a tune that is encoded in ABC NOTATION.

20 How to install ABC4J

To have `ABC4J` available one can follow the next steps:

1. Create a folder to keep `ABC4J`.
2. Use a searching machine to find `ABC4J` on the web. Currently, it is code.google.com/p/abc4j/
(Verified 14/VI/2018)

3. Download, from the download section, *abc4j_v0.5.jar*, and *abc4jApiDoc_v0.5.zip*. Place them in the *ABC4J* folder.
4. Open *Eclipse*.
5. Optional: open a *Java Project* with name *ABC4JV5*. Or reuse project *javaEvolV8*.
6. Go to the menu *Window* → *show view* → *navigator*.
7. In the window of *navigator*, posit the cursor over the name of your project. Right click and select *Properties* → *Java Build Path*, select the *Libraries tab*, and click *Add External JARs*. Find *abc4j_v0.5.jar* and add them to your project.

The documentation can made visible to Eclipse as follows:

1. In the window of *navigator*, posit the cursor over the name of your project. Select *Import* → *General* → *Archive File* → *Browse*. Find the file *abc4jApiDoc_v0.5.zip* and click it, next *OK* and *finish*.
2. Verify through the menu *Window* → *Show view* → *Package Explorer* that *ABC4J* and its documentation appear in the referenced libraries.

3.2 Basic tasks at once

ABC4J allows us to play an ABC-musiText using a very simple and intuitive code.

21 *Let us play the scale.*

We can run now our first program that plays the scale.

```
//Program H21 FirstTest
//This programs plays an ABC tune
//that comes as a string,
//as a musiText.
//Turn on multimedia speakers.

import java.io.IOException;
//The ABC4J API comes here:
import abc.midi.TunePlayer;
import abc.notation.Tune;
```

```

import abc.parser.TuneParser;

public class FirstTest {

    public static void main(String[] args) throws IOException {
        String musiText
            = "X:0\n"
            + "T:A simple scale exercise\n"
            + "M:4/4\n"
            + "K:C\n"
            + "CDEFGABc4 cBAGFEDC4";
        //The musiText is transformed into a tune,
        //an ABC4J musical object
        Tune tune = new TuneParser().parse(musiText);
        //The player converts the tune into
        //a midi message that is sent to
        //hardwared sound synthesizers.
        //If problems appear,
        //an input-out exception must be thrown.
        TunePlayer player = new TunePlayer();
        player.start();
        player.play(tune);
    }
}

```

22 Exercise. Run the program: right click over the Editor Window, where your program is, and choose: Run file else use Shift + F6.

23 Comparison: ABC tune vs. ABC musiText

The scale as an ABC tune:

```

X:0
T:A simple scale exercise
M:4/4
K:C
CDEFGABc4 cBAGFEDC4

```

Regulative or control substrings are distinguished in ABC NOTATION by the semicolon after a single letter that is a reserved symbol of the notation:

X:0 means that we have an ABC tune and the 0 that is the first in a book.

T: stand for the title of the tune.

M: indicates the measure: 4/4 means 4 quarters per measure.

K: indicates the key, C stands for C major.

The scale as an ABC4J-musiText: it is the tune as a Java string but with the explicit inclusion of `\n` which is a Java regulative signal for ends of line and carrier.

```
String musiText =
"X:0\n" +
"T:A simple scale exercise\n" +
"M:4/4\n" +
"K:C\n" +
"CDEFGABc4 cBAGFEDC4";
```

24 Encapsulation

Let us rewrite the same code in a better style, one in which important functions are encapsulated in corresponding method:

```
//Program H24 PlayABCmusiText
//plays an ABC tune, no lyrics.
import java.io.IOException;

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneParser;

public class PlayABCmusiText {

    //The musiText is transformed into a tune,
    //an ABC4J musical object
    private static Tune ABCmusiTextToTune(String musiText) {

        Tune tune = new TuneParser().parse(musiText);
        return tune;
    }

    //A tune is played
    private static void playTune(Tune tune) {
        //The player converts the tune into
```

```

        //a midi message that is sent to
        //hardwared sound synthesizers.
        //If problems appear,
        //an input-out exception must be thrown.
        TunePlayer player = new TunePlayer();
        player.start();
        player.play(tune);
    }

    public static void main(String[] args) throws IOException {
        String musiText;

        musiText
            = "X:0\n"
            + "T:Song \n"
            + "M:4/4\n"
            + "K:C\n"
            + "dd2ddedGB4 BBBAA4 AdAB4 AAAAAB4 AG4"
            + "BBA4A4BAG GGGA4B4A4G4";

        Tune tune = ABCmusiTextToTune(musiText);
        //Displays the title of the tune
        System.out.println("Title = " + tune.getTitles()[0]);
        // and its key
        System.out.println("Key = " + tune.getKey().toLitteralNotation());
        playTune(tune);
    }
} //End of main class H24 PlayABCMusiText

```

25 Exercise. Run the program. Remember that you are guaranteed to have a happy starting. If by whatever reason you cannot successfully run this program, express your complaint to the Author in order to negotiate a reliably solution:

jose@evoljava.com

26 We can also read the tune from a file. You must set the file call to your own directory.

You can use ABCExplorer to compose and save a tune to a file. Or you can get a copy of a tune from

<http://abcnotation.com/>

and save it to an appropriate directory. Or you can use a text editor to compose an abc tune and save it with *.txt* or *.abc* extension.

Get the string with the directory where you have saved the tune and insert it in the appropriate place of the next code.

```
//Program H26 PlayABCFile
//Reads an ABC NOTATION tune from a file
//and plays it back.

import java.io.FileNotFoundException;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;

import abc.midi.TunePlayer;
import abcnotation.Tune;
import abc.parser.TuneBook;
import abc.parser.TuneParser;
import java.io.File;

public class PlayABCFile {

    static String nameOfFile =
        "/home/jose/MusicAll/abcScores/"
        + "El_Espíritu_de_Dios_está_en_este_lugarNoLyrics.abc";

    //A tune is read from a file that contains one tune
    private static Tune ABCfileToTune(String nameOfFile)
        throws FileNotFoundException, IOException {

        //The file is read into a string
        String musitext =
            new String(Files.readAllBytes(Paths.get(nameOfFile)));
        System.out.println("musitext = \n" + musitext);

        Tune tune = new TuneParser().parse(musitext);
```

```
return tune;
}

//Book = various uniquely numbered tunes in tandem in the same file.
//The existence of a book is checked and one tune is picked up.
private static Tune tuneFromABCBook(String nameOfFile)
    throws FileNotFoundException {

    //creates a file from the path to the abc
    //file containing the tune to be played
    //Link file call to your own directory.
    File abcFile = new File(nameOfFile);
    //creates a tunebook from the previous file
    TuneBook book = new TuneBook(abcFile);

    //show details about the tunes that are loaded
    System.out.println("Number of tunes in book.abc : "
        + book.size());
    //A book possibly contains various tunes.
    //Choose one and play it back
    int tuneIndex = book.getHighestReferenceNumber();
    Tune tune = book.getTune(tuneIndex);
    return tune;
}

//A tune is played
private static void playTune(Tune tune) {
    //The player converts the tune into
    //a midi message that is sent to
    //hardwared sound synthesizers.
    //If problems appear,
    //an input-out exception must be thrown.
    TunePlayer player = new TunePlayer();
    player.start();
    player.play(tune);
}

public static void main(String[] args) throws IOException {
    Tune tune = ABCfileToTune(nameOfFile);
    //Tune tune = tuneFromABCBook(nameOfFile);
}
```

```

        playTune(tune);
    }
} //End of main class H26 PlayABCFile

```

27 Exercise. *Run the program.*

28 Exercise. *What does happen if we merge the two previous codes? A tune can be input as a `musiText` else as a tune in a file. We have the two corresponding codes in the previous programs. Merge them to compose a class or program that first plays a `musiText` and then a tune from a file. Pay attention to the result. **Answer***

29 Research. *We have verified that two tunes that presumably must be played one after the other, are indeed played in parallel. This bug must be solved with the Java Thread technology. Make a research about that and keep it in mind for future work.*

30 Where is the stave? It is here:

```

//Program H30 ABCmusiTextToStave
//Plays and draws an ABCmusiText.
//Neither regulative commands or lyrics are allowed.
import java.io.IOException;

import javax.swing.JFrame;

import abc.midi.TunePlayer;
import abcnotation.Tune;
import abc.parser.TuneParser;
import abc.ui.swing.JScoreComponent;

public class ABCmusiTextToStave {

//A musiText is transformed into a tune
    private static Tune ABCmusiTextToTune() {
        //Tune as musiText
        String musiText = "X:0\n"
            + "T:The scale \n"
            + "M:4/4\n"
            + "K:C\n"
    }
}

```

```

        + "CDEFGABc4 cBAGF - EDC8";
//From String to Tune
Tune tune = new TuneParser().parse(musiText);
return tune;
}

//A tune is drawn
//Try tunes without regulative instructions or lyrics
private static void DrawTune(Tune tune) {
    //creates a component that draws the melody on a stave
    JScoreComponent jscore = new JScoreComponent();
    jscore.setJustification(true);
    jscore.setTune(tune);
    JFrame j = new JFrame();
    j.add(jscore);
    j.pack();
    j.setVisible(true);
}

//A tune is played
private static void playTune(Tune tune) {
    //Displays the title of the tune
    System.out.println("Title = " + tune.getTitles()[0]);
    // and its key
    System.out.println("Key = "
        + tune.getKey().toLitteralNotation());
    //The player converts the tune into
    //a midi message that is sent to
    //hardwared sound synthesizers.
    //If problems appear,
    //an input-out exception must be thrown.
    TunePlayer player = new TunePlayer();
    player.start();
    player.play(tune);
}

public static void main(String[] args) throws IOException {
    Tune tune = ABCmusiTextToTune();
    DrawTune(tune);
    playTune(tune);
}

```

```

    }
} //End of main class H30 ABCmusiTextToStave

```

31 Exercise. *Run the program. Pay attention to the fact that one can draw many times the same tune and that a new stave will pop up for each time. These drawings can be closed one by one else all at the same time by clicking on the red squared stop button of the console.*

32 Exercise. *Reuse previous programs to draw a tune from a file. **Answer***

3.3 The ABC4J MusiText Class

We have treated musiTexts as instances of String. This is correct but does not correspond to what we have in our mind: a musiText is not an ordinary string, it is string that is supported with specific to it methods or procedures. The whole ensemble -musiTexts and methods- works like a unity. Since this complaint appear universally in every case of programming, modern languages are provided with the possibility of encapsulating those indivisible unities. Let us show in three steps how this is done. The first step is encapsulation of methods, the second is the full fledged definition of the class and the third is the instantiation of that class for a specific need in a client program.

33 Encapsulation

Our first step is to put in a class all the methods associated with musiTexts but we delete the main method: the idea is that of a toolbox.

```

//Program H33 MusiTextEncapsulation
//A toolBox with the methods associated with musiTexts.
import javax.swing.JFrame;

import abc.midi.TunePlayer;
import abcnotation.Tune;
import abc.parser.TuneParser;
import abc.ui.swing.JScoreComponent;

public class MusiTextEncapsulation {

```

```
//A musiText is transformed into a tune
private static Tune ABCmusiTextToTune() {
    //Tune as musiText
    String musiText = "X:0\n"
        + "T:The scale \n"
        + "M:4/4\n"
        + "K:C\n"
        + "CDEFGABc4 cBAGF - EDC8";
    //From String to Tune
    Tune tune = new TuneParser().parse(musiText);
    return tune;
}

//A tune as Tune is drawn
//Try tunes without regulative instructions or lyrics
private static void DrawTune(Tune tune) {
    //creates a component that draws the melody on a stave
    JScoreComponent jscore = new JScoreComponent();
    jscore.setJustification(true);
    jscore.setTune(tune);
    JFrame j = new JFrame();
    j.add(jscore);
    j.pack();
    j.setVisible(true);
}

//A tune is played
private static void playTune(Tune tune) {
    //Displays the title of the tune
    System.out.println("Title = " + tune.getTitles()[0]);
    // and its key
    System.out.println("Key = "
        + tune.getKey().toLitteralNotation());
    //The player converts the tune into
    //a midi message that is sent to
    //hardwared sound synthesizers.
    //If problems appear,
    //an input-out exception must be thrown.
    TunePlayer player = new TunePlayer();
}
```

```

        player.start();
        player.play(tune);
    }

} //End of class H33 MusiTextEncapsulation

```

34 Exercise. *Try to run this toolBox. Relate the result with the fact that it lacks the main method.*

35 Full fledged definition. *We add variables and the constructor methods.*

```

//Program H35 MusiText
//The MusiText class:
//Encapsulation of variables, methods and constructors.
import javax.swing.JFrame;

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneParser;
import abc.ui.swing.JScoreComponent;

public class MusiText {

    String musiText;

    //Constructor
    //This specifies how a musiText is initialized.
    //Here, it acquires information from a String
    MusiText(String a) {
        musiText = a;
    }

    //One can initialize it by default
    MusiText() {
        musiText
            = "X:0\n"

```

```

        + "T:The scale \n"
        + "M:4/4\n"
        + "K:C\n"
        + "CDEFGABc4 cBAGF - EDC8";
    }

//A musiText is transformed into a tune
    private static Tune ABCmusiTextToTune(String musiText) {

        //From String to Tune
        Tune tune = new TuneParser().parse(musiText);
        return tune;
    }

//A tune as Tune is drawn
//Try tunes without regulative instructions or lyrics
    private static void DrawTune(Tune tune) {
        //creates a component that draws the melody on a stave
        JScoreComponent jscore = new JScoreComponent();
        jscore.setJustification(true);
        jscore.setTune(tune);
        JFrame j = new JFrame();
        j.add(jscore);
        j.pack();
        j.setVisible(true);
    }

//A tune is played
    private void playTune(Tune tune) {
        //Displays the title of the tune
        System.out.println("Title = " + tune.getTitles()[0]);
        // and its key
        System.out.println("Key = "
            + tune.getKey().toLitteralNotation());
        //The player converts the tune into
        //a midi message that is sent to
        //hardware sound synthesizers.
        //If problems appear,
        //an input-out exception must be thrown.
        TunePlayer player = new TunePlayer();
    }

```

```

        player.start ();
        player.play (tune);
    }

} //End of class H35 MusiText

```

36 Exercise. Pay attention to the constructors, i.e. as the form as one initializes an instance of the class. Observe that there could be various initialization procedures, hence there can be various constructors. Also notice that we have *MusiText* and *musiText*. The first refers to the class, to a prototype to instantiate variables, the second to a variable.

37 MusiTexts in use. How *musiTexts* are used in ordinary life is shown in the next program. For pedagogical reasons, the *MusiText* class is inserted as an inner class of the client program, nevertheless, Java projects are intended to be a collection of classes that serve a given purpose, each one in its own file, so that compilation goes faster because non touched classes are not recompiled.

```

//Program H37 MusiTextClient1
//The class MusiText is instantiated
//to enable ordinary functionality.
//The encapsulated MusiText class is inserted as
//an inner class and a main method is added.

import java.io.IOException;

import javax.swing.JFrame;

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneParser;
import abc.ui.swing.JScoreComponent;

public class MusiTextClient1 {

    //*****
    //The MusiText class is inserted as an inner class
    //*****

```

```
public class MusiText1 {

    String musiText;

    //Constructor
    //This specifies how a musiText is initialized.
    //Here, it acquires information from a String
    MusiText1(String a) {
        musiText = a;
    }

    //One can initialize it by default
    MusiText1() {
        musiText
            = "X:0\n"
            + "T:The scale \n"
            + "M:4/4\n"
            + "K:C\n"
            + "CDEFGABc4 cBAGF - EDC8";
    }

    //A musiText is transformed into a tune
    Tune ABCmusiTextToTune(String musiText) {

        //From String to Tune
        Tune tune = new TuneParser().parse(musiText);
        return tune;
    }

    //A tune as Tune is drawn
    //Try tunes without regulative instructions or lyrics
    void DrawTune(Tune tune) {
        //creates a component that draws the melody on a stave
        JScoreComponent jscore = new JScoreComponent();
        jscore.setJustification(true);
        jscore.setTune(tune);
        JFrame j = new JFrame();
        j.add(jscore);
        j.pack();
        j.setVisible(true);
    }
}
```

```

    }

    //A tune is played
    void playTune(Tune tune) {
        //Displays the title of the tune
        System.out.println("Title = " + tune.getTitles()[0]);
        // and its key
        System.out.println("Key = "
            + tune.getKey().toLitteralNotation());
        //The player converts the tune into
        //a midi message that is sent to
        //hardwared sound synthesizers.
        //If problems appear,
        //an input-out exception must be thrown.
        TunePlayer player = new TunePlayer();
        player.start();
        player.play(tune);
    }

} //End of class MusiText

//*****
//***** Definition and default initialization
//***** of an instantiation of MusiText
//*****
static final private MusiTextClient1 A = new MusiTextClient1();
static final private MusiText1 M = A.new MusiText1();

//*****
//***** Main method of the client program
//*****
public static void main(String[] args) throws IOException {
    System.out.println(M.musiText);
    Tune tune = M.ABCmusiTextToTune(M.musiText);
    M.DrawTune(tune);
    M.playTune(tune);
}

} //End of main class H37 MusiTextClient1

```

38 Exercise. Run the program. Verify that it contains a main method. Compare that main with the analogous ones in the programs written in the old ordinary style. Pay attention to the restriction that the methods of the toolBox cannot be declared as static. This is because the methods of a toolBox are intended to be used at the moment when they are needed and then forgotten about. So, the associated used memory is volatile.

39 Exercise. Add to the MusiText class the possibility to read a tune from a file and use that option in the client program. *Answer*

40 Exercise. The methods of a well designed toolBox must meet, at least, four requirements: names shall perfectly reflect their function, they must process a non-void input, make no reference to global variables, and must be irreducible, i.e. execute a single function that is not worth separating into more elementary sub-functions. This style favors reuse, i.e., evolution, and strongly prevents possibilities for entanglement and ensuing bugs. Revise our MusiText class to made the necessary amendments. *Answer*

3.4 A graphical User Interface

We have gained the possibility to play an ABC tune and to see its score (at least in the simplest cases). Let us add now a graphical user interface and, by the way, the simplest one.

41 The simplest GUI

```
//Program H41 SimplestGUI
//This is a GUI for editing ABC code.
//Can play an ABC tune
//and show its score.
//The score can be rudimentarily edited.
//
//Paste your ABC tune into the text area
//and edit it at pleasure.
//
//A soft introduction to GUI's was done in
//Chapter I of Vol III of the series
//Java for the study of evolution,
```

```

//www.evoljava.com
import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneParser;
import abc.ui.swing.JScoreComponent;

public final class SimplestGUI
    extends JFrame {

    private static final long serialVersionUID = 1L;

    protected JTextArea m_monitor;

    protected JToolBar m_toolBar;

    //creates a simple midi player to play the melody
    private static final TunePlayer PLAYER = new TunePlayer();

    //Initialization
    String musiText
        = "X:0\n"
        + "T:Song \n"
        + "M:4/4\n"
        + "L:1/4"
        + "K:C\n"
        + "dd2d |ded2 |GB3 | "
        + "BBBA | A3 A | dAB2 |"
        + " AAAA | B2A A | G4 | "
        + "Gded- |d edG | B4 | "
        + "BBA2 | A A3 | BAG2|"
        + "GGGA-|A2 B2 | A4 | G4";

    //Constructor
    public SimplestGUI() {

```

```

        super.setSize(450, 350);

//Text area = editing window
        m_monitor = new JTextArea();
        JScrollPane ps = new JScrollPane(m_monitor);
        super.getContentPane().add(ps, BorderLayout.CENTER);
        m_monitor.append(musiText);

        JMenuBar menuBar = createMenuBar();
        super.setJMenuBar(menuBar);

        WindowListener wndCloser = new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        };

        super.addWindowListener(wndCloser);

        super.setVisible(true);
    } //End constructor

//=====
//=====MENU BAR=====
//=====
//The menu bar and listeners
    protected JMenuBar createMenuBar() {
        final JMenuBar menuBar = new JMenuBar();

        //File menu
        JMenu mFile = new JMenu("File");
        mFile.setMnemonic('f');

        //New file
        ImageIcon iconNew = new ImageIcon("file_new.gif");

        Action actionNew = new AbstractAction("New", iconNew) {
            private static final long serialVersionUID = 1L;

```

```

        //what shall be done if the new menu is clicked on
        @Override
        public void actionPerformed(ActionEvent e) {
            m_monitor.setText("");
        }
    };

//item is a variable for diverse itemMenus
JMenuItem item = mFile.addActionNew();
mFile.add(item);

//Exit menu
Action actionExit;
actionExit = new AbstractAction("Exit") {
    private static final long serialVersionUID = 1L;
    //what shall be done if the exit menu is clicked on

    @Override
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
};

item = mFile.addActionExit();
item.setMnemonic('x');
menuBar.add(mFile);

m_toolBar = new JToolBar();

//ShowPlay menu
JMenu mShowPlay = new JMenu("ShowPlay");
mShowPlay.setMnemonic('e');
menuBar.add(mShowPlay);

//Show text menu
ImageIcon iconShowText
    = new ImageIcon("ShowPlay_ShowText.gif");
Action actionShowText
    = new AbstractAction("ShowText", iconShowText) {
    private static final long serialVersionUID = 1L;

```

```

        @Override
        public void actionPerformed(ActionEvent e) {
            String ss = m_monitor.getText();
            System.out.println(ss);
        }
    };

    JMenuItem showText = mShowPLay.add(actionShowText);
    mShowPLay.add(showText);

//=== Show tune's score menu
    ImageIcon iconShowTune
        = new ImageIcon("ShowPLay_ShowTune.gif");
    Action actionShowTune
        = new AbstractAction("ShowTune", iconShowTune) {
        private static final long serialVersionUID = 1L;

        @Override
        public void actionPerformed(ActionEvent e) {
            String ss = m_monitor.getText();
            System.out.println(ss);
            Tune tune;
            tune = ABCmusiTextToTune(ss);
            DrawTune(tune);
        }
    };

    JMenuItem showTune = mShowPLay.add(actionShowTune);
    mShowPLay.add(showTune);

    ImageIcon iconPlayTune
        = new ImageIcon("ShowPLay_PlayTune.gif");
    Action actionPlayTune;
    actionPlayTune = new AbstractAction("PlayTune", iconPlayTune) {
        private static final long serialVersionUID = 1L;

        @Override
        public void actionPerformed(ActionEvent e) {
            String ss = m_monitor.getText();

```

```

        System.out.println(ss);
        Tune tune;
        tune = ABCmusiTextToTune(ss);
        playTune(tune);
    }
};

JMenuItem play = mShowPlay.add(actionPlayTune);
mShowPlay.add(play);

//==Show score and play tune-menu
    ImageIcon iconShowAndPlayTune = new
ImageIcon("ShowPlay_ShowAndPlayTune.gif");
    Action actionShowAndPlayTune
        = new AbstractActionImpl("ShowAndPlayTune",
iconShowAndPlayTune);

    JMenuItem ShowAndPlay
        = mShowPlay.add(actionShowAndPlayTune);
mShowPlay.add(ShowAndPlay);

//=====Panic menu
    JMenu mPanic = new JMenu("Panic");
mPanic.setMnemonic('A');
menuBar.add(mPanic);

    ImageIcon iconPanic
        = new ImageIcon("ShowPlay_Panic.gif");
Action actionPanic;
actionPanic = new AbstractAction("Panic", iconPanic) {
    private static final long serialVersionUID = 1L;

    @Override
    public void actionPerformed(ActionEvent e) {
        PLAYER.stopPlaying();
    }
};

JMenuItem Panic = mPanic.add(actionPanic);
mPanic.add(Panic);

```

```

//=====
//The toolbar is added to the container
    getContentPane().add(m_toolBar, BorderLayout.NORTH);

    return menuBar;
} //End of menu bar construction

//=====
//      Some ABC tools
//=====
//A musiText is transformed into a tune
    private static Tune ABCmusiTextToTune(String musiText) {
        Tune tune = new TuneParser().parse(musiText);
        return tune;
    }

//An ABC-tune is drawn
    private static void DrawTune(Tune tune) {
        //creates a component that draws the melody on a musical staff
        JScoreComponent jscore = new JScoreComponent();
        jscore.setJustification(true);
        jscore.setTune(tune);
        JFrame j = new JFrame();
        j.add(jscore);
        j.pack();
        j.setVisible(true);
    }

//A tune as Tune is played
    private static void playTune(Tune tune) {
        //display the title of the tune
        System.out.println("Title = " + tune.getTitles()[0]);
        //and its key
        System.out.println("Key = " + tune.getKey().toLitteralNotation());
        PLAYER.start();
        PLAYER.play(tune);
    }

    public static void main(String argv[]) {

```

```

SimplestGUI simplestGUI = new SimplestGUI();
//Title in the menu bar
simplestGUI.setTitle("Simplest GUI for musiTexts");
}

private class AbstractActionImpl extends AbstractAction {

    public AbstractActionImpl(String name, Icon icon) {
        super(name, icon);
    }
    private static final long serialVersionUID = 1L;

    @Override
    public void actionPerformed(ActionEvent e) {
        String ss = m_monitor.getText();
        System.out.println(ss);
        Tune tune;
        tune = ABCmusiTextToTune(ss);
        DrawTune(tune);
        playTune(tune);
    }
}
} //End of main class H41 SimplestGUI

```

42 Exercise. Run the program and verify that it can be used to show a tune over a stave, to play it and that additionally a playing tune can be stopped at will. And, what about the possibility to edit it? You have now the opportunity and duty of becoming a great composer!

43 Exercise. Try to simplify this program but conserving all its functions. Would you agree that the simplest Java GUI that shows and plays a tune has as minimum 100 lines? *Answer*

44 Exercise. Rewrite the previous code for a GUI using the MusiText Class. Do you feel that this is an additional optional sophistication or rather an important spin in programming? *Answer*

3.5 Conclusion

We have seen ABC4J as an example of a high level programming language for music: no much code is needed to play a tune or to see its score. Nevertheless, at its present stage of development, ABC4J present some difficulties: it cannot dealt with neither lyrics or regulative commands, without which ordinary music cannot be imagined. Nevertheless, it is very useful such as it is. In particular, it has allowed us to dress the concept of musiText into suitable processing procedures and so we conformed the class MusiText in which we encapsulated the

variable that carries the musical information, the constructors to initialize the instantiations and the associated processing methods. How this technology is used in ordinary life was illustrated with a client program which was next embedded in a GUI.

Chapter 4

The first theory of music

Music is everything

45 Introduction and objective. *The sound of a tender wind or of rain is pleasant for everybody. Nevertheless, they will be ignored in our discussions: a complete theory of music is beyond our possibilities. Nevertheless, we can formulate tractable theories of music that allow us to partially understand what music is. Our procedure is as follows: we formulate a theory about music, next we test it in composition. If the theory stands the test and we can produce good musical pieces, we are done. Else, we must reformulate the theory or invent another one to be tested anew. To begin with, we present here the **first theory** about music and test how good it is to produce acceptable tunes. We also will look forward for possible applications.*

4.1 Formulation and predictions

What is the theory? Which are its predictions? Which are its pedagogical implications? How good it is? Which are its immediate corrections? Let us see.

46 The theory. *The **first theory of music** reads: notes are pleasant and so are strings of sound formed with them.*

47 Predictions. *Our theory forces some predictions about music composition that are very simple to test.*

1. *Strings of notes concatenated at random are pleasant.*
2. *Any string of symbols can be translated into music, in particular, it is a must to listen to the music of DNA.*

3. *To compose a very good tune, take a good one, make a list of its notes and use it as the alphabet to feed the random generator.*

48 Pedagogical implications. *Students must be taught to recognize notes.*

Let us develop this program step by step.

4.2 The sound of randomness

The **first theory** of music says that all musiTexts must be pleasant. As a consequence, one can synthesize musiTexts with random concatenation of notes to see what happens.

Java allows us to very easily generate pseudo randoms strings of musical notes. The idea is to set an alphabet from which notes could be drawn at random.

49 The code. *Here, randomness is made into a generator of random strings of notes taken from the scale. All notes have the same duration and volume.*

```
//Program H49 RandomMusic1
//Random notes of equal duration are played.
//The scale is used as the alphabet.
//The inbuilt Java random generator is used.
import java.io.IOException;
import java.util.Random;

import javax.swing.JFrame;

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneParser;
import abc.ui.swing.JScoreComponent;

public class RandomMusic1 {

    //We turn on the inbuilt random generator:
    private static final Random R = new Random();

    //The musiText is transformed into a tune,
```

```
//an ABC4J musical object
private static Tune ABCmusiTextToTune(String musiText) {

    Tune tune = new TuneParser().parse(musiText);
    return tune;
}

//A tune is played
private static void playTune(Tune tune) {
    //The player converts the tune into
    //a midi message that is sent to
    //hardware sound synthesizers.
    //If problems appear,
    //an input-out exception must be thrown.
    TunePlayer player = new TunePlayer();
    player.start();
    player.play(tune);
}

//A tune as Tune is drawn
private static void DrawTune(Tune tune) {
//creates a component that draws the melody on a stave
    JScoreComponent jscore = new JScoreComponent();
    jscore.setJustification(true);
    jscore.setTune(tune);
    JFrame j = new JFrame();
    j.add(jscore);
    j.pack();
    j.setVisible(true);
}

//Randomness is made into a composer
static private String randomNotes() {
    String alphabet = "CDEFGAB";
    //Number of notes
    int n = 10;
    //Output initialization
    String stringOfNotes = "";
    for (int i = 0; i < n; i++) {
        //A random integer less than the
```

```

        //number of chars in the alphabet is thrown
        int k = R.nextInt(alphabet.length());
        //The integer is changed into a note
        char c = alphabet.charAt(k);
        //The note is added to the output
        stringOfNotes = stringOfNotes + c;
    }
    return stringOfNotes;
}

public static void main(String[] args) throws IOException {
    //Initialization
    String header
        = "X:0\n"
        + "T:Song \n"
        + "M:4/4\n"
        + "L:1/8\n"
        + "K:C\n";
    String ss = randomNotes();
    String musiText = header + ss;

    Tune tune;
    tune = ABCmusiTextToTune(musiText);
    DrawTune(tune);
    playTune(tune);
}
} //End of main class H49 RandomMusic1

```

50 Exercise. *Run the program and play with it. Verify else refute the conclusions of the Author: random music generated with the scale as the alphabet and with notes of equal duration has nothing to do with music. More rigorously:*

*the appearing of wonderful musical pieces composed as random music with notes of equal duration escapes the possibilities of the Author given his very limited patience. Thus our **first theory** of music with notes of equal duration and volume has been falsified: music that human beings classify as nice is not in the realm of this theory. Therefore, we must devise a new theory. Our proposal is to continue with the first theory but to include duration as the next important factor that together with pitch will enrich our theory.*

51 *Enlarging the alphabet and including duration.*

The simplicity of the previous code rests on the fact that a note is denoted by a single symbol. That is not the case for the ABC NOTATION in which two symbols are required to include sharps and flats. So, let us make a super-alphabet on our own:

We keep the usual notation for ordinary notes: ABCDEFGabcdefg. But for accidentals we use a single letter as follows, where ordinary ABC NOTATION appears in the first line and ours comes below it:

```
A, _B, B, C #C D _E E F #F G #G A _B B c #c d _e e f #f g #g a _b b
R S T C M D N E F O G P A Q B c m d n e f o g p a q b
```

From this super-alphabet one chooses a subset to form a working alphabet. Say, the previous program was made with CDEFGAB. If one wants a tune with a flavor of blues one must use CbEFbGGAc, where the b stands for flat, E flat and G flat. Using our super-alphabet, that scale reads CNFOGAc.

Our convention forces us to use an interpreter. This is implemented in the next program. But, judging by the results of the previous program, we can predict that nothing good will appear. To foretell this problem, we introduce the next two variations to see what happens: first, an alphabet for variations is introduced. Durations are represented in ABC by numbers, say, 12345678. We will conserve this as the super-alphabet from which one can take a subset, say 1248. Second: We will allow a non uniform distribution for the random generator. This is done as follows: if we want note C to be as twice frequent as the rest, our alphabet will be CCDEFGAB. And so on. Our code follows:

```
//Program H51 RandomMusic2
//Random notes are played.
//One can vary the alphabet,
//the duration of notes and
//their relative frequencies.
//The inbuilt Java random generator is used.
import java.io.IOException;
import java.util.Random;

import javax.swing.JFrame;
```

```

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneParser;
import abc.ui.swing.JScoreComponent;

public class RandomMusic2 {

    //We turn on the inbuilt random generator:
    private static final Random R = new Random();

    //Our one symbol notation for notes :
    //A, _B, B, C #C D _E E F #F G #G
    //R S T C M D N E F O G P

    //A _B B c #c d _e e f #f g #g a _b b
    //A Q B c m d n e f o g p a q b
    private static final String SUPER_ALPHABET
        = "RSTAMBCNDOEFGQambcndoeftpqg";

    private static final String ALPHABET = "RTCDEOGabcdeo";

    private static final String DURATIONS_ALPHABET
        = "1111222222448";

    //The musiText is transformed into a tune,
    //an ABC4J musical object
    private static Tune ABCmusiTextToTune(String musiText) {
        Tune tune = new TuneParser().parse(musiText);
        return tune;
    }

    //A tune is played
    private static void playTune(Tune tune) {
        //The player converts the tune into
        //a midi message that is sent to
        //hardwared sound synthesizers.
        //If problems appear,
        //an input-out exception must be thrown.
        TunePlayer player = new TunePlayer();
        player.start();
    }
}

```

```
        player.play(tune);
    }

//A tune as Tune is drawn
    private static void DrawTune(Tune tune) {
//creates a component that draws the melody on a stave
        JScoreComponent jscore = new JScoreComponent();
        jscore.setJustification(true);
        jscore.setTune(tune);
        JFrame j = new JFrame();
        j.add(jscore);
        j.pack();
        j.setVisible(true);
    }

//Randomness is made into a composer
    static private String randomNotes() {
        //Number of notes
        int n = 100;
        //Output initialization
        String stringOfNotes = "";
        for (int i = 0; i < n; i++) {
            //A note is generated:
            //A random integer less than the
            //number of chars in the alphabet is thrown
            int k = R.nextInt(ALPHABET.length());
            //The interger is changed into a note
            char c = ALPHABET.charAt(k);
            //The note is added to the output
            stringOfNotes = stringOfNotes + c;
            //Duration is generated
            k = R.nextInt(DURATIONS_ALPHABET.length());
            //The integer is changed into a duration
            c = DURATIONS_ALPHABET.charAt(k);
            //The note is added to the output
            stringOfNotes = stringOfNotes + c;
        }
        return stringOfNotes;
    }
}
```

```

//A string in our alphabet is rewritten in ABC notation
//Accidents are prefixes in ABC notation.
//Sharp is denoted by ^, flat by _
    static private String interpreter(String ss) {
        ss = ss.replaceAll("R", "A,");
        ss = ss.replaceAll("S", "_B,");
        ss = ss.replaceAll("T", "B,");
        ss = ss.replaceAll("M", "#C");
        ss = ss.replaceAll("M", "#c");
        ss = ss.replaceAll("N", "_E");
        ss = ss.replaceAll("n", "_e");
        ss = ss.replaceAll("O", "^F");
        ss = ss.replaceAll("o", "^f");
        ss = ss.replaceAll("P", "^G");
        ss = ss.replaceAll("p", "^g");
        ss = ss.replaceAll("Q", "_B");
        ss = ss.replaceAll("q", "_b");
        return ss;
    }

    public static void main(String[] args) throws IOException {
        //Initialization
        String header
            = "X:0\n"
            + "T:Song \n"
            + "M:4/4\n"
            + "L:1/8\n"
            + "K:C\n";
        String ss = randomNotes();
        ss = interpreter(ss);
        String musiText = header + ss;
        System.out.println(musiText);

        Tune tune;
        tune = ABCmusiTextToTune(musiText);
        DrawTune(tune);
        playTune(tune);
    }
} //End of main class R51 RandomMusic2

```

52 Exercise . Run the program and play with it. Experiment with modifications of both alphabets, that of notes and that of durations. Change relative frequencies in them. Take into account that silences are very important in music: in ABC NOTATION, a silence is treated as a note and is distinguished by the letter *z* and the duration is expressed by a number, just as an ordinary note. Experiment enough to refute else to agree with the conclusion of the Author: random music with notes of unequal duration is not that boring and by moments it could be interesting. Thus our **first theory** of music with notes of unequal duration has found to be a good starting point for more mature theories with more descriptive power. Translate your conclusion to the realm of biology. How would you apply your conclusion?

Answer

53 Challenge. Introduce line breaks in the ABC code that the tune could be displayed in various lines instead of one.

54 Exercise. Modify the previous program to make a rhythm machine yourself, i.e., a generator of very small musical pieces that are repeated over and over. To fix ideas, work with a 4/4 measure. Verify else refute the conclusion of the Author: a random rhythm machine is really wonderful, i.e., many rhythms sound really good.

Answer

55 Exercise. Add to the solution of the previous exercise the possibility to synthesize tunes either in 4/4 or in 6/8. *Answer*

56 Exercise. Make the previous composing program to insert a bar line at the end of each measure. *Answer*

57 Exercise. In reference to previous programs of random music, make an estimation of the proportion of rhythms that you like. Would you dare to extrapolate your subjective estimation to a proposal about the realm of biology? *Answer*

4.3 DNA as a composer

We already know that tunes whose notes have changing duration are more interesting than those that not. So, let us develop a program that translates DNA strings into a tune in which the duration of notes changes. This is done in first place for structural DNA, encoding for proteins, and then for raw DNA.

58 Our genetic code

Ordinary one-chain-proteins are strings whose links are aminoacids that come in 20 flavors. A notation exists in which to each amino acid corresponds a single letter. Thus, we can imagine a protein as a string whose letters belong in an alphabet with 20 letters:

Amino Acid alphabet = ARND CEQG HILK MFPS TWYV

To fix ideas, let us take two proteins:

Histone H1.11L

<http://www.ebi.ac.uk/pdbe/entry/pdb/1ghc/protein/1>

Protein Data Bank in Europe

Verified 19/VI/2018

```
MAGPSVTEELITKAVSASKERKGLSL
AALKKALAAGGYDVEKNNSRIKLGL
KSLVSKGTLVQTKGTGASGSFRLSK
```

Insuline chains P01308 A and B :

<http://www.genetics.org/content/162/2/527>

The First Sequence: Fred Sanger and Insulin Antony O. W. Stretton Genetics
October 1, 2002 vol. 162 no. 2 527-532

Verified 19/VI/2018

```
A: GIVEQCCTSI  CSLYQLENYC  N
B: FVNQHLCGSH  LVEALELVCG  ERGGFYTPK
```

We will associate a peptide chain with a musiText as follows: we divide the amino acid chain in codons of two elements, whose first amino acid determines a note and the second a duration. Thus, every amino acid plays two roles depending on the position in the codon. The following genetic code shows how amino acids, in the first line, are associated to notes, second line, durations for 4/4 in the third line and durations for 6/8 in the fourth one.

A	R	N	D	C	E	Q	G	H	I	L	K	M	F	P	S	T	W	Y	V
#A	B	C	#C	D	#D	E	F	#F	G	#G	a	#a	b	c	#c	d	#d	e	f
1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	4	4	4	8	8
1	1	1	1	1	1	1	1	1	3	3	3	3	3	3	3	3	6	6	6

Observe that

1. Our association is capricious.

2. The number of possible associations ranges in the order of billions.
3. The relative frequencies of durations cannot be read from the third or fourth line because amino acids are not uniformly represented in proteins.

59 *Our first implementation of this genetic code is shown below. It shows how one can use a protein primary structure to compose the notes and their durations of an ABC tune.*

```
//Program H59 DNAMusic1
//Shows how one can use a protein primary structure
//to compose the notes and their durations of an ABC tune.
//Measures 4/4 and 6/8 are allowed as an add given by the
//developer.
//
//To transcribe strings of amino acids into strings of notes,
//we use a genetic code whose codons have 2 letters:
//The first letter represents a note and
//the second represents a duration.
import java.io.IOException;

import javax.swing.JFrame;

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneParser;
import abc.ui.swing.JScoreComponent;

public class DNAMusic1 {

//Our genetic code: amino acids and
//the corresponding durations:
//Amino acids in line 1; notes in line 2;
//durations for 4/4 in line 3, for 6/8 in line 4.
//A virtual amino acid, Z; has been added to let us
//concatenate two chains with an ABC rest.
// A R N D C E Q G H I L K M F P S T W Y V Z
// C ^C D _E E F ^F G ^G A _B B c ^c d ^d _e e f ^f z
// 1 1 1 1 1 2 2 2 2 2 2 2 2 2 4 4 4 8 8 8
// 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 6 6 6 6
```

```

//Two chains are concatenated with a ZA in the middle
//That will be interpreted as rest of duration 1
    private static final String INSULINE =
        "IVEQCCTSICSLYQLENYCZA"
        + "NFVNQHLCGSHLVEALELVCGERRGGFYTPK";

    private static final String HISTONE
        = "MAGPSVTEELITKAVSASKERKGLSL"
        + "AALKKALAAGGYDVEKNNNSRIKLGL"
        + "KSLVSKGTLVQTKGTGASGSFRLSK";

//The set of amino acids encoded as a string
//Z stands for a separator that will be
//interpreted as rest by ABC
    private static final String AMINO_ACIDS
        = "ARNDCSEQHILKMFPSTWYVZ";

    private static final String[] NOTES
        = {"C", "^C", "D",
           "_E", "E", "F", "^F", "G",
           "^G", "A", "_B", "B", "c",
           "^c", "d", "_e", "e", "f", "^f", "z"};

//Durations for 4/4
    private static final String DURATIONS_ALPHABET44
        = "111112222222222444888";
//Header 4/4
    private static final String HEADER44
        = "X:0\n"
        + "T:Song 4/4\n"
        + "M:4/4\n"
        + "L:1/8\n"
        + "K:C\n";

//Durations for 6/8
    private static final String DURATIONS_ALPHABET68
        = "111111111333333336666";
//Header 6/8
    private static final String HEADER68
        = "X:0\n"

```

```
        + "T:Song 6/8\n"
        + "M:6/8\n"
        + "L:1/8\n"
        + "K:C\n";

//The musiText is transformed into a tune,
//an ABC4J musical object
    private static Tune ABCmusiTextToTune(String musiText) {

        Tune tune = new TuneParser().parse(musiText);
        return tune;
    }

//A tune is played
    private static void playTune(Tune tune) {
        //The player converts the tune into
        //a midi message that is sent to
        //hardwared sound synthesizers.
        //If problems appear,
        //an input-out exception must be thrown.
        TunePlayer player = new TunePlayer();
        player.start();
        player.play(tune);
    }

//A tune as Tune is drawn
    private static void DrawTune(Tune tune) {
//creates a component that draws the melody on a stave
        JScoreComponent jscore = new JScoreComponent();
        jscore.setJustification(true);
        jscore.setTune(tune);
        JFrame j = new JFrame();
        j.add(jscore);
        j.pack();
        j.setVisible(true);
    }

//Amino acids in the first position of a codon
//are interpreted as notes.
//Interpretation is done amino acid per amino acid.
```

```

static private String notesInterpreter(char a) {
    //The index of a is looked at the list of amino acids
    int i = AMINO_ACIDS.indexOf(a);
    //The corresponding note is looked at the array of notes
    String ss = NOTES[i];
    return ss;
}

//Amino acids in the second position of a codon
//are interpreted as durations of notes.
//Interpretation is done amino acid per amino acid.
//measure = 44 for 4/4; measure = 68 for 6/8;
static private char
    durationsInterpreter(char a, int measure) {
    //The index of a is looked at the list of amino acids
    int i = AMINO_ACIDS.indexOf(a);
    //The corresponding duration is looked
    //at the list of durations
    char c;
    if (measure == 44) {
        c = DURATIONS_ALPHABET44.charAt(i);
    } else {
        c = DURATIONS_ALPHABET68.charAt(i);
    }
    return c;
}

//A protein is transformed into a tune.
//It is divided in codons with 2 letters,
//the first is interpreted as a note,
//the second as a duration
static private String interpreter(String s, int measure) {
    String v = "";
    int l = s.length();
    for (int i = 0; i < l - 2; i = i + 2) {
        char a = s.charAt(i);
        String note = notesInterpreter(a);
        i++;
        a = s.charAt(i);
        char duration = durationsInterpreter(a, measure);
    }
}

```

```

        v = v + note + duration;
    }
    return v;
}

public static void main(String[] args) throws IOException {
    String header;
    //Measure 44 for 4/4, 68 for 6/8
    int measure = 44;
    //String ss = INSULINE;
    String ss = HISTONE;
    int l = ss.length();
    //if l is odd (not even), the last amino acid is trimmed.
    if (!(l == 0 % 2)) {
        l = l - 1;
        ss = ss.substring(0, l);
    }
    String ABCstring = interpreter(ss, measure);
    System.out.println("ABCstring = " + ABCstring);
    if (measure == 44) {
        header = HEADER44;
    } else {
        header = HEADER68;
    }
    String musiText = header + ABCstring;
    Tune tune;
    tune = ABCmusiTextToTune(musiText);
    DrawTune(tune);
    playTune(tune);
}
} //End of main class H59 DNAMusic1

```

60 Exercise. Run the program and play with it.

61 Exercise. What happens if in the previous program one deletes the first amino acid? *Answer*

62 Challenge. Find an encoding that will let you hear the degree of hydrophobicity of a protein. And what about globularity, electronegativity, acidity?

63 Challenge. *We have lost interest in music in which all notes have the same duration. But maybe we are leaving a good opportunity here: would you investigate what it can give of itself in regard with proteins?*

64 Exercise. *Non structural DNA strings.*

Apart from encoding for a protein, DNA also must indicate the genetic regulation of each exon, and moreover it must be structured in such a way that the process of specific localization of genes must not be hindered. So, it is interesting to listen at those sequence that in ancient times were called junk DNA. Develop and run a program to solve this task. Choose one of two options: use the natural genetic code to associate a protein sequence to a DNA sequence and reuse the previous program, else invent your own genetic code that associates an ABC tune to a DNA string. This second option is the chosen by the Author in the answer. **Answer**

65 Challenge. *The genetic code that we chose in the previous program produces a musically uninteresting tune. Would you try another genetic code to improve the musical sense of output tunes?*

66 Exercise. *Show that there are infinitely many forms as one can use DNA to synthesize an ABC tune. Would you see the biological implications of this freedom that we have stressed in many occasions? **Answer***

4.4 Recognizing pitches

Our **first theory** has an immediate pedagogical implication: because notes (pitches + duration) are all to music, so, students must be taught to recognize notes, pitches in first place. This task is almost impossible for almost everybody. In fact, the Author grew with the conviction that music was not for him. Nevertheless, formal music neared its charm somehow to him. Anyway, he cannot recognize a playing note by its name. By contrast, some persons can recognize at once not only notes but also chords, intervals, instruments and voices. Nevertheless, it happens that the distance among the Author and those experts can be filled with many small virtues that use to appear and grow with work. One of them shall be to recognize pitches. The following program implements such a purpose.

67 An application to help the Author to recognize notes.

```
//Program H67 PicthRecognizing1
//The student of music is given a test
```

```

//for pitch recognizing.

import java.util.Random;

import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JScrollPane;
import javax.swing.ListSelectionModel;

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneParser;

import javax.swing.event.ListSelectionEvent;
import java.awt.*;

public class PitchRecognizing1 extends JFrame {

    private static final long serialVersionUID = 1L;

    private static final String[] NOTES
        = {"A", "_B", "B", "C", "^C", "D", "_E",
          "E", "F", "^F", "G", ^G", "A", "_B", "B",
          "c", ^c", "d", "_e", "e", "f", ^f",
          "g", ^g", "a", "_b", "b", "c'"};

    //Every ABC tune must have a correct HEADER
    private static final String HEADER
        = "X:0\n"
        + "T:Song 4/4\n"
        + "M:4/4\n"
        + "L:1/8\n"
        + "K:C\n";

    //The note that must be recognized as a tune
    static private Tune tuneTest;
    //The note that must be recognized as a String
    static private String noteTest;
    //A JList to be displayed
    private static final JList LIST = new JList(NOTES);
    //Counter of trials before success

```

```

static private int counter = 1;

//Pane to display the LIST
Container contentpane;

//Constructor: this is the first task of the program
public PitchRecognizing1() {
    //Title in the upper bar of the pane
    super("Listen to the sound and select the pitch");
    contentpane = super.getContentPane();
    contentpane.setLayout(new FlowLayout());

    LIST.setSelectionMode(ListSelectionMode.SINGLE_SELECTION);
    contentpane.add(new JScrollPane(LIST));
    //The answer by the student is captured and assessed
    LIST.addListSelectionListener((ListSelectionEvent e) \rightarrow
        String noteAnswer
            = NOTES[LIST.getSelectedIndex()];
        String NOTESBoth
            = noteTest + "2" + noteAnswer + "2";
        Tune tuneBoth = noteToTune(NOTESBoth);
        playTune(tuneBoth);
        if (noteAnswer.contentEquals(noteTest)) {
            contentpane.setBackground(Color.GREEN);
            System.out.println("\n "
                + "Your answer = " + noteAnswer
                + " is RIGHT!");
            System.out.println("You needed "
                + counter + " trials");
        } else {
            contentpane.setBackground(Color.RED);
            System.out.println("\n Your answer = "
                + noteAnswer + " was WRONG");
            counter = counter + 1;
        }
    });
    super.setSize(400, 200);
    super.setVisible(true);
}

```

```
//We turn on the inbuilt random generator:
private static final Random R = new Random();

static private boolean print;

//creates a simple midi PLAYER to play the melody
private static final TunePlayer PLAYER = new TunePlayer();

//The musiText is transformed into a tune,
//an ABC4J musical object
private static Tune ABCmusiTextToTune(String musiText) {
    Tune tune = new TuneParser().parse(musiText);
    return tune;
}

//A tune is played
private static void playTune(Tune tune) {
    PLAYER.play(tune);
}

//The note is converted into an ABC tune
static private Tune noteToTune(String note) {
    String musiText = HEADER + note;
    Tune tune = ABCmusiTextToTune(musiText);
    return tune;
}

//Output a random note from the chosen alphabet
static private String randomNote() {
    if (print) {
        System.out.println("RandomNote");
    }
    int k;
    //A random integer less than the
    //number of NOTES is thrown
    k = R.nextInt(NOTES.length);
    //The integer is changed into a note
    String n = NOTES[k];
    return n;
}
```

```

//A random note to be recognized is played
static private void poseQuestion() {
    noteTest = randomNote();
    tuneTest = noteToTune(noteTest + noteTest + noteTest);
    playTune(tuneTest);
}

//The number of questions is defined
static private void examination() {
    for (int i = 0; i < 2; i++) {
        poseQuestion();
    }
}

public static void main(String[] args) {
    PLAYER.start();
    print = false;
    //Instantiation of the application
    PitchRecognizing1 t = new PitchRecognizing1();
    t.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    examination();
}

} //End of main class H67 PicthRecognizing1

```

68 Exercise. *Run the program and play with the code. Play enough to agree else disagree with the opinions of the Author:*

1. *The program suffers from two bugs. First: it gets frozen in the first single question and passes not to a second one. Second: the application reads and assesses the answer of the User twice and therefore the container blasts twice and this causes headache. Let us remember forever: there is no good software without a fierce battle with bugs. In other words, every good software has a rich history of bugs whose correction generated more bugs.*
2. *The program is pretty useless for teaching because it overburdens the User with so many options. Nevertheless, it could be good for checking whether or not the User is an expert.*

3. *In spite of aforementioned problems, the program looks very interesting for the User: it is worth being improved. In particular, given the human nature that longs for reinforcement instead of failure, the program must offer a gradual increase of complexity. Let us notice that we are modifying our software to produce a better one, so we are evolving it, but our changes are not blind, rather they are guided but what we know about human nature, so we are completely sure that planned changes will make it more effective. Thus, we left Darwinian evolution long ago and get residence at the realm of applied Lamarckism. Answer*

69 *An improved version of our helping software to learn pitch recognizing. Contrary to the previous program, the next one poses a very simple test about pitch recognizing and so it is expected to be a help for those users that, like the Author, are nearly deaf. The bug of double reading of the answer of the User was fixed.*

```
//Program H69 PicthRecognizing2
//The student of music is given a
//very simple test
//for pitch recognizing.
//Simplicity enables success and therefore learning.
//Problem: complexity is not increased anyhow.

import java.util.Random;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.ListSelectionModel;

import abc.midi.TunePlayer;
import abcnotation.Tune;
import abc.parser.TuneParser;

import javax.swing.event.ListSelectionEvent;
import java.awt.*;
```

```

public class PitchRecognizing2 extends JFrame {

    private static final long serialVersionUID = 1L;

    private static final String[] NOTES
        = {"C", "c", "c'"};

    //Every ABC tune must have a correct HEADER
    private static final String HEADER
        = "X:0\n"
        + "T:Song 4/4\n"
        + "M:4/4\n"
        + "L:1/8\n"
        + "K:C\n";

    //The note that must be recognized as a tune
    static private Tune tuneTest;
    //The note that must be recognized as a String
    static private String noteTest;
    //A JList to be displayed
    private static final JList LIST = new JList(NOTES);
    private JLabel barMessage;
    //Counter of trials before success
    static private int counter = 1;
    static private int oldK = -1;
    static private Integer lag = 8;

    private final JPanel mPanel;
    //Constructor: this is the first task of the program

    public PitchRecognizing2() {
        //Title in the upper bar of the pane
        super("Listen to the sound and select the pitch");
        mPanel = new JPanel();
        mPanel.setBackground(Color.gray);
        super.add(mPanel, BorderLayout.CENTER);
        barMessage = new JLabel("Click on your selection");
        super.add(barMessage, BorderLayout.SOUTH);
        LIST.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        mPanel.add(new JScrollPane(LIST));
        //The answer by the student is captured and assessed
    }
}

```

```

LIST.addListSelectionListener((ListSelectionEvent e) \rightarrow {
    String noteAnswer
        = NOTES[LIST.getSelectedIndex()];
    String NOTESBoth
        = noteTest + "2" + noteAnswer + "2";
    Tune tuneBoth = noteToTune(NOTESBoth);
    playTune(tuneBoth);
    if (noteAnswer.contentEquals(noteTest)) {
        //To avoid double response from the system
        if (!e.getValueIsAdjusting()) {
            //mPanel.setBackground(Color.GREEN);
            barMessage.setText(
                String.format(
                    "Your answer = " + noteAnswer
                    + " is RIGHT!" + "  Number of trials = "
                    + counter
                )
            );
            lag = 8;
            poseQuestion(lag);
            counter = 1;
        }
    } else {
        //To avoid double response from the system
        if (!e.getValueIsAdjusting()) {
            //mPanel.setBackground(Color.RED);
            barMessage.setText(String.format("Your answer = " +
noteAnswer
                + " was WRONG!"));
            counter++;
        }
    }
});
super.setSize(400, 200);
super.setVisible(true);
}

//We turn on the inbuilt random generator:
private static final Random R = new Random();

```

```

//creates a simple midi PLAYER to play the melody
private static final TunePlayer PLAYER = new TunePlayer();

//The musiText is transformed into a tune,
//an ABC4J musical object
private static Tune ABCmusiTextToTune(String musiText) {
    Tune tune = new TuneParser().parse(musiText);
    return tune;
}

//A tune is played
private static void playTune(Tune tune) {
    PLAYER.play(tune);
}

//The note is converted into an ABC tune
static private Tune noteToTune(String note) {
    String musiText = HEADER + note;
    Tune tune = ABCmusiTextToTune(musiText);
    return tune;
}

//A random note from the chosen alphabet is output
static private String randomNote() {
    String note;
    int newK;
    //A random integer less than the
    //number of NOTES is thrown
    newK = R.nextInt(NOTES.length);
    //the new challenge is different than the prior one
    if (oldK == newK) {
        newK = (newK + 1) % 3;
    }
    //The integer is changed into a note
    note = NOTES[newK];
    oldK = newK;
    return note;
}

//A random note to be recognized is played

```

```

static private void poseQuestion(Integer sleep) {
    String s = sleep.toString();
    noteTest = randomNote();
    tuneTest = noteToTune("z" + s + noteTest + noteTest + noteTest);
    playTune(tuneTest);
}

public static void main(String[] args) {
    PLAYER.start();
    //Instantiation of the application
    PitchRecognizing2 t = new PitchRecognizing2();
    t.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    lag = 1;
    poseQuestion(lag);
}

} //End of main class H69 PicthRecognizing2

```

70 Exercise. Run the program and play with the code. Test the power of the program with yourself and/or with other person. Agree else disagree with the conclusion of the Author: the idea of grading complexity of the test promises to be a marvelous help for teaching pitch recognizing.

71 Exercise. Develop a program to implement a series of tests about pitch recognizing with a soft increase of complexity of posed challenges. More concretely, let us define the **workingSet** as the set whose notes the User must recognize and the **level of complexity of the current test** as the size of that set. The first working set has only two notes and so it has level two. If the User passes this starting level, he or she enters the third level whose workingSet has a new note. And so on. It is a terrible problem to decide the best way as the adding process must be implemented so, try to make your best. And forget not to devise a rule for students to graduate from a given level to the next. *Answer*

72 Gradualism has a tremendous importance for design. Our strategy to convert the extremely difficult task of training the ear to distinguish pitches into a wonderful game has been to resort to the evolutionary strategy of **gradualism**. It is an strategy of design that consists in dividing a great and complex

target in a chain of nested sub-targets with a soft and solvable increasing degree of complexity. The reader is invited by experimenting on himself to judge how powerful is gradualism to help us in pitch recognizing.

73 The first theory of music and gradualism. From the stand of the listener, the *first theory* of music say that every note is musical and therefore a tune is no more than a succession of notes an that the difference among tunes can be ascribed to the difference in the proportion of notes. The immediate pedagogical implication is that recognizing of pitches is the one and only ingredient of musical pedagogy. It happens that this task can be implemented in a perfectly gradualist fashion, growing complexity in tiny amounts. Our actual proposal, good as it is, shall not be the best one. So, the reader is invited to implement his or her own ideas.

74 Gradualism is the core of the evolutionary theory of the origin of species. The evolutionary theory o the origin of species reads: natural history shows that gathering of small change is the connection among species. In other words, every living being can be connected with the origin of life and beyond through a path joined by small change, by gradualism. Because of gradualism, evolution seems to be an irrefutable fact of nature. Anymore: the evolutionary theory of the origin of species is obviously false: evolution as a biological phenomenon is a crude reality that has no magic and so it can solve some problems rather easily but other rather slowly. Given the extreme complexity of life in each one of its folds and if evolution were the creator of the species, the ensuing prediction would be that as the fossil record as modern populations should be plagued with every sort of imperfections, i.e., of malfunctions and malformations. Since nothing like that is found in nature, the evolutionary theory is obviously false.

75 The elegance of Java. In regard with the previous exercise of devising a program that helps the User to learn pitch recognizing by going through a series of levels of complexity, we have seen that one can do many things with the nails and go on. Nevertheless, for every task Java offers the challenge to also make things with elegance and high quality. In general this is very demanding, so one prefers to restrain oneself to what is strictly necessary. In our case, we need to implement a seamless passing from one level of complexity to the next and so we will dispense with the need of terminating the application to begin anew.

```
//Program H75 PicthRecognizing4
//A training facility
//for pitch recognizing.
```

```
//A soft increase in complexity enables success
//and therefore learning.
//The elegance of Java allows us to
//pass from a very simple test
//to the following level of complexity and so on
//without killing the application to begin a new one.
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

import java.util.Random;

import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.ListSelectionModel;

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneParser;

public class PitchRecognizing4 extends JFrame {

    private static final long serialVersionUID = 1L;

    //This is the set of all notes we use:
    private static final String[] NOTES
        = {"A,", "C", "c",};

    /*
    private static String[] NOTES =
    {"A,", "_B,", "B,", "C", "^C",
```

```

"D", "_E", "E", "F", "^F",
"G", ^G", "A", "_B", "B",
  "c",    ^c", "d", "_e", "e",
  "f", ^f",  "g", ^g",  "a",
  "_b", "b", "c'");

//To present notes to the student in that natural order
//is most possibly not the most efficient way for teaching.
//Our proposal is to try to observe the greatest distance
//among notes, such as in the following ordered LIST.
//The workingSet for level k takes
//the first k elements of this set:

/*
private static String[] notesWorkingSet =
{"C",  "c",  "c'", "G", "g",
 "E", "e",  "A", "A", "a",
 "D",  "d",  "F", "f",  "B",
 "B", "b", ^F", ^f", "_B",
 "_B", "_b",  ^C", ^c",  ^G",
 ^g", "_E",  "_e"};
*/
//In consequence, the next vector keeps the order
//of appearance of notes in the GRADEd levels of complexity:

/*
private static int[] NOTES_ORDER =
{3, 15, 27, 10, 22,
 7, 19, 0, 12, 24,
 5, 17, 8, 20, 2,
 14, 26, 9, 21, 1,
 13, 25, 4, 16, 11,
 23, 6, 18};

*/
private static final int[] NOTES_ORDER
    = {1, 2, 0};

//For level of complexity k, the working set

```

```
//takes the first k notes of notesWorkingSet.
//But the chosen elements must be presented
//in the JList in natural order.
//All this meshing bookkeeping is difficult.
//Our choice is to MARK the chosen elements
//in the set notes to guide ourselves .
private static final boolean[] MARK = new boolean[NOTES.length];

//Each test has a complexity level
//which is the size of the working set.
private static int levelOfComplexity = 2;

//Number of trials granted before failure declaration
private static int tolerance;

//The User must declare which degree of expertise
//he or she wants to play with:
//0 (beginner), 1(intermediate), 2 (advanced), 3 (expert).
private static int expertise;

//To graduate from the current level, you need
//a number of points.
private static int neededPoints;

//To make a point you need to find the pitch
//before tolerance is wasted.
//Actual number of gained points
private static int gainedPoints = 0;

//Every ABC tune must have a correct HEADER
private static final String HEADER
    = "X:0\n"
    + "T:Song 4/4\n"
    + "M:4/4\n"
    + "L:1/8\n"
    + "K:C\n";

//The note that must be recognized as a tune
private static Tune tuneTest;
//The note that must be recognized as a String
```

```

private static String noteTest;

private static final JPanel OUTERPANEL = new JPanel();

//Counter of trials before success
private static int counterOfTrials = 1;
private static String oldNote = "";
//A rest prior the playing of a tune.
private static Integer lag = 8;

//We turn on the inbuilt random generator:
private static final Random R = new Random();

private static final JButton GRADE = new JButton("");
private static final JButton REPEAT = new JButton("Repeat");
private static final JButton SHOWREPORT = new JButton("Show report");
private static String Report = "";
//MODEL is a modifiable LIST
private static final DefaultListModel MODEL = new DefaultListModel();
//the JLIST is built on image of MODEL
private static final JList LIST = new JList(MODEL);
static JFrame f = new JFrame();

//creates a simple midi PLAYER to play the melody
private static final TunePlayer PLAYER = new TunePlayer();

//The musiText is transformed into a tune,
//an ABC4J musical object
private static Tune ABCmusiTextToTune(String musiText) {
    Tune tune = new TuneParser().parse(musiText);
    return tune;
}

//A tune is played
private static void playTune(Tune tune) {
    PLAYER.play(tune);
}

//The note is converted into an ABC tune
private static Tune noteToTune(String note) {

```

```

        String musiText = HEADER + note;
        Tune tune = ABCmusiTextToTune(musiText);
        return tune;
    }

//The number of granted trials before failure declaration
//depends on expertise and the complexity level
    static private int tolerance(int levelOfComplexity,
        int expertise) {
        int l;
        l = 6 - expertise;
        return l;
    }

//A complexity level defines each working set
//of notes to be recognized.
//The starting set has only two notes
    private static void initializeModel() {
        //The first two elements of notesWorkingSet are MARKed
        MARK[NOTES_ORDER[0]] = true;
        MARK[NOTES_ORDER[1]] = true;
        MODEL.add(0, NOTES[NOTES_ORDER[0]]);
        MODEL.add(1, NOTES[NOTES_ORDER[1]]);
    }

    private static void report() {
        /*System.out.println("Position " +
        "of new note = " +
        NOTES_ORDER[levelOfComplexity-1] );
        System.out.println("New noteTest "
        + notes[NOTES_ORDER[
            levelOfComplexity-1]] );*/

        Report = "Expertise level = " + expertise
            + "\nLevel of complexity = "
            + "\nNumber of notes under Test = "
            + MODEL.size()
            + "\nYou gain a point if you find "
            + "\nthe answer before "
            + tolerance + " trials."
    }

```

```

        + "\nNeeded number of points"
        + "\nto pass to the next level = "
        + neededPoints;
    System.out.println(Report);
}

//Old information is erased and
//a new start is set up
private static void reinitializeModel() {
    System.out.println("Reinitialization");
    MODEL.clear();
    for (int i = 0; i < NOTES.length; i++) {
        MARK[i] = false;
    }
    initializeModel();
    lag = 8;
}

//The next level of complexity is generated
//A new note is inserted in the correct place
private static void updateModel() throws Exception {
    //begin a new count
    gainedPoints = 0;
    //Notes remain to be recognized? Yes= continue
    if (levelOfComplexity < NOTES.length) {
        //A new note must be accommodated in the right order
        //Counter of notes less than the new one
        int counter = 0;
        //Position in notes[] of the new note
        int n = NOTES_ORDER[levelOfComplexity];
        //To find the place of insertion,
        //all active notes less than the new one are counted
        for (int i = 0; i < n; i++) {
            if (MARK[i]) {
                counter++;
            }
        }
        //The new note is inserted in the MODEL
        //after all notes less than it.
        MODEL.add(counter, NOTES[n]);
    }
}

```

```

        //The added note is MARKed
        MARK[n] = true;
        //We have a new level of complexity
        levelOfComplexity++;
        System.out.println("CONGRATULATIONS: "
            + "you are one level up!");
    } else //No = Pass to the next level of expertise
    {
        System.out.println("NEW LEVEL OF EXPERTISE");
        expertise++;
        levelOfComplexity = 2;
        reinitializeModel();
    }
    //Rest, silence
    lag = 8;
    //Number of granted trials before failure declaration
    tolerance = tolerance(levelOfComplexity, expertise);
    //Number of points that are needed to graduate
    //from the current level of complexity
    neededPoints = 4 * levelOfComplexity;
    report();
}

//Outputs a random note from the chosen alphabet
private static String randomNote() {
    String newNote;
    int l = MODEL.getSize();
    //A random integer less than the
    //number of allowed notes is thrown
    int i = R.nextInt(l);
    newNote = (String) MODEL.get(i);
    //the new challenge must be different than the prior one
    //because of selected index
    if (newNote.contentEquals(oldNote)) {
        int j = (i + 1) % l;
        newNote = (String) MODEL.get(j);
    }
    oldNote = newNote;
    return newNote;
}

```

```

    }

//A random note to be recognized is played
static private void poseQuestion(Integer lag) {
    String s = lag.toString();
    noteTest = randomNote();
    System.out.println("Random noteTest = " + noteTest);
    tuneTest = noteToTune("z" + s + noteTest + noteTest + noteTest);
    playTune(tuneTest);
}

static private void maketest() {
    LIST.addMouseListener(
        new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                if (expertise <= 5) {
                    if (e.getClickCount() == 1) {
                        int index = LIST.locationToIndex(e.getPoint());
                        String noteAnswer
                            = (String) MODEL.get(index);
                        String notesBoth
                            = noteTest + "2" + noteAnswer + "2";
                        Tune tuneBoth = noteToTune(notesBoth);
                        playTune(tuneBoth); //To avoid double response
system

//If answer is correct
if (noteAnswer.contentEquals(noteTest)) {
    gainedPoints++;
    GRADE.setBackground(Color.GREEN);

//GRADE.setText("");
System.out.println("Your answer " + noteAn.
    + " is RIGHT!" + " Number of tria.
    + counterOfTrials
    + " Gained points = " + gainedPoint
lag = 8;

//Promotion?

```

```

        if (gainedPoints == neededPoints) {
            try {
                updateModel();
            } catch (Exception e1) {
            }
        }

        poseQuestion(lag);
        counterOfTrials = 1;

    } else //if answer is wrong
    {
        System.out.println("Your answer = " + noteAnswer
            + " was WRONG!");
        counterOfTrials++;
        //If tolerance is exceeded, begin anew
        if (counterOfTrials > tolerance) {
            gainedPoints = 0;
        }
        GRADE.setBackground(Color.RED);
        //GRADE.setText("");
    }
}
} //End of if =expertise;
else {
    System.out.println("TEST OVER: CONGRATULATIONS");
    GRADE.setBackground(Color.GREEN);
    REPEAT.setBackground(Color.GREEN);
    SHOWREPORT.setBackground(Color.GREEN);
    JOptionPane.showMessageDialog(null,
        "TEST OVER: CONGRATULATIONS");
}
}
}
);
}

public static void main(String[] args) {
    PLAYER.start();
    //Rest before the first question

```

```

lag = 1;
//Expertise:
//1 (deaf), 2(beginner), 3(intermediate),
//4 (advanced), 5 (expert).
expertise = 5;
//Number of points that are needed to graduate
//from the second initial level of complexity
neededPoints = 4;
//Number of trials granted before failure declaration
tolerance = tolerance(levelOfComplexity, expertise);
OUTERPANEL.setLayout(new BorderLayout(OUTERPANEL, BorderLayout.X_AXIS));

//The JList goes here
JPanel leftPanel = new JPanel();
//Buttons go here
JPanel rightPanel = new JPanel();

leftPanel.setLayout(new BorderLayout());
rightPanel.setLayout(new BorderLayout(rightPanel, BorderLayout.Y_AXIS));

LIST.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
LIST.setBorder(BorderFactory.createEmptyBorder(2, 2, 2, 2));
leftPanel.add(new JScrollPane(LIST));
leftPanel.setBorder(BorderFactory.
    createEmptyBorder(20, 20, 20, 20));
if ((expertise <= 5) & (levelOfComplexity < NOTES.length)
    & (levelOfComplexity >= 2)) {
    //There is no MARKed note at the beginning
    for (int i = 0; i < NOTES.length; i++) {
        MARK[i] = false;
    }
    initializeModel();
    report();
    //Begin training
    poseQuestion(lag);

    maketest();
} else {
    System.out.println("Test over");
}

```

```

//The posed question must be REPEATED
REPEAT.addActionListener((ActionEvent e) \rightarrow {
    tuneTest = noteToTune("z1" + noteTest + noteTest + noteTest);
    playTune(tuneTest);
});

//Information is displayed
SHOWREPORT.addActionListener((ActionEvent e) \rightarrow {
    JOptionPane.showMessageDialog(null, Report);
});

rightPanel.add(GRADE);
rightPanel.add(REPEAT);
rightPanel.add(SHOWREPORT);
rightPanel.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 20));

OUTERPANEL.add(leftPanel);
OUTERPANEL.add(rightPanel);

f.add(OUTERPANEL);

f.setSize(500, 250);
f.setLocationRelativeTo(null);
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
f.setVisible(true);

}
} //End of main class H75 PicthRecognizing4

```

76 Exercise. *Run the program and play with the code and enlarge the set of allowed notes.*

77 Initialization at will. *The previous program devises a test that must be run from start to end in one single session. This is senseless because such a task may take a whole life of hard work. Let us remedy this bug of design: one must be allowed to begin at whatever level of complexity or expertise one desires. The code follows:*

```
//Program H77 PicthRecognizing5
//A training facility
//for pitch recognizing.
//We have five levels of expertise from 1 to 5 and
//25 levels of complexity from 2 to 27.
//That level is determined by the number of NOTES
//that are in the training set.
//A soft increase in complexity enables success
//and therefore learning.
//The elegance of Java allows us to
//pass from a very simple test
//to the following level of complexity and so on
//without killing the application to begin a new one.
//One can choose expertise and complexity level at will
//else go from start to end as if in one session.

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

import java.util.Random;

import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.ListSelectionModel;

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneParser;
```

```

public class PitchRecognizing5 extends JFrame {

    private static final long serialVersionUID = 1L;

    //This is the set of all NOTES we use:
    private static final String[] NOTES
        = {"A,", "_B,", "B,", "C", "^C",
          "D", "_E", "E", "F", ^F",
          "G", ^G", "A", "_B", "B",
          "c", ^c", "d", "_e", "e",
          "f", ^f", "g", ^g", "a",
          "_b", "b", "c'"};

    //To present NOTES to the student in that natural order
    //is most possibly not the most efficient way for teaching.
    //Our proposal is to try to observe the greatest distance
    //among NOTES, such as in the following ordered LIST.
    //The workingSet for level k takes
    //the first k elements of this set
    //(but must be displayed in the natural order):
    /*
private static String[] NOTESWorkingSet =
{"C", "c", "c'", "G", "g",
"E", "e", "A", "A", "a",
"D", "d", "F", "f", "B",
"B", "b", ^F", ^f", "_B",
"_B", "_b", ^C", ^c", ^G",
^g", "_E", "_e"};
    */
    //In consequence, the next vector keeps the order
    //of appearance of NOTES in the GRADED levels of complexity:
    private static final int[] NOTES_ORDER
        = {3, 15, 27, 10, 22,
          7, 19, 0, 12, 24,
          5, 17, 8, 20, 2,
          14, 26, 9, 21, 1,
          13, 25, 4, 16, 11,
          23, 6, 18};

```

```

//For level of complexity k, the working set
//takes the first k NOTES of NOTESWorkingSet.
//But the chosen elements must be presented
//in the JList in natural order.
//All this meshing bookkeeping is difficult.
//Our choice is to MARK the chosen elements
//in the set NOTES to guide ourselves.
private static final boolean[] MARK = new boolean[NOTES.length];

//Each test has a complexity level
//which is the size of the working set.
private static int levelOfComplexity = 2;

//Number of trials granted before failure declaration
private static int tolerance;

//The User must declare which degree of expertise
//he or she wants to play with:
//0 (beginner), 1(intermediate), 2 (advanced), 3 (expert).
private static int expertise;

//To graduate from the current level, you need
//a number of points.
private static int neededPoints;

//To make a point you need to find the pitch
//before tolerance is wasted.
//Actual number of gained points
private static int gainedPoints = 0;

//Every ABC tune must have a correct HEADER
private static final String HEADER
    = "X:0\n"
    + "T:Song 4/4\n"
    + "M:4/4\n"
    + "L:1/8\n"
    + "K:C\n";

//The note that must be recognized as a tune
private static Tune tuneTest;

```

```
//The note that must be recognized as a String
private static String noteTest;

private static final JPanel OUTER_PANEL = new JPanel();

//Counter of trials before success
private static int counterOfTrials = 1;
private static String oldNote = "";
//A rest prior the playing of a tune.
private static Integer lag = 8;

//We turn on the inbuilt random generator:
private static final Random R = new Random();

private static final JButton GRADE = new JButton("");
private static final JButton REPEAT = new JButton("Repeat");
private static final JButton SHOW_REPORT
    = new JButton("Show report");
private static final JButton CHOOSE_EXPERTISE
    = new JButton("Choose expertise");
private static final JButton CHOOSE_COMPLEXITY
    = new JButton("Choose complexity");
private static String report = "";

//MODEL is a modifiable LIST
private static final DefaultListModel MODEL
    = new DefaultListModel();
//the JLIST is built on image of MODEL
private static final JList LIST = new JList(MODEL);
static JFrame f = new JFrame();

public PitchRecognizing5() {
    OUTER_PANEL.setLayout(new BorderLayout(OUTER_PANEL,
        BorderLayout.X_AXIS));
    //The JList goes here
    JPanel leftPanel = new JPanel();
    //Buttons go here
    JPanel rightPanel = new JPanel();

    leftPanel.setLayout(new BorderLayout());
```

```

rightPanel.setLayout(new BorderLayout(rightPanel,
    BorderLayout.Y_AXIS));

LIST.setSelectionMode(
    ListSelectionMode.SINGLE_SELECTION);
LIST.setBorder(
    BorderFactory.createEmptyBorder(2, 2, 2, 2));
leftPanel.add(new JScrollPane(LIST));
leftPanel.setBorder(BorderFactory.
    createEmptyBorder(20, 20, 20, 20));
if ((expertise <= 5)
    & (levelOfComplexity < NOTES.length)
    & (levelOfComplexity >= 2)) {
    //There is no MARKed note at the beginning
    for (int i = 0; i < NOTES.length; i++) {
        MARK[i] = false;
    }
    initializeModel(expertise,
        levelOfComplexity);
    //Begin training
    poseQuestion(lag);

    maketest();
} else {
    System.out.println("Test over");
}

//Button REPEAT
//The posed question must be REPEATED
REPEAT.addActionListener((ActionEvent e) \rightarrow {
    tuneTest = noteToTune("z1"
        + noteTest + noteTest + noteTest);
    playTune(tuneTest);
});

//Information is displayed
SHOW_REPORT.addActionListener((ActionEvent e) \rightarrow {
    JOptionPane.showMessageDialog(
        null, report(expertise, levelOfComplexity));
});

```

```
//Information is displayed
CHOOSE_EXPERTISE.addActionListener((ActionEvent e) \rightarrow {
    String s1 = JOptionPane.showInputDialog(
        "Enter your level of expertise, "
        + "1 (least) ,2,3,4,5 (best)", 0);
    String s = s1.trim();
    //translation from char to number
    expertise = s.charAt(0) - 48;
    System.out.println("Expertise = " + expertise);
});

//Information is displayed
CHOOSE_COMPLEXITY.addActionListener((ActionEvent e) \rightarrow {
    String s1 = JOptionPane.showInputDialog(
        "Enter desired level of complexity, "
        + " from 2 (least) to " + NOTES.length
        + "(max)", 2);
    String s = s1.trim();
    int a = s.charAt(0) - 48;
    levelOfComplexity = a;
    if (s.length() > 1) {
        int b = s.charAt(1) - 48;
        levelOfComplexity = a * 10 + b;
    }
    System.out.println("Complexity = "
        + levelOfComplexity);
    initializeModel(expertise, levelOfComplexity);
    poseQuestion(levelOfComplexity);
});

rightPanel.add(GRADE);
rightPanel.add(REPEAT);
rightPanel.add(SHOW_REPORT);
rightPanel.add(CHOOSE_EXPERTISE);
rightPanel.add(CHOOSE_COMPLEXITY);
rightPanel.setBorder(
    BorderLayout.createEmptyBorder(0, 0, 0, 20));

OUTER_PANEL.add(leftPanel);
```

```

    OUTER_PANEL.add(rightPanel);

    f.add(OUTER_PANEL);

    f.setSize(500, 250);
    f.setLocationRelativeTo(null);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.setVisible(true);

}

//creates a simple midi PLAYER to play the melody
    private static final TunePlayer PLAYER = new TunePlayer();

//The musiText is transformed into a tune,
//an ABC4J musical object
    private static Tune ABCmusiTextToTune(String musiText) {
        Tune tune = new TuneParser().parse(musiText);
        return tune;
    }

//A tune is played
    private static void playTune(Tune tune) {
        PLAYER.play(tune);
    }

//The note is converted into an ABC tune
    private static Tune noteToTune(String note) {
        String musiText = HEADER + note;
        Tune tune = ABCmusiTextToTune(musiText);
        return tune;
    }

//The number of granted trials before failure declaration
//depends on expertise and the complexity level
    static private int tolerance(int levelOfComplexity,
        int expertise) {
        int l;
        l = 6 - expertise;
        return l;
    }

```

```

    }

//A complexity level defines each working set
//of NOTES to be recognized.
//The starting set has only two NOTES
    private static void initializeModel() {
        //The first two elements of NOTESWorkingSet are MARKed
        MARK[NOTES_ORDER[0]] = true;
        MARK[NOTES_ORDER[1]] = true;
        MODEL.add(0, NOTES[NOTES_ORDER[0]]);
        MODEL.add(1, NOTES[NOTES_ORDER[1]]);
    }

//A complexity level defines each working set
//of NOTES to be recognized.
//The starting set has only two NOTES
    private static void initializeModel(int expertise,
        int levelOfComplexity) {
        reinitializeModel();
        //The corresponding elements of NOTESWorkingSet are MARKed
        for (int i = 2; i < levelOfComplexity; i++) {
            insertNote(i);
        }
    }

    private static String report(int expertise,
        int levelOfComplexity) {
        /*System.out.println("Position " +
"of new note = " +
NOTES_ORDER[levelOfComplexity-1] );
System.out.println("New noteTest "
+ NOTES[NOTES_ORDER[
    levelOfComplexity-1]] );*/

        report = "Expertise level = " + expertise
            + "\nLevel of complexity = "
            + "\nNumber of NOTES under Test = "
            + levelOfComplexity
            + "\nYou gain a point if you find "
            + "\nthe answer before "

```

```

        + tolerance + " trials."
        + "\nNeeded number of points"
        + "\nto pass to the next level = "
        + neededPoints;
    System.out.println(report);
    return report;
}

//Old information is erased and
//a new start is set up
private static void reinitializeModel() {
    System.out.println("Reinitialization");
    MODEL.clear();
    for (int i = 0; i < NOTES.length; i++) {
        MARK[i] = false;
    }
    initializeModel();
    lag = 8;
}

private static void insertNote(int k) {
    //A new note must be accommodated in the right order
    //Counter of NOTES less than the new one
    int counter = 0;
    //Position in NOTES[] of the new note
    int n = NOTES_ORDER[k];
    //To find the place of insertion,
    //all active NOTES less than the new one are counted
    for (int i = 0; i < n; i++) {
        if (MARK[i]) {
            counter++;
        }
    }
    //The new note is inserted in the MODEL
    //after all NOTES less than it.
    MODEL.add(counter, NOTES[n]);
    //The added note is MARKed
    MARK[n] = true;
}

```

```
//The next level of complexity is generated
//A new note is inserted in the correct place
private static void updateModel() throws Exception {
    //begin a new count
    gainedPoints = 0;
    //Notes remain to be recognized? Yes= continue
    if (levelOfComplexity < NOTES.length) {
        insertNote(levelOfComplexity);
        //We have a new level of complexity
        levelOfComplexity++;
        System.out.println("CONGRATULATIONS: "
            + "you are one level up!");
    } else //No = Pass to the next level of expertise
    {
        System.out.println("NEW LEVEL OF EXPERTISE");
        expertise++;
        levelOfComplexity = 2;
        reinitializeModel();
    }
    //Rest, silence
    lag = 8;
    //Number of granted trials before failure declaration
    tolerance = tolerance(levelOfComplexity,
        expertise);
    //Number of points that are needed to graduate
    //from the current level of complexity
    neededPoints = 4 * levelOfComplexity;
    report(expertise, levelOfComplexity);
}

//Outputs a random note from the chosen alphabet
private static String randomNote() {
    String newNote;
    int l = MODEL.getSize();
    //A random integer less than the
    //number of allowed NOTES is thrown
    int i = R.nextInt(l);
    newNote = (String) MODEL.get(i);
    //the new challenge must be different than the prior one
    //because of selected index
```

```

        if (newNote.contentEquals(oldNote)) {
            int j = (i + 1) % 1;
            newNote = (String) MODEL.get(j);
        }
        oldNote = newNote;
        return newNote;
    }

//A random note to be recognized is played
static void poseQuestion(Integer lag) {
    String s = lag.toString();
    noteTest = randomNote();
    System.out.println("Random noteTest = "
        + noteTest);
    tuneTest = noteToTune("z" + s + noteTest + noteTest + noteTest);
    playTune(tuneTest);
}

static private void maketest() {
    LIST.addMouseListener(
        new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                if (expertise <= 5) {
                    if (e.getClickCount() == 1) {
                        int index = LIST.locationToIndex(e.getPoint());
                        String noteAnswer
                            = (String) MODEL.get(index);
                        String NOTESBoth
                            = noteTest + "2" + noteAnswer + "2";
                        Tune tuneBoth = noteToTune(NOTESBoth);
                        playTune(tuneBoth);

                        //If answer is correct
                        if (noteAnswer.contentEquals(noteTest)) {
                            gainedPoints++;
                            GRADE.setBackground(Color.GREEN);

                            //GRADE.setText("");
                            System.out.println("Your answer " + noteAn

```

```

        + " is RIGHT!" + " Number of trials = "
        + counterOfTrials
        + " Gained points = " + gainedPoints);
    lag = 8;

    //Promotion?
    if (gainedPoints == neededPoints) {
        try {
            updateModel();
        } catch (Exception e1) {
        }
    }

    poseQuestion(lag);
    counterOfTrials = 1;

} else //if answer is wrong
{
    System.out.println("Your answer = "
        + noteAnswer + " was WRONG!");
    counterOfTrials++;
    //If tolerance is exceeded, begin anew
    if (counterOfTrials > tolerance) {
        gainedPoints = 0;
    }
    GRADE.setBackground(Color.RED);
    //GRADE.setText("");
}
}
} //End of if =expertise;
else {
    System.out.println("TEST OVER: CONGRATULATIONS");
    GRADE.setBackground(Color.GREEN);
    REPEAT.setBackground(Color.GREEN);
    SHOW_REPORT.setBackground(Color.GREEN);
    JOptionPane.showMessageDialog(null,
        "TEST OVER: CONGRATULATIONS");
}
}
}
}

```

```

    );
}

public static void trigger() {
    PLAYER.start();
    //Rest before the first question
    lag = 1;
    //Expertise:
    //1 (deaf), 2 (beginner), 3 (intermediate),
    //4 (advanced), 5 (expert).
    expertise = 5;
    //Number of points that are needed to graduate
    //from the second initial level of complexity
    neededPoints = 4;
    //Number of trials granted before failure declaration
    tolerance = tolerance(levelOfComplexity, expertise);
    //Instantiation of application
    PitchRecognizing5 x = new PitchRecognizing5();
    x.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    poseQuestion(lag);
}

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

} //End of main class H77 PicthRecognizing5

```

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

79 Challenge. *The previous program is very poor in feedback. Try to remedy this deficit.*

80 Intrigue. *In spite of personal and cultural biases, there are many musical pieces that everybody likes and that one classify as nice, elegant, beautiful. Let us turn this fact into biology: Can you say that animals, say, a horse, a lion, a cock, a mouse or a bath are elegant? Can you justify you answer from the stand of the evolutionary theory? On the other hand, can you say that the genome that*

encodes for those animals is elegant, i.e. that follows a very elegant style of software development? Can you justify your answer from the stand of the evolutionary theory?

81 One more challenge. *Our final aim is not music: it is evolution, which can also be found in every learning process and particularly in that of pitch recognizing. One can imagine it as an evolutionary process that builds a neural circuitry for the recognizing of pitches and that enables the carrier person to classify as expert. Possibly, the process builds in the brain of the student a population of neural circuits together with their programming that must be subjected to mutation and recombination to compete for performance. From the stand of evolution the important point for us is to understand and quantify the difficulty of the learning task. We expect the User to say something like: the number of trials that I needed was an exponential function in the degree of complexity. So, we need to keep a record of the needed number of trials to solve each and every question, together with the level of complexity of the question. This implies management of the hard disk.*

4.5 The statistical version of the first theory

The **first theory** of music is yet very primitive. At the beginning we considered pitches only. Then, we added duration. At this stage, we considered that any string of notes with different duration sounds musical. To test this theory we developed a program to synthesize precisely that type of strings. We discovered that most strings sound ugly but that some strokes inside long strings may be pleasant. From this, a rhythm machine was tested to appreciate the beauty of short repetitive strings and in effect, a not nil proportions of random rhythms were found to be pleasant. This is perceived as progress and warms our hearts along this journey but we know that we are rather crude. So, is there an easy correction to be considered?

82 The statistical version of the first theory.

Let us present the **statistical first theory** as an immediate correction to the first theory: the beauty of music resides in the frequencies as notes (pitch + duration) are used in composition. According to this theory, music is determined by the frequencies as notes appear in the tune regardless of any other condition, say, the order as notes appear along the tune, or their intensity.

4.6 Musical analysis

The purpose of **musical analysis** is to study wonderful musical pieces to try to produce more of the same. The statistical theory of music readily lends itself to musical analysis: since we enjoy an inheritance with many wonderful musical compositions, all we must do is to register the frequencies of notes of a nice tune to next feed the random generator with the corresponding frequency table. Alternatively, one divides the tune in tokens, notes + duration, and permutes it to obtain a new tune with the very same distribution of notes.

83 Exercise. *Develop a program that implements the musical analysis induced by the **statistical first theory** of music. The input shall be an ABC tune, with nothing else apart from notes and duration. The output shall be a tune with the same statistical distribution of the frequency of notes as a model one. Since we need to feed the program with a complex input, we prefer to do that through a GUI, so our solution to this exercise will be included below as an updating to our GUI. We implement the permutation method.*

84 Challenge. *Instead of permutations, feed the random generator with the frequency table of your preferred songs. If you take this challenge, you must learn to feed the random generator with a frequency table. This is explained in vol V version 2 of this series, *The scientific method with Java*, Sec 8.7 pg 141.*

85 Exercise. *Use the GUI to estimate the power of our statistical theory to compose music. Play enough to agree else disagree with the conclusion of the Author: this theory is perfectly useless to compose music. If you agree, would you conclude that the correct extrapolation to biology is that blind reordering of any genome can only lead to exactly the same form of life (with insignificant changes) else to death? *Answer**

4.7 Updating our GUI

Let us update our GUI with the programs developed in this chapter together with the promised facility to keep record of user's performance in pitch recognizing + a permutator of musitexts to serve as a lab for the **statistical first theory of music**.

86 The programs developed for this chapters are added to our GUI. *We include a new menu: StatMusic with the items: NotesEqualDuration, NotesUnequalDuration, ProteinToTune, DNAToTune, PitchRecognizing, MusicalAnalysis. To execute either of ProteinToTune or DNAToTune a protein or DNA sequence must have been*

previously pasted into the textArea or editor window. The possibility to keep a personal record of user's performance in pitch recognizing is allowed. This, of course, demands hard disk management and the corresponding file menu.

```
//Program H86 FirstTheoryGUI
//A note consists of pitch + duration.
//First theory of music:
//All notes are pleasant separately and
//so are when they form strings.
//Predictions:
//1) Notes concatenated at random are pleasant.
//2) Every string is musical, DNA included.
//We test the theory through generation of
//tunes produced by random notes else by DNA
// or a protein sequence that
//must be pasted in the text area.
//
//For random notes we use the inbuilt Java generator
//of random numbers.
//
//The program also can play an ABC tune
//and show its score.
//The score can be rudimentarily edited.
//
//The program includes a training facility
//for pitch recognizing and for
//keeping record of user's performance.
//
//A facility is also provided to test
//the Statistical First Theory of music:
//what distinguishes a nice tune from ugly ones
//is the frequencies as notes are played
//and nothing else.
//
//A soft introduction to GUI's was done in
//Chapter I of Vol III of the series
//Java for the study of evolution,
//www.evoljava.com
```

```
//Here we use Swing to devise our GUI.
//This platform is mature and in current use by 2018.
//But a modern platform is also available since 2015:
//JavaFX that can be learned from Vol XII Fossils
//in the Area of Morphogenesis.
import java.awt.event.*;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.text.DateFormat;
import java.util.Random;

import javax.swing.*;

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneBook;
import abc.parser.TuneParser;
import abc.ui.swing.JScoreComponent;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.ListSelectionModel;
```

```

import javax.swing.WindowConstants;

public final class FirstTheoryGUI extends JFrame {

    private static final long serialVersionUID = 1L;

    //We turn on the inbuilt random generator:
    static private Random r = new Random();

    //Our one symbol notation for notes :
    //A, _B, B, C #C D _E E F #F G #G
    //R S T C M D N E F O G P

    //A _B B c #c d _e e f #f g #g a _b b
    //A Q B c m d n e f o g p a q b

    //All possible notes we use in one letter notation
    /*static private String superAlphabet =
        "RSTAMBCNDOEFPGQambcndoeftpq";*/
    static private String alphabet = "RTCDEOGabcdeo";

    static private String durationsAlphabet = "1111222222448";

    //=====Variables for PitchRecognizing=====
    //*****
    //***** Definition and default initialization
    //***** of an instantiation of MusiText
    //*****
    static private final FirstTheoryGUI A = new FirstTheoryGUI();
    static private final MusiText3 N = A.new MusiText3();

    protected JTextArea myTextArea;

    protected JToolBar myToolBar;

    protected static JFileChooser myChooser;

    static private File myFile = new File("");

    //creates a simple midi PLAYER to play the melody

```

```

static private final TunePlayer PLAYER = new TunePlayer();

static private boolean print = false;

//Initialization
String musiText2
    = "X:0\n"
    + "T:Song \n"
    + "M:4/4\n"
    + "L:1/4"
    + "K:C\n"
    + "dd2d |ded2 |GB3 | "
    + "BBBA | A3  A | dAB2 |"
    + " AAAA | B2A A | G4 | "
    + "Gded- |d edG | B4 | "
    + "BBA2 | A  A3 | BAG2|"
    + "GGGA-|A2 B2 | A4 | G4";

String Help = "READ ME  \n "
    + "-. THIS IS THE TEXT AREA. CLEAR IT before any "
    + "operation: \n "
    + "  Click on it and Use control + A + X \n"
    + "-. TO OPEN AN ABC file with a tune: \n"
    + "  File \rightarrow Open...\n"
    + "-. TO SAVE AN ABC tune:\n"
    + "  The suffix of ABC tunes is  .abc  \n"
    + "  File \rightarrow Save...\n"
    + "  TO PLAY A VERY SIMPLE ABC TUNE:\n"
    + "-. Paste THE ABC tune into the editor window and\n "
    + "  ABCMusic \rightarrow ShowAndPlayTune\n"
    + "-. TO PLAY A PROTEIN SEQUENCE in one letter notation:\n"
    + "  Paste the sequence to the text area and\n"
    + "  FirstTheory \rightarrow ProteinToTune\n "
    + "-. TO PLAY A DNA SEQUENCE:\n"
    + "  Paste the sequence to the text area and\n"
    + "  FirstTheory \rightarrow DNAToTune\n "
    + "-. TO TRAIN YOUR EAR WITH PITCH RECOGNIZING: \n"
    + "  FirstTheory \rightarrow PitchRecognizing: \n"
    + "  CHANGE continuously CLICKING FINGER AND HAND. \n"
    + "  keep in mind that learning means an \n"

```

```

+ "    evolutionary process in the brain to build and \n"
+ "    program neural networks to respond adequately. \n"
+ "-. TO KEEP A RECORD of your performance \n"
+ "    in Pitch recognizing and to study how complexity\n"
+ "    is battled by your brain, \n"
+ "    which is powered by evolution: \n"
+ "    a. Open a new account with your name and \n"
+ "        use the suffix    .ear    and \n"
+ "        In the emerging window choose: New account \n"
+ "    b. Reopen your account for every session:\n"
+ "        In the emerging window choose: \n    "
+ "Continue training.\n "
+ "    c. To end session and update your account:\n"
+ "        In the emerging window choose: \n    "
+ "save account & Close. \n "
+ "    d. As you progress, you rise up from low \n    "
+ "        complexity, ony two notes, to higher complexity\n"
+ "        up to 27 notes. \n    "
+ "        Gain points and go up. \n"
+ "        For a given level of complexity you have \n    "
+ "        8 degrees    of expertise, from 0 to 7. \n    "
+ "        More expertise means less \n    "
+ "        tolerance, less opportunities to commit errors. \n"
+ "        Concentration is determinant. \n    "
+ "    e. If you need more training, \n"
+ "        you have two options: \n"
+ "        to return to a lower level of complexity or \n"
+ "        to augment persistence of the test."
+ "-. Use control + A to SELECT ALL,\n "
+ "-. Use control + V to PASTE the content\n "
+ "    of the clipboard.\n "
+ "-. TO STOP THE PLAYER: Panic \rightarrow Panic \n"
+ "-. ACCELERATORS: \n"
+ "    Alt + the underscored letter of the menu";

    static private boolean equalDuration;
//Constructor

    public FirstTheoryGUI() {
//Title in the menu bar

```

```

        super("A GUI for stat music");
        super.setSize(450, 350);

//Text area = editing window
        myTextArea = new JTextArea();
        JScrollPane ps = new JScrollPane(myTextArea);
        super.getContentPane().add(ps, BorderLayout.CENTER);
        myTextArea.append(Help);

        JMenuBar menuBar = createMenuBar();
        setJMenuBar(menuBar);

        WindowListener wndCloser = new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        };

        addWindowListener(wndCloser);

        setVisible(true);
    } //End constructor

//=====
//=====MENU BAR=====
//=====
//The menu bar and LISTeners
    protected JMenuBar createMenuBar() {
        final JMenuBar menuBar = new JMenuBar();

        //File menu
        JMenu mFile = new JMenu("File");
        mFile.setMnemonic('f');

        //New file
        ImageIcon iconClearTextArea = new ImageIcon("file_ClearTextAre

        Action actionClearTextArea = new AbstractAction("ClearTextArea
        iconClearTextArea) {

```

```

        private static final long serialVersionUID = 1L;

        //what shall be done if the new menu is clicked on
        @Override
        public void actionPerformed(ActionEvent e) {
            myTextArea.setText("");
        }
    };

//item is a variable for diverse itemMenus
JMenuItem clearTextArea = mFile.add(actionClearTextArea);
mFile.add(clearTextArea);

myChooser = new JFileChooser();
myChooser.setCurrentDirectory(new File("."));

ImageIcon iconOpen = new ImageIcon("file_open.gif");
Action actionOpen = new AbstractAction("Open...", iconOpen) {

    private static final long serialVersionUID = 1L;

    @Override
    public void actionPerformed(ActionEvent e) {
        readFileIntoTextArea();
    }

};

JMenuItem open = mFile.add(actionOpen);
mFile.add(open);

ImageIcon iconSave = new ImageIcon("file_save.gif");
Action actionSave = new AbstractAction("Save...", iconSave) {
    private static final long serialVersionUID = 1L;

    @Override
    public void actionPerformed(ActionEvent e) {
        saveTextAreaToFile();
    }
};

```

```
JMenuItem save = mFile.addActionSave();
mFile.addAction(save);

//Exit menu
Action actionExit = new AbstractAction("Exit") {
    private static final long serialVersionUID = 1L;
    //what shall be done if the exit menu is clicked on

    @Override
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
};

JMenuItem exit = mFile.addActionExit();
exit.setMnemonic('x');
menuBar.addAction(mFile);

myToolBar = new JToolBar();

//=====
//ABCmusic menu
JMenu mABCmusic = new JMenu("ABCmusic");
mABCmusic.setMnemonic('a');
menuBar.addAction(mABCmusic);

//Show text menu
ImageIcon iconPrintTextArea
    = new ImageIcon("ABCmusic_PrintTextArea.gif");
Action actionPrintTextArea
    = new AbstractAction("PrintTextArea",
        iconPrintTextArea) {
    private static final long serialVersionUID = 1L;

    @Override
    public void actionPerformed(ActionEvent e) {
        String ss = myTextArea.getText();
        System.out.println(ss);
    }
}
```

```

};

JMenuItem PrintTextArea = mABCmusic.add(actionPrintTextArea);
mABCmusic.add(PrintTextArea);

//=== Show tune's score menu
ImageIcon iconShowTune
    = new ImageIcon("ABCmusic_ShowTune.gif");
Action actionShowTune
    = new AbstractAction("ShowTune", iconShowTune) {
    private static final long serialVersionUID = 1L;

    @Override
    public void actionPerformed(ActionEvent e) {
        String ss = myTextArea.getText();
        System.out.println(ss);
        //Get header, score and lyrics
        getComponentsABC(ss);
        Tune tune;
        tune = N.ABCmusiTextToTune(headerABC + "\n" + scoreABC);
        N.DrawTune(tune);
    }
};

JMenuItem showTune = mABCmusic.add(actionShowTune);
mABCmusic.add(showTune);

ImageIcon iconPlayTune
    = new ImageIcon("ABCmusic_PlayTune.gif");
Action actionPlayTune
    = new AbstractAction("PlayTune", iconPlayTune) {
    private static final long serialVersionUID = 1L;

    @Override
    public void actionPerformed(ActionEvent e) {
        String ss = myTextArea.getText();
        System.out.println(ss);
        //Get header, score and lyrics
        getComponentsABC(ss);
        Tune tune;

```

```

        tune = N.ABCmusiTextToTune(headerABC + "\n" + scoreABC);
        N.playTune(tune);
    }
};

JMenuItem play = mABCmusic.add(actionPlayTune);
mABCmusic.add(play);

//==Show score and play tune-menu
ImageIcon iconShowAndPLayTune
    = new ImageIcon("ShowPLay_ShowAndPLayTune.gif");
Action actionShowAndPLayTune
    = new AbstractAction("ShowAndPLayTune", iconShowAndPLayTune);
private static final long serialVersionUID = 1L;

@Override
public void actionPerformed(ActionEvent e) {
    String ss = myTextArea.getText();
    System.out.println(ss);
    //Get header, score and lyrics
    getComponentsABC(ss);
    Tune tune;
    tune = N.ABCmusiTextToTune(headerABC + "\n" + scoreABC);
    N.DrawTune(tune);
    N.playTune(tune);
}
};

JMenuItem ShowAndPLay
    = mABCmusic.add(actionShowAndPLayTune);
mABCmusic.add(ShowAndPLay);

//=====
//ShowPLay menu
JMenu mFirstTheory = new JMenu("FirstTheory");
mFirstTheory.setMnemonic('d');
menuBar.add(mFirstTheory);

//=====
//NotesEqualDuration menu

```

```

    ImageIcon iconNotesEqualDuration
        = new ImageIcon("NotesEqualDuration_ShowText.gif");
    Action actionNotesEqualDuration
        = new AbstractAction("NotesEqualDuration",
            iconNotesEqualDuration) {
        private static final long serialVersionUID = 1L;

        @Override
        public void actionPerformed(ActionEvent e) {
            equalDuration = true;
            try {
                N.mainRandomMusic(equalDuration);
            } catch (IOException ex) {

                Logger.getLogger(FirstTheoryGUI.class.getName()).log(Level.SEVERE, null, ex);
            }

        }
    };

    JMenuItem NotesEqualDuration
        = mFirstTheory.add(actionNotesEqualDuration);
    mFirstTheory.add(NotesEqualDuration);

//=====
//NotesEqualDuration menu
    ImageIcon iconNotesUnequalDuration
        = new ImageIcon("NotesUnequalDuration_ShowText.gif");
    Action actionNotesUnequalDuration
        = new AbstractAction("NotesUnequalDuration",
            iconNotesUnequalDuration) {
        private static final long serialVersionUID = 1L;

        @Override
        public void actionPerformed(ActionEvent e) {
            equalDuration = false;
            try {
                N.mainRandomMusic(equalDuration);
            } catch (IOException ex) {

```

```

Logger.getLogger(FirstTheoryGUI.class.getName()).log(Level.SEVERE, null,
    }
}
};

JMenuItem NotesUnequalDuration
    = mFirstTheory.addActionNotesUnequalDuration();
mFirstTheory.addAction(NotesUnequalDuration);

//=====
//ProteinToTune menu
ImageIcon iconProteinToTune
    = new ImageIcon("ProteinToTune_ShowText.gif");
Action actionProteinToTune
    = new AbstractAction("ProteinToTune", iconProteinToTune);
private static final long serialVersionUID = 1L;

@Override
public void actionPerformed(ActionEvent e) {
    String protein = myTextArea.getText();
    try {
        N.mainProt(protein);
    } catch (IOException ex) {}

Logger.getLogger(FirstTheoryGUI.class.getName()).log(Level.SEVERE, null,
    }
}
};

JMenuItem ProteinToTune
    = mFirstTheory.addActionProteinToTune();
mFirstTheory.addAction(ProteinToTune);

//=====
//DNAToTune menu
ImageIcon iconDNAToTune
    = new ImageIcon("DNAToTune_ShowText.gif");
Action actionDNAToTune

```

```

        = new AbstractAction("DNAToTune", iconDNAToTune) {
    private static final long serialVersionUID = 1L;

    @Override
    public void actionPerformed(ActionEvent e) {

        String dna = myTextArea.getText();
        try {
            N.mainDNA(dna);
        } catch (IOException ex) {

Logger.getLogger(FirstTheoryGUI.class.getName()).log(Level.SEVERE, null, ex);
        }

    }
};

JMenuItem DNAToTune = mFirstTheory.add(actionDNAToTune);
mFirstTheory.add(DNAToTune);

//=====
//Pitch recognizing
ImageIcon iconPitchRecognizing
    = new ImageIcon("PitchRecognizing_ShowText.gif");
Action actionPitchRecognizing
    = new AbstractAction("PitchRecognizing",
        iconPitchRecognizing) {
    private static final long serialVersionUID = 1L;

    @Override
    public void actionPerformed(ActionEvent e) {
        //Instantiation of application
        PitchRecognizing6 p = new PitchRecognizing6();
        p.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
        //Rest before the first question
        int lag = 1;
        PitchRecognizing6.poseQuestion(lag);
    }
};

```

```

JMenuItem PitchRecognizing
    = mFirstTheory.add(actionPitchRecognizing);
mFirstTheory.add(PitchRecognizing);

//=====
//Statistical version of the first theory
ImageIcon iconStatVersion
    = new ImageIcon("StatVersion_ShowText.gif");
Action actionStatVersion
    = new AbstractAction("StatVersion",
        iconStatVersion) {
    private static final long serialVersionUID = 1L;

    @Override
    public void actionPerformed(ActionEvent e) {
        String ss = myTextArea.getText();
        System.out.println(ss);
        StatVersion(ss);
    }
};

JMenuItem StatVersion
    = mFirstTheory.add(actionStatVersion);
mFirstTheory.add(StatVersion);

//=====
//=====Panic menu
JMenu mPanic = new JMenu("Panic");
mPanic.setMnemonic('p');
menuBar.add(mPanic);

ImageIcon iconPanic
    = new ImageIcon("ShowPLay_Panic.gif");
Action actionPanic
    = new AbstractAction("Panic", iconPanic) {
    private static final long serialVersionUID = 1L;

    @Override
    public void actionPerformed(ActionEvent e) {
        PLAYER.stopPlaying();
    }
};

```

```

        }
    };

    JMenuItem Panic = mPanic.add(actionPanic);
    mPanic.add(Panic);
//=====

//The toolbar is added to the container
    getContentPane().add(myToolBar, BorderLayout.NORTH);

    return menuBar;
} //End of menu bar construction

private void saveTextAreaToFile() {
    FirstTheoryGUI.this.repaint();
    if (myChooser.showSaveDialog(FirstTheoryGUI.this)
        != JFileChooser.APPROVE_OPTION) {
        return;
    }
    Thread runner = new Thread() {
        @Override
        public void run() {
            File fChosen = myChooser.getSelectedFile();

            try (FileWriter out = new FileWriter(fChosen)) {
                myTextArea.write(out);
            } catch (IOException ex) {

                Logger.getLogger(FirstTheoryGUI.class.getName()).log(Level.SEVERE, null, ex);
            }

            myFile = fChosen;
        }
    };
    runner.start();
}

private String readFileIntoTextArea() {
    FirstTheoryGUI.this.repaint();
    if (myChooser.showOpenDialog(FirstTheoryGUI.this)

```

```

        != JFileChooser.APPROVE_OPTION) {
            return "";
        }
        Thread runner = new Thread() {
            @Override
            public void run() {
                File fChosen = myChooser.getSelectedFile();

                try (FileReader in = new FileReader(fChosen)) {
                    myTextArea.read(in, null);
                } catch (FileNotFoundException ex) {

Logger.getLogger(FirstTheoryGUI.class.getName()).log(Level.SEVERE, null,
                ex);

                Logger.getLogger(FirstTheoryGUI.class.getName()).log(Level.SEVERE, null,
                ex);

                myFile = fChosen;
            }

        };
        runner.start();
        return myTextArea.getText();
    }

//*****
//The MusiText class is inserted as an inner class
//*****
    public class MusiText3 {

        String musiText;

        //Constructor
        //This specifies how a musiText is initialized.
        //Here, it acquires information from a String
        MusiText3(String a) {
            musiText = a;
        }
    }

```

```

//One can initialize it by default
MusiText3() {
    musiText
        = "X:0\n"
        + "T:The scale \n"
        + "M:4/4\n"
        + "K:C\n"
        + "CDEFGABc4 cBAGF - EDC8";
}

//=====
//Randomness is made into a composer
String randomNotes(boolean equalDuration) {
    //Number of notes
    int n = 100;
    //Output initialization
    String stringOfNotes = "";
    for (int i = 0; i < n; i++) {
        //A note is generated:
        //A random integer less than the
        //number of chars in the alphabet is thrown
        int k = r.nextInt(alphabet.length());
        //The integer is changed into a note
        char c = alphabet.charAt(k);
        //The note is added to the output
        stringOfNotes = stringOfNotes + c;
        if (equalDuration); else { //Duration is generated
            k = r.nextInt(durationsAlphabet.length());
            //The integer is changed into a duration
            c = durationsAlphabet.charAt(k);
            //The note is added to the output
            stringOfNotes = stringOfNotes + c;
        }
    }
    return stringOfNotes;
}

//A string in our alphabet is rewritten in ABC notation
//Accidents are prefixes in ABC notation.
//Sharp is denoted by ^, flat by _

```

```

private String interpreter(String ss) {
    ss = ss.replaceAll("R", "A,");
    ss = ss.replaceAll("S", "_B,");
    ss = ss.replaceAll("T", "B,");
    ss = ss.replaceAll("M", "#C");
    ss = ss.replaceAll("M", "#c");
    ss = ss.replaceAll("N", "_E");
    ss = ss.replaceAll("n", "_e");
    ss = ss.replaceAll("O", "^F");
    ss = ss.replaceAll("o", "^f");
    ss = ss.replaceAll("P", "^G");
    ss = ss.replaceAll("p", "^g");
    ss = ss.replaceAll("Q", "_B");
    ss = ss.replaceAll("q", "_b");
    return ss;
}

void mainRandomMusic(boolean equalDuration)
    throws IOException {
    //Initialization
    String header
        = "X:0\n"
        + "T:Song \n"
        + "M:4/4\n"
        + "L:1/8\n"
        + "K:C\n";
    String ss = randomNotes(equalDuration);
    ss = interpreter(ss);
    String musiText2 = header + ss;

    Tune tune;
    tune = ABCmusiTextToTune(musiText2);
    DrawTune(tune);
    playTune(tune);
}

//=====
//===== Protein to ABC tune
//=====
//Our genetic code:  amino acids and

```

```

//the corresponding durations:
//Amino acids in line 1; notes in line 2;
//durations for 4/4 in line 3, for 6/8 in line 4.
//A virtual amino acid, Z; has been added to let us
//concatenate two chains with an ABC rest.
//A R N D C E Q G H I L K M F P S T W Y V Z
//C ^C D _E E F ^F G ^G A _B B c ^c d ^d _e e f ^f z
//1 1 1 1 1 2 2 2 2 2 2 2 2 2 4 4 4 8 8 8
// 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 6 6 6 6
//Two chains are concatenated with a ZA in the middle
//That will be interpreted as rest of duration 1
/*private String insuline = "IVEQCCTSICSLYQLENYCZA" +
    "NFVNQHLCGSHLVEALELVCGERRGFFYTPK";*/

/*private String histone = "GSHMGPRRRSRKPEAPRRRSP" +
    "SPTPTPGPSRRGPSLGASSHQHSRRRQGWLKEIRKLQKSTHLLIRKLPF" +
    "SRLAREICVKFTRGVDFNWQAQALLALQEAEEAFLVHLFEDAYLLTLHAG" +
    "RVTLPKDVQLARRIRGLEEGLG"; */
//The set of amino acids encoded as a string
//Z stands for a separator that will be
//interpreted as rest by ABC
private final String aminoAcids
    = "ARNDCEQGHILKMFPSTWYVZ";

private final String[] Notes
    = {"C", "^C", "D",
        "_E", "E", "F", "^F", "G",
        "^G", "A", "_B", "B", "c",
        "^c", "d", "_e", "e", "f", "^f", "z"};

//Durations for 4/4
private final String durationsAlphabet44
    = "1111122222222222444888";

//Header 4/4
private final String header44
    = "X:0\n"
    + "T:Song 4/4\n"
    + "M:4/4\n"
    + "L:1/8\n"
    + "K:C\n";

```

```

//Durations for 6/8
    private final String durationsAlphabet68
        = "11111111113333333336666";
//Header 6/8
    private final String header68
        = "X:0\n"
        + "T:Song 6/8\n"
        + "M:6/8\n"
        + "L:1/8\n"
        + "K:C\n";

//Amino acids in the first position of a codon
//are interpreted as notes.
//Interpretation is done amino acid per amino acid.
    private String notesInterpreterProt(char a) {
//The index of a is looked at the LIST of amino acids
        int i = aminoAcids.indexOf(a);
//The corresponding note is looked at the array of notes
        String ss = Notes[i];
        return ss;
    }

//Amino acids in the second position of a codon
//are interpreted as durations of notes.
//Interpretation is done amino acid per amino acid.
//measure = 44 for 4/4; measure = 68 for 6/8;
    private char durationsInterpreterProt(char a, int measure) {
//The index of a is looked at the LIST of amino acids
        int i = aminoAcids.indexOf(a);
//The corresponding duration is looked
//at the LIST of durations
        char c;
        if (measure == 44) {
            c = durationsAlphabet44.charAt(i);
        } else {
            c = durationsAlphabet68.charAt(i);
        }
        return c;
    }
}

```

```

//A protein is transformed into a tune.
//It is divided in codons with 2 letters,
//the first is interpreted as a note,
//the second as a duration
    private String interpreterProt(String s, int measure) {
        String v = "";
        int l = s.length();
        for (int i = 0; i < l - 2; i = i + 2) {
            char a = s.charAt(i);
            String note = notesInterpreterProt(a);
            i++;
            a = s.charAt(i);
            char duration = durationsInterpreterProt(a, measure);
            v = v + note + duration;
        }
        return v;
    }

private void mainProt(String protein) throws IOException {
    String header;
    //Measure 44 for 4/4, 68 for 6/8
    int measure = 44;
    int l = protein.length();
    //if l is odd (not even), the last amino acid is trimmed.
    if (!(l == 0 % 2)) {
        l = l - 1;
        protein = protein.substring(0, l);
    }
    String ABCstring = interpreterProt(protein, measure);
    System.out.println("ABCstring = " + ABCstring);
    if (measure == 44) {
        header = header44;
    } else {
        header = header68;
    }
    String musiText3 = header + ABCstring;
    Tune tune;
    tune = ABCmusiTextToTune(musiText3);
    DrawTune(tune);
}

```

```

        playTune(tune);
    }

//=====
//===== DNA to ABC tune
//=====
//Our genetic code:  In the first line we see our 16 codons
//with a pair of bases.  In the second line we see the
//associated notes.  In the third the duration for 4/4
//and in the last line the durations for 6/8.
//A rest is included as a note.
//AA AT AC AG TA TA TC TG CA CT CC CG GA GT GC GG
//C #C  D _E  E  F #F  G #G  A _B  B  c #c  d  z
// 1  1  1  1  2  2  2  2  2  2  2  4  4  4  8  4
// 1  1  1  1  1  1  1  3  3  3  3  3  3  3  6  3
//The set of codons encoded as a string.
//Z stands for a separator that will be
//interpreted as rest by ABC
    private final String Codons
        = "AA-AT-AC-AG-TA-TA-TC-TG-CA-CT-CC-CG-GA-GT-GC-GG-";

//Codons are taken by pairs.
//The first codon
//is interpreted as  a note.
//Interpretation is done codon per codon.
    private String notesInterpreterDNA(String cd) {
//The index of cd is looked at the LIST of amino acids
        cd = cd + "-";
        int i = Codons.indexOf(cd);
        i = i / 3;
        System.out.println("i = " + i);
//The corresponding note is looked at the array of notes
        String ss = Notes[i];
        return ss;
    }

//Codons are taken by pairs.
//The second codon
//is interpreted as  a duration.
//Interpretation is done codon per codon.

```

```

        private char durationsInterpreterDNA(String cd,
            int measure) {
            //The index of cd is looked at the LIST of amino acids
            cd = cd + "-";
            int i = Codons.indexOf(cd);
            i = i / 3;
//The corresponding duration is looked
//at the LIST of durations
            char c;
            if (measure == 44) {
                c = durationsAlphabet44.charAt(i);
            } else {
                c = durationsAlphabet68.charAt(i);
            }
            return c;
        }

//A dna sequence is transformed into a tune.
//The sequence is divided in pair of codons
//each one with 2 letters,
//the first codon is interpreted as a note,
//the second as a duration
        private String interpreterDNA(String s, int measure) {
            String v = "";
            int l = s.length();
            System.out.println("l = " + l);
            for (int i = 0; i < l - 4; i = i + 4) {
                String cd = s.substring(i, i + 2);
                System.out.println("cd = " + cd);
                String note = notesInterpreterDNA(cd);
                cd = s.substring(i + 2, i + 4);
                System.out.println("cd = " + cd);
                char duration = durationsInterpreterDNA(cd, measure);
                v = v + note + duration;
            }
            return v;
        }

public void mainDNA(String adn) throws IOException {

```

```

String header;
//Measure 44 for 4/4, 68 for 6/8
int measure = 44;
if (measure == 44) {
    header = header44;
} else {
    header = header68;
}
/*
//OTTHUMT00000257844 intron 7:_unprocessed_pseudogene
String adn =
"GTCCCCTCCAAATGTGTCCTGGACAGCCCTGGGACCATGGGACAGACCTAGCCCAGGAAG"+
"CCCATGGGGACCTCGACACCAGGCCCTGCCAGGCCCATCAGGACTAAGAAGGAGATCAG"+
"GAGATACCCCCACTGCAACATGGGCACCTGTATCTGGAAGGAACCAGGGTGGCACCCACC"+
"AGCTTCCCCGCAGCCCCAGATACCTTGAAGATCCCATTCAATGGGGAAGCTGGTCCCAT"+
"CAAGGCCACAAGAGAGAGACGTGAGCACGGCATCCCTGCAGCCCCAGACAGCCAGCAGTA"+
"TGACCTGGGGCACAATACCCAGGCCAAGAGTGGCCTGGCCTAGACCCCAGCCCTGGGAA"+
"GAGAGGGGTACCAGGGCTGTGGTGGGCTCCCAGCTGCAGTCAGCATGACCATGCAGGGGA"+
"CCCTGGCACTGCAGAGCTCTGCATTGTCACAATTCACCTTACAGAACCGGGGAGAGATCT"+
"CGAAGCCTTGCAGCCACTGCTGCAGCCACAGCAGGTGCCACCACCACGTGCCTTCCAGCT"+
"GCCAGTGCAGGCTCTCCTGACAGCAGGGCACCCCTGCAGCCCTCACCTCAGCAGCTCAGAC"+
"TCTGAAGGTATCTGTTGTTTCATGTTCCCAGTGCTCCCACCAAAGAGAATTACTACAGAGA"+
"ATACTCTTGGTCCTGAGGTGGGCAAACAGCATGCCCTGTGGATGGAGCCGGCTTCTTTC"+
"CCAGCCACCATGCTTGCTGGATGGGCCCGCAGAGCAAGTGGCCATGAGACGGGCATGGGG"+
"ACGGGAGAGGCTCAGCATCCTGCACATGCCCTTACCAAGGCTCACCAGGCTGACACCACT"+
"GCTGAGCGCCTTGTCTGTGGAAAACAGATTACACCCAACCCCTGGGATGGCACC";
*/
//adn is is trimmed to a length multiple of 4
int l = adn.length();
int f = l / 4;
int g = f * 4;
adn = adn.substring(0, g);

String ABCstring = interpreterDNA(adn, measure);
System.out.println("ABCstring = " + ABCstring);
String musiText4 = header + ABCstring;
Tune tune;
tune = ABCmusiTextToTune(musiText4);
DrawTune(tune);
playTune(tune);

```

```

    }

//=====PITCH RECOGNIZING=====
//=====ABC tools=====
//A tune from a file is read into an usable tuneBook
TuneBook ABCfileToBookTune(File abcFile)
    throws FileNotFoundException {
    //creates a tunebook from the previous file
    TuneBook book = new TuneBook(abcFile);
    return book;
}

//A tune is retrieved from a TuneBook
Tune RetrieveTuneFromBook(TuneBook book) {
    //A book possibly contains various tunes.
    // Choose one, pick its index (in the X: field) and
    //assign it to tuneIndex else define
    //tuneIndex = book.getHighestReferenceNumber();
    int tuneIndex = book.getHighestReferenceNumber();
    Tune tune = book.getTune(tuneIndex);
    return tune;
}

//An indexed tune is retrieved from a TuneBook
Tune RetrieveTuneFromBook(TuneBook book, int tuneIndex) {
    //A book possibly contains various tunes.
    // Choose one and pick its index (in the X: field)
    Tune tune = book.getTune(tuneIndex);
    return tune;
}

//A musiText is transformed into a tune
Tune ABCmusiTextToTune(String musiText) {
    //From String to Tune
    Tune tune = new TuneParser().parse(musiText);
    return tune;
}

//A tune as Tune is drawn
//Try tunes without regulative instructions or lyrics

```

```

void DrawTune(Tune tune) {
    //creates a component that draws the melody on a stave
    JScoreComponent jscore = new JScoreComponent();
    jscore.setJustification(true);
    jscore.setTune(tune);
    JFrame j = new JFrame();
    j.add(jscore);
    j.pack();
    j.setVisible(true);
}

//A tune is played
void playTune(Tune tune) {
    //Displays the title of the tune
    System.out.println("Title = " + tune.getTitles()[0]);
    // and its key
    System.out.println("Key = "
        + tune.getKey().toLitteralNotation());
    PLAYER.start();
    PLAYER.play(tune);
}
} //End of inner class MusiText

//=====
//=====PITCH RECOGNIZING=====
//=====
//Program h76 PicthRecognizing5
//A training facility
//for pitch recognizing.
//We have seven levels of expertise from 0 to 7 and
//25 levels of complexity from 2 to 27.
//That level is determined by the number of notes
//that are in the training set.
//A soft increase in complexity enables success
//and therefore learning.
//The elegance of Java allows us to
//pass from a very simple test
//to the following level of complexity and so on
//without killing the application to begin a new one.
//One can choose expertise and complexity level at will

```

```

//else go from start to end as if in one session.
    static class PitchRecognizing6 extends JFrame {

        private static final long serialVersionUID = 1L;

//This is the set of all notes we use:
        private static final String[] NOTES
            = {"A,", "_B,", "B,", "C", "^C",
              "D", "_E", "E", "F", "^F",
              "G", ^G", "A", "_B", "B",
              "c", ^c", "d", "_e", "e",
              "f", ^f", "g", ^g", "a",
              "_b", "b", "c'"};

//To present notes to the student in that natural order
//is most possibly not the most efficient way for teaching.
//Our proposal is to try to observe the greatest distance
//among notes, such as in the following ordered LIST.
//The workingSet for level k takes
//the first k elements of this set
// (but must be displayed in the natural order):
        /*
        private static String[] notesWorkingSet =
            {"C", "c", "c'", "G", "g",
            "E", "e", "A", "A", "a",
            "D", "d", "F", "f", "B,",
            "B", "b", ^F", ^f", "_B,",
            "_B", "_b", ^C", ^c", ^G",
            ^g", "_E", "_e"};
        */

//In consequence, the next vector keeps the order
//of appearance of notes in the graded levels of complexity:
        private static final int[] NOTES_ORDER
            = {3, 15, 27, 10, 22,
              7, 19, 0, 12, 24,
              5, 17, 8, 20, 2,
              14, 26, 9, 21, 1,
              13, 25, 4, 16, 11,
              23, 6, 18};
    }

```

```

//For level of complexity k, the working set
//takes the first k notes of notesWorkingSet.
//But the chosen elements must be presented
//in the JList in natural order.
//All this meshing bookkeeping is difficult.
//Our choice is to mark the chosen elements
//in the set notes to guide ourselves.
    private static final boolean[] MARK = new boolean[NOTES.length]

//To make a point you need to find the pitch
//before tolerance is wasted.
//Actual number of gained points
    private static int gainedPointsThisSeries = 0;

//Every ABC tune must have a correct header
    private static final String HEADER
        = "X:0\n"
        + "T:Song 4/4\n"
        + "M:4/4\n"
        + "L:1/8\n"
        + "K:C\n";

//The note that must be recognized as a tune
    private static Tune tuneTest;
//The note that must be recognized as a String
    private static String noteTest;

    private static final JPanel OUTER_PANEL = new JPanel();

//Counter of trials before success
    private static int counterOfTrials = 1;
    private static String oldNote = "";

//The User must declare which degree of expertise
//he or she wants to play with:
//Expertise:
//0 (deaf), 1(beginner), 2(intermediate),
//3 (in the right path), 4 (advanced),
//5 (very advanced), 6 (quasi-expert) , 7(expert).
    private static int expertise = 0;

```

```
        private static final int EXPERTISE_LEVEL = 8;

//Each test has a complexity level
//which is the size of the working set.
//Index runs from 0 to 27.
        private static int levelOfComplexity = 2;

//Number of trials granted before failure declaration
        private static int tolerance;

//To graduate from the current level, you need
//a number of points = persistence * levelOfComplexity
        private static int neededPoints;
        private static int persistence = 4;

//A rest prior the playing of a tune.
        private static Integer lag = 8;

//We turn on the inbuilt random generator:
        private static final Random R = new Random();

        private static final JButton OK = new JButton("");
        private static final JButton GRADE = new JButton("");
        private static final JButton REPEAT
            = new JButton("Repeat question");
        private static final JButton PLAY_NOTES
            = new JButton("Play notes");
        private static final JButton SHOW_STATUS
            = new JButton("Show my status");
        private static final JButton CHOOSE_PERSISTENCE
            = new JButton("Choose persistence");
        private static final JButton CHOOSE_EXPERTISE
            = new JButton("Choose expertise");
        private static final JButton CHOOSE_COMPLEXITY
            = new JButton("Choose complexity");
        private static String report = "";
        private static final JButton NEW_ACCOUNT
            = new JButton("New account");
        private static final JButton CONTINUE_TRAINING
            = new JButton("Continue training");
```

```

        private static final JButton SAVE_ACCOUNT
            = new JButton("Save account");
//MODEL is a modifiable LIST
        private static final DefaultListModel MODEL
            = new DefaultListModel();
//the JList is built on image of MODEL
        private static final JList LIST = new JList(MODEL);
        private static final JFrame F = new JFrame();

//Matrix to keep the number of trials made to pass from
//the given level of complexity to the next
        private static final int[][] MEMORY
            = new int[NOTES.length][EXPERTISE_LEVEL];
//Number of tries of today
//(in the current levels of complexity and expertise)
        private static int trialsOfThisSeries = 0;
//Flag that indicates that a new level has been gained
        private static boolean graded = false;
//Date
        private static String date;
//Is the User keeping a record of his or her performance?
        private static final boolean MEMORIZING = false;

        static private final TunePlayer PLAYER = new TunePlayer();

public PitchRecognizing6() {
    //Rest before the first question
    lag = 1;
    //Expertise:
    //0 (deaf), 1(beginner), 2(intermediate),
    //3 (in the right path), 4 (advanced),
    //5 (very advanced), 6 (semi-expert), 7(expert).
    expertise = EXPERTISE_LEVEL - 2;
    //Number of points that are needed to graduate
    //from the second initial level of complexity
    neededPoints = persistence * levelOfComplexity;
    //Number of trials granted before failure declaration
    tolerance = tolerance(expertise);
    OUTER_PANEL.setLayout(new BorderLayout(OUTER_PANEL,
        BorderLayout.X_AXIS));
}

```

```

//The MEMORY matrix is zeroed
initializeMemory();
PLAYER.start();
//The JList goes here
JPanel leftPanel = new JPanel();
//Buttons go here
JPanel rightPanel = new JPanel();

leftPanel.setLayout(new BorderLayout());
rightPanel.setLayout(new BoxLayout(rightPanel,
    BoxLayout.Y_AXIS));

LIST.setSelectionMode(
    ListSelectionMode.SINGLE_SELECTION);
LIST.setBorder(
    BorderFactory.createEmptyBorder(2, 2, 2, 2));
leftPanel.add(new JScrollPane(LIST));
leftPanel.setBorder(BorderFactory.
    createEmptyBorder(20, 20, 20, 20));
if (((expertise <= EXPERTISE_LEVEL)
    & (levelOfComplexity < NOTES.length))
    & (levelOfComplexity >= 2)) {
    //There is no MARKed note at the beginning
    for (int i = 0; i < NOTES.length; i++) {
        MARK[i] = false;
    }
    initializeModel(persistence, expertise,
        levelOfComplexity);
    //Begin training
    poseQuestion(lag);

    maketest();
} else {
    System.out.println("Test over");
}

//Button REPEAT
//The posed question must be REPEATED
REPEAT.addActionListener((ActionEvent e) \rightarrow {
    tuneTest = noteToTune("z1"

```

```

        + noteTest + noteTest + noteTest);
    playTune(tuneTest);
});

//
PLAY_NOTES.addActionListener((ActionEvent e) \rightarrow {
    String m = "";
    for (int i = 0; i < MODEL.size(); i++) {
        m = m + MODEL.get(i).toString() + "4";
    }
    System.out.println(m);
    Tune mTune = noteToTune(m);
    playTune(mTune);
});

//Information is displayed
SHOW_STATUS.addActionListener((ActionEvent e) \rightarrow {
    JOptionPane.showMessageDialog(
        null, report(persistence,
            expertise, levelOfComplexity));
});

//Test can be short for playing else
//longer for better training. Default is 4.
CHOOSE_PERSISTENCE.addActionListener((ActionEvent e) \rightarrow {
    String s1 = JOptionPane.showInputDialog(
        "Enter desired level of persistence, "
        + "1 (for low training), 1, ... , 9 "
        + "(for very hard trainng)", 4);
    String s = s1.trim();
    //translation from char to number
    persistence = s.charAt(0) - 48;
    System.out.println("Persistence = " + persistence);
});

//Experts need less tolerance than beginners
CHOOSE_EXPERTISE.addActionListener((ActionEvent e) \rightarrow {
    String s1 = JOptionPane.showInputDialog(
        "Enter your level of expertise, "
        + "0 (deaf), 1, 2, 3, 4, 5, 6, 7 (expert)", 0);

```

```

String s = s1.trim();
//translation from char to number
expertise = s.charAt(0) - 48;
System.out.println("Expertise = " + expertise);

if (MEMORIZING) //The MEMORY matrix is updated
{
    updateMemory(expertise, levelOfComplexity,
        trialsOfThisSeries);
}
//New count
trialsOfThisSeries = 0;
gainedPointsThisSeries = 0;
});

//The training set has 3 notes at the beginning,
//4 at the next level and so on until 27.
CHOOSE_COMPLEXITY.addActionListener((ActionEvent e) \rightarrow {
String s1 = JOptionPane.showInputDialog(
    "Enter desired level of complexity, "
    + " from 2 (least) to " + NOTES.length
    + "(max)", 2);
String s = s1.trim();
int a = s.charAt(0) - 48;
levelOfComplexity = a;
if (s.length() > 1) {
    int b = s.charAt(1) - 48;
    levelOfComplexity = a * 10 + b;
}
System.out.println("Complexity = "
    + levelOfComplexity);

if (MEMORIZING) //The MEMORY matrix is updated
{
    updateMemory(expertise, levelOfComplexity,
        trialsOfThisSeries);
}
//new count
trialsOfThisSeries = 0;
gainedPointsThisSeries = 0;

```

```

        initializeModel(persistence, expertise,
                        levelOfComplexity);
        poseQuestion(levelOfComplexity);
    });

    //You are allowed to play but if you are decided to
    //become an expert, open a new account
    //to keep a record of your performance.
    NEW_ACCOUNT.addActionListener((ActionEvent e) \rightarrow
        JOptionPane.showMessageDialog(null,
            "Begin a new file with suffix .ear, say"
            + "BraveSmith.ear, "
            + "where you will keep your account. "
            + "Next, choose the note else click on Repeat.");
    //DateFormat is inquired
    DateFormat dF = DateFormat.
        getDateInstance(DateFormat.MEDIUM);
    //date is transformed into a string
    date = dF.format(new java.util.Date());
    System.out.println(date);
    //date is written to TextArea
    A.myTextArea.setText(date);
    //A new file is initialized with
    //the date when the User began
    //this experiment with complexity
    A.saveTextAreaToFile();

    //Memory is cleared
    initializeMemory();
});

//Future expert reopens his or her account
//on each session
CONTINUE_TRAINING.addActionListener((ActionEvent e) \rightarrow
    //TextArea is cleared
    A.myTextArea.setText("");
    JOptionPane.showMessageDialog(null,
        "Open the file "
        + "where you have your account");
    //A chosen file is read

```

```

        A.readFileIntoTextArea();
        String input = A.readFileIntoTextArea();
        //String is interpreted as information,
        //mainly as numbers,
        //and kept in MEMORY[][]
        ParseReadString("\n" + input);
    });

    //Memory of performance is saved to file
    SAVE_ACCOUNT.addActionListener((ActionEvent e) \rightarrow {
        saveMemory();
    });

    //Buttons are added to rightPanel
    rightPanel.add(OK);
    rightPanel.add(GRADE);
    rightPanel.add(REPEAT);
    rightPanel.add(PLAY_NOTES);
    rightPanel.add(SHOW_STATUS);
    rightPanel.add(CHOOSE_PERSISTENCE);
    rightPanel.add(CHOOSE_EXPERTISE);
    rightPanel.add(CHOOSE_COMPLEXITY);
    rightPanel.add(NEW_ACCOUNT);
    rightPanel.add(CONTINUE_TRAINING);
    rightPanel.add(SAVE_ACCOUNT);

    rightPanel.setBorder(
        BorderFactory.createEmptyBorder(0, 0, 0, 20));

    OUTER_PANEL.add(leftPanel);
    OUTER_PANEL.add(rightPanel);

    F.add(OUTER_PANEL);

    F.setSize(500, 550);
    F.setLocationRelativeTo(null);
    F.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    F.setVisible(true);

} //End of constructor of PitchRecognizing6

```

```

private void saveMemory() {
    A.myTextArea.setText("");
    updateMemory(expertise, levelOfComplexity,
        trialsOfThisSeries);
    MEMORYToTextArea();
    if (myFile.exists()) {
        Thread runner = new Thread() {
            @Override
            public void run() {

                try (FileWriter out = new FileWriter(myFile)) {
                    A.myTextArea.write(out);
                } catch (IOException ex) {

                    Logger.getLogger(FirstTheoryGUI.class.getName()).log(Level.SEVERE, null,
                        ex);
                }
            }
        };
        runner.start();
    }
}

//String s that encodes for a number is decoded into number
private static int stringToNumber(String s) {
    int k = s.length();
    int number = 0;
    for (int j = 0; j < k; j++) {
        char c = s.charAt(j);

        int l = Character.getNumericValue(c);
        int q = j + 1;
        int number1 = (int) (l * Math.pow(10, k - q));
        number = number + number1;
    }
    return number;
}

//The string captured from the file

```

```

//is interpreted as information
private void ParseReadString(String input) {
    System.out.println("\nInput string = " + input);
    //Input is divided in lines, which are separated by \n
    String[] tokens = input.split("\n");
    for (String t : tokens) {
        System.out.println(t);
    }
    //Each line is deciphered according to
    //the saved pattern ( a shift of one is needed)
    date = tokens[1];
    persistence = stringToNumber(tokens[2].trim());
    expertise = stringToNumber(tokens[3].trim());
    levelOfComplexity = stringToNumber(tokens[4].trim());
    neededPoints = persistence * levelOfComplexity;
    int nTokens = tokens.length;
    // System.out.println("Number of tokens = " + nTokens);
    /*
for (int i = 0; i < nTokens; i++)
System.out.println("Token " + i + " " + tokens[i]);
    */
    for (int i = 5; i < nTokens; i++) {
        String[] tokens2 = tokens[i].split("\t");
        for (int j = 1; j < 9; j++) {
            //System.out.println("NTokens2 = " + tokens2.length);
            MEMORY[i - 5][j - 1] = stringToNumber(tokens2[j].trim());
        }
    }

    System.out.println("Memory numeric");
    for (int i = 0; i < NOTES.length; i++) {
        System.out.printf("%n %10d %10d %10d %10d %10d %10d"
            + " %10d %10d %10d ",
            i, MEMORY[i][0], MEMORY[i][1],
            MEMORY[i][2], MEMORY[i][3], MEMORY[i][4],
            MEMORY[i][5], MEMORY[i][6], MEMORY[i][7]);
    }

    initializeModel(persistence, expertise, levelOfComplexity);
    report(persistence, expertise, levelOfComplexity);

```

```

    }

//Memory matrix is zeroed
private void initializeMemory() {
    for (int i = 0; i < NOTES.length; i++) {
        for (int j = 0; j < 5; j++) {
            MEMORY[i][j] = 0;
        }
    }
}

//The trials of the session are
//added to the Memory matrix.
private void updateMemory(int expertise,
    int levelOfComplexity,
    int trialsOfThisSeries) {
    int i = levelOfComplexity;
    int j = expertise;
    MEMORY[i][j] = MEMORY[i][j] + trialsOfThisSeries;
}

//The MEMORY is written to
//the console and the text area.
private void MEMORYToTextArea() {
    System.out.println("\nDATA SAVED: \n");
    //MEMORY is displayed on the console
    System.out.println("Beginning date = " + date);
    System.out.printf("Persistence = %3d    "
        + "Expertise = %3d    "
        + "level of complexity = %3d",
        persistence, expertise, levelOfComplexity);
    System.out.printf("\nMemory matrix: number of trials \n"
        + "needed to pass from a given level of complexity
        + " to the next. Seven levels of expertise in colu
        + "28 levels of complexity in rows. \n");
    for (int i = 0; i < NOTES.length; i++) {
        System.out.printf("%n %10d %10d    %10d %10d    "
            + "%10d %10d %10d %10d    %10d ",
            i, MEMORY[i][0], MEMORY[i][1],
            MEMORY[i][2], MEMORY[i][3],

```

```

        MEMORY[i][4], MEMORY[i][5], MEMORY[i][6],
        MEMORY[i][7]);
    }
    //MEMORY is written to a string
    String sMemory = "" + date + "\n";
    sMemory = sMemory + persistence + "\n"
        + expertise + "\n"
        + levelOfComplexity + "\n";
    for (int i = 0; i < NOTES.length; i++) {
        for (int j = 0; j < EXPERTISE_LEVEL; j++) {
            /*Integer k = MEMORY[i][j];
String s = k.toString(); */
            sMemory = sMemory + "\t" + MEMORY[i][j];
        }
        sMemory = sMemory + "\n";
    }
    //MEMORY is written to the text area
    A.myTextArea.setText(sMemory);
}

//The musiText is transformed into a tune,
//an ABC4J musical object
    private static Tune ABCmusiTextToTune(String musiText) {
        Tune tune = new TuneParser().parse(musiText);
        return tune;
    }

//A tune is played
    private static void playTune(Tune tune) {
        PLAYER.play(tune);
    }

//The note is converted into an ABC tune
    private static Tune noteToTune(String note) {
        String musiText = HEADER + note;
        Tune tune = ABCmusiTextToTune(musiText);
        return tune;
    }

//The number of granted trials before failure declaration

```

```

//depends on expertise and the complexity level
private int tolerance(int expertise) {
    int l;
    l = EXPERTISE_LEVEL - expertise;
    return l;
}

//A complexity level defines each working set
//of notes to be recognized.
//The starting set has only two notes
private void initializeModel() {
//The first two elements of notesWorkingSet are marked
    MARK[NOTES_ORDER[0]] = true;
    MARK[NOTES_ORDER[1]] = true;
    MODEL.add(0, NOTES[NOTES_ORDER[0]]);
    MODEL.add(1, NOTES[NOTES_ORDER[1]]);
}

//A complexity level defines each working set
//of notes to be recognized.
//The starting set has only two notes
private void initializeModel(int persistence,
                             int expertise,
                             int levelOfComplexity) {
    reinitializeModel();
//The corresponding elements of notesWorkingSet are marked
//and inserted into the MODEL
    for (int i = 2; i < levelOfComplexity; i++) {
        insertNote(i);
    }
}

private String report(int persistence,
                     int expertise,
                     int levelOfComplexity) {
    /*System.out.println("Position " +
"of new note = " +
NOTES_ORDER[levelOfComplexity-1] );
System.out.println("New noteTest "
+ notes[NOTES_ORDER[

```

```

        levelOfComplexity-1]] );*/
        neededPoints = persistence * levelOfComplexity;
        tolerance = tolerance(expertise);
        report = "Persistence = " + persistence
            + "\nExpertise level = " + expertise
            + "\nLevel of complexity"
            + "\n=Number of notes under Test = "
            + levelOfComplexity
            + "\nYou gain a point if you find "
            + "\nthe answer before "
            + tolerance + " trials."
            + "\nNeeded number of points"
            + "\nto pass to the next level = "
            + neededPoints;
        System.out.println(report);
        return report;
    }

//Old information is erased and
//a new start is set up
    private void reinitializeModel() {
        System.out.println("\nReinitialization");
        MODEL.clear();
        for (int i = 0; i < NOTES.length; i++) {
            MARK[i] = false;
        }
        initializeModel();
        lag = 8;
    }

    private void insertNote(int k) {
//A new note must be accommodated in the right order
//Counter of notes less than the new one
        int counter = 0;
//Position in notes[] of the new note
        int n = NOTES_ORDER[k];
//To find the place of insertion,
//all active notes less than the new one are counted
        for (int i = 0; i < n; i++) {
            if (MARK[i]) {

```

```

        counter++;
    }
}
//The new note is inserted in the MODEL
//after all notes less than it.
MODEL.add(counter, NOTES[n]);
//The added note is marked
MARK[n] = true;
}

//The next level of complexity is generated
//A new note is inserted in the correct place
//Bookkeeping is done
private void updateModel() throws Exception {
    //The MEMORY matrix is updated
    updateMemory(expertise, levelOfComplexity,
        trialsOfThisSeries);
    //Lack expertise? Yes= continue
    if (expertise < EXPERTISE_LEVEL - 1) {
        {
            System.out.println("NEW LEVEL OF EXPERTISE");
            expertise++;
            GRADE.setBackground(Color.BLUE);
        }
    } else //No = Pass to the next level of complexity
    {
        if (levelOfComplexity < NOTES.length - 1) {
            insertNote(levelOfComplexity);
            //We have a new level of complexity
            levelOfComplexity++;
            GRADE.setBackground(Color.BLUE);
            //
            System.out.println("CONGRATULATIONS: "
                + "you are one level up!");
            expertise = 0;
        }
    }
}

//Rest, silence
lag = 8;

```

```

//Number of granted trials before failure declaration
    tolerance = tolerance(expertise);
//Number of points that are needed to graduate
//from the current level of complexity
    neededPoints = persistence * levelOfComplexity;
//begin a new count of gainedPointsThisSeries
//and trialsOfMoment
    gainedPointsThisSeries = 0;
    trialsOfThisSeries = 0;
    report(persistence, expertise, levelOfComplexity);
    graded = true;
}

//Outputs a random note from the chosen alphabet
    private static String randomNote() {
        String newNote;
        int l = MODEL.getSize();
//A random integer less than the
//number of allowed notes is thrown
        int i = r.nextInt(l);
        newNote = (String) MODEL.get(i);
//the new challenge must be different than the prior one
//because of selected index
        if (newNote.contentEquals(oldNote)) {
            int j = (i + 1) % l;
            newNote = (String) MODEL.get(j);
        }
        oldNote = newNote;
        return newNote;
    }

//A random note to be recognized is played
    static void poseQuestion(Integer lag) {
        String s = lag.toString();
        noteTest = randomNote();
        System.out.println("Random noteTest = "
            + noteTest);
        tuneTest = noteToTune("z" + s + noteTest + noteTest + noteTest);
        playTune(tuneTest);
        if ((trialsOfThisSeries == 2)

```

```

        & (graded)) {
            GRADE.setBackground(Color.GRAY);
        }
    }

private void maketest() {
    LIST.addMouseListener(
        new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                if ((expertise <= EXPERTISE_LEVEL)
                    & (levelOfComplexity < NOTES.length)) {
                    if (e.getClickCount() == 1) {
                        trialsOfThisSeries++;
                        System.out.println("Tries of moment thus :
                            + trialsOfThisSeries);
                        int index = LIST.locationToIndex(e.getPoint
                        String noteAnswer
                            = (String) MODEL.get(index);
                        String notesBoth
                            = noteTest + "2" + noteAnswer + "2"
                        Tune tuneBoth = noteToTune(notesBoth);
                        playTune(tuneBoth);

                        //If answer is correct
                        if (noteAnswer.contentEquals(noteTest)) {
                            gainedPointsThisSeries++;
                            OK.setBackground(Color.GREEN);

                            //GRADE.setText("");
                            System.out.println("Your answer " + noteTest
                                + " is RIGHT!" + " Number of t

                                + counterOfTrials
                                + " Gained points = " +
gainedPointsThisSeries);

                            lag = 8;

                            //Promotion?
                            if (gainedPointsThisSeries == neededPo

```



```

//=====MUSICAL ANALYSIS=====
//A very simple ABC tune is received as input,
//it is divided in tokens, component notes,
//which are kept in the string vector TokenABC.
//To synthesize a new tune with the same distribution
//we simply disorder the tune. To that aim,
//we permute many pair of tokens in TokenABC.
//=====
    private static String[] TokenABC = new String[10000];
    private static int nTokensABC = 0;
    private static String headerABC;
    private static String bodyABC;
    private static String scoreABC;
    private static String lyricsABC;

//Delete all spaces
    private static String deleteAllSpaces(String s) {
        int length = s.length();
        char c;
        String x = "";
        for (int i = 0; i < length; i++) {
            c = s.charAt(i);
            if (c == ' '); else {
                x = x + c;
            }
        }
    }
//System.out.println("LILY===== \n" + x);
    return x;
}

//An ABC string is divided in tokens,
//each one determined by a note
//or a bar or slur
    private static int tokenizeScoreABC(String s) {
        s = deleteAllSpaces(s);
        s = s + " ";
        int length = s.length() - 1;
        System.out.println("New s V5= \n" + s);
        System.out.println("Length = " + length);
        String token = "";

```

```
String symbolsABC = "-|";
String accidentsABC = "_^";
String notesABC = "CDEFGABcdefgabz,'"";
String durationNumbers = "123456789/";
String starters = "-|_^CDEFGABcdefgabz";
nTokensABC = 0;
boolean nextCharIsStarter = false;

for (int i = 0; i < length; i++) {
    char c = s.charAt(i);
    String a = "" + c;
    char nextChar = s.charAt(i + 1);

    String NextChar = "" + nextChar;
    if (starters.contains(NextChar)) {
        nextCharIsStarter = true;
    }

    System.out.println("char = " + a);

    //If a is a symbol, it is a token
    if (symbolsABC.contains(a)) {
        TokenABC[nTokensABC] = a;
        nTokensABC++;
        //Begin new token
        token = "";
        nextCharIsStarter = false;
    } else //a is part of a token
    {
        //if a is an accident, begin accumulation
        if (accidentsABC.contains(a)) {
            token = a;
            nextCharIsStarter = false;
        }

        //if a is a note: accumulate
        if (notesABC.contains(a)) {
            token = token + a;
        }
    }
}
```

```

//if a is a duration number, accumulate
if (durationNumbers.contains(a)) {
    token = token + a;
}

//if a is followed by a starter: end accumulation
//and begin a new token.
//Exception: a is an accident
if ((nextCharIsStarter)
    & (!(accidentsABC.contains(a)))) {
    TokenABC[nTokensABC] = token;
    nTokensABC++;
    token = "";
    nextCharIsStarter = false;
}

//if a is the trailing char, close accumulation
if (i == length - 1) {
    TokenABC[nTokensABC] = token;
    nTokensABC++;
}

} //End of else
} //EndFor

nTokensABC--;
if (print) {
    System.out.println(
        "Number of tokens = " + " " + nTokensABC);
    System.out.println("ABC-score tokens");
    int nMeasure = 0;
    for (int i = 0; i < nTokensABC; i++) {
        System.out.print(i + " " + TokenABC[i]);
        if (TokenABC[i].contentEquals("|")) {
            System.out.print(nMeasure);
            nMeasure++;
        }
        System.out.println();
    }
    System.out.println(

```

```

        "Length of input score = " + " " + length);
    }
    return nTokensABC;
}

//A transposition is the interchange of
//place between two items.
private static void transposition(int nTokensABC) {
    int m = r.nextInt(nTokensABC);
    boolean flag = false;
    //Sites must be different
    int n = 0;
    while (!(flag)) {
        n = r.nextInt(nTokensABC);
        if (!(n == m)) {
            flag = true;
        }
    }
    if (print) {
        System.out.println("Permutation of "
            + m + " and " + n);
    }
    String a = TokenABC[m];
    String b = TokenABC[n];
    TokenABC[n] = a;
    TokenABC[m] = b;
}

//Mixer: a sequence of transpositions creates variability.
//The output is encoded in a string, a chromosome.
private static void mixer(int numberOfTransp,
    int nTokensABC) {
    for (int i = 0; i < numberOfTransp; i++) {
        transposition(nTokensABC);
    }
}

//An ABC tune is separated into header + body
private static void getComponentsABC(String tuneABC) {

```

```

    int i = tuneABC.indexOf("K:");
    System.out.println("K: at = " + i);
    String h = tuneABC.substring(i);
    int j = h.indexOf("\n");
    System.out.println("eol at = " + j);
    headerABC = tuneABC.substring(0, i + j);
    System.out.println("Header = " + headerABC);

    bodyABC = tuneABC.substring(i + j);
    System.out.println("bodyABC = " + bodyABC);
//Lyrics is separated from score
    decomposeBodyABC(bodyABC);
}

//The body of a tune is decomposed into score + lyrics
private static void decomposeBodyABC(String body) {
    scoreABC = "";
    lyricsABC = "";
    String lineTune;
    int l;
    int nBeginS = 0;
    int nEndS;
    String tailBody = body;
    boolean accumulate = true;

    while (accumulate) {
        if (tailBody.contains("\n")) {
            System.out.println("Tail of body = \n" + tailBody);
            int length = tailBody.length();
            System.out.println("length = " + length);
            System.out.println("Beginning = " + nBeginS);
            l = tailBody.indexOf("\n") + 1;
            nEndS = l;
            System.out.println("end = " + nEndS);
            lineTune = tailBody.substring(nBeginS, nEndS);
            tailBody = tailBody.substring(nEndS);
            System.out.println("line of tune: " + lineTune);
            if (lineTune.contains("w:")) {
                lyricsABC = lyricsABC + lineTune;
            } else {

```

```

        scoreABC = scoreABC + lineTune;
    }
    } else {
        accumulate = false;
    }
} //End of while
if (tailBody.contains("w:")) {
    lyricsABC = lyricsABC + tailBody;
} else {
    scoreABC = scoreABC + tailBody;
}
System.out.println("Score: \n" + scoreABC);
System.out.println("Lyrics: \n" + lyricsABC);
}

private static String assembleTuneABC() {
    String song = "";
    for (int i = 0; i < nTokensABC; i++) {
        song = song + TokenABC[i] + " ";
    }
    return song;
}

//A very simple ABC tune, s, is received as input.
//It is first divided into header plus body,
//which is next divided into score + lyrics.
//Next, the score is divided in tokens, component notes.
//To synthesize a new tune with the same distribution
//we simply mix the score. To that aim,
//we permute many pair of tokens in TokenABC.
private static void StatVersion(String tuneABC) {
    //Get header, score and lyrics
    getComponentsABC(tuneABC);
    //Divide the score in tokens
    nTokensABC = tokenizeScoreABC(scoreABC);
    //The score is mixed with the help of transpositions.
    //Number of transpositions
    int numberOfTransp = 20;
    mixer(numberOfTransp, nTokensABC);
    //A new tune is reassembled

```

```

        String song = headerABC + "\n" + assembleTuneABC();
        A.myTextArea.setText(song);
    }

//*****
//***** Main method *****
//*****
    public static void main(String[] args) throws IOException {
//The instructions in here are executed at the beginning,
//but the GUI is always ready
//to fulfill the desires of the User.
        print = true;
    }

} //End of main class H86 FirstTheoryGUI

/*
 * X:14
T: Canción de cuna
M: 4/4
L: 1/8
K: C
C> CCEG2G2 |
w: A-rru-rrú mi ni-ña,
A> AAAG4 |
w: a-rru-rrú mi sol,
   FFF           FE2E2 |   D> D DDC4   |
w: duér-me-te pe-da-zo de mi co-ra-zón.
C> CCEG2G2 |
w: Van pa-san-do nu-bes,
A> AAAG4 |
w: co-pos de .al-go-dón,
   EGE           GD2D2 | E> F EDC4   |
w: duér-me-te pe-da-zo de mi co-ra-zón.
C/2z/2CCEG2G2 |
w: Pa-lo-mi-tas vie-nen,
A> AAAG4 |
w: pa-lo-mi-tas van,
E^FG E^F2D2 | ^C D E^CD4 |
w: duér-me-te pe-da-zo de mi co-ra-zón.

```

```
FEE          DD2C2 |   E< F GBc4   |]
w:Duér-me-te pe-da-zo de mi co-ra-zón.*/*
```

87 Exercise. *Run the program and play with the code. Enjoy the facility for training the ear with pitch recognizing and open your account to keep a record of your performance: you will educate your brain to recognizes pitches. This amounts to a building and continuous restructuring of neural networks: this is evolution. So, learning implies a hidden evolutionary process. Therefore, while you practice music, you are also making an experiment in evolution. That is why it is very important to keep a record: that would allow you to gauge the power of evolution to get rid of complexity. The Author always have wondered how it is possible to differentiate one note from another as experts do. Practicing with this GUI has allowed him to conclude that in the same way as one distinguishes blue from red, so one can educate the brain to distinguish C from c at once. But to achieve such a tiny advance a lot of training has been needed for him. When notes are added, the resulting complexity seem to explode and huge levels of training have been necessary to pass to the next level.*

88 Challenge. *Furnish our GUI with suitable interfaces to define control parameters.*

89 Challenge. *Furnish the GUI developed in the previous program with automatic recognizing. Example: if the User pastes a DNA sequence, thanks to automatic recognizing, only the corresponding menuItem, DNAToTune, must be enabled while all others must be disabled. Additionally, and if you prefer, the corresponding tune must begin to play at once.*

4.8 Conclusion

Our purpose is to understand music. This means to devise a theory and to use it as guide for composition. If the theory is correct, we will produce nice music. In this regard we were spinning around the **1-theory** that says: each note in isolation is pleasant and music is just a succession of notes. This theory predicts that random notes drawn at random would sound well. For notes of equal duration, this was readily rejected. But with notes of different duration it was found that a non nil proportion of musiTexts were pleasant to the Author. An immediate application of the theory was to devise a rhythm generator, i.e. short musiTexts that are repeated over and over. Surprisingly, many rhythms were found to be good, possibly in

greater proportion than when musiTexts were played only once. Thus, we conclude that human beings have a biologically given propensity for repetitive sounds and dancing. A slight variation of the code allows one to convert a DNA or protein string into a tune. Since our first theory was found highly defective, we proposed the **statistical version of the first theory of music** as a correction: the quality of musical pieces is determined by the frequencies of notes regardless of the order as they appear along the tune. We tested the theory by rearranging the order as notes appear along a tune. No encouraging result in this regard was found by the Author. Anyway, defective as the 1-theory could be, it has an application to pedagogy: students must learn to recognize pitches. In consequence, we devised a program to help students to recognize notes. Let us notice now that learning implies the building and continuous restructuring of appropriate neural networks: this is evolution. That is why the aforementioned application will allow every one in our community to run an evolutionary experiment with him or herself. In consequence, a facility to keep a record of personal performance was furnished. On the other hand, to allow notes of different duration is a first step in regulation. So, some questions with biological importance arise: can DNA based life exist without regulation of gene expression? More mechanistically: how shall be a milieu in order that DNA based life without regulation of gene expression could exist? On another hand, our creative work allows too many options and one of them must be capriciously chosen. This observation immediately forces another question: if life arose by evolution, where are those infinitely many possible genetic codes together with the many other alternatives to DNA and RNA?

Chapter 5

Evolutionary Darwinian music

Darwinian Evolution and the human composer as an ensemble

90 Objective: *to learn how one can program evolution to compose music. Evolution will be implemented here as random mutation + recombination + selection. The first two operations are done automatically but selection will be exercised by the composer.*

5.1 Putting evolution to help

In this section we try out the power of Darwinian evolution to synthesize musiTexts with interesting musical content.

91 Notes of equal duration

To implement evolution we might proceed as follows: an automatic mechanism will propose musiTexts whose notes all have the same duration. Next, the composer listens at composed musiTexts to score them. MusiTexts will be chosen according to their scores. Selected ones are subjected to mutation and recombination whose products will be used for the formation of the new generation that enters a new round of selection. And so on until the composer decides to stop the process.

The following code implements that idea and allows one to experiment to see what happens :

```
//Program H91 DarwinMusic1
//Darwinian evolution is made into a composer.
//Notes have equal duration.
```

```

//WARNING:
//Close all instances of this program
//before running a new one.
import java.io.IOException;
import java.util.Random;

import javax.swing.JFrame;
import javax.swing.JOptionPane;

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneParser;
import abc.ui.swing.JScoreComponent;

public class DarwinMusic1 {

    //The PLAYER converts the tune into
    //a midi message that is sent to
    //hardwared sound synthesizers.
    //If problems appear,
    //an input-out exception must be thrown.
        private static final TunePlayer PLAYER = new TunePlayer();

    //Our one symbol notation. Notes in the first line,
    //our one letter notation in the second.
    //A, _B, B, C #C D _E E F #F G #G A _B B c #c d _e e f #f g #g a _b
    //R S T C M D N E F O G P A Q B c m d n e f o g p a q
        private static final String SUPER_ALPHABET
            = "RSTAMBCNDOEFGQambcndoe fpgq";

        private static final String ALPHABET = "RTCDEOGabcdeo";

    //Number of time unities per measure
        static private int maxTime;
    //The ABC header
        static private String header;

    //Number of time unities per measure 4/4
        private static final int MAX_TIME44 = 8;

```

```

//Header 4/4
    private static final String HEADER44
        = "X:0\n"
        + "T:Song 4/4\n"
        + "M:4/4\n"
        + "L:1/8\n"
        + "K:C\n";

//Number of time unities per measure 6/8
    private static final int MAX_TIME68 = 6;

//Header 6/8
    private static final String HEADER68
        = "X:0\n"
        + "T:Song 6/8\n"
        + "M:6/8\n"
        + "L:1/8\n"
        + "K:C\n";

    static private boolean print;

//We turn on the inbuilt random generator:
    private static final Random R = new Random();

//The musiText is transformed into a tune,
//an ABC4J musical object
    private static Tune ABCmusiTextToTune(String musiText) {
        Tune tune = new TuneParser().parse(musiText);
        return tune;
    }

//A tune is played
    private static void playTune(Tune tune) {
        PLAYER.start();
        PLAYER.play(tune);
    }

//A tune as Tune is drawn
    private static void DrawTune(Tune tune) {
//creates a component that draws the melody on a stave

```

```

        JScoreComponent jscore = new JScoreComponent();
        jscore.setJustification(true);
        jscore.setTune(tune);
        JFrame j = new JFrame();
        j.add(jscore);
        j.pack();
        j.setVisible(true);
    }

//Output a random note from the chosen ALPHABET
    static private char randomNote() {
        int k;
        //A random integer less than the
        //number of chars in the ALPHABET is thrown
        k = R.nextInt(ALPHABET.length());
        //The integer is changed into a symbol of the ALPHABET
        char c = ALPHABET.charAt(k);
        return c;
    }

//Randomness is made into a composer
    static private String randomNotes(int nNotes) {
        //Output initialization
        String stringOfNotes = "";
        //nNotes are generated
        for (int i = 0; i < nNotes; i++) {
            char c = randomNote();
            //The note is added to the output
            stringOfNotes = stringOfNotes + c;
        }
        return stringOfNotes;
    }

//A string in our ALPHABET is rewritten in ABC notation.
//Accidents are prefixes in ABC notation.
//Sharp is denoted by ^, flat by _
    static private String interpreter(String ss) {
        ss = ss.replaceAll("R", "A,");
        ss = ss.replaceAll("S", "_B,");
        ss = ss.replaceAll("T", "B,");
    }

```

```

        ss = ss.replaceAll("M", "#C");
        ss = ss.replaceAll("M", "#c");
        ss = ss.replaceAll("N", "_E");
        ss = ss.replaceAll("n", "_e");
        ss = ss.replaceAll("O", "^F");
        ss = ss.replaceAll("o", "^f");
        ss = ss.replaceAll("P", "^G");
        ss = ss.replaceAll("p", "^g");
        ss = ss.replaceAll("Q", "_B");
        ss = ss.replaceAll("q", "_b");
        return ss;
    }

//=====
//==== Evolution=====
//=====
//Strings that represent ABC tunes
//are synthesized at random
//selected, recombined, mutated
//reproduced.
//This is a fair model of evolution:
//strings encode for an ABC tune (genotype)
//which are listened at by the User (phenotype)
//who gives scores to tunes
//that direct reproduction.
//Global variables.
//They are used all throughout the whole class.
//The number of individuals must be
//less than N_MAX
    static final int N_MAX = 1000;

//Ind is a population of individuals
    static String Ind[] = new String[N_MAX];
//A helping array to keep a clone of Ind1
    static String IndClone[] = new String[N_MAX];
//The score of each tune is kept here
    static double Fitness[] = new double[N_MAX];
//Fitness is used to give a rank to individuals
//that define their reproductive possibilities
    static int Ordered[] = new int[N_MAX];

```

```

//The population has NUMB_IND members.
    static private final int NUMB_IND = 4;
    static private int generation;

    /* We generate NUMB_IND individuals (strings)
maxTime notes long.
Sequences are completely random */
    private static void Initialization() {
        if (print) {
            System.out.println("ORIGINAL POPULATION \n");
        }
        for (int i = 0; i < NUMB_IND; i++) {
            Ind[i] = randomNotes(maxTime);
            if (print) {
                System.out.println("Individual " + i
                    + " = " + Ind[i]);
            }
        }
        for (int i = 0; i < NUMB_IND; i++) {
            Ordered[i] = 0;
        }
    }

//The User assess each tune by giving a score.
//ind is a string that encodes for a tune
//in one-letter notation.
    private static double assessInd(String ind) {
        //ind is translated to ABC notation
        String music = interpreter(ind);
        if (print) {
            System.out.println("Music = " + music);
        }
        //An ABC musiText is assembled
        String musiText = header + music;
        System.out.println("\nmusiText \n" + musiText);
        //musiText is made into an ABC tune
        Tune tune;
        tune = ABCmusiTextToTune(musiText);
        //The score can eventually be drawn

```

```

        //DrawTune(tune);
        playTune(tune);
        // first string entered by user
        String string1;

        //Score given by the User: selection
        double score;
        // We open a dialog box;
        // We obtain the score from User;
        // It is read as String.
        string1 = JOptionPane.showInputDialog(
            "Enter score of tune, "
            + "0 (worst) ,1,2,3,4,5 (best)", 0);
        // We convert String to number of type double
        score = Double.parseDouble(string1);
        System.out.println("Score = " + score);
        return score;
    }

//Individuals are sorted by fitness.
//The fittest is the first.
    private static void sorting() {
        int Champ;
//FitnessCopy[] is a copy of Fitness[]
//used as workbench.
        double FitnessCopy[] = new double[N_MAX];
        for (int i = 0; i < NUMB_IND; i++) {
            Fitness[i] = 1000000;
            FitnessCopy[i] = 1000000;
        }
        if (print) {
            System.out.println("\nSCORING \n");
        }
        for (int i = 0; i < NUMB_IND; i++) {
            if (print) {
                System.out.println("\nind " + i + " " + Ind[i]);
            }
            Fitness[i] = assessInd(Ind[i]);
            FitnessCopy[i] = Fitness[i];
        }
    }

```

```

if (print) {
    System.out.println("\nIndividuals and scores ");
    for (int i = 0; i < NUMB_IND; i++) {
        System.out.println("Individual " + i + " = " + Ind[i]
            + " Score = " + FitnessCopy[i]);
    }
}

//We sort individuals by fitness
// Fitness 0 means the worst, 5 is perfect
if (print) {
    System.out.println("\nSORTING: from the best to the worst:");
}
for (int i = 0; i < NUMB_IND; i++) {
    Champ = 0;
    for (int j = 0; j < NUMB_IND; j++) {
        if (FitnessCopy[j] > FitnessCopy[Champ]) {
            Champ = j;
        }
    }
    //The array Order classifies individuals by fitness.
    //The fittest is number zero
    Ordered[i] = Champ;
    FitnessCopy[Champ] = 0;
    if (print) {
        System.out.println("ind number " + Ordered[i] + " "
            + Ind[Ordered[i]] + " score " + Fitness[Ordered[i]]);
    }
}
//IndClone is a copy of Ind1
//for (int i = 0; i < NUMB_IND; i++)
System.arraycopy(Ind, 0, IndClone, 0, NUMB_IND);

}

//Negative selection:
//Fitness determines probability of death.
//The probability of dead is higher,
//the lower is the fitness.
//Death = replacement of individual by ""

```

```

    private static void death() {
//Min and max are found
        double min = 100;
        double max = -100;
        for (int j = 0; j < NUMB_IND; j++) {
            if (Fitness[j] > max) {
                max = Fitness[j];
            }
            if (Fitness[j] < min) {
                min = Fitness[j];
            }
        }

        if (print) {
            System.out.println(
                "\nFitness determines death probability");
        }
//Fitness determines probability of death
        for (int j = 0; j < NUMB_IND; j++) {
            //Death probability y and fitness x are related by:
            //y = ax +b
            //y(min) = 1, (the worst must die)
            //y(max) = 0, (the best must live)
            //a = 1/(min - max)
            //b = -a max = -max/(min -max)
            double probDeath
                = -Fitness[j] / (max - min) + max / (max - min);
            if (print) {
                System.out.print("ind " + j);
                System.out.print(" Fitness = " + Fitness[j]);

                System.out.println(" ProbDeath = " + probDeath);
            }
            double p = R.nextDouble();
//Death = replacement of individual by ""
            if (p <= probDeath) {
                Ind[j] = "";
            }
        }
        if (print) {

```

```

        System.out.println("The best ind is number " + Ordered[0]
    }
}

//A random not nil individual is chosen.
private static int randomIndiv() {
    boolean cloned = false;
    int k = 0;
    while (cloned == false) {
        k = R.nextInt(NUMB_IND);
        if (!(Ind[k].contentEquals(""))) {
            cloned = true;
        }
    }
    if (print) {
        System.out.println(" cloned from old ind " + k);
    }
    return k;
}

//The pre-population of the new generation
//is built with clones
//of living individuals sampled at random
//with replacement.
private static void Reproduction() {
    if (print) {
        System.out.println("\nREPRODUCTION\n");
    }
    for (int j = 0; j < NUMB_IND; j++) {
        if (print) {
            System.out.print("New ind " + j);
        }
        IndClone[j] = Ind[randomIndiv()];
    }
}
//for (int j = 0; j < NUMB_IND; j++)
//Ind[j] = IndClone[j];
System.arraycopy(IndClone, 0, Ind, 0, NUMB_IND);
}

```

```

//Recombination is enabled.
//individual l is recombined with
//another taken at random.
    private static String recInd(int l) {
        //individual l
        String indl = IndClone[l];
        String head;
        String tail;
        String recombinant ;
        int m = indl.length();
        //Mutation site
        int m1 ;

        //Random site for recombination
        m1 = R.nextInt(m);
        //The head is taken from ind l
        head = indl.substring(0, m1);

        //Random number
        int k = R.nextInt(NUMB_IND);
        //Partner at random
        String partner = IndClone[k];
        //The tail is taken from the partner
        tail = partner.substring(m1);
        //Recombination
        recombinant = head + tail;
        if (print) {
            System.out.println("Indl      = " + indl);
            System.out.println("Head      = " + head);
            System.out.println("Partner   = " + partner);
            System.out.println("Tail      = " + tail);
            System.out.println("Recomb    = " + recombinant + "\n");
        }
        return recombinant;
    }

//The definitive population is a recombinant one.
    private static void recombination() {
        if (print) {
            System.out.println("\nRECOMBINATION\n");
        }
    }

```

```

    }
    //IndClone is a copy of Ind
    System.arraycopy(Ind, 0, IndClone, 0, NUMB_IND);

    //Ind1 is a recombinant version of Ind2
    for (int l = 0; l < NUMB_IND; l++) {
        Ind[l] = recInd(l);
    }
}

//String a is mutated at a random place.
//The string is divided in three regions:
//head + mutation site + tail
private static String pointMutationInd(String a) {
    if (print) {
        System.out.println("Old string = " + a);
    }
    int m = a.length();
    String z = "";
    if (m == 0) {
        return z;
    } else {
        int l = R.nextInt(a.length());
        //Head + mutation site
        z = a.substring(0, l) + randomNote();
        //If mutation is in the last place
        if (l == m - 1); //do nothing
        //else add the tail
        else {
            z = z + a.substring(l + 1);
        }
    }
    if (print) {
        System.out.println("New string = " + z + "\n");
    }
    return z;
}

//All strings are mutated at one site
private static void mutation() {

```

```

        if (print) {
            System.out.println("\nMUTATION");
        }
        // All the individuals mutate.
        for (int j = 0; j < NUMB_IND; j++) {
            //v=original individual
            String v = Ind[j];
            // w = mutated individual.
            String w = pointMutationInd(v);
            Ind[j] = w; //Replacement
        }
    }

//Evolution = selection + reproduction
//+recombination + mutation
    private static void Dynamics() {
        System.out.println("\nGENERATION = " + generation + " \n");
        sorting();
        death();
        Reproduction();
        mutation();
        recombination();
    }

//Evolution is made into a co-composer
    public static void evoMusic() {
        Initialization();

        int NGen = 999;
        for (int n = 1; n <= NGen; n++) {
            generation = n;
            Dynamics();
        }
    }

    public static void main(String[] args) throws IOException {
        print = false;
        //Measure 44 for 4/4, 68 for 6/8
        int measure = 68;
        switch (measure) {

```

```

        case 44:
            maxTime = 2 * MAX_TIME44;
            header = HEADER44;
            break;

        case 68:
            maxTime = 2 * MAX_TIME68;
            header = HEADER68;
            break;
    }
    evoMusic();
}
} //End of main class H91 DarwinMusic1

```

92 Exercise. *Run the program and play with it. To better understand the code, activate the printing option in the main method at the end of the program. Play enough to accept else reject the conclusion of the Author: musiTexts with notes of equal duration has nothing to do with music. Nevertheless, to compose our first program that makes evolution into a compositor was an excellent exercise that expects to be evolved for the better.*

93 Exercise. *Our model of evolution was sorting, death, reproduction, mutation, recombination, in that order. Is that the only possible model? Is that biologically correct? What about the methods that implement those aforementioned tasks? *Answer**

94 Notes with different duration

The next code allows us to see what evolution gives from itself if we add the possibility to encode notes with different duration. Our strategy is to work with super-codons with two codons: one for the note and the other for the duration. Seemingly, all evolutionary processes that were devised for the program with notes of equal duration also work for this new setting with the exception of mutation, which must be correspondingly updated.

```

//Program H94 DarwinMusic2
//Darwinian evolution is made into a composer.

```

```

//Different durations of notes are allowed.
//There is no theoretical guide to compose music.
//WARNING:
//Close all instances of this program
//before running a new one.
//The program outputs ABC musiTexts.
import java.io.IOException;
import java.util.Random;

import javax.swing.JFrame;
import javax.swing.JOptionPane;

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneParser;
import abc.ui.swing.JScoreComponent;

public class DarwinMusic2 {

    //The PLAYER converts the tune into
    //a midi message that is sent to
    //hardwared sound synthesizers.
    //If problems appear,
    //an input-out exception must be thrown.
    private static final TunePlayer PLAYER = new TunePlayer();

    //Our one symbol notation. Notes in the first line,
    //our one letter notation in the second.
    //A, _B, B, C #C D _E E F #F G #G A _B B c #c d _e e f #f g #g a _b b
    //R S T C M D N E F O G P A Q B c m d n e f o g p a q b
    /*private static final String SUPER_ALPHABET
        = "RSTAMBCNDOEFPGQambcndoeifpgq";*/
    //Actual alphabet for work
    private static final String NOTES_ALPHABET = "RTCDEOGabcdeo";

    //Number of time unities per measure
    static private int maxTime;
    //The ABC header
    static private String header;

```

```

//The working alphabet of durations.
    static private String durationsAlphabet;

//Number of time unities per measure 4/4
    private static final int MAX_TIME44 = 8;

//Durations for 4/4
    private static final String DURATIONS_ALPHABET44 = "1111222222448"

//Header 4/4
    private static final String HEADER44
        = "X:0\n"
        + "T:Song 4/4\n"
        + "M:4/4\n"
        + "L:1/8\n"
        + "K:C\n";

//Number of time unities per measure 6/8
    private static final int MAX_TIME68 = 6;

//Durations for 6/8
    private static final String DURATIONS_ALPHABET68 = "111111336";

//Header 6/8
    private static final String HEADER68
        = "X:0\n"
        + "T:Song 6/8\n"
        + "M:6/8\n"
        + "L:1/8\n"
        + "K:C\n";

    static private boolean print;
    static private boolean printMutation;

//We turn on the inbuilt random generator:
    private static final Random R = new Random();

//The musiText is transformed into a tune,
//an ABC4J musical object
    private static Tune ABCmusiTextToTune(String musiText) {

```

```
        Tune tune = new TuneParser().parse(musiText);
        return tune;
    }

//A tune is played
    private static void playTune(Tune tune) {
        PLAYER.play(tune);
    }

//A tune as Tune is drawn
    private static void DrawTune(Tune tune) {
//creates a component that draws the melody on a stave
        JScoreComponent jscore = new JScoreComponent();
        jscore.setJustification(true);
        jscore.setTune(tune);
        JFrame j = new JFrame();
        j.add(jscore);
        j.pack();
        j.setVisible(true);
    }

//Output a random note from the chosen alphabet
    static private char randomNote() {
        if (printMutation) {
            System.out.println("RandomNote");
        }
        int k;
        //A random integer less than the
        //number of chars in the alphabet is thrown
        k = R.nextInt(NOTES_ALPHABET.length());
        //The integer is changed into a symbol of the alphabet
        char c = NOTES_ALPHABET.charAt(k);
        return c;
    }

//Output a random duration from the chosen alphabet
    static private char randomDuration() {
        if (printMutation) {
            System.out.println("RandomDuration");
        }
    }
}
```

```

        int k;
//A random integer less than the
//number of chars in the alphabet is thrown
        k = R.nextInt(durationsAlphabet.length());
//The integer is changed into a symbol of the alphabet
//The integer is changed into a
//char that represents a duration number
        char c = durationsAlphabet.charAt(k);

        return c;
    }

//Randomness is made into a composer.
//Notes occupy even places in the music of the musiText.
//durations odd places.
    static private String randomNotes(int maxTime,
        String header, String durationsAlphabet) {
        boolean go = true;
//Output initialization
        String stringOfNotes = "";
        int k;
//CumulativeTime must not surpass n
        for (int i = 0; i < 7; i++) {
            //A random note is generated:
            char c = randomNote();
            //The note is added to the output
            stringOfNotes = stringOfNotes + c;
            //A random duration is generated
            c = randomDuration();
            //The duration is added to the output
            stringOfNotes = stringOfNotes + c;
        }
        return stringOfNotes;
    }

//A string in our alphabet is rewritten in ABC notation.
//Accidents are prefixes in ABC notation.
//Sharp is denoted by ^, flat by _
    static private String interpreter(String ss) {
        ss = ss.replaceAll("R", "A,");
    }

```

```

        ss = ss.replaceAll("S", "_B,");
        ss = ss.replaceAll("T", "B,");
        ss = ss.replaceAll("M", "#C");
        ss = ss.replaceAll("M", "#c");
        ss = ss.replaceAll("N", "_E");
        ss = ss.replaceAll("n", "_e");
        ss = ss.replaceAll("O", "^F");
        ss = ss.replaceAll("o", "^f");
        ss = ss.replaceAll("P", "^G");
        ss = ss.replaceAll("p", "^g");
        ss = ss.replaceAll("Q", "_B");
        ss = ss.replaceAll("q", "_b");
        return ss;
    }

//=====
//==== Evolution=====
//=====
//Strings that represent ABC tunes
//are synthesized at random
//selected, recombined, mutated
//reproduced.
//This is a fair model of evolution:
//strings encode for an ABC tune (genotype)
//which are listened at by the User (phenotype)
//who gives scores to tunes
//that direct reproduction.
//Global variables.
//They are used all throughout the whole class.
//The number of individuals must be
//less than N_MAX
    static final int N_MAX = 1000;

//Ind is a population of individuals
    static final String IND[] = new String[N_MAX];
//A helping array to keep a clone of Ind1
    static final String IND_CLONE[] = new String[N_MAX];
//The score of each tune is kept here
    static final double FITNESS[] = new double[N_MAX];
//FITNESS is used to give a rank to individuals

```

```

//that define their reproductive possibilities
    static final int ORDERED[] = new int[N_MAX];
//The population has NUMB_IND members.
    static private final int NUMB_IND = 4;
    static private int generation;

    /* We generate NUMB_IND individuals (strings)
maxTime notes long.
Sequences are completely random */
    private static void Initialization() {
        if (print) {
            System.out.println("ORIGINAL POPULATION \n");
        }
        for (int i = 0; i < NUMB_IND; i++) {
            IND[i] = randomNotes(maxTime, header, durationsAlphabet);
            if (print) {
                System.out.println("Individual " + i
                    + " = " + IND[i]);
            }
        }
        for (int i = 0; i < NUMB_IND; i++) {
            ORDERED[i] = 0;
        }
    }

//The User assess each tune by giving a score.
//ind is a string that encodes for a tune
//in one-letter notation.
    private static double assessInd(String ind) {
        //ind is translated to ABC notation
        String music = interpreter(ind);
        if (print) {
            System.out.println("Music = " + music);
        }
        //An ABC musiText is assembled
        String musiText = header + music;
        System.out.println("\n musiText \n" + musiText);
        //musiText is made into an ABC tune
        Tune tune;

```

```

    tune = ABCmusiTextToTune(musiText);
    //The score can eventually be drawn
    //DrawTune(tune);
    playTune(tune);
    // first string entered by user
    String string1;

    //Score given by the User: selection
    double score;
    // We open a dialog box;
    // We obtain the score from User;
    // It is read as String.
    string1 = JOptionPane.showInputDialog(
        "Enter score of tune, "
        + "0 (worst) ,1,2,3,4,5 (best)", 0);
    // We convert String to number of type double
    score = Double.parseDouble(string1);
    System.out.println("Score = " + score);
    return score;
}

//Individuals are sorted by fitness.
//The fittest is the first.
    private static void sorting() {
        int Champ;
//FITNESSCopy[] is a copy of FITNESS[]
//used as workbench.
        double FITNESSCopy[] = new double[N_MAX];
        for (int i = 0; i < NUMB_IND; i++) {
            FITNESS[i] = 1000000;
            FITNESSCopy[i] = 1000000;
        }
        if (print) {
            System.out.println("\nSCORING \n");
        }
        for (int i = 0; i < NUMB_IND; i++) {
            if (print) {
                System.out.println("\nind " + i + " " + IND[i]);
            }
            FITNESS[i] = assessInd(IND[i]);

```

```

        FITNESSCopy[i] = FITNESS[i];
    }
    if (print) {
        System.out.println("\nIndividuals and scores ");
        for (int i = 0; i < NUMB_IND; i++) {
            System.out.println("Individual " + i + " = " + IND[i]
                + " Score = " + FITNESSCopy[i]);
        }
    }

    //We sort individuals by fitness
    // FITNESS 0 means the worst, 5 is perfect
    if (print) {
        System.out.println("
            + "\nSORTING: from the best to the worst: ");
    }
    for (int i = 0; i < NUMB_IND; i++) {
        Champ = 0;
        for (int j = 0; j < NUMB_IND; j++) {
            if (FITNESSCopy[j] > FITNESSCopy[Champ]) {
                Champ = j;
            }
        }
        //The array Order classifies individuals by fitness.
        //The fittest is number zero
        ORDERED[i] = Champ;
        FITNESSCopy[Champ] = 0;
        if (print) {
            System.out.println("ind number " + ORDERED[i] + " "
                + IND[ORDERED[i]] + " score " + FITNESS[ORDERED[i]]);
        }
    }
    //IND_CLONE is a copy of Ind1
    System.arraycopy(IND, 0, IND_CLONE, 0, NUMB_IND);

}

//Negative selection:
//FITNESS determines probability of death.
//The probability of dead is higher,
```

```

//the lower is the fitness.
//Death = replacement of individual by ""
    private static void death() {
//Min and max are found
        double min = 100;
        double max = -100;
        for (int j = 0; j < NUMB_IND; j++) {
            if (FITNESS[j] > max) {
                max = FITNESS[j];
            }
            if (FITNESS[j] < min) {
                min = FITNESS[j];
            }
        }

        if (print) {
            System.out.println(
                "\nFITNESS determines death probability");
        }
//FITNESS determines probability of death
        for (int j = 0; j < NUMB_IND; j++) {
            //Death probability y and fitness x are related by:
            //y = ax + b
            //y(min) = 1, (the worst must die)
            //y(max) = 0, (the best must live)
            //a = 1/(min - max)
            //b = -a max = -max/(min -max)
            double probDeath
                = -FITNESS[j] / (max - min) + max / (max - min);
            if (print) {
                System.out.print("ind " + j);
                System.out.print(" FITNESS = " + FITNESS[j]);

                System.out.println(" ProbDeath = " + probDeath);
            }
            double p = R.nextDouble();
//Death = replacement of individual by ""
            if (p <= probDeath) {
                IND[j] = "";
            }
        }
    }

```

```

    }
    if (print) {
        System.out.println("
            + "The best ind is number " + ORDERED[0]);
    }
}

//A not nil random individual is chosen.
private static int randomIndiv() {
    boolean cloned = false;
    int k = 0;
    while (cloned == false) {
        k = R.nextInt(NUMB_IND);
        if (!"".equals(IND[k])) {
            cloned = true;
        }
    }
    if (print) {
        System.out.println(
            " cloned from old ind " + k);
    }
    return k;
}

//The pre-population of the new generation
//is built with clones
//of living individuals sampled at random
//with replacement.
private static void Reproduction() {
    if (print) {
        System.out.println("\nREPRODUCTION\n");
    }
    for (int j = 0; j < NUMB_IND; j++) {
        if (print) {
            System.out.print("New ind " + j);
        }
        IND_CLONE[j] = IND[randomIndiv()];
    }
}

```

```

        System.arraycopy(IND_CLONE, 0, IND, 0, NUMB_IND);
    }

//Recombination is enabled.
//individual 1 is recombined with
//another taken at random.
    private static String recInd(int l) {
        //individual 1
        String ind1 = IND_CLONE[l];
        String head ;
        String tail ;
        String recombinant ;
        int m = ind1.length();
        //Mutation site
        int m1 ;

        //Random site for recombination
        m1 = R.nextInt(m);
        //The head is taken from ind 1
        head = ind1.substring(0, m1);

        //Random number
        int k = R.nextInt(NUMB_IND);
        //Partner at random
        String partner = IND_CLONE[k];
        //The tail is taken from the partner
        tail = partner.substring(m1);
        //Recombination
        recombinant = head + tail;
        if (print) {
            System.out.println("Ind1      = " + ind1);
            System.out.println("Head      = " + head);
            System.out.println("Partner   = " + partner);
            System.out.println("Tail      = " + tail);
            System.out.println("Recomb    = " + recombinant + "\n");
        }
        return recombinant;
    }

//The definitive population is a recombinant one.

```

```

private static void recombination() {
    if (print) {
        System.out.println("\nRECOMBINATION\n");
    }
    //Ind2 is a copy of Ind1
    System.arraycopy(IND, 0, IND_CLONE, 0, NUMB_IND);

    //Ind1 is a recombinant version of Ind2
    for (int l = 0; l < NUMB_IND; l++) {
        IND[l] = recInd(l);
    }
}

//String a is mutated at a random place.
//The string is divided in three regions:
//head + mutation site + tail
private static String pointMutationInd(String a) {
    if (printMutation) {
        System.out.println(
            "Old string = " + a);
    }
    int m = a.length();
    String z = "";
    if (m == 0) {
        return z;
    } else {
        int l = R.nextInt(a.length());
        char c ;
        //A number l is even if l modulus 2 is zero
        //if l is even, we have a note
        if (l % 2 == 0) {
            c = randomNote();
        } else {
            c = randomDuration(); //else a duration
        }
        if (printMutation) {
            System.out.println(
                "l = " + l + " c = " + c);
        }
    }
    //Head + mutation site

```

```

        z = a.substring(0, l) + c;
        //If mutation is in the last place
        if (l == m - 1); //do nothing
        //else add the tail
        else {
            z = z + a.substring(l + 1);
        }
    }
    if (printMutation) {
        System.out.println(
            "New string = " + z + "\n");
    }
    return z;
}

//All strings are mutated at one site
private static void mutation() {
    if (printMutation) {
        System.out.println("\nMUTATION\n");
    }
    // All the individuals mutate.
    for (int j = 0; j < NUMB_IND; j++) {
        //v=original individual
        String v = IND[j];
        // w = mutated individual.
        String w = pointMutationInd(v);
        IND[j] = w; //Replacement
    }
}

//Evolution = selection + reproduction
//+recombination + mutation
private static void Dynamics() {
    System.out.println("\nGENERATION = " + generation + " \n");
    sorting();
    death();
    Reproduction();
    mutation();
    recombination();
}

```

```

//Evolution is made into a co-composer
public static void evoMusic() {
    Initialization();
    int NGen = 999;
    for (int n = 1; n <= NGen; n++) {
        generation = n;
        Dynamics();
    }
}

public static void main(String[] args) throws IOException {
    PLAYER.start();
    print = false;
    printMutation = false;
    //Measure 44 for 4/4, 68 for 6/8
    int measure = 68;
    switch (measure) {
        case 44:
            maxTime = 2 * MAX_TIME44;
            header = HEADER44;
            durationsAlphabet = DURATIONS_ALPHABET44;
            break;

        case 68:
            maxTime = 2 * MAX_TIME68;
            header = HEADER68;
            durationsAlphabet = DURATIONS_ALPHABET68;
            break;
    }
    evoMusic();
}
} //End of main class H94 DarwinMusic2

```

95 Exercise. *Run the program and play with the code. Hints: change the alphabet of durations that an allegro, preferentially with short notes, could appear. Or change the alphabet of notes that a song with low pitch could preferentially arise.*

Play enough to agree else disagree with the conclusion of the Author: strings of notes of unequal duration can eventually have an interesting sound and with the selection of some random changes that quality might be enhanced although not too much. Concretely, an evolutionary process with a population of 4 individuals that are subject to mutation and selection over some 10 to 15 generations are enough to produce a relatively interesting musiText. Anyway, to end with high quality music is seemingly a matter of eternal patience that goes beyond 20 generations. An apparent problem is that the Author is seemingly inconsistent over time. So, the need arises to devise and include automatic scores for musical quality.

5.2 Conclusion

Science longs for the kind of simplicity that covers as much universality as possible. In this regard, we rapidly rejected the simplest theory of music: notes as separate entities are all to music. This theory is plainly false for notes of equal duration but when notes are allowed to vary duration, the theory begins to be interesting although very primitive. That frame was kind enough to allow us to implement Darwinian evolution as a quality improver of tunes that were synthesized at random. Nevertheless, our patience was not enough to have evolution helping us in the art of musical composition.

Chapter 6

Evolutionary Lamarckian music

Teaching Evolution to be a better helper

96 Introduction. *We have seen that Darwinian evolution could be eventually a help for a musician if he or she has a lot of patience that most humans lack. What else do we have?*

97 Objective: *to arm evolution with a simple hint from human knowledge in order to be a better helper for music composition. With this we enter the realm of Lamarckian evolution.*

6.1 Lamarckian evolution

Darwin and Lamarck are both evolutionist but quite different one from the other.

98 Two insights

Our examination in hindsight of the ideas of **Darwin** and **Lamarck** runs as follows (See, Vol VIII):

In Darwinian evolution (updated to include the idea of Mendel about genes) initialization of the population and changes of its genetic composition are all at random and the only feedback from the environment is through differential death and/or reproduction. If that feedback is consistent over time, individuals are expected to get fitted to it. We have witnessed through the examination of the two previous programs that Darwinian evolution is very slow for a typical human being that aspires to composing music just on the fly. Thus, we need something else. So, what is there apart from Darwinism?

The powerful mind of Lamarck postulated that evolution to be a reliable explanation of extant life should be helped somehow. Actually, he envisaged alchemy as the frame to add details to this idea. So, he rejected Darwinism in favor of an extended evolution that involves the spiritual universe. In relation with us, the brilliant idea is that evolution can be helped. This idea defines Lamarckism in our community: if one knows that the final product of evolution is to be fitted to the environment, then one can accelerate that process if only one makes more probably the correct changes that eventually can increase the fitness of individuals.

Let us give the first step to show how to use Lamarckism in our musical project.

99 *The interval space*

We have worked with a very natural, simple and not too bad definition of music: it is a succession of notes, which are described by pitch and duration. Concretely: music is a succession of instructions that say what key of the piano to press and for how much time. Let us now pay attention to a very simple change of stand that is based on human experience and being one of the most elementary is also one of the most powerful principles of music design: music is, apart from the duration of each note, a succession of instructions to rise or lower the pitch.

We can understand the whole deal if we consider the ABC-musiText CECEC4. It is a tiny tune. Now, let consider GBGBG4. It is another musiText. What they have in common? They encode the same tune but the second form has a higher pitch than the former. Formally, we say: GBGBG4 is the trasposition of CECEC4 7 semitones up, because the difference in pitch between two neighboring keys of the piano is a semitone and if we add 7 semitones to each note of CECEC4, we get GBGBG4.

When we say that CECEC4 and GBGBG4 encode the same tune, we are declaring that music is not a succession of sounds but of **intervals**, i.e., of instructions to low or to rise pitch. Of course, we need an initial note to begin the process. Let us see one form as this can be done operationally:

If we begin with C and go 4 semitones up, we get E, and then we must lower 4 semitones to get C and then 4 semitones up to get E and All this can be denoted as follows:

$$CECEC4 = C \rightarrow +4S \rightarrow -4S \rightarrow +4S \rightarrow -4S, duration \times 4.$$

If we declare that the plus sign + is redundant, then we could write:

$$CECEC4 = C \rightarrow 4S \rightarrow -4S \rightarrow 4S \rightarrow -4S, duration \times 4.$$

We can compress the notation one step further:

$$CECEC4 = C4S - 4S4S - 4S4.$$

By the same token, we have:

$$CEGC4 = C4S3S - 7S4$$

As we see, to say that music is a succession of sounds is equivalent to say that music is an initial note ensued by a succession of intervals. Nevertheless, the second formulation is very appropriate to include a Lamarckian hint, i.e., a set of instructions that is based on the study of the human nature and that accelerates the process of composition of pleasant music.

To discover the biologically given hint, we hear and score each interval. The next ABC tune is appropriate to do that:

```
X:1
T:Intervals
K: C
%%0S
CCCCCCCCCCCCCCC
%%1S
C^CC^C C^CC^CC^CC^CCCC
%%2S
CDCDCDCDCDCDCDC
%%3S
C_EC_EC_EC_EC_EC_EC_EC_EC
%%4S
CECECECECECECE
%%5S
CFCFCFCFCFCFC
%%6S
C^FC^FC^FC^FC^FC^FC^FC
%%7S
CGCGCGCGCG
%%8S
C^GC^GC^GC^GC^GC^GC^GC^GC
%%9S
CACACACACACACACACACAC
%%10S
C_BC_BC_BC_BC_BC_BC_B
%%11
CBCBCBCBCBCBCBCBC
%%12
CcCcCcCcCcCcCcCcCcCc
```

After some work, the Author ended with the following classification of intervals beginning with the most pleasant and ended with the most unpleasant:

```

X:1
T:Intervals by rank according to pleasantness
K: C
%%=====
%%The best: rank one.
%%0S
CCCCCCCCCCCCCCC
%%4S
CECECECECECECECCCC
%%5S
CFCFCFCFCFCFCCCC
%%7S
CGCGCGCGCGCCCC
%%12
CcCcCcCcCcCcCcCcCcCcCcCCCC
%%=====
%%Rank two:
%%9S
CACACACACACACACACACACACACCCCC
%%2S
CDCDCDCDCDCDCDCDCDCDC
%%=====
%%Rank three
%%8S
C^GC^GC^GC^GC^GC^GC^GC^GC^GC^GCCCCC
%%3S
C_EC_EC_EC_EC_EC_EC_EC_EC_EC_ECCCC
%%6S
C^FC^FC^FC^FC^FC^FC^FC^FC^FC^FCCCC
%%=====
%%Rank four:
%%1S
C^CC^C C^CC^CC^CC^CCCC
%%10S
C_BC_BC_BC_BC_BC_BC_BC_BCCCC
%%11
CBCBCBCBCBCBCBCBCBCBCBCCCC

```

100 Exercise. *Humans are coarsely the same but nevertheless small differences exist. So, a problem that touches the human nature may have many solutions and even the same individual can propose different solution over time. This is particularly true for music. So, play the tune above with the intervals and rank them according to pleasantness such as you perceive them. Most possibly you will be very uncertain in regard with the correct position of some intervals. Do not worry too much about that, rather accept yourself such as you are and take a decision without too much thought.*

101 Challenge. *Our guess in relation with the previous classification has been to assume that if a given upward interval is pleasant in some measure, then the corresponding downward one also is pleasant in the same measure. Would you test this assumption?*

102 Challenge. *Devise an application to help students to recognize the interval between a pair of notes. Hint: reuse the code for pitch recognizing of previous chapter.*

103 Evolution over intervals. *We implement evolution but not over notes but over intervals. Pleasant intervals are given higher probability of appearance. A problem appears: our musITexts encode for instruction to lower or rise pitch. But given that the ensemble of notes is bounded, one cannot assure that overflowing does not occur when one implements mutation. The Reader is invited to study the code to see how these troubles were dealt with although unsatisfactorily. The code follows:*

```
//Program H103 LamarckMusic1
//Lamarckian evolution is made into a composer.
//Different durations of notes are allowed.
//Theoretical guide to composing music:
//Notes are pleasant in isolation but when
//they go one after another in a tune,
//the interval, the number of semitones in
//within notes, is what matters.
//Our encoding in interval notation:
//Example: T6+33-51+91+83-21-51+01
// an initial note + durations: T6,
// plus a string of codons with three letters:
//the first is a sign to indicate the direction
//of the interval: up else down,
```

```

//the second is an interval in semitones,
//the third is a duration.
//Example:
//T6+33 means: begins with B, with a duration of 6,
//prepare to rise pitch,
//move 3 semitones up to D
//and play it for 3 units of time.
//The program outputs ABC musiTexts.
//The performance of this program leaves
//to much to be desired.
//The reason is that the interval notation
//is not appropriate for mutation
//because any mutation affects all that comes next.
//Overflow problems arise, say resultant notes
//are outside allowed range and/or scale.
//To search for something better is a must for the Reader.
import java.util.Random;

import javax.swing.JFrame;
import javax.swing.JOptionPane;

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneParser;
import abc.ui.swing.JScoreComponent;

public class LamarckMusic1 {

    //The PLAYER converts the tune into
    //a midi message that is sent to
    //hardware sound synthesizers.
    //If problems appear,
    //an input-out exception must be thrown.
    private static final TunePlayer PLAYER = new TunePlayer();

    //Our one symbol notation. Notes in the first line,
    //our one letter notation in the second.
    //A, _B, B, C #C D _E E F #F G #G A _B B c #c d _e e f #f g #g a _b
    //R S T C M D N E F O G P A Q B c m d n e f o g p a q
    private static final String SUPER_ALPHABET

```

```

        = "RSTCMDNEFOGPAQBcmdnefogpaqb";
//Actual alphabet for work (scale G)
    private static final String ACTUAL_ALPHABET = "RTCDEOGABcdeo";

//The ABC header
    static private String header;

//The working alphabet of durations.
    static private String durationsAlphabet;

//Durations for 4/4
    private static final String DURATIONS_ALPHABET44 = "1111222222448";

//Header 4/4
    private static final String HEADER44
        = "X:0\n"
        + "T:Song 4/4\n"
        + "M:4/4\n"
        + "L:1/8\n"
        + "K:C\n";

//Durations for 6/8
    private static final String DURATIONS_ALPHABET68 = "111111336";

//Header 6/8
    private static final String HEADER68
        = "X:0\n"
        + "T:Song 6/8\n"
        + "M:6/8\n"
        + "L:1/8\n"
        + "K:C\n";

    /*
    *
    Intervals in semitones by rank according to pleasantness
    (According to the Author):
    The best: rank zero: 0, 4, 5, 7, 12.
    Rank one: 9, 2
    %%Rank two: 8, 3, 6.
    %%Rank three: 1, 10, 11

```

```

        */

    /*
    * The 13 INTERVALS in one letter notation:
    0 1 2 3 4 5 6 7 8 9 10 11 12
    0 1 2 3 4 5 6 7 8 9  a  b  c
        */
        private static final String INTERVALS = "0123456789abc";
//Rank of the 13 INTERVALS of the scale
//{0,3,1,2,0,0,2,0,2,1,3,3,0}

        private static final String RANK_ZERO = "0457c";
        private static final String RANK_ONE = "92";
        private static final String RANK_TWO = "836";
        private static final String RANK_THREE = "1ab";
//probabilities of appearance depends on rank:
        private static final double PROB[] = {0.4, 0.3, 0.2, 0.1};

//The interval can be upwards (1) or downwards (-1);
        static private char signInterval = 1;
        static private int lastNoteIndex;
        static private boolean print;
        static private boolean printThis;

//We turn on the inbuilt random generator:
        private static final Random R = new Random();

//The musiText is transformed into a tune,
//an ABC4J musical object
        private static Tune ABCmusiTextToTune(String musiText) {
            Tune tune = new TuneParser().parse(musiText);
            return tune;
        }

//A tune is played
        private static void playTune(Tune tune) {
            PLAYER.play(tune);
        }

//A tune as Tune is drawn

```

```
private static void DrawTune(Tune tune) {
//creates a component that draws the melody on a stave
    JScoreComponent jscore = new JScoreComponent();
    jscore.setJustification(true);
    jscore.setTune(tune);
    JFrame j = new JFrame();
    j.add(jscore);
    j.pack();
    j.setVisible(true);
}

//Output a random letter from the chosen alphabet
static private char randomLetter(String alphabet) {
    int k;
    //A random integer less than the
    //number of chars in the alphabet is thrown
    k = R.nextInt(alphabet.length());
    //The integer is changed into a letter of the alphabet
    char c = alphabet.charAt(k);
    return c;
}

//An interval, a number, is proposed
//with a probability according to its rank.
static private int proposeInterval() {
    int k;
    Character f = ' ';
    double w = R.nextDouble();
    if (w > 1 - PROB[3]) {
        f = randomLetter(RANK_THREE);
    }
    if (w < PROB[0] + PROB[1] + PROB[2]) {
        f = randomLetter(RANK_TWO);
    }
    if (w < PROB[0] + PROB[1]) {
        f = randomLetter(RANK_ONE);
    }
    if (w < PROB[0]) {
        f = randomLetter(RANK_ZERO);
    }
}
```

```
k = Character.getNumericValue(f);
if (f == 'a') {
    k = 10;
}
if (f == 'b') {
    k = 11;
}
if (f == 'c') {
    k = 12;
}
if (f == 'd') {
    k = 13;
}
if (f == 'e') {
    k = 14;
}
if (f == 'f') {
    k = 15;
}
if (f == 'g') {
    k = 16;
}
if (f == 'h') {
    k = 17;
}
if (f == 'i') {
    k = 18;
}
if (f == 'j') {
    k = 19;
}
if (f == 'k') {
    k = 20;
}
if (f == 'l') {
    k = 21;
}
if (f == 'm') {
    k = 22;
```

```

    }
    if (f == 'n') {
        k = 23;
    }
    if (f == 'o') {
        k = 24;
    }
    if (print) {
        System.out.println("w = " + w + " k = " + k);
    }
    return k;
}

//Output a random interval as string:
//the first char is the sign: + upwards, - downwards.
//the second is the interval as a one letter number.
//The rank of INTERVALS determines its
//probability of occurrence.
//The output must be compatible with both
//SUPER_ALPHABET and ACTUAL_ALPHABET,
//this is determined by actualNoteIndex
//the index of the last note in the SUPER_ALPHABET
    static private String randomInterval() {
        if (print) {
            System.out.println("RandomInterval");
        }
    }
//Output
String randomInterval = "";
String newNote;
int newNoteIndex = lastNoteIndex;
int direction;
boolean generated = false;
char intervalOneLetter;
while (generated == false) {
    // a direction is created
    double p = R.nextDouble();
    if (p > 0.5) {
        signInterval = '+';
        direction = 1;
    } else {

```

```

        signInterval = '-';
        direction = -1;
    }
    //a random number is generated
    int n = proposeInterval();

    int newTryIndex = newNoteIndex + direction * n;
    //The corresponding note is checked for compatibility
    if ((newTryIndex >= 0)
        & (newTryIndex < SUPER_ALPHABET.length())) {
        newNote = "" + SUPER_ALPHABET.charAt(newTryIndex
    );
        //Is the new note in the actual alphabet of notes?
        if (ACTUAL_ALPHABET.contains(newNote)) {
            generated = true;
            newNoteIndex = newTryIndex;
            lastNoteIndex = newNoteIndex;
            //Interval in one letter notation
            intervalOneLetter = INTERVALS.charAt(n);
            randomInterval = ""
                + signInterval + intervalOneLetter;
            if (print) {
                System.out.println("New note = " + newNote);
            }
        }
    }
    return randomInterval;
}

//Randomness is made into a composer.
//This method outputs a random musiText in interval notation:
//The first note comes as a note in one letter notation
//plus a duration
//Notes occupy even places in the music of the musiText
//while durations go in odd places.
static private String randomMTIN() {
    //MusiText in interval notation
    String m;
    //An initialNote begins the musiText

```

```

String c = "" + randomLetter(ACTUAL_ALPHABET);
if (print) {
    System.out.println("First letter = " + c);
}
int firstNoteIndex = SUPER_ALPHABET.indexOf(c);
//Duration is added
m = "" + c + randomLetter(durationsAlphabet);
lastNoteIndex = firstNoteIndex;
//For the following notes
//INTERVALS and durations are next added
for (int i = 0; i < 7; i++) {
    //An interval is generated
    c = randomInterval();
    //The interval is added to the output
    m = m + c;
    //A random duration is generated
    c = "" + randomLetter(durationsAlphabet);
    //The duration is added to the output
    m = m + c;
}
if (print) {
    System.out.println("Random ind = " + m);
}
return m;
}

//Input: a musiText in interval one letter notation
//output: a musiText in ABC one letter notation
private static String transcription(String a) {
    int m = a.length();
    //Output musiText in one letter ABC notation
    String z;
    //The first two letters encode the initial note
    //and duration, these are already in ABC one letter notation
    z = a.substring(0, 2);
    int newNoteIndex = SUPER_ALPHABET.indexOf(a.charAt(0));
    //The rest of the string encodes INTERVALS and durations.
    //Our codons have three letters:
    //one for the direction of the interval,

```

```
//the other for the interval,  
//and the third for duration  
  
char newNote;  
int k = 0;  
for (int i = 2; i < m;) {  
    //The first letter is a sign:  
    //the direction of the interval.  
    char c = a.charAt(i++);  
    //The second letter denotes an interval  
    Character f = a.charAt(i++);  
    //The third note represents a duration  
    char third = a.charAt(i++);  
    if (print) {  
        System.out.println("char 0,1,2 = " + c  
            + " " + f + " " + third);  
    }  
    //Direction is decoded  
    int direction;  
    if (c == '+') {  
        direction = 1;  
    } else {  
        direction = -1;  
    }  
    //Interval is decoded  
    int l = Character.getNumericValue(f);  
    if ((l >= 0) & (l < 10)) {  
        k = l;  
    } else {  
        if (f == 'a') {  
            k = 10;  
        }  
        if (f == 'b') {  
            k = 11;  
        }  
        if (f == 'c') {  
            k = 12;  
        }  
        if (f == 'd') {  
            k = 13;  
        }  
    }  
}
```

```
    }
    if (f == 'e') {
        k = 14;
    }
    if (f == 'f') {
        k = 15;
    }
    if (f == 'g') {
        k = 16;
    }
    if (f == 'h') {
        k = 17;
    }
    if (f == 'i') {
        k = 18;
    }
    if (f == 'j') {
        k = 19;
    }
    if (f == 'k') {
        k = 20;
    }
    if (f == 'l') {
        k = 21;
    }
    if (f == 'm') {
        k = 22;
    }
    if (f == 'n') {
        k = 23;
    }
    if (f == 'o') {
        k = 24;
    }
}
if (print) {
    System.out.println(" k = " + k);
}

newNoteIndex = newNoteIndex + direction * k;
```

```

        newNote = SUPER_ALPHABET.charAt(newNoteIndex);
        //The new note is assembled
        z = z + newNote + third;

    }
    if (print) {
        System.out.println("mTIntNotation = " + a
            + "\n mT in one letter notation = = " + z + "\n");
    }
    return z;
}

//A musiText in interval notation is rewritten
//as a normal ABC musiText.
//Notes occupy even positions, beginning from zero,
//and durations odd ones.
//Accidents are prefixes in ABC notation.
//Sharp is denoted by ^, flat by _
    static private String interpreter(String ss) {
        ss = ss.replaceAll("R", "A,");
        ss = ss.replaceAll("S", "_B,");
        ss = ss.replaceAll("T", "B,");
        ss = ss.replaceAll("M", "#C");
        ss = ss.replaceAll("M", "#c");
        ss = ss.replaceAll("N", "_E");
        ss = ss.replaceAll("n", "_e");
        ss = ss.replaceAll("O", "^F");
        ss = ss.replaceAll("o", "^f");
        ss = ss.replaceAll("P", "^G");
        ss = ss.replaceAll("p", "^g");
        ss = ss.replaceAll("Q", "_B");
        ss = ss.replaceAll("q", "_b");
        //ss = ss.replaceAll("+", "");
        //ss = ss.replaceAll("-", "");
        return ss;
    }

//=====
//==== Evolution=====
//=====

```

```

//Strings that represent ABC tunes
//are synthesized at random
//selected, recombined, mutated
//reproduced.
//This is a fair model of evolution:
//strings encode for an ABC tune (genotype)
//which are listened at by the User (phenotype)
//who gives scores to tunes
//that direct reproduction.
//Global variables.
//They are used all throughout the whole class.
//The number of individuals must be
//less than N_MAX
    static final int N_MAX = 1000;

//Ind is a population of individuals
    static String Ind[] = new String[N_MAX];
//A helping array to keep a clone of Ind1
    static String IndClone[] = new String[N_MAX];
//The score of each tune is kept here
    static double Fitness[] = new double[N_MAX];
//Fitness is used to give a rank to individuals
//that define their reproductive possibilities
    static int Ordered[] = new int[N_MAX];
//The population has NUM_IND members.
    private static final int NUM_IND = 4;
    static private int generation;

    /* We generate NUM_IND individuals (strings)
maxTime notes long.
Sequences are completely random */
    private static void Initialization() {
        if (print) {
            System.out.println("ORIGINAL POPULATION \n");
        }
        for (int i = 0; i < NUM_IND; i++) {
            Ind[i] = randomMTIN();
            if (print) {
                System.out.println("Individual " + i

```

```

        + " = " + Ind[i]);
    }
}
for (int i = 0; i < NUM_IND; i++) {
    Ordered[i] = 0;
}
}

private static void printPopulation() {
    for (int i = 0; i < NUM_IND; i++) {
        System.out.println(Ind[i]);
    }
}

//The User assess each tune by giving a score.
//ind is a string that encodes for a tune
//in one-letter notation.
private static double assessInd(String indNI) {
    //A musicText in Interval Notation
    //is translated into normal ABC notation
    String ABCMusitext = transcription(indNI);
    //ind is translated to ABC notation
    String music = interpreter(ABCMusitext);
    if (print) {
        System.out.println("Music = " + music);
    }
    //An ABC musiText is assembled
    String musiText = header + music;
    System.out.println("\nmusiText \n" + musiText);
    //musiText is made into an ABC tune
    Tune tune;
    tune = ABCmusiTextToTune(musiText);
    //The score can eventually be drawn
    //DrawTune(tune);
    playTune(tune);
    // first string entered by user
    String string1;

    //Score given by the User: selection
    double score;

```

```

    // We open a dialog box;
    // We obtain the score from User;
    // It is read as String.
    string1 = JOptionPane.showInputDialog(
        "Enter score of tune, "
        + "0 (worst) ,1,2,3,4,5 (best)", 0);
    // We convert String to number of type double
    score = Double.parseDouble(string1);
    System.out.println("Score = " + score);
    return score;
}

//Individuals are sorted by fitness.
//The fittest is the first.
private static void sorting() {
    if (printThis) {
        printPopulation();
    }
    int Champ;
//FitnessCopy[] is a copy of Fitness[]
//used as workbench.
    double FitnessCopy[] = new double[N_MAX];
    for (int i = 0; i < NUM_IND; i++) {
        Fitness[i] = 1000000;
        FitnessCopy[i] = 1000000;
    }
    if (print) {
        System.out.println("\nSCORING \n");
    }
    for (int i = 0; i < NUM_IND; i++) {
        if (print) {
            System.out.println("\nind = " + i + " " + Ind[i]);
        }
        Fitness[i] = assessInd(Ind[i]);
        FitnessCopy[i] = Fitness[i];
    }
    if (print) {
        System.out.println("\nIndividuals and scores ");
        for (int i = 0; i < NUM_IND; i++) {
            System.out.println("Individual " + i + " = " + Ind[i]

```

```

        + " Score = " + FitnessCopy[i]);
    }
}

//We sort individuals by fitness
// Fitness 0 means the worst, 5 is perfect
if (print) {
    System.out.println("
        + "\nSORTING: from the best to the worst: ");
}
for (int i = 0; i < NUM_IND; i++) {
    Champ = 0;
    for (int j = 0; j < NUM_IND; j++) {
        if (FitnessCopy[j] > FitnessCopy[Champ]) {
            Champ = j;
        }
    }
    //The array Order classifies individuals by fitness.
    //The fittest is number zero
    Ordered[i] = Champ;
    FitnessCopy[Champ] = 0;
    if (print) {
        System.out.println("ind number " + Ordered[i] + " "
            + Ind[Ordered[i]] + " score " + Fitness[Ordered[i]]);
    }
}
//IndClone is a copy of Ind
System.arraycopy(Ind, 0, IndClone, 0, NUM_IND);

}

//Negative selection:
//Fitness determines probability of death.
//The probability of dead is higher,
//the lower is the fitness.
//Death = replacement of individual by ""
private static void death() {
//Min and max are found
    double min = 100;
    double max = -100;
}

```

```

    for (int j = 0; j < NUM_IND; j++) {
        if (Fitness[j] > max) {
            max = Fitness[j];
        }
        if (Fitness[j] < min) {
            min = Fitness[j];
        }
    }

    if (print) {
        System.out.println(
            "\nFitness determines death probability");
    }
//Fitness determines probability of death
    for (int j = 0; j < NUM_IND; j++) {
        //Death probability y and fitness x are related by:
        //y = ax +b
        //y(min) = 1, (the worst must die)
        //y(max) = 0, (the best must live)
        //a = 1/(min - max)
        //b = -a max = -max/(min -max)
        double probDeath;
        if ((max - min) == 0) {
            probDeath = 0.1;
        } else {
            probDeath = -Fitness[j] / (max - min) + max / (max - min);
        }
        if (print) {
            System.out.print("ind " + j);
            System.out.print(" Fitness = " + Fitness[j]);

            System.out.println(" ProbDeath = " + probDeath);
        }
        double p = R.nextDouble();
//Death = replacement of individual by ""
        if (p <= probDeath) {
            Ind[j] = "";
        }
    }
    if (print) {

```

```

        System.out.println("
            + "The best ind is number " + Ordered[0]);
    }
}

//A not nil random individual is chosen.
private static int randomIndiv() {
    boolean cloned = false;
    int k = 0;
    while (cloned == false) {
        k = R.nextInt(NUM_IND);
        if (!"".equals(Ind[k])) {
            cloned = true;
        }
    }
    if (print) {
        System.out.println(
            " cloned from old ind " + k);
    }
    return k;
}

//The pre-population of the new generation
//is built with clones
//of living individuals sampled at random
//with replacement.
private static void Reproduction() {
    if (print) {
        System.out.println("\nREPRODUCTION\n");
    }
    for (int j = 0; j < NUM_IND; j++) {
        if (print) {
            System.out.print("New ind " + j);
        }
        IndClone[j] = Ind[randomIndiv()];
    }
    //IndClone is a copy of Ind
    System.arraycopy(IndClone, 0, Ind, 0, NUM_IND);
}

```

```

//Recombination is enabled.
//individual l is recombined with
//another taken at random.
    private static String recInd(int l) {
        //individual l
        String indl = IndClone[l];
        String head;
        String tail;
        String recombinant;
        int m = indl.length();
        //Mutation site
        int m1;

        //Random site for recombination
        m1 = R.nextInt(m);
        //The head is taken from ind l
        head = indl.substring(0, m1);

        //Random number
        int k = R.nextInt(NUM_IND);
        //Partner at random
        String partner = IndClone[k];
        //The tail is taken from the partner
        tail = partner.substring(m1);
        //Recombination
        recombinant = head + tail;
        if (print) {
            System.out.println("Indl      = " + indl);
            System.out.println("Head      = " + head);
            System.out.println("Partner  = " + partner);
            System.out.println("Tail     = " + tail);
            System.out.println("Recomb   = " + recombinant + "\n");
        }
        return recombinant;
    }

//We found no remedy against recombination:
//it is always mortal for the interval notation.
    private static void recombination() {

```

```

    if (print) {
        System.out.println("\nRECOMBINATION\n");
    }
    //IndClone is a copy of Ind
    System.arraycopy(Ind, 0, IndClone, 0, NUM_IND);

    //Ind1 is a recombinant version of Ind2
    for (int l = 0; l < NUM_IND; l++) {
        Ind[l] = recInd(l);
    }
}

//Returns the interval of a codon
private static int intervalCodon(String codon) {
    int output;
//System.out.println( "Input (intervalCodon) = " + codon);
    //The first letter is a sign:
    //the direction of the interval.
    char c = codon.charAt(0);
    //The second letter denotes an interval
    Character f = codon.charAt(1);

    if (print) {
        System.out.println("char 0,1 = " + c
            + " " + f + " ");
    }
    //Direction is decoded
    int direction;
    if (c == '+') {
        direction = 1;
    } else {
        direction = -1;
    }
    //Interval is decoded
    int l = Character.getNumericValue(f);
    int k = 0;
    if ((l >= 0) & (l < 10)) {
        k = l;
    } else {
        if (f == 'a') {

```

```
        k = 10;
    }
    if (f == 'b') {
        k = 11;
    }
    if (f == 'c') {
        k = 12;
    }
    if (f == 'd') {
        k = 13;
    }
    if (f == 'e') {
        k = 14;
    }
    if (f == 'f') {
        k = 15;
    }
    if (f == 'g') {
        k = 16;
    }
    if (f == 'h') {
        k = 17;
    }
    if (f == 'i') {
        k = 18;
    }
    if (f == 'j') {
        k = 19;
    }
    if (f == 'k') {
        k = 20;
    }
    if (f == 'l') {
        k = 21;
    }
    if (f == 'm') {
        k = 22;
    }
    if (f == 'n') {
        k = 23;
    }
```

```

    }
    if (f == 'o') {
        k = 24;
    }
}
if (print) {
    System.out.println(" k = " + k);
}
output = direction * k;
return output;
}

//The start has the initial note and a duration
private static String mutateStart(String a) {
    if (printThis) {
        System.out.println("input  = " + a);
    }
    String h = transcription(a);
    if (printThis) {
        System.out.println("input T = " + h);
    }
    int length = SUPER_ALPHABET.length();
    String output;
    //Mutate note else duration?
    double f = R.nextDouble();
    if (f < 0.5) //note
    {
        char s = a.charAt(0);

        int indexNote = SUPER_ALPHABET.indexOf(s);
        //An shift is generated
        Integer shift = R.nextInt(6);
        String newNote;
        int newIndex;
        //To avoid some overflow problems.
        if ((indexNote + shift > length)
            || (indexNote + shift > INTERVALS.length())
            || (indexNote + shift < 0)) {
            shift = -shift;
        }
    }
}

```

```

newIndex = indexNote + shift;
if (printThis) {
    System.out.println(
        " indexNote = " + indexNote
        + " shift = " + shift
        + " newIndex = " + newIndex);
}
newNote = "" + SUPER_ALPHABET.charAt(newIndex);

if (printThis) {
    System.out.println("Old note = " + s
        + " New note " + newNote);
}
String codon = a.substring(2, 5);
System.out.println("First Codon = " + codon);
int interval = intervalCodon(codon);
Integer newInterval = interval - shift;
//Compensation: indexNote + interval = newIndex + newInterval;
if (printThis) {
    System.out.println(
        " indexNote = " + indexNote
        + " shift = " + shift
        + " interval = " + interval
        + " newInterval = " + newInterval);
}

String newI = intToOneLetterNot(newInterval);

output = "" + newNote + a.charAt(1)
        + newI + a.substring(4);
if (printThis) {
    System.out.println("output (note) "
        + " = " + output);
}
} else //duration
{
    //Duration is changed
    output = "" + a.charAt(0)
        + randomLetter(durationsAlphabet)
        + a.substring(2);
}

```

```

        if (printThis) {
            System.out.println("output (duration)"
                + " = " + output);
        }
    }

    h = transcription(output);
    if (print) {
        System.out.println("output T = " + h);
    }
    return output;
}

//The interval as Integer is transcribed
//to one letter notation
private static String intToOneLetterNot(Integer intInput) {
    String sign;
    if (intInput > 0) {
        sign = "+";
    } else {
        sign = "-";
    }
    intInput = Math.abs(intInput);
    String newI2 = intInput.toString();
    if (intInput == 10) {
        newI2 = "a";
    }
    if (intInput == 11) {
        newI2 = "b";
    }
    if (intInput == 12) {
        newI2 = "c";
    }
    if (intInput == 13) {
        newI2 = "d";
    }
    if (intInput == 14) {
        newI2 = "e";
    }
    if (intInput == 15) {

```

```
        newI2 = "f";
    }
    if (intInput == 16) {
        newI2 = "g";
    }
    if (intInput == 17) {
        newI2 = "h";
    }
    if (intInput == 18) {
        newI2 = "i";
    }
    if (intInput == 19) {
        newI2 = "j";
    }
    if (intInput == 20) {
        newI2 = "k";
    }
    if (intInput == 21) {
        newI2 = "l";
    }
    if (intInput == 22) {
        newI2 = "m";
    }
    if (intInput == 23) {
        newI2 = "n";
    }
    if (intInput == 24) {
        newI2 = "o"
            + ""
            + "";
    }
    String output = sign + newI2;
    return output;
}
```

```
//The string a is mutated at two contiguous sites.
//Why? We use musITets in interval notation,
//instruction to rise or lower pitch so,
//a mutation may cause notes to get outside
//established sets.
```

```

//The remedy is to make balancing changes at two
//contiguous sites in such a way that no harm
//is caused beyond mutation site.
//Example 1: INTERVALS are mutated
//input    B6+01-y1+61-41+91-41+01
//output:  B6+01-y1+61-21+71-41+01
//Notice:  -4 + 9 = 5 = -2 + 7.
//Example 2: durations are mutated
//input    B6+01-y1+61-41+91-41+01
//output:  B6+01-y1+61-43+91-41+01
//Notice:  we pose no restriction on durations
//Example 3: initial note mutated
//input    B6+01-y1+61-41+91-41+01
//output:  A6+21-y1+61-21+71-41+01
//Notice:  from B to A there are 2 semitones which are
//reestablished in the following note.
//The string is divided in three regions:
//head + mutation sites + tail.
    private static String twoPointsMutationInd(String a) {
        if (printThis) {
            System.out.println(
                "\nInput string = " + a);
        }
        boolean mutateInterval = false;
        boolean mutateDuration = false;
        int m = a.length();
        String output = "";
        if (m == 0) {
            return output;
        } else {
            int l = R.nextInt(a.length());
            if (printThis) {
                System.out.println(
                    "Place mutation = " + l);
            }

            //Is l at the beginning?
            if (l < 2) {
                output = mutateStart(a);
            } else //pick codon and mutate it.

```

```
{
    int numberCodon1 = (l - 2) / 3;
    int numberCodons = (m - 2) / 3;
    int beginCodon1 = numberCodon1 * 3 + 2;
    int endCodon1 = beginCodon1 + 3;
    String codon1 = a.substring(beginCodon1, endCodon1);
    if (printThis) {
        System.out.println("codon1 = " + codon1
            + " codon number = " + numberCodon1
            + " Number of codons = " + numberCodons
            + " a.length = " + m
            + " endCodon = " + endCodon1);
    }
    //Mutate interval else duration?
    double f = R.nextDouble();
    if (printThis) {
        System.out.println("f = " + f);
    }
    if (f < 0.5) {
        mutateDuration = true;
    } else {
        mutateInterval = true;
    }

    if (mutateDuration) {
        //Duration is changed
        output = "" + a.substring(0, beginCodon1 + 2)
            + randomLetter(durationsAlphabet)
            + a.substring(beginCodon1 + 3);
        if (printThis) {
            System.out.println(
                "Output (duration) = " + output + "\n");
        }
        return output;
    }

    if (mutateInterval) {
        int interval1 = intervalCodon(codon1);
        Integer newInterval1 = 0;
        if (interval1 > 1) {
```

```

        newIntervall1 = R.nextInt(intervall1) % 6;
    }

    String newInt1 = intToOneLetterNot(newIntervall1);

    output = "" + a.substring(0, beginCodon1)
            + newInt1 + a.substring(beginCodon1 + 2);
    if (printThis) {
        System.out.println(
            "\n newIntervall1 = " + newIntervall1 + "
            + "Output (u) = " + output + "\n");
    }
    //Is codon1 the final one?
    //Yes = end work
    if (numberCodon1 == numberCodons - 1) {
        return output;
    } else //No: make a coordinate mutation in next site
    {
        String codon2 = a.substring(endCodon1, endCodon1 + 3);
        if (printThis) {
            System.out.println("Codon 2= " + codon2);
        }
        int interval2 = intervalCodon(codon2);

        //Compensation:
        Integer newInterval2 = intervall1 + interval2 -
newIntervall1;

        if (printThis) {
            System.out.println(
                " int 1 = " + intervall1
                + " int 2 = " + interval2
                + " newIntervall1 = " + newIntervall1
                + " newInterval2 = " + newInterval2);
        }
        String newI2 = intToOneLetterNot(newInterval2);

        output = "" + output.substring(0, endCodon1)
                + newI2 + output.substring(endCodon1 + 3);
        if (printThis) {
            System.out.println(

```

```

        "Output (interval) = " + output + "\n");
    }
    } //end else
}
String h1 = transcription(a);
String h2 = transcription(output);
if (printThis) {
    System.out.println(
        "Input (twoPointsM) = " + a + "\n"
        + "Output (twoPointsM) = " + output + "\n"
        + "Input (twoPointsM) = " + h1 + "\n"
        + "Output (twoPointsM) = " + h2 + "\n");
}

    return output;
} //End no void input
} //end method

//All strings are mutated.
private static void mutation() {
    if (print) {
        System.out.println("\nMUTATION\n");
    }
    // All the individuals mutate.
    for (int j = 0; j < NUM_IND; j++) {
        //v=original individual
        String v = Ind[j];
        // w = mutated individual.
        String w = twoPointsMutationInd(v);
        Ind[j] = w; //Replacement
    }
}

//Evolution = selection + reproduction
//+recombination + mutation
private static void Dynamics() {
    System.out.println("\nGENERATION = " + generation + " \n");
    sorting();
    death();
}

```

```

        Reproduction();
        mutation();
        //recombination();
    }

//Evolution is made into a co-composer
    public static void evoMusic() {
        Initialization();
        int NGen = 999;
        for (int n = 1; n <= NGen; n++) {
            generation = n;
            Dynamics();
        }
    }

    public static void main(String[] args) {
        PLAYER.start();
        print = false;
        printThis = true;
        //Measure 44 for 4/4, 68 for 6/8
        int measure = 68;
        switch (measure) {
            case 44:
                header = HEADER44;
                durationsAlphabet = DURATIONS_ALPHABET44;
                break;
            case 68:
                header = HEADER68;
                durationsAlphabet = DURATIONS_ALPHABET68;
                break;
        }
        evoMusic();

        /*
    printThis = true;
    durationsAlphabet = DURATIONS_ALPHABET44;
    twoPointsMutationInd("T6+33-51+91+83-21-51+01" );
        */
    }
} //End of main class H103 LamarckMusic1

```

104 Exercise. *Run the program and play with the code. Play enough to accept else reject the conclusion of the Author: the directive of composing tunes taking care of observing pleasant intervals is definitively better than our former theory that claimed that notes per se are all to music. Nevertheless, after an initial tune has been assembled by randomness, further evolution is on one hand very difficult to implement (our proposed solution is naive) and on the other, it is highly unproductive.*

105 Exercise. *The previous program is not good. When one finds a case like this, a good idea is to begin from scratch to trace own path. Anyway, while ideas come to the mind, one can try to improve the existent code. So, the most simple task among possible ones is to simplify the method `intToOneLetterNot(Integer intInput)` and related code: it includes a lengthy substitution process that can be reduced to some few lines. Try it out. *Answer**

106 Graduation. *You have two tasks to graduate from this volume:*

- 1. The **Theory of the Interval** (that intervals are all to music) gives rise to a corresponding statistical version and the ensuing musical analysis: the statistical registration of the used intervals of wonderful pieces must produce a frequency table that could be used to feed the random generator to compose wonderful `musiTexts`. Else, one can encode a tune in interval notation and then subject it to mixing. Choose a methodology and implement it that one could test this theory. Hint: this is more easily done when one uses a GUI. So, study how the statistical version of the first theory was implemented in our GUI as a mixing procedure and follow the example.*
- 2. Update our GUI with all the programs of this chapter. Our answer is given in `Finch`, a program that appears separately in the Art section:*

<http://www.evoljava.com/DownloadsByAreas.php>

(Verified 3/VII/2018)

107 Challenge. *Add to this volume the needed material from previous ones to make it into a self contained book, one that could serve now to teach Java to teenagers, boys and girls. They are the biologists of the future.*

6.2 Conclusion

We imagined that we could help evolution if we include a hint from the human nature: what matters in music is not the notes but the intervals, measured in semitones, among them. It seems to us that the idea is fruitful and that the musiTexts that were synthesized at random following that directive are pleasant in greater frequency than for the former theory. Nevertheless, when we tried to implement evolution in the frame of this new hint, we found that troubles were greater than benefits: on one hand, our implementation was naive and in second place we did not find an ore deposit. Therefore, we claim that music is a challenge worth careful attention by our community and that it will give rise to a wonderful laboratory to study the battle of evolution against complexity.

Chapter 7

Summary

A **musiText** is a text in plain format that represents music and that can be played by computers. This idea has been implemented in many forms and we have worked with the ABC NOTATION. Musitexts allow us to propose music as a tractable parable of life: as a DNA string encodes for an organism, a musiText encodes for a musical piece. Therefore, music is a model of life in which the genetic aspect of information appears in first plane and so evolution is a natural and welcome guest. Thanks to the effort of many, many people, it has been possible for us to incarnate these ideas into a (basic as yet) musical laboratory for practicing the scientific method and for the study of evolution.

Our immediate purpose is to understand music. This means to devise a theory and to use it as a guide for composition. If the theory is correct, we will produce nice music. We hope that this challenge is tractable in any case more than to scientifically testing a theory of life. In this regard we were spinning around the **First theory of music** that says: each note in isolation is pleasant and music is just a succession of notes. This theory predicts that random notes drawn at random would sound well. For notes of equal duration, this was readily rejected. But with notes of different duration it was found that a non nil proportion of musiTexts were pleasant to the Author. An immediate application of this finding was to devise a rhythm generator, i.e. short musiTexts that were repeated over and over. Surprisingly, many rhythms were found to be good, possibly in greater proportion than when musiTexts were played only once. Thus, we conclude that human beings have a biologically given propensity for repetitive sounds and dancing. A slight variation of the code allows one to convert a DNA or protein string into a tune.

Since our first theory was found highly defective, we proposed the **statistical version of the first theory** as a correction: the quality of musical pieces is deter-

mined by the frequencies of notes regardless of the order as they appear along the tune. We tested the theory by rearranging the order as notes appear along a tune. No encouraging result in this regard was found.

Defective as the 1-theory could be, it has an application to pedagogy: students must learn to recognize pitches. In consequence, we devised a program to help students to recognize notes. Let us notice now that learning amounts to the building and continuous restructuring of appropriate neural networks : this is evolution. Now, it is full fledged because evolution amounts to selection over the fruits of mutation plus recombination. Selection means success and failure, mutation means change and recombination results from the turning on and off of neural subnetworks. That is why the aforementioned application will allow every one in our community to run an evolutionary experiment with him or herself.

Let us see the witness of the Author about the developed facility to help people training the ear to distinguish pitches:

According to my personal experience, the created facility is marvelous. Let us explain why: thanks to the judgments of my teachers, I acquired the firm conviction that in relation with music my best option was to remain in silence. Nevertheless, battling with formal music as a crude approximation to the art of music, I somehow was able to awake a musical feeling that grew with the years. Anyway, to listen at a musical note to classify it as a C or G or F is at present time so a mysterious task as levitation could be. But working with the aforementioned facility has allowed me to see a light across the tunnel: after some thousand trials while I tuned the program, I correctly classified the right note among a set of 3 notes (Ccc') and my proportion of successes in classifying notes in a set of 4 notes (CGcc') or 5 (CGcgc') was high. So, I was very expectant to see what can be achieved in the long run and in consequence I tried to work some 15 minutes a day some four days per week.

Result after some 4 years of work: the accurateness of my sense of hearing varies from day to day ranging from rare perfection to rare dumbness (this shows that my neural networks are always trying new configurations, always evolving but without finding an optimum). Most of the time, some few trials are necessary to match a note. The necessary quantity depends mainly on the complexity of the testing set but for less than 7 notes, two or three trials may suffice. In spite of my failure, I can feel music more finely, more sensitively, more accurately. And this rejoices me a lot. A lot. So, the effectiveness of my neural networks to classify pitches remains quite poor but anyway sufficient to enrich my life a lot. And I continue to work albeit in other fronts.

In conclusion, my neural networks continuously change and are not good to

match instantaneously a pitch but are good to do that in 2 or 3 trials for a testing set of low complexity. So, how are they structured? The author feels very intrigued in this regard.

And, what about the power of evolution as a help to compose good music? Our stand was the simplest theory of music: notes as separate entities are all to music. This theory was found to be very primitive, nevertheless that frame was kind enough to allow us to implement Darwinian evolution as a quality improver of tunes that were synthesized at random. Nevertheless, we found that our patience was not enough to have evolution helping us in the art of musical composition.

Our failure led us to imagine that we can help evolution if we include a hint from the human nature: what matters in music is not the notes but the intervals, measured in semitones, among them. It seems to us that the idea is fruitful and that the musiTexts that were synthesized at random following that directive are pleasant in greater frequency than for the former theory. Nevertheless, when we tried to implement evolution in the frame of this new hint, that classify as Lamarckian evolution, we found that troubles were greater than benefits: if one encodes a musical piece as an initial note plus a string of instructions to lower or rise the pitch and/or duration, the result is a text that creates historical dependence and so the concept of point mutation is lost. Nevertheless, mutations can be made if one creates two contiguous coordinated mutations. Resultant evolution was assessed as follows: if someone can use this type of evolution to produce good musical pieces, the glory is only his or her because we cannot imagine that this could be made without very hard and extenuating work.

Our experience allows us to formulate simple but relevant questions in biology:

1. To allow notes of different duration is a first step in regulation and a necessary one to produce musiTexts with potential musical concern. So: can DNA based life exist without regulation of gene expression? More mechanistically: how shall be a milieu in order that DNA based life without regulation of gene expression could exist as an stable phenomenon?
2. On another hand, our creative work allows too many options and one of them must be capriciously chosen. This observation immediately forces another question: if life arouse by evolution, where are those infinitely many possible genetic codes together with the many other alternatives to DNA and RNA?
3. Evolution seems to work, say learning is possible, but it is too slow, too expensive in trials. What is the human intelligence that is so fast to create things maybe without trials and errors? How can one enhance that miracle?

4. Given the exceeding complexity of life forms, where is the witness of the fossil record and of modern populations of the infinitely many trials leading to nothing, malfunction and malformation?
5. Intrigue: In spite of personal and cultural biases, there are many musical pieces that everybody likes and that one classifies as nice, elegant, beautiful. Let us turn this fact into biology: Can you say that animals, say, a horse, a lion, a cock, a mouse or a bat are elegant? Can you justify your answer from the stand of the evolutionary theory? On the other hand, can you say that the genome that encodes for those animals is elegant, i.e. that follows a very elegant style of software development? Can you justify your answer from the stand of the evolutionary theory?
6. A look to our many programs of this series that implement evolution shows that they are all different. So, evolution can be implemented in infinitely many ways. Which is the best? The best solution to complex problems in general cannot be found but instead many very good solutions can be chased. So, many different implementations of evolution should exist on Earth if only natural law were the cause of the origin of life and of its diversification. Where are they?

In conclusion, we claim that music as a parable is a challenge worth careful attention by our community: it is very difficult but with the appropriate methodological insight, it seems to be (almost) tractable and promising to teach us a lot about both evolution and the human nature.

Answers to exercises

Problems of Chapter 3

28 pag 22. The code that plays first a musiText and then a tune from a file follows. The tunes are not played one after the other but in parallel.

```
//Program H28 Play2ABCTunes
//plays two  tunes
//one from a musiText and the other from an ABC file.
//Result:
//BUG = the  two tunes are played in parallel
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneBook;
import abc.parser.TuneParser;
import java.nio.file.Files;
import java.nio.file.Paths;

public class Play2ABCTunes {

    //An ABC-musiText is converted into a tune
    private static Tune ABCmusiTextToTune() {
        //musiText declaration
        String musiText
```

```

        = "X:0\n"
        + "T:The scale \n"
        + "M:4/4\n"
        + "K:C\n"
        + "CDEFGABc4 cBAGFEDC8";
    Tune tune = new TuneParser().parse(musiText);
    return tune;
}

//A tune is read from a file
private static Tune ABCFileToTune()
    throws FileNotFoundException, IOException {

    String nameOfFile =
"/home/jose/MusicAll/abcScores/El_Espíritu_de_Dios_está_en_este_
c";

    //loading from the file
    TuneBook book = new TuneBook(new File(nameOfFile));
    //The file is read into a string
    String musitext = new String(Files.readAllBytes(Paths.get(nameOfFile)));
    System.out.println("musitext = \n" + musitext);

    Tune tune = new TuneParser().parse(musitext);
    return tune;

}

//A tune is played
private static void playTune(Tune tune) {

// display the title of the tune
    System.out.println("Title = " + tune.getTitles()[0]);
// and its key
    System.out.println("Key = "
        + tune.getKey().toLiteralNotation());

// creates a simple midi player to play the melody
    TunePlayer player = new TunePlayer();

```

```

        player.start();
        player.play(tune);
    }

    public static void main(String[] args) throws IOException {
        Tune tune;
        tune = ABCmusiTextToTune();
        playTune(tune);
        tune = ABCFileToTune();
        playTune(tune);
    }
} //End of main class H28 Play2ABCTunes

```

32 pag 24. The following program reads a tune from a file and draws it.

```

//Program H32 ABCfileToStave()
//Reads a simple ABC tune from a file,
//plays and draws its score.
//Neither regulative commands or lyrics are allowed.
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;

import javax.swing.JFrame;

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneBook;
import abc.ui.swing.JScoreComponent;

public class ABCfileToStave {

    //Change this

```

```

static String nameOfFile
    = "/home/jose/MusicAll/abcScores/"
    + "El_Espíritu_de_Dios_está_en_este_lugarNoLyrics.ab

//A tune is read from a file
private static Tune ABCfileToTune()
    throws FileNotFoundException {

    //creates a file from the path to the abc
    //file containing the tune to be played
    File abcFile = new File(nameOfFile);
    //creates a tunebook from the previous file
    TuneBook book = new TuneBook(abcFile);

    //show details about the tunes that are loaded
    System.out.println("# of tunes in book.abc : "
        + book.size());
    //A book possibly contains various tunes.
    // Choose one, pick its index (in the X: field) and
    //assign it to tuneIndex else define
    //tuneIndex = book.getHighestReferenceNumber();
    int tuneIndex = book.getHighestReferenceNumber();
    Tune tune = book.getTune(tuneIndex);
    return tune;
}

//A tune is drawn
private static void DrawTune(Tune tune) {
    //creates a component that draws the melody on a stave
    JScoreComponent jscore = new JScoreComponent();
    jscore.setJustification(true);
    jscore.setTune(tune);
    JFrame j = new JFrame();
    j.add(jscore);
    j.pack();
    j.setVisible(true);
}

```

```

//A tune is played
private static void playTune(Tune tune) {
    //Displays the title of the tune
    System.out.println("Title = "
        + tune.getTitles()[0]);
    // and its key
    System.out.println("Key = "
        + tune.getKey().toLitteralNotation());
    //The player converts the tune into
    //a midi message that is sent to
    //hardwared sound synthesizers.
    //If problems appear,
    //an input-out exception must be thrown.
    TunePlayer player = new TunePlayer();
    player.start();
    player.play(tune);
}

public static void main(String[] args) throws IOException {
    Tune tune = ABCfileToTune();
    DrawTune(tune);
    playTune(tune);
}
} //End of main class H32 ABCfileToStave()

```

39 pag 31. Reading a tune from a file is enabled:

```

//Program H39 MusiTextClient2
//The class MusiText is instantiated
//to enable ordinary functionality.
//The encapsulated MusiText class is inserted as
//an inner class and a main method is added.
//The inner class MusiText is furnished with the
//possibility to read a tune from a file.

```

```

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;

import javax.swing.JFrame;

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneBook;
import abc.parser.TuneParser;
import abc.ui.swing.JScoreComponent;

public class MusiTextClient2 {

    //Change this
    static String nameOfFile
        = "/home/jose/MusicAll/abcScores/"
        + "El_Espíritu_de_Dios_está_en_este_lugarNoLyrics.ab

    //*****
    //The MusiText class is inserted as an inner class
    //*****
    public class MusiText2 {

        String musiText;

        //Constructor
        //This specifies how a musiText is initialized.
        //Here, it acquires information from a String
        MusiText2(String a) {
            musiText = a;
        }

        //One can initialize it by default
        MusiText2() {
            musiText
                = "X:0\n"

```

```

        + "T:The scale \n"
        + "M:4/4\n"
        + "K:C\n"
        + "CDEFGABc4 cBAGF - EDC8";
    }

Tune ABCfileToTune(String nameOfFile)
    throws FileNotFoundException {

    //creates a file from the path to the abc
    //file containing the tune to be played
    File abcFile = new File(nameOfFile);
    //creates a tunebook from the previous file
    TuneBook book = new TuneBook(abcFile);

    //show details about the tunes that are loaded
    System.out.println("Number of tunes in book.abc : "
        + book.size());
    //A book possibly contains various tunes.
    // Choose one, pick its index (in the X: field) and
    //assign it to tuneIndex else define
    //tuneIndex = book.getHighestReferenceNumber();
    int tuneIndex = book.getHighestReferenceNumber();
    Tune tune = book.getTune(tuneIndex);
    return tune;
}

//A musiText is transformed into a tune
Tune ABCmusiTextToTune(String musiText) {

    //From String to Tune
    Tune tune = new TuneParser().parse(musiText);
    return tune;
}

//A tune as Tune is drawn
//Try tunes without regulative instructions or lyrics
void DrawTune(Tune tune) {

```

```

        //creates a component that draws the melody on a sta
        JScoreComponent jscore = new JScoreComponent();
        jscore.setJustification(true);
        jscore.setTune(tune);
        JFrame j = new JFrame();
        j.add(jscore);
        j.pack();
        j.setVisible(true);
    }

    //A tune is played
    void playTune(Tune tune) {
        //Displays the title of the tune
        System.out.println("Title = " + tune.getTitles()[0])
        // and its key
        System.out.println("Key = "
            + tune.getKey().toLitteralNotation());
        //The player converts the tune into
        //a midi message that is sent to
        //hardwared sound synthesizers.
        //If problems appear,
        //an input-out exception must be thrown.
        TunePlayer player = new TunePlayer();
        player.start();
        player.play(tune);
    }

} //End of class MusiText

//*****
//***** Definition and default initialization
//***** of an instantiation of MusiText
//*****
static final private MusiTextClient2 A = new MusiTextClient2
static final private MusiTextClient2.MusiText2 M = A.new Mus

//*****
//***** Main method of the client program

```

```

//*****
public static void main(String[] args) throws IOException {
    //System.out.println(M.musiText);
    //Tune tune = M.ABCmusiTextToTune(M.musiText);

    Tune tune = M.ABCfileToTune(nameOfFile);
    M.DrawTune(tune);
    M.playTune(tune);
}
} //End of class H39 MusiTextClient2

```

40 pag 31. A clean and evolvable toolbox:

```

//Program H40 MusiTextClient3
//The class MusiText is instantiated
//to enable ordinary functionality.
//The encapsulated MusiText class is inserted as
//an inner class and a main method is added.
//The inner class MusiText is furnished with the
//possibility to read a tune from a file.
//The inner class MusiText has been depured:
//names shall perfectly reflect their function,
//each method processes a nonvoid input,
//no reference to global variables is made,
//every method implements a single indivisible function,
//This style favors reuse, i.e., evolution,
//and strongly prevents possibilities for
//entanglement and ensuing bugs.
//Other desirable properties:
//variables shall have as short scope as possible,
//methods must be complete,
//i.e., all possible functions must be implemented.
//To that aim, Java provides the possibility
//to define constructors and methods in various ways
//with different lists of arguments.

```

```
//This is known as overloading.

//In real life no one subjects him or herself to this slavery
//because a good style prescindible inside Java
//and it demands time, concentration and effort.
//But a good style pays on the long range.
//That is why languages have been designed in which
//all this and more is imprescindible:
//this is the functional style of programming.
//It has been implemented
//in Functional Java beginning since Java 8
//and more strictly, elegantly and imprescindible
//in Clojure (Vol XVII version 2), a language of the Java clan.

//The most loved functional programming
//by 2017 and 2018 is Rust.

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;

import javax.swing.JFrame;

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneBook;
import abc.parser.TuneParser;
import abc.ui.swing.JScoreComponent;

public class MusiTextClient3 {

    //Change this
    static String nameOfFile
        = "/home/jose/MusicAll/abcScores/"
        + "El_Espíritu_de_Dios_está_en_este_lugarNoLyrics.ab
```

```

//*****
//The MusiText class is inserted as an inner class
//*****
public class MusiText3 {

    String musiText;

    //Constructor
    //This specifies how a musiText is initialized.
    //Here, it acquires information from a String
    MusiText3(String a) {
        musiText = a;
    }

    //One can initialize it by default
    MusiText3() {
        musiText
            = "X:0\n"
            + "T:The scale \n"
            + "M:4/4\n"
            + "K:C\n"
            + "CDEFGABc4 cBAGF - EDC8";
    }

    //A tune from a file is read into an usable tuneBook
    TuneBook ABCfileToBookTune(File abcFile) throws FileNotFoundException {
        //creates a tunebook from the previous file
        TuneBook book = new TuneBook(abcFile);
        return book;
    }

    //A tune is retrieved from a TuneBook
    Tune RetrieveTuneFromBook(TuneBook book) {
        //A book possibly contains various tunes.
        // Choose one, pick its index (in the X: field) and
        //assign it to tuneIndex else define
        //tuneIndex = book.getHighestReferenceNumber();
    }
}

```

```

        int tuneIndex = book.getHighestReferenceNumber();
        Tune tune = book.getTune(tuneIndex);
        return tune;
    }

    //An indexed tune is retrieved from a TuneBook
    Tune RetrieveTuneFromBook(TuneBook book, int tuneIndex)
    //A book possibly contains various tunes.
    // Choose one and pick its index (in the X: field)
    Tune tune = book.getTune(tuneIndex);
    return tune;
}

//A musiText is transformed into a tune
Tune ABCmusiTextToTune(String musiText) {
    //From String to Tune
    Tune tune = new TuneParser().parse(musiText);
    return tune;
}

//A tune as Tune is drawn
//Try  tunes without regulative instructions or lyrics
void DrawTune(Tune tune) {
    //creates a component that draws the melody on a sta
    JScoreComponent jscore = new JScoreComponent();
    jscore.setJustification(true);
    jscore.setTune(tune);
    JFrame j = new JFrame();
    j.add(jscore);
    j.pack();
    j.setVisible(true);
}

//A tune  is played
void playTune(Tune tune) {
    //Displays the title of the tune
    System.out.println("Title = " + tune.getTitles()[0])
    // and its key

```

```

        System.out.println("Key = "
            + tune.getKey().toLiteralNotation());
        //The player converts the tune into
        //a midi message that is sent to
        //hardwared sound synthesizers.
        //If problems appear,
        //an input-out exception must be thrown.
        TunePlayer player = new TunePlayer();
        player.start();
        player.play(tune);
    }
} //End of inner class MusiText

//*****
//***** Definition and default initialization
//***** of an instantiation of MusiText
//*****
static final private MusiTextClient3 A = new MusiTextClient3();
static final private MusiTextClient3.MusiText3 M = A.new MusiText3(

//*****
//***** Main method of the client program
//*****
public static void main(String[] args) throws IOException {
    String whatToPlay = "file";
    //whatToPlay = "musitext";

    //System.out.println(M.musiText);
    //Tune tune = M.ABCmusiTextToTune(M.musiText);
    Tune tune;
    if (whatToPlay.equals("file")) {
        File abcFile = new File(nameOfFile);
        TuneBook book = M.ABCfileToBookTune(abcFile);
        tune = M.RetrieveTuneFromBook(book);
    } else {
        System.out.println(M.musiText);
        tune = M.ABCmusiTextToTune(M.musiText);
    }
}

```

```

        M.DrawTune(tune);
        M.playTune(tune);
    }

} //End of main class H40 MusiTextClient3

```

43 pag 38. The concept of minimum program is ill defined in Java. The reason is that one can enrich Java with appropriate APIs such that a given program can be simplified to just one line (apart from necessary headers and enclosing code).

44 pag 38. The class MusiText has been included as an inner class of the code for the GUI. One usually feels that the object approach to programming is just a sophistication but this sensation fades away if one allows Netbenas or Eclipse or IntelliJ IDEA to help one in the developing work: once we have instantiated a given object, say `n` as an instance of MusiText in line 46, one acquires the possibility to type `n .` (the instance and a period) and the IDE (Integrated Development Environment) immediately answers with all possible services and variables that the object owns but without any single trace of garbage. At this moment one perceives the importance of objects: the keep united what naturally goes together. The code follows:

```

//Program H44 MusitextGUI
//This is a GUI for editing ABC code.
//Can play an ABC tune
//and show its score.
//The score can be rudimentarily edited.
//
//Paste your ABC tune into the text area
//
//A soft introduction to GUI's was done in
//Chapter I of Vol III of the series
//Java for the study of evolution,
//www.evoljava.com

```

```

import java.awt.*;
import java.awt.event.*;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;

import javax.swing.*;

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneBook;
import abc.parser.TuneParser;
import abc.ui.swing.JScoreComponent;

public final class MusiTextGUI extends JFrame {

    private static final long serialVersionUID = 1L;

    //Change this
    static String nameOfFile
        = "/home/jose/MusicAll/abcScores/"
        + "El_Espíritu_de_Dios_está_en_este_lugarNoLyrics.abc";

    //*****
    //***** Definition and default initialization
    //***** of an instantiation of MusiText
    //*****
    static private final MusiTextGUI A = new MusiTextGUI();
    static private final MusiText N = A.new MusiText();

    protected JTextArea m_monitor;

    protected JToolBar m_toolBar;

    //creates a simple midi player to play the melody
    static private TunePlayer player = new TunePlayer();

    //Initialization

```

```

String musiText2
    = "X:0\n"
    + "T:Song \n"
    + "M:4/4\n"
    + "L:1/4"
    + "K:C\n"
    + "dd2d |ded2 |GB3 | "
    + "BBBA | A3  A | dAB2 |"
    + " AAAA | B2A A | G4 | "
    + "Gded- |d edG | B4 | "
    + "BBA2 | A  A3 | BAG2|"
    + "GGGA-|A2 B2 | A4 | G4";

//Constructor
    public MusiTextGUI() {
//Title in the menu bar
        super.setTitle("A GUI for the class MusiTexts");
        super.setSize(450, 350);

//Text area = editing window
        m_monitor = new JTextArea();
        JScrollPane ps = new JScrollPane(m_monitor);
        super.getContentPane().add(ps, BorderLayout.CENTER);
        m_monitor.append(musiText2);

        JMenuBar menuBar = createMenuBar();
//JMenuBar menuBar = new JMenuBar();
        super.setJMenuBar(menuBar);

        WindowListener wndCloser = new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        };

        super.addWindowListener(wndCloser);

```

```

        super.setVisible(true);
    }//End constructor

//=====
//=====MENU BAR=====
//=====
//The menu bar and listeners
    protected JMenuBar createMenuBar() {
        final JMenuBar menuBar = new JMenuBar();

        //File menu
        JMenu mFile = new JMenu("File");
        mFile.setMnemonic('f');

        //New file
        ImageIcon iconNew = new ImageIcon("file_new.gif");

        Action actionNew = new AbstractAction("New", iconNew) {
            private static final long serialVersionUID = 1L;

            //what shall be done if the new menu is clicked on
            @Override
            public void actionPerformed(ActionEvent e) {
                m_monitor.setText("");
            }
        };

        //item is a variable for diverse itemMenus
        JMenuItem item = mFile.add(actionNew);
        mFile.add(item);

        //Exit menu
        Action actionExit = new AbstractAction("Exit") {
            private static final long serialVersionUID = 1L;
            //what shall be done if the exit menu is clicked on

            @Override
            public void actionPerformed(ActionEvent e) {

```

```

        System.exit(0);
    }
};

item = mFile.addActionExit();
item.setMnemonic('x');
menuBar.add(mFile);

m_toolBar = new JToolBar();

//ShowPlay menu
JMenu mShowPlay = new JMenu("ShowPlay");
mShowPlay.setMnemonic('e');
menuBar.add(mShowPlay);

//Show text menu
ImageIcon iconShowText
    = new ImageIcon("ShowPlay_ShowText.gif");
Action actionShowText
    = new AbstractAction("ShowText", iconShowText) {
    private static final long serialVersionUID = 1L;

    @Override
    public void actionPerformed(ActionEvent e) {
        String ss = m_monitor.getText();
        System.out.println(ss);
    }
};

JMenuItem showText = mShowPlay.addActionShowText();
mShowPlay.add(showText);

//=== Show tune's score menu
ImageIcon iconShowTune
    = new ImageIcon("ShowPlay_ShowTune.gif");
Action actionShowTune
    = new AbstractAction("ShowTune", iconShowTune) {
    private static final long serialVersionUID = 1L;

```

```

        @Override
        public void actionPerformed(ActionEvent e) {
            String ss = m_monitor.getText();
            System.out.println(ss);
            Tune tune;
            tune = N.ABCmusiTextToTune(ss);
            N.DrawTune(tune);
        }
    };

    JMenuItem showTune = mShowPLAY.add(actionShowTune);
    mShowPLAY.add(showTune);

    ImageIcon iconPlayTune
        = new ImageIcon("ShowPLAY_PlayTune.gif");
    Action actionPlayTune;
    actionPlayTune = new AbstractAction("PlayTune", iconPlayTune) {
        private static final long serialVersionUID = 1L;

        @Override
        public void actionPerformed(ActionEvent e) {
            String ss = m_monitor.getText();
            System.out.println(ss);
            Tune tune;
            tune = N.ABCmusiTextToTune(ss);
            N.playTune(tune);
        }
    };

    JMenuItem play = mShowPLAY.add(actionPlayTune);
    mShowPLAY.add(play);

    //==Show score and play tune-menu
    ImageIcon iconShowAndPLAYTune = new
    ImageIcon("ShowPLAY_ShowAndPLAYTune.gif");
    Action actionShowAndPLAYTune
        = new AbstractAction("ShowAndPLAYTune", iconShowAndPLAYTune);

```

```

private static final long serialVersionUID = 1L;

@Override
public void actionPerformed(ActionEvent e) {
    String ss = m_monitor.getText();
    System.out.println(ss);
    Tune tune;
    tune = N.ABCmusiTextToTune(ss);
    N.DrawTune(tune);
    N.playTune(tune);
}
};

JMenuItem ShowAndPLay
    = mShowPLay.add(actionShowAndPLayTune);
mShowPLay.add(ShowAndPLay);

//=====Panic menu
JMenu mPanic = new JMenu("Panic");
mPanic.setMnemonic('A');
menuBar.add(mPanic);

ImageIcon iconPanic
    = new ImageIcon("ShowPLay_Panic.gif");
Action actionPanic
    = new AbstractAction("Panic", iconPanic) {
private static final long serialVersionUID = 1L;

@Override
public void actionPerformed(ActionEvent e) {
    player.stopPlaying();
}
};

JMenuItem Panic = mPanic.add(actionPanic);
mPanic.add(Panic);

//=====

```

```

//The toolbar is added to the container
    getContentPane().add(m_toolBar, BorderLayout.NORTH);

    return menuBar;
} //End of menu bar construction

//*****
//The MusiText class is inserted as an inner class
//*****
    public class MusiText {

        String musiText;

        //Constructor
        //This specifies how a musiText is initialized.
        //Here, it acquires information from a String
        MusiText(String a) {
            musiText = a;
        }

        //One can initialize it by default
        MusiText() {
            musiText
                = "X:0\n"
                + "T:The scale \n"
                + "M:4/4\n"
                + "K:C\n"
                + "CDEFGABc4 cBAGF - EDC8";
        }

        //A tune from a file is read into an usable tuneBook
        TuneBook ABCfileToBookTune(File abcFile)
            throws FileNotFoundException {
            //creates a tunebook from the previous file
            TuneBook book = new TuneBook(abcFile);
            return book;
        }
    }

```

```

//A tune is retrieved from a TuneBook
Tune RetrieveTuneFromBook(TuneBook book) {
    //A book possibly contains various tunes.
    // Choose one, pick its index (in the X: field) and
    //assign it to tuneIndex else define
    //tuneIndex = book.getHighestReferenceNumber();
    int tuneIndex = book.getHighestReferenceNumber();
    Tune tune = book.getTune(tuneIndex);
    return tune;
}

//An indexed tune is retrieved from a TuneBook
Tune RetrieveTuneFromBook(TuneBook book, int tuneIndex)
    //A book possibly contains various tunes.
    // Choose one and pick its index (in the X: field)
    Tune tune = book.getTune(tuneIndex);
    return tune;
}

//A musiText is transformed into a tune
Tune ABCmusiTextToTune(String musiText) {
    //From String to Tune
    Tune tune = new TuneParser().parse(musiText);
    return tune;
}

//A tune as Tune is drawn
//Try tunes without regulative instructions or lyrics
void DrawTune(Tune tune) {
    //creates a component that draws the melody on a sta
    JScoreComponent jscore = new JScoreComponent();
    jscore.setJustification(true);
    jscore.setTune(tune);
    JFrame j = new JFrame();
    j.add(jscore);
    j.pack();
    j.setVisible(true);
}

```

```

//A tune is played
void playTune(Tune tune) {
    //Displays the title of the tune
    System.out.println("Title = " + tune.getTitles()[0]);
    // and its key
    System.out.println("Key = "
        + tune.getKey().toLitteralNotation());
    player.start();
    player.play(tune);
}
} //End of inner class MusiText

//*****
//***** Main method *****
//*****
//These instructions are executed at the beginning,
//but the GUI is always ready
//to fulfill the desires of the User.
    public static void main(String[] args) throws IOException {
        //Creates an open a file
        File abcFile = new File(nameOfFile);
        TuneBook book = N.ABCfileToBookTune(abcFile);
        Tune tune = N.RetrieveTuneFromBook(book);
        N.DrawTune(tune);
        N.playTune(tune);
    }

} //End of main class H44 MusitextGUI

```

Problems of Chapter 4

52 pag 49. We found that random music composed of notes of unequal duration could be interesting by moments. From this we draw the general lesson, applicable to design, that randomness alone is pretty useless but if some restrictions are wisely imposed then randomness may get into a powerful tool for design. In our case, the restrictions were given by intro-

ducing a well suited alphabet, including notes and duration, and by allowing change of relative frequencies.

54 pag 49. A rhythm machine:

```
//Program H54 RandomMusic3
//Randomness is made into a generator of rhythms.
import java.io.IOException;
import java.util.Random;

import javax.swing.JFrame;

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneParser;
import abc.ui.swing.JScoreComponent;

public class RandomMusic3 {

    //We turn on the inbuilt random generator:
    private static final Random R = new Random();

    //Our one symbol notation for notes :
    //A, _B, B, C #C D _E E F #F G #G
    //R S T C M D N E F O G P

    //A _B B c #c d _e e f #f g #g a _b b
    //A Q B c m d n e f o g p a q b

    private static final String SUPER_ALPHABET
        = "RSTAMBCNDOEFPGQambcndoeftpqg";

    private static final String ALPHABET = "RTCDEOGabcdeo";

    private static final String DURATIONS_ALPHABET = "111122222222

//The musiText is transformed into a tune,
```

```

//an ABC4J musical object
private static Tune ABCmusiTextToTune(String musiText) {

    Tune tune = new TuneParser().parse(musiText);
    return tune;
}

//A tune is played
private static void playTune(Tune tune) {
    //The player converts the tune into
    //a midi message that is sent to
    //hardwared sound synthesizers.
    //If problems appear,
    //an input-out exception must be thrown.
    TunePlayer player = new TunePlayer();
    player.start();
    player.play(tune);
}

//A tune as Tune is drawn
private static void DrawTune(Tune tune) {
//creates a component that draws the melody on a stave
    JScoreComponent jscore = new JScoreComponent();
    jscore.setJustification(true);
    jscore.setTune(tune);
    JFrame j = new JFrame();
    j.add(jscore);
    j.pack();
    j.setVisible(true);
}

//Randomness is made into a composer
static private String randomNotes() {
    //Number of time unities per measure
    int limit = 8;
    boolean go = true;
    //Output initialization
    String stringOfNotes = "";
}

```

```

int k;
//CumulativeTime must not surpass n
for (int cumulativeTime = 0; cumulativeTime < limit;) {
    //A note is generated:
    //A random integer less than the
    //number of chars in the alphabet is thrown
    k = R.nextInt(ALPHABET.length());
    //The integer is changed into a note
    char c = ALPHABET.charAt(k);
    //The note is added to the output
    stringOfNotes = stringOfNotes + c;
    //A duration is generated
    //but the limit time must not be surpassed
    while (go) {
        //An integer is generated
        k = R.nextInt(DURATIONS_ALPHABET.length());
        //The integer is changed into a
        //char that represents a duration number
        c = DURATIONS_ALPHABET.charAt(k);
        //the char is reread as a number
        int nc = (int) c - 48;
        cumulativeTime = cumulativeTime + nc;
        //System.out.println(nc + " " + cumulativeTime);
        //if cumulativeTime does not surpass the allowed
        if (cumulativeTime <= limit) { //the duration is
the output
            stringOfNotes = stringOfNotes + c;
            go = false; //leave the while loop
        } //else give another try
        else {
            cumulativeTime = cumulativeTime - nc;
        }
    }
    go = true;
}
return stringOfNotes;
}

```

```

//A string in our alphabet is rewritten in ABC notation
//Accidents are prefixes in ABC notation.
//Sharp is denoted by ^, flat by _
    static private String interpreter(String ss) {
        ss = ss.replaceAll("R", "A,");
        ss = ss.replaceAll("S", "_B,");
        ss = ss.replaceAll("T", "B,");
        ss = ss.replaceAll("M", "#C");
        ss = ss.replaceAll("M", "#c");
        ss = ss.replaceAll("N", "_E");
        ss = ss.replaceAll("n", "_e");
        ss = ss.replaceAll("O", "^F");
        ss = ss.replaceAll("o", "^f");
        ss = ss.replaceAll("P", "^G");
        ss = ss.replaceAll("p", "^g");
        ss = ss.replaceAll("Q", "_B");
        ss = ss.replaceAll("q", "_b");
        return ss;
    }

public static void main(String[] args) throws IOException {
    //Initialization
    String header
        = "X:0\n"
        + "T:Song \n"
        + "M:4/4\n"
        + "L:1/8\n"
        + "K:C\n";
    String ss = randomNotes();
    ss = interpreter(ss);
    System.out.println("Rhythm = " + ss);
    for (int i = 0; i < 4; i++) {
        ss = ss + ss;
    }
    String musiText = header + ss;
    Tune tune;
    tune = ABCmusiTextToTune(musiText);
    DrawTune(tune);
}

```

```

        playTune(tune);
    }
} //End of main class H54 RandomMusic3A

```

55 pag 49.

```

//Program H55 RandomMusic4
//Randomness is made into a generator of rhythms.
//Measures 4/4 and 6/8 are allowed.
import java.io.IOException;
import java.util.Random;

import javax.swing.JFrame;

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneParser;
import abc.ui.swing.JScoreComponent;

public class RandomMusic4 {

    //We turn on the inbuilt random generator:
    private static final Random R = new Random();

    //Our one symbol notation for notes :
    //A, _B, B, C #C D _E E F #F G #G
    //R S T C M D N E F O G P
    //A _B B c #c d _e e f #f g #g a _b b
    //A Q B c m d n e f o g p a q b
    private static final String SUPER_ALPHABET
        = "RSTAMBCNDOEFPGQambcndoeffpgq";

    private static final String ALPHABET = "RTCDEOGabcdeo";

    //Number of time unities per measure 4/4

```

```

    private static final int LIMIT44 = 8;
//Durations for 4/4
    private static final String
        SUPER_ALPHABET44 = "1111222222448";
//Header 4/4
    private static final String HEADER44
        = "X:0\n"
        + "T:Song 4/4\n"
        + "M:4/4\n"
        + "L:1/8\n"
        + "K:C\n";

//Number of time unities per measure 6/8
    private static final int LIMIT68 = 6;
//Durations for 6/8
    private static final String SUPER_ALPHABET68 = "11133336";
//Header 6/8
    private static final String HEADER68
        = "X:0\n"
        + "T:Song 6/8\n"
        + "M:6/8\n"
        + "L:1/8\n"
        + "K:C\n";

//The musiText is transformed into a tune,
//an ABC4J musical object
    private static Tune ABCmusiTextToTune(String musiText) {

        Tune tune = new TuneParser().parse(musiText);
        return tune;
    }

//A tune is played
    private static void playTune(Tune tune) {
        //The player converts the tune into
        //a midi message that is sent to
        //hardwared sound synthesizers.
        //If problems appear,

```

```

        //an input-out exception must be thrown.
        TunePlayer player = new TunePlayer();
        player.start();
        player.play(tune);
    }

//A tune as Tune is drawn
    private static void DrawTune(Tune tune) {
//creates a component that draws the melody on a stave
        JScoreComponent jscore = new JScoreComponent();
        jscore.setJustification(true);
        jscore.setTune(tune);
        JFrame j = new JFrame();
        j.add(jscore);
        j.pack();
        j.setVisible(true);
    }

//Randomness is made into a composer
    static private String randomNotes(int limit,
        String header, String durationsAlphabet) {
        boolean go = true;
        //Output initialization
        String stringOfNotes = "";
        int k;
        //CumulativeTime must not surpass n
        for (int cumulativeTime = 0; cumulativeTime < limit;) {
            //A note is generated:
            //A random integer less than the
            //number of chars in the alphabet is thrown
            k = R.nextInt(ALPHABET.length());
            //The integer is changed into a note
            char c = ALPHABET.charAt(k);
            //The note is added to the output
            stringOfNotes = stringOfNotes + c;
            //A duration is generated
            //but the limit time must not be surpassed
            while (go) {

```

```

//An integer is generated
k = R.nextInt(durationsAlphabet.length());
//The integer is changed into a
//char that represents a duration number
c = durationsAlphabet.charAt(k);
//the char is reread as a number
int nc = Character.getNumericValue(c);
cumulativeTime = cumulativeTime + nc;
//System.out.println(nc + " " + cumulativeTime);
//if cumulativeTime does not surpass the allowed time
//the duration is concatenated to the output
if (cumulativeTime <= limit) {
    stringOfNotes = stringOfNotes + c;
    go = false; //leave the while loop
} //else give another try
else {
    cumulativeTime = cumulativeTime - nc;
}
}
go = true;
}
return stringOfNotes;
}

//A string in our alphabet is rewritten in ABC notation
//Accidents are prefixes in ABC notation.
//Sharp is denoted by ^, flat by _
static private String interpreter(String ss) {
    ss = ss.replaceAll("R", "A,");
    ss = ss.replaceAll("S", "_B,");
    ss = ss.replaceAll("T", "B,");
    ss = ss.replaceAll("M", "#C");
    ss = ss.replaceAll("m", "#c");
    ss = ss.replaceAll("N", "_E");
    ss = ss.replaceAll("n", "_e");
    ss = ss.replaceAll("O", "^F");
    ss = ss.replaceAll("o", "^f");
    ss = ss.replaceAll("P", "^G");
}

```

```

        ss = ss.replaceAll("p", "^g");
        ss = ss.replaceAll("Q", "_B");
        ss = ss.replaceAll("q", "_b");
        return ss;
    }

    public static void main(String[] args) throws IOException {
        int limit = 0;
        String header = "";
        String durationsAlphabet = "";
        //Measure 44 for 4/4, 68 for 6/8
        int measure = 44;
        switch (measure) {
            case 44:
                limit = LIMIT44;
                header = HEADER44;
                durationsAlphabet = SUPER_ALPHABET44;
                break;

            case 68:
                limit = LIMIT68;
                header = HEADER68;
                durationsAlphabet = SUPER_ALPHABET68;
                break;
        }
        String ss = randomNotes(limit, header, durationsAlphabet);
        ss = interpreter(ss);
        System.out.println("Rhythm = " + ss);
        for (int i = 0; i < 4; i++) {
            ss = ss + ss;
        }
        String musiText = header + ss;
        Tune tune;
        tune = ABCmusiTextToTune(musiText);
        DrawTune(tune);
        playTune(tune);
    }
} //End of main class H55 RandomMusic4

```

56 pag 49.

The bar line is inserted after each copy of the generated pattern:

```
//Program H56 RandomMusic5
//Randomness is made into a generator of rhythms.
//Measures 4/4 and 6/8 are allowed.
//A bar line is inserted after each measure.

import java.io.IOException;
import java.util.Random;

import javax.swing.JFrame;

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneParser;
import abc.ui.swing.JScoreComponent;

public class RandomMusic5
{
    private static final Random R = new Random();

    //Our one symbol notation for notes :
    //A, _B, B, C #C D _E E F #F G #G
    //R S T C M D N E F O G P
    //A _B B c #c d _e e f #f g #g a _b b
    //A Q B c m d n e f o g p a q b
    private static final String SUPER_ALPHABET
        = "RSTAMBCNDOEFPGQambcndoeofpgq";

    private static final String ALPHABET = "RTCDEOGabcdeo";

    //Number of time unities per measure 4/4
    private static final int LIMIT44 = 8;
```

```

//Durations for 4/4
    private static final String
        SUPER_ALPHABET44 = "1111222222448";
//Header 4/4
    private static final String HEADER44
        = "X:0\n"
        + "T:Song 4/4\n"
        + "M:4/4\n"
        + "L:1/8\n"
        + "K:C\n";

//Number of time unities per measure 6/8
    private static final int LIMIT68 = 6;
//Durations for 6/8
    private static final String SUPER_ALPHABET68 = "11133336";
//Header 6/8
    private static final String HEADER68
        = "X:0\n"
        + "T:Song 6/8\n"
        + "M:6/8\n"
        + "L:1/8\n"
        + "K:C\n";

//The musiText is transformed into a tune,
//an ABC4J musical object
    private static Tune ABCmusiTextToTune(String musiText) {

        Tune tune = new TuneParser().parse(musiText);
        return tune;
    }

//A tune is played
    private static void playTune(Tune tune) {
        //The player converts the tune into
        //a midi message that is sent to
        //hardwared sound synthesizers.
        //If problems appear,
        //an input-out exception must be thrown.
    }

```

```

        TunePlayer player = new TunePlayer();
        player.start();
        player.play(tune);
    }

//A tune as Tune is drawn
    private static void DrawTune(Tune tune) {
//creates a component that draws the melody on a stave
        JScoreComponent jscore = new JScoreComponent();
        jscore.setJustification(true);
        jscore.setTune(tune);
        JFrame j = new JFrame();
        j.add(jscore);
        j.pack();
        j.setVisible(true);
    }

//Randomness is made into a composer
    static private String randomNotes(int limit,
        String header, String durationsAlphabet) {
        boolean go = true;
        //Output initialization
        String stringOfNotes = "";
        int k;
        //CumulativeTime must not surpass n
        for (int cumulativeTime = 0; cumulativeTime < limit;) {
            //A note is generated:
            //A random integer less than the
            //number of chars in the alphabet is thrown
            k = R.nextInt(ALPHABET.length());
            //The integer is changed into a note
            char c = ALPHABET.charAt(k);
            //The note is added to the output
            stringOfNotes = stringOfNotes + c;
            //A duration is generated
            //but the limit time must not be surpassed
            while (go) {
                //An integer is generated

```

```

        k = R.nextInt(durationsAlphabet.length());
        //The integer is changed into a
        //char that represents a duration number
        c = durationsAlphabet.charAt(k);
        //the char is reread as a number
        int nc = Character.getNumericValue(c);
        cumulativeTime = cumulativeTime + nc;
        //System.out.println(nc + " " + cumulativeTime
        //if cumulativeTime does not surpass the allowed
        //the duration is concatenated to the output
        if (cumulativeTime <= limit) {
            stringOfNotes = stringOfNotes + c;
            go = false; //leave the while loop
        } //else give another try
        else {
            cumulativeTime = cumulativeTime - nc;
        }
    }
    go = true;
}
return stringOfNotes;
}

//A string in our alphabet is rewritten in ABC notation
//Accidents are prefixes in ABC notation.
//Sharp is denoted by ^, flat by _
static private String interpreter(String ss) {
    ss = ss.replaceAll("R", "A,");
    ss = ss.replaceAll("S", "_B,");
    ss = ss.replaceAll("T", "B,");
    ss = ss.replaceAll("M", "#C");
    ss = ss.replaceAll("M", "#c");
    ss = ss.replaceAll("N", "_E");
    ss = ss.replaceAll("n", "_e");
    ss = ss.replaceAll("O", "^F");
    ss = ss.replaceAll("o", "^f");
    ss = ss.replaceAll("P", "^G");
    ss = ss.replaceAll("p", "^g");
}

```

```

        ss = ss.replaceAll("Q", "_B");
        ss = ss.replaceAll("q", "_b");
        return ss;
    }

    public static void main(String[] args) throws IOException {
        int limit = 0;
        String header = "";
        String durationsAlphabet = "";
        //Measure 44 for 4/4, 68 for 6/8
        int measure = 44;
        switch (measure) {
            case 44:
                limit = LIMIT44;
                header = HEADER44;
                durationsAlphabet = SUPER_ALPHABET44;
                break;

            case 68:
                limit = LIMIT68;
                header = HEADER68;
                durationsAlphabet = SUPER_ALPHABET68;
                break;
        }
        String ss = randomNotes(limit, header, durationsAlphabet);
        String pattern = interpreter(ss);
        System.out.println("Pattern = " + pattern);
        String rhythm = "";
        for(int i = 0; i < 10 ; i++)
            rhythm = rhythm + pattern + "|";
        System.out.println("Rhythm = " + rhythm);
        String musiText = header + rhythm;
        Tune tune;
        tune = ABCmusiTextToTune(musiText);
        DrawTune(tune);
        playTune(tune);
    }
} //End of main class H56 RandomMusic5

```

57 pag 49.

The Author is very ambivalent, sometimes he likes rather few rhythms while sometimes he likes many of them if only he can imagine a situation where they could fit. Anyway, 50% is a good estimation for the proportion of random rhythms generated by RandomMusic5 that he likes. The implications that the Author sees are:

1. Random musical strings that encode musical notes all of the same duration are in general uninteresting.
2. Random musical strings that encode musical notes of different durations may produce patterns some 6 notes long that are interesting in a proportion of, say, greater than 10%.
3. The cost for the developer to pass from a zero proportion of interesting strings to a nonzero proportion is rather low: some 40 lines of code that must be added to a previous one of 100 lines that runs over an extremely powerful IDE.
4. Extrapolations: to allow a change in the duration of notes is the first step into regulation. It is not known to us a form of life that would correspond with strings of notes of the same duration, i.e., a form of life without gene regulation. All forms of life include sophisticated forms of regulation that tentatively could be mimicked by our music in which the duration of notes can be changed. Can we explain the lack of forms of unregulated genomes? Can that be done in a scientific way that future generations will accept our proof? An exit to this conundrum would be as follows: early forms of life were of course deprived of regulation loops but those forms rapidly disappeared in the struggle for life against regulated ones. Why is there no fossils of that forms of life? Fossils are difficult to find for every form of life and so sampling effects are responsible for the lack of aforementioned early forms of life. Now, is this all that science can produce? Where is the simulations made by the eJCommunity to study what happens to a cell when every kind of regulation is erased? (Strong warning: a simulation of gene regulation must include the molecular level because molecular recognizing is at the basis of pairing of DNA with

the DNA-protein ensembles that direct gene regulation and moreover it is also at the basis of the pairing of enzymes and reactants.)

61 pag 55. If the first amino acid of the given protein is deleted at the beginning, a new tune of very different sound results.

64 pag 56. We associate an ABC tune to a DNA string through our own genetic code as follows: we take codons of two bases and to the first base of a codon we associate a note while a duration is associated to the second.

Our genetic code is shown below. In the first line we see our 16 codons with a pair of bases. In the second line we see the associated notes. In the third the duration for 4/4 and in the last line the durations for 6/8. A rest is included as a note.

```
//Program H64 DNAMusic2
//Shows how one can use a DNA sequence
//to compose the notes and their durations of an ABC tune.
//Measures 4/4 and 6/8 are allowed.
//A bar line is inserted after each measure.
import java.io.IOException;

import javax.swing.JFrame;

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneParser;
import abc.ui.swing.JScoreComponent;

public class DNAMusic2 {

//Our genetic code: In the first line we see our 16 codons
//with a pair of bases. In the second line we see the
//associated notes. In the third the duration for 4/4
//and in the last line the durations for 6/8.
//A rest is included as a note.
// AA AT AC AG TA TA TC TG CA CT CC CG GA GT GC GG
// C #C D _E E F #F G #G A _B B c #c d z
```

```

// 1 1 1 1 2 2 2 2 2 2 2 4 4 4 8 4
// 1 1 1 1 1 1 1 3 3 3 3 3 3 3 6 3
//The set of codons encoded as a string.
//Z stands for a separator that will be
//interpreted as rest by ABC
    static private final String CODONS
        = "AA-AT-AC-AG-TA-TA-TC-TG-CA-CT-CC-CG-GA-GT-GC-GG-"

    private static final String[] NOTES
        = {"C", "^C", "D",
           "_E", "E", "F", "^F", "G",
           "^G", "A", "_B", "B", "c",
           "^c", "d", "_e", "e", "f", "^f", "z"};

//Durations for 4/4
    private static final String DURATIONS_ALPHABET44
        = "1111122222222222444888";
//Header 4/4
    private static final String HEADER44
        = "X:0\n"
          + "T:Song 4/4\n"
          + "M:4/4\n"
          + "L:1/8\n"
          + "K:C\n";

//Durations for 6/8
    private static final String DURATIONS_ALPHABET68
        = "11111111113333333336666";
//Header 6/8
    private static final String HEADER68
        = "X:0\n"
          + "T:Song 6/8\n"
          + "M:6/8\n"
          + "L:1/8\n"
          + "K:C\n";

//The musiText is transformed into a tune,
//an ABC4J musical object

```

```

private static Tune ABCmusiTextToTune(String musiText) {

    Tune tune = new TuneParser().parse(musiText);
    return tune;
}

//A tune is played
private static void playTune(Tune tune) {
    //The player converts the tune into
    //a midi message that is sent to
    //hardwared sound synthesizers.
    //If problems appear,
    //an input-out exception must be thrown.
    TunePlayer player = new TunePlayer();
    player.start();
    player.play(tune);
}

//A tune as Tune is drawn
private static void DrawTune(Tune tune) {
//creates a component that draws the melody on a stave
    JScoreComponent jscore = new JScoreComponent();
    jscore.setJustification(true);
    jscore.setTune(tune);
    JFrame j = new JFrame();
    j.add(jscore);
    j.pack();
    j.setVisible(true);
}

//Codons are taken by pairs.
//The first codon
//is interpreted as a note.
//Interpretation is done codon per codon.
    static private String notesInterpreter(String cd) {
        //The index of cd is looked at the list of amino acids
        cd = cd + "-";
        int i = CODONS.indexOf(cd);

```

```

        i = i / 3;
        //System.out.println("i = " + i);
        //The corresponding note is looked at the array of notes
        String ss = NOTES[i];
        return ss;
    }

//Codons are taken by pairs.
//The second codon
//is interpreted as a duration.
//Interpretation is done codon per codon.
    static private char durationsInterpreter(String cd,
        int measure) {
        //The index of cd is looked at the list of amino acids
        cd = cd + "-";
        int i = CODONS.indexOf(cd);
        i = i / 3;
        //The corresponding duration is looked
        //at the list of durations
        char c;
        if (measure == 44) {
            c = DURATIONS_ALPHABET44.charAt(i);
        } else {
            c = DURATIONS_ALPHABET68.charAt(i);
        }
        return c;
    }

//A dna sequence is transformed into a tune.
//It is divided in pair of codons
//each one with 2 letters,
//the first codon is interpreted as a note,
//the second as a duration
    static private String interpreter(String s, int measure) {
        String v = "";
        int l = s.length();
        System.out.println("l = " + l);
        for (int i = 0; i < l - 4; i = i + 4) {

```

```

        String cd = s.substring(i, i + 2);
        //System.out.println("cd = " + cd);
        String note = notesInterpreter(cd);
        cd = s.substring(i + 2, i + 4);
        //System.out.println("cd = " + cd);
        char duration = durationsInterpreter(cd, measure);
        v = v + note + duration;
    }
    return v;
}

public static void main(String[] args) throws IOException {

    String header;
    //Measure 44 for 4/4, 68 for 6/8
    int measure = 44;
    if (measure == 44) {
        header = HEADER44;
    } else {
        header = HEADER68;
    }

    //http://www.prodoric.de/seq.php?spos=3938498&epos=3939519&replicon_no=
    //PRODORIC logo Release 8.9
    //Cited 18/VI/2018
    String adn
        =
    "GCGTTCACGGCGAAAGTTCTCTTTTAATGCCGCCATTTTCGGCCATTAACGTATTGAGATGCGCTTTCTGT
    C\n"
        +
    "CAACCGTTGCCAGTAGGCAGGCTTGCGGCTCCAGACAGATACGCATAACCAGAAAGTGATCGATGACCTG
    C\n"
        +
    "TCTTCTGTTCATCCACCAGGTAAGCAATTCCTGATCAAGAAAATTCAGTTCGATTGTGGCATGACCCGAG
    G\n"
        +
    "CGGTTCGCGGTAAAACCATCCCTTTTGCCGTTAACGTTTTGACCGCTTCGCGTACCGCTGTACGACTCACT

```

```

T\n"
      +
"CGCCCAGCTCAATCTCACCGGGCAAATGGTGCCGGGTTTCATATTCACCTTTTAAGATCCGTT
C\n"
      +
"AGAACATACGAAAGGTTTTTCTGTGCAGCTAACTGTTGTGCGCTTAAAGGCATTACTTATCTT
C\n"
      +
"CTCCTTAGTATGCCACCAGGAAGTGTGATTACGGTTGCAAAAACGGCAAATTGCTTGTTTTAT
T\n"
      +
"TTTGCTGAAAAAATGCGCGGTCAGAAAATTATTTTAAATTTCTCTTGTCAGGCCGGAATAAC
C\n"
      +
"CACTGACACGGAACAACGGCAAACACGCCCGGGTCAGCGGGGTTCTCCTGAGAACTCCGGC
A\n"
      +
"ATGCTTGACTCTGTAGCGGGAAAGCGTATTATGCACACCCCGCGCCGCTGAGAAAAAGCAAAG
C\n"
      +
"AATTTATCAGACAATCTGTGTGGGCACTCGAAGATACGGATTCTTAACGTCGCAAGACGAAAA
A\n"
      +
"AGAGTGAACACGTAATTCATTACGAAGTTTAATTCTTTGAGCATCAAACCTTTTAAATTGAAGA
G\n"
      +
"ATTGAACGCTGGCGGCAGGCCTAACACATGCAAGTCGAACGGTAAACAGGAAACAGCTTGCTG"
      //adn is is trimmed to a length multiple of 4
      int l = adn.length();
      int f = l / 4;
      int g = f * 4;
      adn = adn.substring(0, g);

      String ABCstring = interpreter(adn, measure);
      System.out.println("ABCstring = " + ABCstring);
      String musiText = header + ABCstring;
      Tune tune;
      tune = ABCmusiTextToTune(musiText);

```

```

        DrawTune(tune);
        playTune(tune);
    }
} //End of main class H64 DNAMusic2

```

66 pag 56. Just to illustrate how large is the possible variability of the translation machinery from DNA to tunes, let us figure out codons with one basis, two, three, four or of whatever number of bases. This amount to infinitely many forms of using DNA to synthesize ABC tunes. Now, by a handful of reasons one would prefer rather short codons to large ones. But even under the restriction of codons of equal length, there are many forms as variability of genetic codes and translation can arise, say, by shifting or permuting the associated notes. Given that we have just one biological genetic code (or nearly so), science has tried to explain this by resorting to natural selection. Nevertheless, one expects many equally functional variants: ¿Where are they?

71 pag 65.

Our guess to grade the complexity of the series of tests has been to try to maximize the easiness of each test by keeping the notes of the working set as separated as possible. To devise the rule for passing the current level we consider that a human being loves achievable challenges and hates recurrent failure. So, we define the tolerance as the number of trials that one is allowed to make without being classified as wrong. So, if one finds the target pitch in within the tolerance number of trials, one succeeds. To pass to the next level one must have a series of uninterrupted successes. The corresponding number is the runOfSuccesses. Aforementioned numbers depend on the complexity level and on the expertise of the User (deaf, beginner, intermediate, advanced and expert). The conclusion of the Author is: pitch recognizing is extremely difficult and the program seems to be a marvelous help if only the student has enough patience.

```

//Program H71 PicthRecognizing3
//A training facility
//for pitch recognizing.
//Once a complexity level and a degree of expertise

```

```

//have been chosen,
//a succession of questions is given to the User.
//A certain tolerance is admitted.
//Problem: there is no the way to go to the upper level

import java.util.Random;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.ListSelectionModel;

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneParser;

import javax.swing.event.ListSelectionEvent;
import java.awt.*;

public class PitchRecognizing3 extends JFrame {

    private static final long serialVersionUID = 1L;

    //This is the set of all NOTES.
    //The WORKING_SET for level k takes
    //the first k elements of this set.
    private static final String[] NOTES
        = {"C", "c", "c'", "G", "g", "E", "e",
          "A", "A", "a", "D", "d", "F", "f",
          "B", "B", "b", "^F", "^f", "_B", "_B", "_b",
          "^C", "^c", "^G", "^g", "_E", "_e"};

    //Each test has a complexity level
    //which is the size of the working set.
    private static final int LEVEL_OF_COMPLEXITY = 5;

```

```

//The actual set of NOTES determined by
// the level of complexity
    private static final String[] WORKING_SET
        = new String[LEVEL_OF_COMPLEXITY];

//Number of trials granted before failure declaration
static private int tolerance;

//The User must declare which degree of expertise
//he or she wants to play with:
//0 (beginner), 1(intermediate), 2 (advanced), 3 (expert).
static private int expertise;

//To graduate from the current level, you need
//a run of success, without a single failure in within.
static private int runOfSuccesses;
//Actual number of contiguous successes
static private int contiguousSuccesses = 0;

//Every ABC tune must have a correct HEADER
private static final String HEADER
    = "X:0\n"
    + "T:Song 4/4\n"
    + "M:4/4\n"
    + "L:1/8\n"
    + "K:C\n";

//The note that must be recognized as a tune
static private Tune tuneTest;
//The note that must be recognized as a String
static private String noteTest;
//A JList to be displayed
private JList list = new JList(WORKING_SET);
private JLabel barMessage;
//Counter of trials before success
static private int counterOfTrials = 1;
static private int oldK = -1;

```

```

//A rest prior the playing of a tune.
static private Integer lag = 8;

private JPanel mPanel;
//Constructor: this is the first task of the program

public PitchRecognizing3() {
    //Title in the upper bar of the pane
    super("Listen to the sound and select the pitch");
    mPanel = new JPanel();
    mPanel.setBackground(Color.gray);
    super.add(mPanel, BorderLayout.CENTER);
    barMessage = new JLabel("Click on your selection");
    super.add(barMessage, BorderLayout.SOUTH);
    list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    mPanel.add(new JScrollPane(list));
    //The answer by the student is captured and assessed
    list.addListSelectionListener((ListSelectionEvent e) \ri
        String noteAnswer
            = NOTES[list.getSelectedIndex()];
        String NOTESBoth
            = noteTest + "2" + noteAnswer + "2";
        Tune tuneBoth = noteToTune(NOTESBoth);
        playTune(tuneBoth);
        //If answer is correct
        if (noteAnswer.contentEquals(noteTest)) {
            //To avoid double response from the system
            if (!e.getValueIsAdjusting()) {
                //mPanel.setBackground(Color.GREEN);
                barMessage.setText(
                    String.format(
                        "Your answer = " + noteAnswer
                            + " is RIGHT!" + " |
                    + counterOfTrials
                )
            );
            lag = 8;

```

```

poseQuestion(lag, LEVEL_OF_COMPLEXITY);
counterOfTrials = 1;
contiguousSuccesses++;
if (contiguousSuccesses == runOfSuccesses) {
    barMessage.setText(
        String.format(
            "CONGRATULATIONS: you can pass"
            + " to the following le
        )
    );
    mPanel.removeAll();
}
}
} else //if answer is wrong
{
    //To avoid double response from the system
    if (!e.getValueIsAdjusting()) {
        //mPanel.setBackground(Color.RED);
        barMessage.setText(String.format("Your answer = " +
noteAnswer
            + " was WRONG!"));
        counterOfTrials++;
        //If tolerance is exceeded
        if (counterOfTrials > tolerance) {
            contiguousSuccesses = 0; //begin anew
        }
    }
}
});
super.setSize(400, 200);
super.setVisible(true);
}

//We turn on the inbuilt random generator:
private static final Random R = new Random();

//creates a simple midi PLAYER to play the melody
private static final TunePlayer PLAYER = new TunePlayer();

```

```

//The musiText is transformed into a tune,
//an ABC4J musical object
private static Tune ABCmusiTextToTune(String musiText) {
    Tune tune = new TuneParser().parse(musiText);
    return tune;
}

//A tune is played
private static void playTune(Tune tune) {
    PLAYER.play(tune);
}

//The note is converted into an ABC tune
static private Tune noteToTune(String note) {
    String musiText = HEADER + note;
    Tune tune = ABCmusiTextToTune(musiText);
    return tune;
}

//A complexity level defines each working set
//of NOTES to be recognized.
static private String[] BuildWorkingSet(int level) {
    //for (int i = 0; i < level; i++)
    System.arraycopy(NOTES, 0, WORKING_SET, 0, level);
    return WORKING_SET;
}

//A random note from the chosen alphabet is output
static private String randomNote() {
    String note ;
    int newK;
    //A random integer less than the
    //number of allowed NOTES is thrown
    newK = R.nextInt(LEVEL_OF_COMPLEXITY);
    //the new challenge is different than the prior one
    if (oldK == newK) {
        newK = (newK + 1) % LEVEL_OF_COMPLEXITY;
    }
}

```

```

    }
    //The integer is changed into a note
    note = NOTES[newK];
    oldK = newK;
    return note;
}

//A string in our alphabet is rewritten in ABC notation.
//Accidents are prefixes in ABC notation.
//Sharp is denoted by ^, flat by _
static private String interpreter(String ss) {
    ss = ss.replaceAll("R", "A,");
    ss = ss.replaceAll("S", "_B,");
    ss = ss.replaceAll("T", "B,");
    ss = ss.replaceAll("M", "#C");
    ss = ss.replaceAll("M", "#c");
    ss = ss.replaceAll("N", "_E");
    ss = ss.replaceAll("n", "_e");
    ss = ss.replaceAll("O", "^F");
    ss = ss.replaceAll("o", "^f");
    ss = ss.replaceAll("P", "^G");
    ss = ss.replaceAll("p", "^g");
    ss = ss.replaceAll("Q", "_B");
    ss = ss.replaceAll("q", "_b");
    return ss;
}

//The number of granted trials before failure declaration
//depends on expertise and the complexity level
static private int salvationNumber(int level,
    int expertise) {
    int grace = 0;
    if ((level <= 5)) {
        grace = level;
    }
    if ((level > 5)) {
        grace = 6 - expertise;
    }
}

```

```

        return grace;
    }

//A random note to be recognized is played
static private void poseQuestion(Integer sleep, int level) {
    String s = sleep.toString();
    noteTest = randomNote();
    tuneTest
        = noteToTune("z" + s + noteTest + noteTest + noteTest);
    playTune(tuneTest);
}

public static void main(String[] args) {
    PLAYER.start();
    //Rest before the first question
    lag = 1;
    //Expertise:
    //1 (deaf), 2 (beginner), 3 (intermediate),
    //4 (advanced), 5 (expert).
    expertise = 1;
    //Number of contiguous successes to graduate
    //from the current level
    runOfSuccesses = 2 * LEVEL_OF_COMPLEXITY;
    //Number of trials granted before failure declaration
    tolerance
        = salvationNumber(LEVEL_OF_COMPLEXITY, expertise);
    System.out.println("Level of complexity = "
        + LEVEL_OF_COMPLEXITY);
    System.out.println("Expertise level = " + expertise);
    System.out.println("Needed run of successes"
        + " to pass to the following level = " + runOfSu
    System.out.println("You succeed if you find "
        + "the answer before " + tolerance + " trials"

    BuildWorkingSet(LEVEL_OF_COMPLEXITY);
    //Instantiation of the application
    PitchRecognizing3 t = new PitchRecognizing3();
    t.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

```

```

        poseQuestion(lag, LEVEL_OF_COMPLEXITY);
    }
} //End of main class H71 PicthRecognizing3

```

85 pag 92. An attempt to extrapolate our negative experiences to biology is a rampant failure: the phenomenon of local DNA inversions (which is a very specific type of permutation) is well known to cause speciation in flies. Because the mechanism is quite simple, the Author agrees that this evolutionary explanation is the correct one: after inversion, recombination is not allowed because of mispairing and so, there is separation of lineages and mutation causes further divergence that being insignificant is anyway important.

Problems of Chapter 5

93 pag 160. A look to our many programs of this series that implement evolution shows that they are all different. So, evolution can be implemented in infinitely many ways. Which is the best? The best solution to complex problems in general cannot be found but instead many very good solutions can be chased. So, many different implementations of evolution should exist on Earth if only natural law were the cause of the origin of life and of its diversification. Where are they?

Problems of Chapter 6

105 pag 211. A slightly simplified version of Program H103 Lamarck-Music1:

```

//Program H105 LamarckMusic2
//Lamarckian evolution is made into a composer.
//Different durations of notes are allowed.
//Theoretical guide to composing music:
//Notes are pleasant in isolation but when
//they go one after another in a tune,
//the interval, the number of semitones in

```

```
//within notes, is what matters.

//Our encoding in interval notation:
//Example: T6+33-51+91+83-21-51+01
// an initial note + durations: T6,
// plus a string of codons with three letters:
//the first is a sign to indicate the direction
//of the interval: up else down,
//the second is an interval in semitones,
//the third is a duration.
//Example:
//T6+33 means: begins with B, with a duration of 6,
//prepare to rise pitch,
//move 3 semitones up to D
//and play it for 3 units of time.
//The program outputs ABC musiTexts.
//The performance of this program leaves
//to much to be desired.
//The reason is that the interval notation
//is not appropriate for mutation
//because any mutation affects all that comes next.
//Overflow problems arise, say resultant notes
//are outside allowed range and/or scale.
//To search for something better is a must for the Reader.
//This is a slightly simplified version of EvoMusic3
import java.util.Random;

import javax.swing.JFrame;
import javax.swing.JOptionPane;

import abc.midi.TunePlayer;
import abc.notation.Tune;
import abc.parser.TuneParser;
import abc.ui.swing.JScoreComponent;

public class LamarckMusic2 {

    //The PLAYER converts the tune into
```

```

//a midi message that is sent to
//hardwared sound synthesizers.
//If problems appear,
//an input-out exception must be thrown.
private static final TunePlayer PLAYER = new TunePlayer();

//Our one symbol notation. Notes in the first line,
//our one letter notation in the second.
//A, _B, B, C #C D _E E F #F G #G A _B B c #c d _e e f #f g #g a _b l
//R S T C M D N E F O G P A Q B c m d n e f o g p a q l
private static final String SUPER_ALPHABET
    = "RSTCMDNEFOGPAQBcmdnefогpaqb";
//Actual alphabet for work (scale G)
private static final String ACTUAL_ALPHABET = "RTCDEOGABcdeo";

//The ABC header
static private String header;

//The working alphabet of durations.
static private String durationsAlphabet;

//Durations for 4/4
private static final String DURATIONS_ALPHABET44 = "1111222222448";

//Header 4/4
private static final String HEADER44
    = "X:0\n"
    + "T:Song 4/4\n"
    + "M:4/4\n"
    + "L:1/8\n"
    + "K:C\n";

//Durations for 6/8
private static final String DURATIONS_ALPHABET68 = "111111336";

//Header 6/8
private static final String HEADER68
    = "X:0\n"

```

```

        + "T:Song 6/8\n"
        + "M:6/8\n"
        + "L:1/8\n"
        + "K:C\n";

    /*
    *
    Intervals in semitones by rank according to pleasantness
    (According to the Author):
    The best: rank zero: 0, 4, 5, 7, 12.
    Rank one: 9,2
    %%Rank two: 8, 3, 6.
    %%Rank three: 1, 10, 11
    */

    /*
    * The 13 INTERVALS in one letter notation:
    0 1 2 3 4 5 6 7 8 9 10 11 12
    0 1 2 3 4 5 6 7 8 9 a b c
    */
    private static final String INTERVALS = "0123456789abc";
    //Rank of the 13 INTERVALS of the scale
    //{0,3,1,2,0,0,2,0,2,1,3,3,0}

    private static final String RANK_ZERO = "0457c";
    private static final String RANK_ONE = "92";
    private static final String RANK_TWO = "836";
    private static final String RANK_THREE = "1ab";
    //probabilities of appearance depends on rank:
    private static final double PROB[] = {0.4, 0.3, 0.2, 0.1};

    //The interval can be upwards (1) or downwards (-1);
    static private char signInterval = 1;
    static private int lastNoteIndex;
    static private boolean print;
    static private boolean printThis;
    static private boolean drawTune;

```

```

//We turn on the inbuilt random generator:
    private static final Random R = new Random();

//The musiText is transformed into a tune,
//an ABC4J musical object
    private static Tune ABCmusiTextToTune(String musiText) {
        Tune tune = new TuneParser().parse(musiText);
        return tune;
    }

//A tune is played
    private static void playTune(Tune tune) {
        PLAYER.play(tune);
    }

//A tune as Tune is drawn
    private static void DrawTune(Tune tune) {
//creates a component that draws the melody on a stave
        JScoreComponent jscore = new JScoreComponent();
        jscore.setJustification(true);
        jscore.setTune(tune);
        JFrame j = new JFrame();
        j.add(jscore);
        j.pack();
        j.setVisible(true);
    }

//Output a random letter from the chosen alphabet
    static private char randomLetter(String alphabet) {
        int k;
        //A random integer less than the
        //number of chars in the alphabet is thrown
        k = R.nextInt(alphabet.length());
        //The integer is changed into a letter of the alphabet
        char c = alphabet.charAt(k);
        return c;
    }

```

```

//Char c is decoded into a number: 'A' is zero.
private static int charToNumber(char c) {
    String s1 = "" + c;
    int number = 0;
    for (int j = 0; j < 26; j++) {
        char d = (char) (j + 97);
        String s2 = "" + d;
        if (s1.equals(s2)) {
            number = j + 10;
        }
    }
    return number;
}

//An interval, a number, is proposed
//with a probability according to its rank.
static private int proposeInterval() {
    int k;
    Character f = ' ';
    double w = R.nextDouble();
    if (w > 1 - PROB[3]) {
        f = randomLetter(RANK_THREE);
    }
    if (w < PROB[0] + PROB[1] + PROB[2]) {
        f = randomLetter(RANK_TWO);
    }
    if (w < PROB[0] + PROB[1]) {
        f = randomLetter(RANK_ONE);
    }
    if (w < PROB[0]) {
        f = randomLetter(RANK_ZERO);
    }

    k = Character.getNumericValue(f);
    if (k > 9) {
        k = charToNumber(f);
    }
}
/* This replaces the following code:

```

```

    if (f == 'a') k = 10;
    if (f == 'b') k = 11;
    if (f == 'c') k = 12;
    if (f == 'd') k = 13;
    if (f == 'e') k = 14;
    if (f == 'f') k = 15;
    if (f == 'g') k = 16;
    if (f == 'h') k = 17;
    if (f == 'i') k = 18;
    if (f == 'j') k = 19;
    if (f == 'k') k = 20;
    if (f == 'l') k = 21;
    if (f == 'm') k = 22;
    if (f == 'n') k = 23;
    if (f == 'o') k = 24;
        */
        if (print) {
            System.out.println("w = " + w + " k = " + k);
        }
        return k;
    }

//Output a random interval as string:
//the first char is the sign: + upwards, - downwards.
//the second is the interval as a one letter number.
//The rank of INTERVALS determines its
//probability of occurrence.
//The output must be compatible with both
//SUPER_ALPHABET and ACTUAL_ALPHABET,
//this is determined by actualNoteIndex
//the index of the last note in the SUPER_ALPHABET
    static private String randomInterval() {
        if (print) {
            System.out.println("RandomInterval");
        }
    }
//Output
    String randomInterval = "";
    String newNote;

```

```

int newNoteIndex = lastNoteIndex;
int direction;
boolean generated = false;
char intervalOneLetter;
while (generated == false) {
    // a direction is created
    double p = R.nextDouble();
    if (p > 0.5) {
        signInterval = '+';
        direction = 1;
    } else {
        signInterval = '-';
        direction = -1;
    }
    //a random number is generated
    int n = proposeInterval();

    int newTryIndex = newNoteIndex + direction * n;
    //The corresponding note is checked for compatibility
    if ((newTryIndex >= 0)
        & (newTryIndex < SUPER_ALPHABET.length())) {
        newNote = "" + SUPER_ALPHABET.charAt(newTryIndex
        );
        //Is the new note in the actual alphabet of note
        if (ACTUAL_ALPHABET.contains(newNote)) {
            generated = true;
            newNoteIndex = newTryIndex;
            lastNoteIndex = newNoteIndex;
            //Interval in one letter notation
            intervalOneLetter = INTERVALS.charAt(n);
            randomInterval = ""
                + signInterval + intervalOneLetter;
            if (print) {
                System.out.println("New note = " + newNo
                );
            }
        }
    }
}

```

```

        return randomInterval;
    }

//Randomness is made into a composer.
//This method outputs a random musiText in interval notation:
//The first note comes as a note in one letter notation
//plus a duration
//Notes occupy even places in the music of the musiText
//while durations go in odd places.
    static private String randomMTIN() {
        //MusiText in interval notation
        String m;
        //An initialNote begins the musiText
        String c = "" + randomLetter(ACTUAL_ALPHABET);
        if (print) {
            System.out.println("First letter = " + c);
        }
        int firstNoteIndex = SUPER_ALPHABET.indexOf(c);
        //Duration is added
        m = "" + c + randomLetter(durationsAlphabet);
        lastNoteIndex = firstNoteIndex;
        //For the following notes
        //INTERVALS and durations are next added
        for (int i = 0; i < 7; i++) {
            //An interval is generated
            c = randomInterval();
            //The interval is added to the output
            m = m + c;
            //A random duration is generated
            c = "" + randomLetter(durationsAlphabet);
            //The duration is added to the output
            m = m + c;
        }
        if (print) {
            System.out.println("Random ind = " + m);
        }
        return m;
    }

```

```

    }

//Input: a musiText in interval one letter notation
//output: a musiText in ABC one letter notation
    private static String transcription(String a) {
        int m = a.length();
        //Output musiText in one letter ABC notation
        String z;
        //The first two letters encode the initial note
        //and duration, these are already in ABC one letter nota
        z = a.substring(0, 2);
        int newNoteIndex = SUPER_ALPHABET.indexOf(a.charAt(0));
        //The rest of the string encodes INTERVALS and durations
        //Our codons have three letters:
        //one for the direction of the interval,
        //the other for the interval,
        //and the third for duration

        char newNote;
        int k;
        for (int i = 2; i < m;) {
            //The first letter is a sign:
            //the direction of the interval.
            char c = a.charAt(i++);
            //The second letter denotes an interval
            Character f = a.charAt(i++);
            //The third note represents a duration
            char third = a.charAt(i++);
            if (print) {
                System.out.println("char 0,1,2 = " + c
                    + " " + f + " " + third);
            }
            //Direction is decoded
            int direction;
            if (c == '+') {
                direction = 1;
            } else {
                direction = -1;
            }
        }
    }
}

```

```

    }
    //Interval is decoded
    int l = Character.getNumericValue(f);
    if ((l >= 0) & (l < 10)) {
        k = l;
    } else {
        k = charToNumber(f);
        /*This replaces:
if (f == 'a') k = 10;
if (f == 'b') k = 11;
if (f == 'c') k = 12;
if (f == 'd') k = 13;
if (f == 'e') k = 14;
if (f == 'f') k = 15;
if (f == 'g') k = 16;
if (f == 'h') k = 17;
if (f == 'i') k = 18;
if (f == 'j') k = 19;
if (f == 'k') k = 20;
if (f == 'l') k = 21;
if (f == 'm') k = 22;
if (f == 'n') k = 23;
if (f == 'o') k = 24;
        */
    }
    if (print) {
        System.out.println(" k = " + k);
    }

    newNoteIndex = newNoteIndex + direction * k;
    newNote = SUPER_ALPHABET.charAt(newNoteIndex);
    //The new note is assembled
    z = z + newNote + third;

}
if (print) {
    System.out.println("mTIntNotation = " + a
        + "\n mT in one letter notation = = " + z + "\n");
}

```

```

    }
    return z;
}

//A musiText in interval notation is rewritten
//as a normal ABC musiText.
//NOtes occupy even positions, beginning from zero,
//and durations odd ones.
//Accidents are prefixes in ABC notation.
//Sharp is denoted by ^, flat by _
    static private String interpreter(String ss) {
        ss = ss.replaceAll("R", "A,");
        ss = ss.replaceAll("S", "_B,");
        ss = ss.replaceAll("T", "B,");
        ss = ss.replaceAll("M", "#C");
        ss = ss.replaceAll("M", "#c");
        ss = ss.replaceAll("N", "_E");
        ss = ss.replaceAll("n", "_e");
        ss = ss.replaceAll("O", "^F");
        ss = ss.replaceAll("o", "^f");
        ss = ss.replaceAll("P", "^G");
        ss = ss.replaceAll("p", "^g");
        ss = ss.replaceAll("Q", "_B");
        ss = ss.replaceAll("q", "_b");
//ss = ss.replaceAll("+", "");
//ss = ss.replaceAll("-", "");
        return ss;
    }

//=====
//==== Evolution=====
//=====
//Strings that represent ABC tunes
//are synthesized at random
//selected, recombined, mutated
//reproduced.
//This is a fair model of evolution:
//strings encode for an ABC tune (genotype)

```

```

//which are listened at by the User (phenotype)
//who gives scores to tunes
//that direct reproduction.
//Global variables.
//They are used all throughout the whole class.
//The number of individuals must be
//less than N_MAX
    static final int N_MAX = 1000;

//Ind is a population of individuals
    static String Ind[] = new String[N_MAX];
//A helping array to keep a clone of Ind1
    static String IndClone[] = new String[N_MAX];
//The score of each tune is kept here
    static double Fitness[] = new double[N_MAX];
//Fitness is used to give a rank to individuals
//that define their reproductive possibilities
    static int Ordered[] = new int[N_MAX];
//The population has NUMB_IND members.
    private static final int NUMB_IND = 4;
    static private int generation;

    /* We generate NUMB_IND individuals (strings)
maxTime notes long.
Sequences are completely random */
    private static void Initialization() {
        if (print) {
            System.out.println("ORIGINAL POPULATION  \n");
        }
        for (int i = 0; i < NUMB_IND; i++) {
            Ind[i] = randomMTIN();
            if (print) {
                System.out.println("Individual " + i
                    + " = " + Ind[i]);
            }
        }
        for (int i = 0; i < NUMB_IND; i++) {

```

```

        Ordered[i] = 0;
    }
}

private static void printPopulation() {
    for (int i = 0; i < NUMB_IND; i++) {
        System.out.println(Ind[i]);
    }
}

//The User assess each tune by giving a score.
//ind is a string that encodes for a tune
//in one-letter notation.
private static double assessInd(String indNI) {
    //A musicText in Interval Notation
    //is translated into normal ABC notation
    String ABCMusitext = transcription(indNI);
    //ind is translated to ABC notation
    String music = interpreter(ABCMusitext);
    if (print) {
        System.out.println("Music = " + music);
    }
    //An ABC musiText is assembled
    String musiText = header + music;
    System.out.println("\nmusiText  \n" + musiText);
    //musiText is made into an ABC tune
    Tune tune;
    tune = ABCmusiTextToTune(musiText);
    //The score can eventually be drawn
    if (drawTune) {
        DrawTune(tune);
    }
    playTune(tune);
    // first string entered by user
    String string1;

    //Score given by the User: selection
    double score;

```

```

// We open a dialog box;
// We obtain the score from User;
// It is read as String.
string1 = JOptionPane.showInputDialog(
    "Enter score of tune, "
    + "0 (worst) ,1,2,3,4,5 (best)", 0);
// We convert String to number of type double
score = Double.parseDouble(string1);
System.out.println("Score = " + score);
return score;
}

//Individuals are sorted by fitness.
//The fittest is the first.
private static void sorting() {
    if (printThis) {
        printPopulation();
    }
    int Champ;
//FitnessCopy[] is a copy of Fitness[]
//used as workbench.
double FitnessCopy[] = new double[N_MAX];
for (int i = 0; i < NUMB_IND; i++) {
    Fitness[i] = 1000000;
    FitnessCopy[i] = 1000000;
}
if (print) {
    System.out.println("\nSCORING \n");
}
for (int i = 0; i < NUMB_IND; i++) {
    if (print) {
        System.out.println("\nind = " + i + " " + Ind[i]);
    }
    Fitness[i] = assessInd(Ind[i]);
    FitnessCopy[i] = Fitness[i];
}
if (print) {
    System.out.println("\nIndividuals and scores ");
}

```

```

        for (int i = 0; i < NUMB_IND; i++) {
            System.out.println("Individual " + i + " = " + I
                + " Score = " + FitnessCopy[i]);
        }
    }

    //We sort individuals by fitness
    // Fitness 0 means the worst, 5 is perfect
    if (print) {
        System.out.println("
            + "\nSORTING: from the best to the worst: ")
    }
    for (int i = 0; i < NUMB_IND; i++) {
        Champ = 0;
        for (int j = 0; j < NUMB_IND; j++) {
            if (FitnessCopy[j] > FitnessCopy[Champ]) {
                Champ = j;
            }
        }
        //The array Order classifies individuals by fitness
        //The fittest is number zero
        Ordered[i] = Champ;
        FitnessCopy[Champ] = 0;
        if (print) {
            System.out.println("ind number " + Ordered[i] +
                + Ind[Ordered[i]] + " score " + Fitness[
        }
    }
    //IndClone is a copy of Ind1
    System.arraycopy(Ind, 0, IndClone, 0, NUMB_IND);

}

//Negative selection:
//Fitness determines probability of death.
//The probability of dead is higher,
//the lower is the fitness.
//Death = replacement of individual by ""

```

```

    private static void death() {
//Min and max are found
        double min = 100;
        double max = -100;
        for (int j = 0; j < NUMB_IND; j++) {
            if (Fitness[j] > max) {
                max = Fitness[j];
            }
            if (Fitness[j] < min) {
                min = Fitness[j];
            }
        }

        if (print) {
            System.out.println(
                "\nFitness determines death probability");
        }
//Fitness determines probability of death
        for (int j = 0; j < NUMB_IND; j++) {
            //Death probability y and fitness x are related by:
            //y = ax +b
            //y(min) = 1, (the worst must die)
            //y(max) = 0, (the best must live)
            //a = 1/(min - max)
            //b = -a max = -max/(min -max)
            double probDeath;
            if ((max - min) == 0) {
                probDeath = 0.1;
            } else {
                probDeath = -Fitness[j] / (max - min) + max / (max - min);
            }
            if (print) {
                System.out.print("ind " + j);
                System.out.print(" Fitness = " + Fitness[j]);

                System.out.println(" ProbDeath = " + probDeath);
            }
            double p = R.nextDouble();

```

```

//Death = replacement of individual by ""
    if (p <= probDeath) {
        Ind[j] = "";
    }
}
if (print) {
    System.out.println("
        + "The best ind is number " + Ordered[0]);
}
}

//A not nil random individual is chosen.
private static int randomIndiv() {
    boolean cloned = false;
    int k = 0;
    while (cloned == false) {
        k = R.nextInt(NUMB_IND);
        if (!"".equals(Ind[k])) {
            cloned = true;
        }
    }
    if (print) {
        System.out.println(
            " cloned from old ind " + k);
    }
    return k;
}

//The pre-population of the new generation
//is built with clones
//of living individuals sampled at random
//with replacement.
private static void Reproduction() {
    if (print) {
        System.out.println("\nREPRODUCTION\n");
    }
    for (int j = 0; j < NUMB_IND; j++) {
        if (print) {

```

```

        System.out.print("New ind " + j);
    }
    IndClone[j] = Ind[randomIndiv()];

}
//Ind is a copy of IndClone
System.arraycopy(IndClone, 0, Ind, 0, NUMB_IND);
}

//Recombination is enabled.
//individual l is recombined with
//another taken at random.
private static String recInd(int l) {
    //individual l
    String indl = IndClone[l];
    String head;
    String tail;
    String recombinant;
    int m = indl.length();
    //Mutation site
    int m1;

    //Random site for recombination
    m1 = R.nextInt(m);
    //The head is taken from ind l
    head = indl.substring(0, m1);

    //Random number
    int k = R.nextInt(NUMB_IND);
    //Partner at random
    String partner = IndClone[k];
    //The tail is taken from the partner
    tail = partner.substring(m1);
    //Recombination
    recombinant = head + tail;
    if (print) {
        System.out.println("Indl      = " + indl);
        System.out.println("Head      = " + head);
    }
}

```

```

        System.out.println("Partner = " + partner);
        System.out.println("Tail    = " + tail);
        System.out.println("Recomb  = " + recombinant + "\n"
    }
    return recombinant;
}

//We found no remedy against recombination:
//it is always mortal for the interval notation.
private static void recombination() {
    if (print) {
        System.out.println("\nRECOMBINATION\n");
    }
    //IndClone is a copy of Ind
    System.arraycopy(Ind, 0, IndClone, 0, NUMB_IND);

    //Ind1 is a recombinant version of Ind2
    for (int l = 0; l < NUMB_IND; l++) {
        Ind[l] = recInd(l);
    }
}

//Returns the interval of a codon
private static int intervalCodon(String codon) {
    int output;
//System.out.println( "Input (intervalCodon) = " + codon);
//The first letter is a sign:
//the direction of the interval.
char c = codon.charAt(0);
//The second letter denotes an interval
Character f = codon.charAt(1);

    if (print) {
        System.out.println("char 0,1 = " + c
            + " " + f + " ");
    }
//Direction is decoded
int direction;

```

```

    if (c == '+') {
        direction = 1;
    } else {
        direction = -1;
    }
    //Interval is decoded
    int l = Character.getNumericValue(f);
    int k;
    if ((l >= 0) & (l < 10)) {
        k = l;
    } else {
        k = charToNumber(f);
    }
    if (print) {
        System.out.println(" k = " + k);
    }
    output = direction * k;
    return output;
}

//The start has the initial note and a duration
private static String mutateStart(String a) {
    if (printThis) {
        System.out.println("input  = " + a);
    }
    String h = transcription(a);
    if (printThis) {
        System.out.println("input T = " + h);
    }
    int length = SUPER_ALPHABET.length();
    String output;
    //Mutate note else duration?
    double f = R.nextDouble();
    if (f < 0.5) //note
    {
        char s = a.charAt(0);

        int indexNote = SUPER_ALPHABET.indexOf(s);

```

```

//An shift is generated
Integer shift = R.nextInt(6);
String newNote;
int newIndex;
//To avoid some overflow problems.
if ((indexNote + shift > length)
    || (indexNote + shift > INTERVALS.length())
    || (indexNote + shift < 0)) {
    shift = -shift;
}
newIndex = indexNote + shift;
if (printThis) {
    System.out.println(
        " indexNote = " + indexNote
        + " shift = " + shift
        + " newIndex = " + newIndex);
}
newNote = "" + SUPER_ALPHABET.charAt(newIndex);

if (printThis) {
    System.out.println("Old note = " + s
        + " New note " + newNote);
}
String codon = a.substring(2, 5);
System.out.println("First Codon = " + codon);
int interval = intervalCodon(codon);
Integer newInterval = interval - shift;
//Compensation: indexNote + interval = newIndex + ne
if (printThis) {
    System.out.println(
        " indexNote = " + indexNote
        + " shift = " + shift
        + " interval = " + interval
        + " newInterval = " + newInterval);
}

String newI = intToOneLetterNot(newInterval);

```

```

        output = "" + newNote + a.charAt(1)
                + newI + a.substring(4);
        if (printThis) {
            System.out.println("output (note) "
                + " = " + output);
        }
    } else //duration
    {
        //Duration is changed
        output = "" + a.charAt(0)
                + randomLetter(durationsAlphabet)
                + a.substring(2);
        if (printThis) {
            System.out.println("output (duration) "
                + " = " + output);
        }
    }
}

h = transcription(output);
if (print) {
    System.out.println("output T = " + h);
}
return output;
}

//The interval as Integer is transcribed
//to one letter notation
private static String intToOneLetterNot(Integer intInput) {
    String sign;
    if (intInput > 0) {
        sign = "+";
    } else {
        sign = "-";
    }
    intInput = Math.abs(intInput);
    String newI2 = intInput.toString();
    int j = intInput;
    char newI;

```

```

        if (intInput > 9) {
            newI = (char) (j + 87);
            if (printThis) {
                System.out.println(
                    j + " = " + newI);
            }
            newI2 = "" + newI;
        }
        /*This replaces the following code:
if (intInput == 10) newI2 = "a";
if (intInput == 11) newI2 = "b";
if (intInput == 12) newI2 = "c";
if (intInput == 13) newI2 = "d";
if (intInput == 14) newI2 = "e";
if (intInput == 15) newI2 = "f";
if (intInput == 16) newI2 = "g";
if (intInput == 17) newI2 = "h";
if (intInput == 18) newI2 = "i";
if (intInput == 19) newI2 = "j";
if (intInput == 20) newI2 = "k";
if (intInput == 21) newI2 = "l";
if (intInput == 22) newI2 = "m";
if (intInput == 23) newI2 = "n";
if (intInput == 24) newI2 = "o" +
    "" +
    "";
        */
        String output = sign + newI2;
        return output;
    }

//The string a is mutated at two contiguous sites.
//Why? We use musiTets in interval notation,
//instruction to rise or lower pitch so,
//a mutation may cause notes to get outside
//established sets.
//The remedy is to make balancing changes at two
//contiguous sites in such a way that no harm

```

```

//is caused beyond mutation site.
//Example 1: INTERVALS are mutated
//input      B6+01-y1+61-41+91-41+01
//output:    B6+01-y1+61-21+71-41+01
//Notice:    -4 + 9 = 5 = -2 + 7.
//Example 2: durations are mutated
//input      B6+01-y1+61-41+91-41+01
//output:    B6+01-y1+61-43+91-41+01
//Notice:    we pose no restriction on durations
//Example 3: initial note mutated
//input      B6+01-y1+61-41+91-41+01
//output:    A6+21-y1+61-21+71-41+01
//Notice:    from B to A there are 2 semitones which are
//reestablished in the following note.
//The string is divided in three regions:
//head + mutation sites + tail.
    private static String twoPointsMutationInd(String a) {
        if (printThis) {
            System.out.println(
                "\nInput string = " + a);
        }
        boolean mutateInterval = false;
        boolean mutateDuration = false;
        int m = a.length();
        String output = "";
        if (m == 0) {
            return output;
        } else {
            int l = R.nextInt(a.length());
            if (printThis) {
                System.out.println(
                    "Place mutation = " + l);
            }

            //Is l at the beginning?
            if (l < 2) {
                output = mutateStart(a);
            } else //pick codon and mutate it.

```

```

{
    int numberCodon1 = (l - 2) / 3;
    int numberCodons = (m - 2) / 3;
    int beginCodon1 = numberCodon1 * 3 + 2;
    int endCodon1 = beginCodon1 + 3;
    String codon1 = a.substring(beginCodon1, endCodon1);
    if (printThis) {
        System.out.println("codon1 = " + codon1
            + " codon number = " + numberCodon1
            + " Number of codons = " + numberCodons
            + " a.length = " + m
            + " endCodon = " + endCodon1);
    }
    //Mutate interval else duration?
    double f = R.nextDouble();
    if (printThis) {
        System.out.println("f = " + f);
    }
    if (f < 0.5) {
        mutateDuration = true;
    } else {
        mutateInterval = true;
    }

    if (mutateDuration) {
        //Duration is changed
        output = "" + a.substring(0, beginCodon1 + 2)
            + randomLetter(durationsAlphabet)
            + a.substring(beginCodon1 + 3);
        if (printThis) {
            System.out.println(
                "Output (duration) = " + output);
        }
        return output;
    }

    if (mutateInterval) {
        int interval1 = intervalCodon(codon1);
    }
}

```

```

Integer newIntervall1 = 0;
if (intervall1 > 1) {
    newIntervall1 = R.nextInt(intervall1) % 6;
}

String newInt1 = intToOneLetterNot(newIntervall1);

output = "" + a.substring(0, beginCodon1)
        + newInt1 + a.substring(beginCodon1 + 2);
if (printThis) {
    System.out.println(
        "\n newIntervall1 = " + newIntervall1 + "
        + "Output (u) = " + output + "\n");
}
//Is codon1 the final one?
//Yes = end work
if (numberCodon1 == numberCodons - 1) {
    return output;
} else //No: make a coordinate mutation in next site
{
    String codon2 = a.substring(endCodon1, endCodon2);
    if (printThis) {
        System.out.println("Codon 2= " + codon2);
    }
    int interval2 = intervalCodon(codon2);

    //Compensation:
    Integer newInterval2 = intervall1 + interval2 -
newIntervall1;

    if (printThis) {
        System.out.println(
            " int 1 = " + intervall1
            + " int 2 = " + interval2
            + " newIntervall1 = " + newIntervall1
            + " newInterval2 = " + newInterval2
        )
    }
    String newI2 = intToOneLetterNot(newInterval2);

```

```

        output = "" + output.substring(0, endCod
            + newI2 + output.substring(endCo
    if (printThis) {
        System.out.println(
            "Output (interval) = " + out
    }
    }//end else
    }
}
String h1 = transcription(a);
String h2 = transcription(output);
if (printThis) {
    System.out.println(
        "Input (twoPointsM) = " + a + "\n"
        + "Output (twoPointsM) = " + output + "\n"
        + "Input (twoPointsM) = " + h1 + "\n"
        + "Output (twoPointsM) = " + h2 + "\n");
}

    return output;
} //End no void input
} //end method

//All strings are mutated.
private static void mutation() {
    if (print) {
        System.out.println("\nMUTATION\n");
    }
    // All the individuals mutate.
    for (int j = 0; j < NUMB_IND; j++) {
        //v=original individual
        String v = Ind[j];
        // w = mutated individual.
        String w = twoPointsMutationInd(v);
        Ind[j] = w; //Replacement
    }
}
}

```

```

//Evolution = selection + reproduction
//+recombination + mutation
    private static void Dynamics() {
        System.out.println("\nGENERATION = " + generation + " \n");
        sorting();
        death();
        Reproduction();
        mutation();
        //recombination();
    }

//Evolution is made into a co-composer
    public static void evoMusic() {
        Initialization();
        int NGen = 999;
        for (int n = 1; n <= NGen; n++) {
            generation = n;
            Dynamics();
        }
    }

    public static void main(String[] args) {
        PLAYER.start();
        print = false;
        printThis = true;
        drawTune = false;

        //int g = charToNumber('a');
        //System.out.println(g);
        //Measure 44 for 4/4, 68 for 6/8
        int measure = 68;
        switch (measure) {
            case 44:
                header = HEADER44;
                durationsAlphabet = DURATIONS_ALPHABET44;
                break;
            case 68:
                header = HEADER68;

```

```
        durationsAlphabet = DURATIONS_ALPHABET68;
        break;
    }
    evoMusic();

    /*
printThis = true;
durationsAlphabet = DURATIONS_ALPHABET44;
twoPointsMutationInd("T6+33-51+91+83-21-51+01" );
    */
}
} //End of main class H105 LamarckMusic2
```

Index

- .jar file, 5
- ABCExplorer, 5
- ABCforBeginners, 6
- ABCJ, 5
- console, 10
- copy, 10
- Darwin, 177
- delete, 10
- evolJava community, 9
- field, 7
- first theory, 41
- first theory of music, 41
- first theory, 42, 44, 49, 56, 66, 91
 - statistical , 91, 92
- gradualism, 65
- interval, 178
- Java Sound API, 13
- Lamarck, 177
- musical analysis, 92
- musiText, 213
- musiTexts, v, 2
- Output, 10
- pasting, 10
- select all, 10
- statistical first theory of music, 92
- theory of music
 - 1, 145
 - first, 213
 - Interval, 211
 - statistical version of the first theory, 146, 213