

JAVA ***FOR*** ***ALIENS***

CLAUDIO DE SIO CESARI

JAVA FOR ALIENS

**LEARN JAVA FROM SCRATCH
AND BECOME A PRO**

Appendices

Java for Aliens - Appendices

Copyright © 2019 by **Claudio De Sio Cesari**

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, without the prior written permission of the author, except in the case of brief quotations permitted by copyright law. For permission requests, write to the author at the address: **claudio@claudiodesio.com**

Editor: Emanuele Giuliani (emanuele@giuliani.mi.it)

First Edition (November, 2019)

Font licenses

Libre Baskerville (<https://fonts.google.com/specimen/Libre+Baskerville>, Impallari Type): OFL

Libre Franklin (<https://fonts.google.com/specimen/Libre+Franklin>, Impallari Type): OFL

Cousine (<https://fonts.google.com/specimen/Cousine>, Steve Matteson): AL

Inconsolata (<https://fonts.google.com/specimen/Inconsolata>, Raph Levien): OFL

Roboto (<https://fonts.google.com/specimen/Roboto>, Christian Robertson): AL

Digits (<https://www.1001fonts.com/digits-font.html>, Dieter Steffmann): FFC

Journal Dingbats 3 (<https://www.1001fonts.com/journal-dingbats-3-font.html>, Dieter Steffmann): FFC

Musicals (<https://www.1001fonts.com/musicals-font.html>, Brain Eaters): FFC

Image licenses

Curiosity icon (https://www.flaticon.com/free-icon/toyger-cat_107975, www.freepik.com): FBL

Alien icon (<http://www.iconarchive.com/show/free-space-icons-by-goodstuff-no-nonsense/alien-4-icon.html>, goodstuffnononsense.com): CC

Trick icon (https://www.flaticon.com/free-icon/magic-wand_1275106, www.flaticon.com/authors/pause08): FBL

License specifications

Open Free License (OFL): https://scripts.sil.org/cms/scripts/page.php?site_id=nrsi&id=OFL_web

Apache License, Version 2.0 (AL): <http://www.apache.org/licenses/LICENSE-2.0>

1001Fonts Free For Commercial Use License (FFC): <https://www.1001fonts.com/licenses/ffc.html>

Flaticon Basic License (FBL): <https://file000.flaticon.com/downloads/license/license.pdf>

CC Attribution 4.0 (CC): <https://creativecommons.org/licenses/by/4.0/legalcode>

Any other trademarks, service marks, product names or named features are assumed to be the property of their respective owners, and are used only for reference. There is no implied endorsement if we use one of these terms.

Table of Contents

Appendix A - Brief history of Java	1
<hr/>	
A.1 Oak, Gosling and the Green Team	1
A.2 Java = Internet	2
A.3 The Reasons for Success	3
A.4 Java... Run Everywhere	4
A.5 Licenses and Oracle Support	4
 Appendix B - Setting the working environment on Microsoft Windows operating systems: installation of the Java Development Kit	 5
<hr/>	
B.1 OpenJDK Download	6
B.2 Oracle JDK Standard Edition Download	7
B.3 Download API Documentation	10
B.4 PATH Environment Variable Setting	12
B.4.1 Windows 10	12
B.4.2 Windows 8	13
B.4.3 Windows 7	13
B.5 Verify the Installation	14
 Appendix C - Basic Commands for Interacting with the Windows Command Line	 17
<hr/>	
C.1 Introduction	17
C.2 Most Used DOS Commands Table	19
C.3 More Information on the Command Prompt	21
 Appendix D - Introduction to the Design Pattern	 23
<hr/>	
D.1 Design Pattern Definition	24
D.2 GoF Book: Formalization and Classification	24
D.3 Examples of Patterns in the Book	26

Appendix E - The CLASSPATH Environment Variable	27
E.1 CLASSPATH	27
E.2 JAR File	29
E.3 CLASSPATH and JAR File	29
Appendix F - Introduction to the Unified Modeling Language	31
F.1 What is UML?	31
F.2 When and Where It Was Born	32
F.3 Why UML Was Born (Why)	32
F.4 Who Created UML (Who)?	33
Appendix G - UML Syntax Reference	35
G.1 UML 1.3 Syntax Reference	35
G.1.1 Use Case Diagram	38
G.1.2 Class Diagram	41
G.1.3 Component & deployment diagram	44
G.1.4 Interaction diagram	46
G.1.5 State Diagram	48
G.1.6 Activity diagram	49
G.1.7 General Purpose Elements	51
Appendix H - Introduction to XML	53
H.1 Markup Languages	53
H.2 The eXtensible Markup Language	54
H.2.1 XML Documents	54
H.2.2 Structure of an XML Document	55
H.2.2.1 Prologue Structure	55
H.2.2.2 Document Structure	56
H.2.3 Characteristics of an XML Document	57
H.2.3.1 Well Formed XML Document	57
H.2.3.2 Valid XML Document	58
Appendix I - Applet	61
I.1 Definition of Applet	62
I.2 Introduction to HTML	63
I.3 Installing Applet	65

Appendix J - Compiling Past Versions of Java Code	67
J.1 Warning	68
J.2 Keywords Used as Identifiers	68
J.3 Current Syntax vs Previous Syntax	68
J.4 The target Option	69
 Appendix K - Introduction to Apache Derby	 71
K.1 Apache Derby Installation	71
K.1.1 Software Download	72
K.1.2 Installation	73
K.1.3 Configuration	73
K.2 Execution and Use of Apache Derby	76
K.2.1 Server Mode	76
K.2.2 Embedded Mode	77
K.2.3 Interactive Console	77
 Appendix L - JavaFX Environment Configuration	 79
L.1 JAVA_HOME Variable Setting	80
L.2 Download JavaFX SDK and Runtime	80
L.3 PATH_TO FX Variable Setting	82
L.4 Configuration Check	83
 Appendix M - EJE (Everyone's Java Editor)	 85
M.1 System Requirements	86
M.2 EJE Installation and Execution	86
M.2.1 For Windows Users	86
M.2.2 For Unix-like Operating Systems (Linux, Solaris, etc.) Users	86
M.2.3 For Users Who Have Problems with These Scripts (Windows 9x/NT/ME/2000/XP & Linux, Solaris)	87
M.3 User Manual	87
M.4 Descriptive table of the main EJE commands	89

Appendix N - Easter Eggs, References and Quotes	97
N.1 Creativity	97
N.1.1 References, Easter Eggs and Quotes	99
N.1.1.1 Volume 1	99
N.1.1.2 Volume 2	100
N.1.2 Exercises References	102
N.1.3 Cover References	103
N.1.3.1 Volume 1	103
N.1.3.2 Volume 2	104
N.1.4 References in the analytical index	105
Appendix O - Bibliography	107
O.1 Resources for Java	108
O.1.1 Books	108
O.1.2 Online Resources	110
O.2 Resources for Java Technologies	111
O.2.1 Resources for Java EE	111
O.2.2 Resources for Android programming	111
O.3 Resources for Object Orientation	112

Appendix A

Brief history of Java

Goals:

At the end of this appendix, the reader should:

- ✓ Understand the history and the importance of Java (Units A.1, A.2, A.3, A.4).
- ✓ Understand the new licensing model (Unit A.5).

The following sections will briefly describe the history of the Java programming language. Over the years, on the Sun Microsystems and Oracle sites, it has been told a bit like a legend, so much so that, at times, it seemed like reading a historical, almost mythological, novel. The protagonists of this story, however, are not heroes that sail oceans and fight weird creatures, but people like James Gosling, principal creator of the Java language, Bill Joy, mythical vice-president of Sun Microsystems in the nineties (also author of the *VI* editor, and father of the *Solaris operating system*) and, in general, the *Green Team*, the group of engineers (including Joy and Gosling) from which all ideas were born. We will never know if the anecdotes are all true, the only thing certain is that today Java is the most used programming language in the world. It is very important to read section A.5 about new Java licensing model.

A.1 Oak, Gosling and the Green Team

Java was born thanks to university research conducted at Stanford University in the early 1990s. A small team of engineers (later nicknamed the *Green Team*) was commissioned by Sun Microsystems to search for new solutions in the field of embedded systems, or processing microsystems that use microprocessors created to meet certain specific purposes. Examples of embedded systems well known today are smartphones, decoders, smart cards and various domestic appliances. The side effect of this research was the creation of a language designed to program these systems, simpler and less heavy than the prevailing language in those years: C++.

In 1992, the *Oak* language was born, whose main architect was James Gosling, today one of the most famous and esteemed “computer gurus” on the planet. The Green Team aimed to create an experimental touch-screen handheld device called “*7” (i.e. “StarSeven”) for home entertainment. For about eighteen months, they closed themselves away in an anonymous office on Sand Hill Road in the city of Menlo Park (Silicon Valley). Gosling created Oak specifically to program this demo, but he had in mind the goal of making the language independent of the platform where the programs had to run.

The name *Oak* was chosen because outside of the office where the Green Team worked, there was an oak tree and they dedicated the language to that tree.

A.2 Java = Internet

At first, Sun decided to take advantage of the experience of *7 to enter fields that seemed strategic in those years, such as domotics and cable TV. However, the times were not yet prepared for topics such as “video on demand” and “smart domestic appliances”. Only a few years later, in fact, people began to request videos via the Internet or TV and even today, home automation is not as widespread as it could be. So, the original idea was put aside in order to focus on a new use for Oak, which turned out to be an amazing language.

In 1993, with the explosion of the Internet in the United States, the idea of using executable code through HTML pages was born. The emergence of applications using CGI (Common Gateway Interface) technology revolutionized the World Wide Web in such a way as to generate an unbelievable increase in users, never seen in the history of mankind.

The market that seemed most appealing then, soon became the Internet itself. In 1994, a browser was created that was briefly called *Web Runner* (from the movie *Blade Runner*, a favorite of Gosling’s) and then, finally, *HotJava*. It was not a top-level browser, but its creation led to the word “Java” being coupled with the word “Internet”.

On May 23, 1995, Oak, after an important revision, was officially renamed *Java*.

***Java* is the name of an Indonesian type of coffee very famous in the United States, which seems to have been Gosling’s favorite.**

At the same time the Netscape Corporation announced it was going to equip its famous browser with the Java Virtual Machine (JVM). It was a software that allows programs written in Java to be run directly within the browser itself, and consequently on every platform on which Netscape

Navigator ran. This meant a new revolution in the Internet world: the pages became interactive on the client computer thanks to *Applet* technology (explained in Appendix I). Users could, for example, use games on web pages and take advantage of programs such as dynamic and interactive chats, which made the web much more attractive.

A.3 The Reasons for Success

Furthermore, in a very short time, Sun Microsystems made available the JDK (Java Development Kit) for free from the website <http://java.sun.com>. Within a few weeks, downloads of JDK 1.02a ran into the thousands and Java began to be the word on everyone's lips. Java's popularity was down to the possibility that it offered of allowing the developers to write small applications on the Internet, called applets, which were secure, interactive and independent from the platform. In the early days, it seemed that Java was the right language to create spectacular websites. But Java was much more than just a tool to make navigation more enjoyable. In addition, tools were quickly developed to achieve certain results with less effort (see Macromedia Flash). However, the popularity that Netscape provided (which in 1995 was a colossus that fought a "browser war" with Microsoft) along with other large companies such as IBM, Oracle, etc., paid off.

Over the years, Java has increasingly become the ideal solution for problems shared by companies operating in different sectors, such as banks, software houses and insurance companies.

With the new millennium, Java technology has conquered new market shares such as smart cards, mobile phones and later smartphones (embedded systems for which it was originally born). These achievements have consolidated the success of the language. In fact, Java has provided a big boost to the spread of these consumer goods in recent years. Furthermore, Java is a leader in the world of web-enterprise development.

Finally, thanks to its "write once, run everywhere" philosophy, Java technology can be run on completely different platforms. It is even potentially executable on devices that have not yet seen the light today!

In 2006, Java became open source and, in 2010, Sun Microsystems was absorbed by Oracle. Oracle, after a long period of adjustment, has begun to evolve the Java platform.

In 2011, Java Release 7 (the first from Oracle) was not as amazing as expected. Many promised features did not find a place in the release, despite the many delays caused by trying to publish everything that was expected. Version 8 (2014), on the other hand, really brought a big step forward, but this version suffered several delays too. From September 2017, with Java Release 9, a new release model has been adopted: every six months a new version will be released, containing all the functionalities ready for production and delaying the functionalities that are not ready to future versions. Hence, Java 10 was published in March 2018, Java 11 in September 2018, Java 12 in March 2019, Java 13 in September 2019, and it is not difficult to predict what the subsequent versions will be and when they will be released.

A.4 Java... Run Everywhere

Today, Java is therefore a powerful and very well-established programming language and its numbers are impressive. It counts the highest number of active developers in the world, over three billion devices use Java technology, 97% of corporate desktops, 100% of Blu-ray players and one billion downloads per year of Java runtime, etc.

Java technology has invaded our daily lives without us even realizing it. It is massively present, for example, in our smartphones, decoders, smart cards, home appliances, even robots that walk on Mars! Very often Java is the invisible engine of the gadgets that we use every day, and it is probably the only programming language whose name is also in the vocabulary of those who know nothing about programming. With Java, we can program applications for the Google Android mobile operating system, most server-side applications use Java technology, and it is one of the most requested language when searching for an IT job!

A.5 Licenses and Oracle Support



As for Oracle's licensing and support, things have changed, and they may change again. Up to Java 8, Oracle provided free security patches and important updates. Now that the release model has changed, Oracle will not provide these updates for each version for free. At the time of writing, Java 11 was declared as the first **Long Term Service (LTS)** version after Java 8. This means that Oracle will publish updates for Java 11 until September 2023 but for a fee. The next LTS version should be the version that will be released in September 2021, Version 17, but it is better to check Oracle site periodically to be sure.

The vast majority of developers, however, will not need to obtain a commercial license with Oracle in order to develop in Java. By downloading the JDK from <http://jdk.java.net>, we can obtain important updates and then update our version every 6 months, keeping up with the new features offered by the new platform.



It is therefore essential not to download the JDK from the Oracle site anymore (unless we need to pay Oracle for support), but rather download it from <http://jdk.java.net>. The Oracle JDK and the OpenJDK are functionally equivalent, but the first is distributed with the classic installer, while the second is distributed as a zipped file (see Appendix B).

Appendix B

Setting the working environment on Microsoft Windows operating systems: installation of the Java Development Kit

Goals:

At the end of this appendix the reader should:

- ✓ Be able to install the Java Development Kit correctly on Windows operating systems (Units B.1, B.2, B.3, B.4, B.5).



Below are the procedures to install the Open JDK and Oracle JDK. As explained at the end of Appendix A, the Oracle JDK needs to be installed only if we plan to create programs that need the support of Oracle (for a fee) when they are up and running. So, it's much more likely that you will have to install the OpenJDK, ignoring the "Oracle JDK Download" section (section B.2). When using this solution, it is always advisable to keep up to date with the latest version, by updating the JDK. This will not only allow us to get all the latest security patches for free, but will also give us the opportunity to preview new features and libraries without having to wait for the next LTS version. Obviously, if you or your company cannot update the JDK version for long periods, and you don't want to accept the risk of not having the latest security patch available, then you can also consider using Oracle JDK.

There are dozens of other free and open source solutions such as those offered by Azul, AdoptOpenJDK, Amazon Corretto, etc. (and some offer support to pay), but with OpenJDK we'll know that we are using something that is functionally identical to the Oracle JDK, for every update.

B.1 OpenJDK Download

Below are the steps to be taken to download and install the OpenJDK on Windows operating systems. This will allow you to program in Java for free.

You can alternatively watch a 2 minutes tutorial on my personal YouTube channel, that explains how to install the OpenJDK on Windows 10 (<https://www.javaforaliens.com/yt/jdk.html>). The video is a quicker solution, but this appendix give you more information. The choice is yours!

Go to <http://jdk.java.net>, and choose the JDK version to download. In general, you should choose the most recent which, in the image B.1 is Version 12, but of course you can choose other versions like 13.

jdk.java.net

Java Development Kit builds, from Oracle

Ready for use: JDK 12 

Early access: [JDK 14](#), [JDK 13](#), [jpackage](#), [OpenJFX](#), [Panama](#), [Valhalla](#), & [JMC](#)

Reference implementations: [Java SE 12](#), [11](#), [10](#), [9](#), [8](#), & [7](#)



© 2019 Oracle Corporation and/or its affiliates
Terms of Use · Privacy · Trademarks

Figure B.1 - Screenshot of <http://jdk.java.net>. The arrow highlights the right link to click on.

We will be redirected to the page shown in Figure B.2, where you have to click on the **zip** link relating to the Windows platform (see arrow).

jdk.java.net

- GA Releases
 - JDK 12
- Early-Access Releases
 - JDK 14
 - JDK 13
 - jpackage
 - OpenJFX
 - Panama
 - Valhalla
 - JMC
- Reference Implementations
 - Java SE 12
 - Java SE 11
 - Java SE 10
 - Java SE 9
 - Java SE 8
 - Java SE 7
- Feedback
 - Report a bug
- Archive

JDK 12.0.1 General-Availability Release

This page provides production-ready open-source builds of the [Java Development Kit, version 12.0.1](#), an implementation of the [Java SE 12.0.1 Platform](#) under the [GNU General Public License, version 2, with the Classpath Exception](#).

Commercial builds of JDK 12.0.1 from Oracle under a [non-open-source license](#), for a wider range of platforms, can be found at the [Oracle Technology Network](#).

Documentation

- Features
- Release notes
- API Javadoc
- Tool & command reference

Build 12 (2019/4/2):

Linux/x64	tar.gz (sha256)	197649562 bytes
macOS/x64	tar.gz (sha256)	189315228
Windows/x64	zip (sha256)	196414289

Notes

- The Alpine Linux build previously available on this page was removed as of JDK 12.0.1 GA. It's not production-ready because it hasn't been tested thoroughly enough to be considered a GA build. Please use the [early-access JDK 13](#) Alpine Linux build in its place.
- To obtain the source code for these builds, clone the JDK 12.0.1 [Mercurial repository](#) and update to the tag `jdk-12.0.1-ga`.
- If you have difficulty downloading any of these files please contact jdk-download-help_ww@oracle.com.

Figure B.2 - JDK 12 download screen. The arrow shows the right link to click on.

Once you have the zipped file, unzip it into the `C:/Program Files/Java` folder (if it does not exist, create it).

Actually, it is possible to decompress the content of the zip file anywhere on your hard disk but, by convention, we should use the `C:/Program Files/Java` directory, because it is the same one used by the Oracle JDK installer.

B.2 Oracle JDK Standard Edition Download

Download the latest version of the Java Development Kit from:
<http://www.oracle.com/technetwork/java/javase/downloads/index.html>.

The file name varies from release to release, but not that much. At the time of

writing, the current version is 12.0.1, and the file name to download for 64-bit systems is `jdk-12.0.1_windows-x64_bin.exe`.

The screenshot shows the Oracle Java SE Downloads page. The page has a navigation bar with 'Menu', 'ORACLE', 'Search', 'Account', 'Country/Region', and 'Contact'. Below the navigation bar is a breadcrumb trail: 'Oracle Technology Network / Java / Java SE / Downloads'. The main content area is titled 'Java SE Downloads' and features a 'DOWNLOAD' button. A red arrow points to the 'Oracle JDK DOWNLOAD' button on the right side of the page. The page also includes a sidebar with links to 'Java SE', 'Java EE', 'Java ME', 'Java SE Subscription', 'Java Embedded', 'Java Card', 'Java TV', 'Community', and 'Java Magazine'. The right sidebar contains links to 'Java SDKs and Tools' and 'Java Resources'.

Figure B.3 - Download screen. The arrow highlights the right link to click on.

As seen in Appendix A, the Java version is updated every six months. At the time of writing, the current version is 12, so every internet address in these pages contains the number 12 for the current java version. You can try to substitute 12 with the last available version of Java to obtain the latest link.

To download the file, you will be asked to accept a license (click on the **Accept License Agreement** option), then will you be allowed to download the correct file. In Figure B.4, the arrows highlight the fundamental steps of the process to be carried out.

Oracle Technology Network / Java / Java SE / Downloads

Java SE Development Kit 12 Downloads
Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

Important Oracle JDK License Update
The Oracle JDK License has changed for releases starting April 16, 2019.
The new [Oracle Technology Network License Agreement for Oracle Java SE](#) is substantially different from prior Oracle JDK licenses. The new license permits certain uses, such as personal use and development use, at no cost – but other uses authorized under prior Oracle JDK licenses may no longer be available. Please review the terms carefully before downloading and using this product. An FAQ is available [here](#).
Commercial license and support is available with a low cost [Java SE Subscription](#).
Oracle also provides the latest OpenJDK release under the open source [GPL License](#) at [jdk.java.net](#).

See also:

- [Java Developer Newsletter](#): From your Oracle account, select **Subscriptions**, expand **Technology**, and subscribe to **Java**.
- [Java Developer Day hands-on workshops \(free\) and other events](#)
- [Java Magazine](#)

JDK 12.0.1 [checksum](#)

Java SE Development Kit 12.0.1
You must accept the [Oracle Technology Network License Agreement for Oracle Java SE](#) to download this software.
☐ Accept License Agreement ☒ Decline License Agreement

Product / File Description	File Size	Download
Linux	154.7 MB	jdk-12.0.1_linux-x64_bin.deb
Linux	162.54 MB	jdk-12.0.1_linux-x64_bin.rpm
Linux	181.18 MB	jdk-12.0.1_linux-x64_bin.tar.gz
macOS	173.4 MB	jdk-12.0.1_osx-x64_bin.dmg
macOS	173.7 MB	jdk-12.0.1_osx-x64_bin.tar.gz
Windows	158.49 MB	jdk-12.0.1_windows-x64_bin.exe
Windows	179.45 MB	jdk-12.0.1_windows-x64_bin.zip

Figure B.4 – JDK download screen.

Once you have obtained the installation file, run it and follow the steps proposed by the installation procedure. This is a very simple wizard with which it will be enough to advance from page to page without choosing options other than those proposed by default. The JDK will then be installed.

At the end of the process, the success of the installation will be confirmed, but we have not finished yet.



Figure B.5 - JDK installation screen.

B.3 Download API Documentation



This step is theoretically optional since API Documentation is always available online at <https://docs.oracle.com/en/java/javase/12/docs/api/index.html>.

However, we recommend that you always have the documentation offline, downloaded to your computer. From Version 9 on, the search functionality is also readily available in the interface.

We have to go to <http://www.oracle.com/technetwork/java/javase/downloads/index.html>.

Also download the “Java API” documentation. Click the download button at the bottom of the page under the **Java SE 12 Documentation** entry, as shown in Figure B.6.


Additional Resources	
NetBeans A powerful integrated development environment for developing applications on the Java platform.	DOWNLOAD
Oracle Java Advanced Management Console Advanced Management Console (AMC) enables desktop administrators to track and manage Java usage across their organization -- understanding which Java versions are used with which applications and managing compatibility/security. AMC is a commercial product available for Java users who license Java SE Subscription. Learn More	DOWNLOAD
Java SE 12 Documentation <ul style="list-style-type: none"> Java SE 12 Documentation Docs Installation Instructions 	 DOWNLOAD
Java SE 11 Documentation <ul style="list-style-type: none"> Java SE 11 Documentation Docs Installation Instructions 	DOWNLOAD
Java SE 8 Documentation <ul style="list-style-type: none"> Java SE 8 Documentation Docs Installation Instructions 	DOWNLOAD
Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files for JDK/JRE	DOWNLOAD

Figure B.6 - Download screen, “Additional Resources” section.

You will be redirected to another page where you can download (after accepting the license) the Java API documentation, as you can see in Figure B.7.

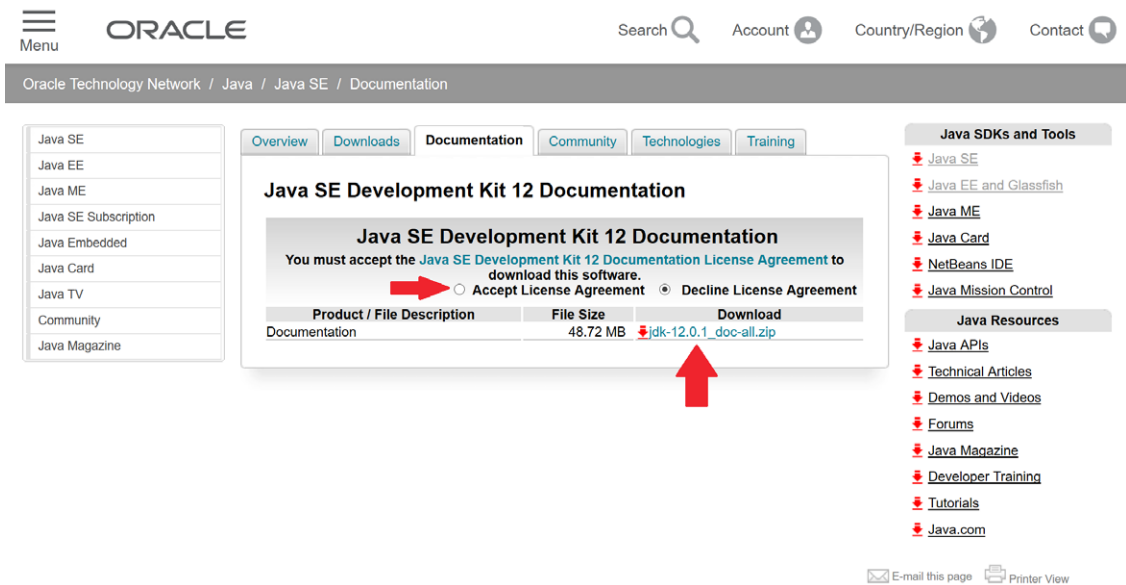


Figure B.7 - Screen to download the documentation.

This is a .zip file currently named `jdk-12.0.1_doc-all.zip`. In this case, the file name varies from one release to the other. Unzip the file in the same folder as the JDK (or any other folder that is easily accessible) in the **Docs** folder (or choose a suitable name). Create a shortcut on the desktop (or any other location that is considered easily accessible) to `index.html` file of the **Docs** folder.

B.4 PATH Environment Variable Setting

To use the compiler (`javac`), the interpreter (`java`) and all the software contained in the JDK **bin** folder, you need to set the **PATH** environment variable, pointing it to the JDK **bin** folder.

B.4.1 Windows 10

For Windows 10 systems, perform the following steps:

1. open the control panel, then click on **System**, then on **Advanced System Settings** in the new window;
2. in the new window that opens, select the **Advanced** tab and click on **Environment variables**;

3. among the **System Variables** (or if you prefer among the **User Variables**) select the **PATH** variable and click **Modify**;
4. on the new window that is shown click on **New**;
5. move to the **Variable value** box and move the cursor to the beginning of the row, then add the path to the JDK bin folder, which should look like this:

```
C:\Program Files\Java\jdk-12.0.1\bin
```

6. after confirming the insertion of the new item, select it and bring it to the beginning of the list by clicking on the **Move up** button;
7. click **OK** and the installation is finished.

B.4.2 Windows 8

For Windows 8 systems, perform the following steps:

1. open the control panel, then click on **System**, then on **Advanced System Settings** in the new window;
2. in the new window that opens, select the **Advanced** tab and click on **Environment variables**;
3. among the **System Variables** (or if you prefer among the **User Variables**) select the **PATH** variable and click **Modify**;
4. on the new window that is presented click on **New**;
5. move to the **Variable value** box and move the cursor to the beginning of the row, then add the path to the JDK bin folder, which should look like this:

```
C:\Program Files\Java\jdk-12.0.1\bin;
```

6. the “;” will allow us to separate our value from other values;
7. click **OK** and the installation is finished.

B.4.3 Windows 7

For Windows 7 systems, perform the following steps:

1. press the **start** button, then right click on **My Computer** and click on **Properties**;
2. select **Advanced System Settings** and click **Environment Variables**;
3. among the **System Variables** (or if you prefer among the **User variables**), select the **PATH** variable and click **Edit**;

4. move to the Variable value box and move the cursor to the beginning of the row, then add the path to the JDK bin folder, which should look like this:

```
C:\Program Files\Java\jdk-12.0.1\bin;
```

5. the “;” will allow us to separate our address from other references;
6. click **OK** and the installation is finished.

B.5 Verify the Installation

To verify that everything has been successful, open a command prompt and, in the text field, available in the Windows **Start** menu, type the command:

```
cmd
```

Once the command prompt is open, to test the version of Java installed, type:

```
java -version
```

If everything is OK, a message like the following should be printed:

```
openjdk version "12.0.1" 2019-04-16
OpenJDK Runtime Environment (build 12.0.1+12)
OpenJDK 64-Bit Server VM (build 12.0.1+12, mixed mode, sharing)
```

Also test that the compiler is correctly installed by typing the command:

```
javac
```

If everything is OK, a message like the following should be printed:

```
Usage: javac <options> <source files>
where possible options include:
  @<filename>                Read options and filenames from file
  -Akey[=value]              Options to pass to annotation processors
  --add-modules <module>(<module>)*
                             Root modules to resolve in addition to the initial
                             modules, or all modules on the module path if <module>
                             is ALL-MODULE-PATH.
  --boot-class-path <path>, -bootclasspath <path>
                             Override location of bootstrap class files
  --class-path <path>, -classpath <path>, -cp <path>
                             Specify where to find user class files and annotation
                             processors
  -d <directory>             Specify where to place generated class files
  -deprecation                Output source locations where deprecated APIs are used
```

```

--enable-preview           Enable preview language features. To be used in
                           conjunction with either -source or --release.
-encoding <encoding>       Specify character encoding used by source files
-endorseddirs <dirs>       Override location of endorsed standards path
-extdirs <dirs>            Override location of installed extensions
-g                         Generate all debugging info
-g:{lines,vars,source}     Generate only some debugging info
-g:none                   Generate no debugging info
-h <directory>            Specify where to place generated native header files
--help, -help, -?         Print this help message
--help-extra, -X          Print help on extra options
-implicit:{none,class}    Specify whether or not to generate class files for
                           implicitly referenced files
-J<flag>                  Pass <flag> directly to the runtime system
--limit-modules <module>(<module>)*
                           Limit the universe of observable modules
--module <module>(<module>)*, -m <module>(<module>)*
                           Compile only the specified module(s), check timestamps
--module-path <path>, -p <path>
                           Specify where to find application modules
--module-source-path <module-source-path>
                           Specify where to find input source files for multiple
                           modules
--module-version <version>
                           Specify version of modules that are being compiled
-nowarn                   Generate no warnings
-parameters               Generate metadata for reflection on method parameters
-proc:{none,only}         Control whether annotation processing and/or compilation
                           is done.
-processor <class1>[,<class2>,<class3>...]
                           Names of the annotation processors to run; bypasses
                           default discovery process
--processor-module-path <path>
                           Specify a module path where to find annotation processors
--processor-path <path>, -processorpath <path>
                           Specify where to find annotation processors
-profile <profile>         Check that API used is available in the specified profile
--release <release>        Compile for a specific release. Supported releases:
                           7, 8, 9, 10, 11, 12
-s <directory>            Specify where to place generated source files
--source <release>, -source <release>
                           Provide source compatibility with specified release.
                           Supported releases: 7, 8, 9, 10, 11, 12
--source-path <path>, -sourcepath <path>
                           Specify where to find input source files
--system <jdk>|none        Override location of system modules
--target <release>, -target <release>
                           Generate class files for specific VM version.
                           Supported versions: 7, 8, 9, 10, 11, 12
--upgrade-module-path <path>
                           Override location of upgradeable modules
-verbose                  Output messages about what the compiler is doing

```

```
--version, -version      Version information
-werror                  Terminate compilation if warnings occur
```

If you have previously installed another version of the JDK (and maybe you won't be aware), to be sure that it's all ok, also type:

```
javac -version
```

That should print the current version (12.0.1 in our case):

```
javac 12.0.1
```

Appendix C

Basic Commands for Interacting with the Windows Command Line

Goals:

At the end of this appendix, the reader should:

- ✓ Be able to use simple DOS commands to move between Windows folders (Units C.1, C.2 and C.3).

C.1 Introduction

Windows is an operating system that originated from another historical operating system called MS DOS (MS means Microsoft). Before becoming independent from DOS, Windows systems were nothing more than a DOS interface. It was possible to open the DOS system from Windows with a **Start** menu item called **DOS Prompt**. Since the release of Windows 7, this entry is gone, and to open a DOS session (which is now also called **command line** or **command prompt**) we can type the statement:

```
cmd
```

in the test field that is visible by clicking the **Start** button, for Windows 8, by simply typing the instruction in the search field at the top right of the desktop, while for Windows 10, in the text field at the bottom left of the desktop. A window similar to the one shown in figure C.1 should appear.

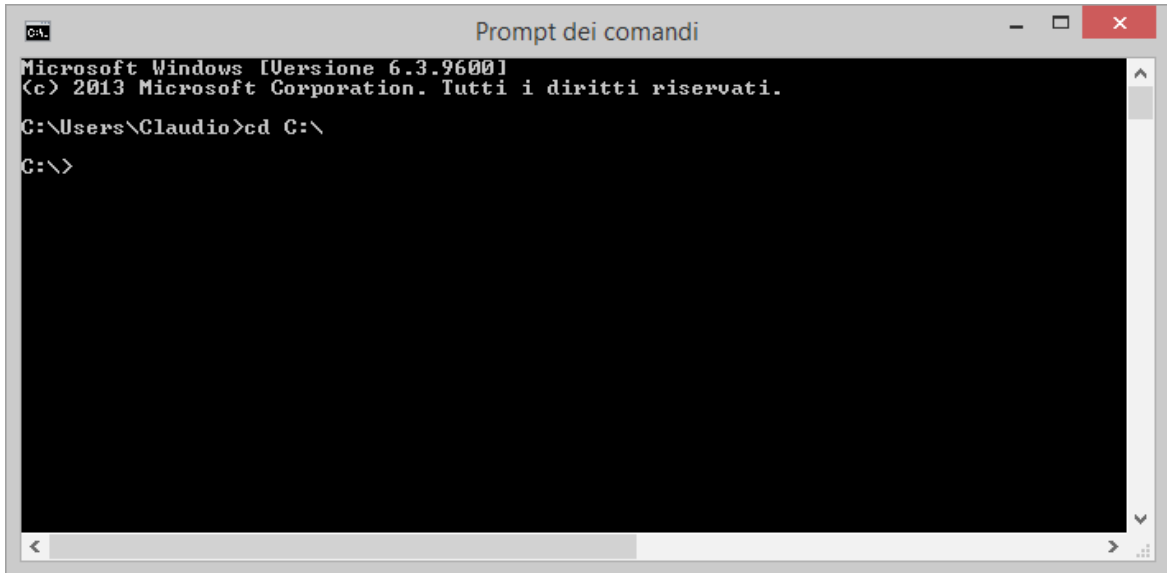


Figure C.1 – Command prompt (Windows 7).

The DOS operating system did not have a graphical window interface. To navigate between folders, or to perform any other type of operation on files, we had to type a command. Below is a list of basic commands that should allow us to approach the study of this book without problems. The commands executed on the prompt are always case insensitive (they do not distinguish between uppercase and lowercase letters).

Each command described must be followed by pressing the Enter key on the keyboard to take effect.

C.2 Most Used DOS Commands Table

Command	Explanation
<pre>cd folderName cd foldersPath</pre>	<p>Used to move to a folder contained in the working folder. If the folder name contains spaces, we must include the folder name in quotation marks. Here are some examples:</p> <pre>cd JavaHandbook cd "Java Handbook"</pre> <p>We can also move to sub-folders:</p> <pre>cd folder/subfolder/subsubfolder</pre> <p>Or move to paths that are located outside the folder where it is located:</p> <pre>cd "../parallelfolder/folder whose name contains spaces"</pre>
<pre>cd ..</pre>	Moves to the folder that contains the working folder.
<pre>dir</pre>	Lists the contents of the current folder with vertical alignment.
<pre>dir /w</pre>	Lists the contents of the current folder with horizontal alignment.
<pre>dir /p</pre>	Lists the contents of the current folder with a vertical alignment. If the files to be listed exceed the visual availability of the window, only the files that fall within the window are initially displayed. When any other key is pressed, the system will display the next screen.
<pre>cd driveName:/...</pre>	<p>Moving to a folder of another drive will not immediately produce the desired result. Suppose, for example, we are in the C:/Test folder, by typing the command:</p> <pre>cd D:/Java</pre> <p>It will not move to the destination folder if we do not type the following command too:</p> <pre>D:</pre> <p>So, to move to another drive, we need an additional command.</p>

control c (that is, the simultaneous pressing of the keys ctrl and c)	It interrupts the ongoing process (useful in the case of an infinite life cycle process).
TAB	<p>The tab key allows us to write the name of a file or folder contained in the folder where it is positioned. By repeatedly pressing the tab key, all the names of the files and folders in the current folder will be listed in alphabetical order. For example, if we are in a folder where there are HelloWorld.java and HelloWorld.class files and the Test folder. Pressing the tab key repeatedly will write HelloWorld.class first, then HelloWorld.java and then Test. Then the cycle starts again.</p> <p>We can also start writing the initial part of the file or folder name we want to write, and then press tab. In this case, the system will complete the word only with compatible alternatives. For example, if we write:</p> <pre>cd t</pre> <p>and then we press tab, the system will complete our command like this:</p> <pre>cd Test</pre> <p>Or if we write:</p> <pre>javac Hell</pre> <p>and then we press tab, the system will complete our command like this:</p> <pre>javac HelloWorld.class</pre> <p>Pressing tab again will give us:</p> <pre>javac HelloWorld.java</pre> <p>The system is smart enough to understand that if we press tab after the command:</p> <pre>cd</pre> <p>only the names of folders (and not files) will be proposed.</p>

Up and Down Arrow keys	Allows us to navigate through the last commands typed.
<code>exit</code>	The command line session is closed.

C.3 More Information on the Command Prompt

Pressing the right button of the mouse on the title bar will allow us to **customize** the command prompt by clicking on the **Properties** menu. From the window that appears, we can customize colors, layouts, fonts and so on.

To paste an externally-copied string at the DOS prompt, we must always press the right button on the title bar, then click on the **Paste** submenu of the **Edit** menu. In the case of Windows 10, we can also paste text using the key combination (shortcut) **ctrl** and **v**.

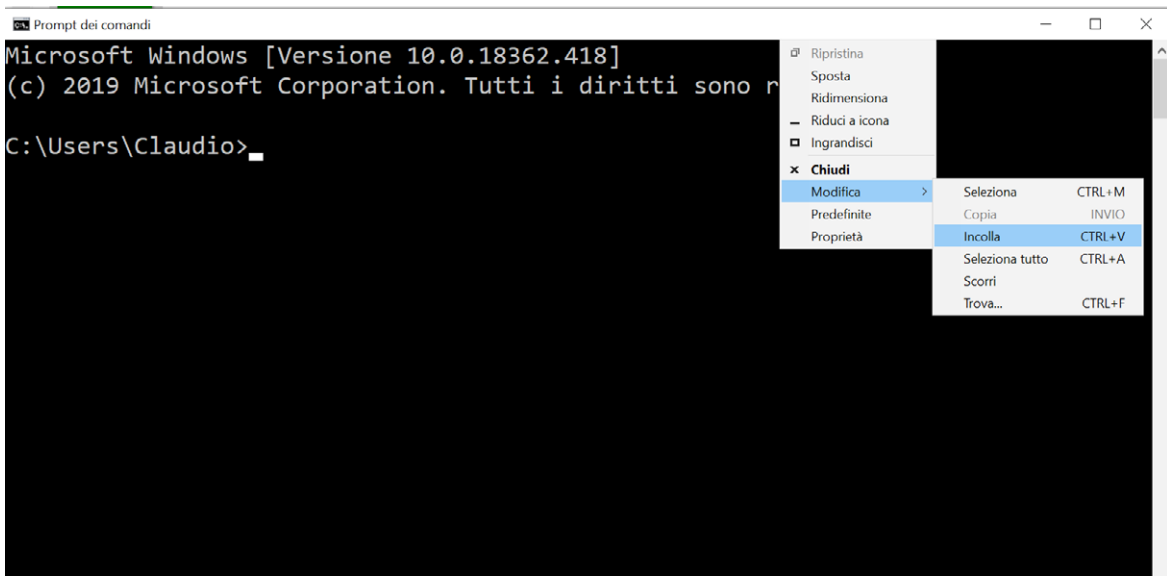


Figure C.2 - Command Prompt Menu (Windows 10).

If our goal is to **copy a path to a folder** (which could also be very long), we can also type the command:

```
cd
```

followed by a space. Then drag from the Windows window interface the folder we want to

move to, by dragging it into the command prompt.

To **copy** a string from the command prompt, we need to:

1. press the right button on the title bar;
2. click on the **Select** submenu of the **Edit** menu (on Windows 10, we can use the key combination **ctrl** and **m**);
3. select the string to copy by dragging the mouse with the left key pressed on the characters to be copied;
4. click on the **Copy** submenu of the **Edit** menu or press the **Enter** key.

At that point, the selected text will be copied, and it is ready to be pasted.

We can also **search** by clicking on the **Find...** sub-menu of the **Edit** menu (on Windows 10, use the key combination **ctrl** and **f**).

Appendix D

Introduction to the Design Pattern

Goals:

At the end of this appendix, the reader should:

- ✓ Know how to define what a Design Pattern is (Units D.1, D.2, D.3).

Applying Object Orientation to complex software development processes involves many difficulties. One of the most critical moments in the development cycle is that in which one passes from the analysis phase to the design phase. In such situations, it is necessary to make particularly delicate design choices, since they could jeopardize the functioning and release date of the software. In this context (but not only), we introduce the concept of *Design Pattern*, imported into software engineering directly from architecture. In fact, the first definition of a pattern was given by Cristopher Alexander, an important Austrian architect (teacher at the University of Berkeley - California), who began to formalize this concept in the 60s. In his book “Pattern Language: Towns, Buildings, Construction” (Oxford University Press, 1977) Alexander defines a pattern as **an architectural solution that can solve problems in heterogeneous contexts**. The formalization of the concept of Design Pattern is widely attributed to the so-called *Gang of Four* (briefly **GoF**). The “gang of four” consists of Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides who, despite coming from three different continents, cataloged - after four years of comparing - the first set of 23 patterns that constituted the fundamental core of the technique. In 1994, their book was considered the reference guide for the pattern community: “Design Patterns: elements of reusable object-oriented software” (Addison-Wesley). Other authors later published texts that extend the number of known patterns.

D.1 Design Pattern Definition

Design Patterns are generic design solutions applicable to recurring problems within heterogeneous contexts. Aware that the above statement may not be clear to the reader who is not familiar with certain situations, we will try to describe the concept by presenting, as well as the theory, also some examples of patterns within the book. In this way, we can better appreciate both the concepts and applicability.

The basic idea, however, is rather simple. It is well known that the goodness of the design is directly proportional to the designer's experience. An expert designer solves the problems that arise by using solutions that have already led to good results in the past. The GoF did nothing but compare its (broad) experience in finding design solutions, thus discovering some obvious intersections. Since these intersections are also solutions that often solve problems in heterogeneous contexts, they can be declared and formalized as a Design Pattern. In this way, solutions to common problems are available, even to designers who do not have extensive experience like that of the GoF. So, we are talking about a real gold mine!

D.2 GoF Book: Formalization and Classification

The GoF has cataloged the patterns using a very precise formalism. Each pattern is presented through a name, the problem to which it can be applied, the solution (not in a particular case) and the consequences. In particular, each pattern is described by the following list of elements that they consider most characteristic:

1. *Name and classification*: important for the vocabulary used in the project.
2. *Purpose*: brief description of what the pattern does and its logical foundation.
3. *Alternative names* (if any): many patterns are known by several names, perhaps because they are “discovered” by different people who gave them different names.
4. *Motivation*: description of a scenario illustrating a design problem and the solution offered.
5. *Applicability*: when the pattern can be applied.
6. *Structure (or model)*: graphical representation of the classes of the pattern using a notation language.

In this book, we will use UML, but in the book of GoF, which was created prior to the definition of UML, the OMT notation language (acronym of “Object Modeling Technique”) created by James Rumbaugh is used, plus Grady Booch’s interaction diagrams.

- 7. *Participants*: classes/objects with their own responsibilities.
- 8. *Collaborations*: how the participants collaborate in order to take responsibility.
- 9. *Consequences*: pros and cons of the application of the pattern.
- 10. *Implementation*: how to implement the pattern.
- 11. *Sample code*: code fragments that help in understanding the pattern.

This book will use Java, but C++ and SmallTalk are used in the book, whose birth pre-dates Java.

- 12. *Related patterns*: relationships with other patterns.
- 13. *Known uses*: examples of real uses of the pattern in existing systems.

The 23 patterns presented in the GoF book are classified into three main categories, based on the purpose of the pattern:

- 1. **Creational patterns**: they propose solutions for creating objects.
- 2. **Structural patterns**: they propose solutions for the structural composition of classes and objects.
- 3. **Behavioral patterns**: they propose solutions to manage the way in which the responsibilities of classes and objects are divided.

Furthermore, a distinction is also made based on the range of action of the pattern:

- 1. **Class patterns**: offer solutions through classes and subclasses in static relationships between them.
- 2. **Object patterns**: offer dynamic solutions based on objects.

We can summarize the 23 patterns of the GOF book through the following table.

Purpose				
		Creational	Structural	Behavioral
Scope	Class	Factory Method	Adapter (class)	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Facade Flyweight Proxy	Chain of responsibility Command Iterator Mediator Memento Observer State Strategy

D.3 Examples of Patterns in the Book

Examples of GOF patterns are described in this book. In particular, after introducing the `static` keyword in section 6.8.5, the **Singleton** pattern, one of the most famous, and used, is described in 6.8.6. In section 9.6.3, a solution based on the **Factory Method** is presented, in a module that talks about exceptions. It is a very famous pattern but often used in a simplified way. The pattern is taken up in section 19.3.1, when we talk about the services created with the `ServiceLoader`.

The **Thread Pool** pattern is described according to its implementation in section 15.6.3.3 when it comes to “Executor Interfaces”.

The same is true for the **Iterator** pattern, which finds an excellent representation in the `Iterator` interface and its implementations. This is all described above in module 18.2.3.2.

The spectacular **Decorator** structural pattern, on the other hand, is described in sufficient detail in section 20.2, to describe the structure of the Input-Output base library.

The **MVC** architectural pattern (acronym for **Model View Controller**) is mentioned in Chapters 23 and 24 when talking about GUIs.

Finally, the previously-mentioned Chapter 23, dedicated to Swing/AWT and GUIs in general, in addition to the **MVC**, also explains the behavior of the **Observer** pattern, which is the basis for the Java event management mechanism, and also shows how the libraries AWT and Swing were constructed using the **Composite** structural pattern.

We have only referred to the pattern known as **DTO** (acronym for **Data Transfer Object**) in section 5.4.1, which features an introduction to software architecture.

Appendix E

The CLASSPATH Environment Variable

Goals:

At the end of this appendix, the reader should:

- ✓ Be able to correctly set the CLASSPATH environment variable (Units E.1, E.3).
- ✓ Understand and know how to create and use JAR files (Units E.2, E.3).

Sometimes it happens that we cannot run a Java program. In fact, it may be necessary to use external libraries, or it could happen that even if we are in the same directory as our executable files, these are not “seen” by the virtual machine. In both cases, it is very probable that the problem results from an incorrect setting of the **CLASSPATH** environment variable.

E.1 CLASSPATH



The **CLASSPATH** environment variable is used at runtime by the virtual machine to find the classes that the program wants to use. To understand it better, let's give an example. Suppose we place our application in the **C:\OurApplication** directory. Now, suppose our application wants to use some classes located within another directory, let's say **C:\OtherClassesToUse**. The virtual machine will not scan all the directories on the hard disk to find the classes it needs, but will only search the directories indicated by the **CLASSPATH** environment variable. In this case, therefore, it is necessary to set the **CLASSPATH** both on the **C:\OtherClassesToUse** directory, and on the **C:\OurApplication** directory, in order to run our application correctly. To reach the goal, there are two solutions:

1. set the CLASSPATH environment variable directly from the operating system;
2. set CLASSPATH only for our application.

The first solution is highly discouraged. It involves permanently setting an operating system environment variable (the procedure is identical to the one described in Appendix B for the **PATH** variable). This would cause the permanent setting of the variable, and will prevent the execution of an application from another directory other than those referenced by the CLASSPATH (unless we set the CLASSPATH variable again).

The second solution is more flexible and is implemented by specifying the value of the CLASSPATH variable with an option of the **java** command. In fact, the **-classpath** (or **-cp**) flag of the **java** command allows us to specify the CLASSPATH, but only for the application we are running. For example, if we are in the **C:\OurApplication** directory and we want to run our application, we need to execute the following command:

```
java -cp .;C:\OtherClassesToUse mypackage.MyMainClass
```

where, with the **-cp** option, we have specified that the classpath must point to the current directory (specified with **“.”**) and to the directory **C:\OtherClassesToUse**.

It is also possible to use relative paths. The following command is equivalent to the previous one:

```
java -cp .;..\OtherClassesToUse mypackage.MyMainClass
```

Note that we have used the “;” as a separator, since we are assuming that we are on a Windows system. On a Unix-Like system, the separator would have been “:”.

The same option can also be used for the **javac** command, if we need to compile files that depend on others already compiled (the CLASSPATH must point to class files, not source files). For example, if to compile the file **MyMainClass.java**, we need classes already compiled that are inside the folder **C:\OtherClassesToUse**, then we will have to use the command:

```
javac -cp.; C:\OtherClassesToUse mypackage.MyMainClass
```

However, consult section 6.6.1 for details on compiling the files belonging to packages using the command line.

E.2 JAR File

It is not uncommon to refer to our application for external class libraries. Class libraries can be distributed within directories, but the most common format in which class libraries are created is the **JAR** format. The term JAR stands for “Java Archive”. This is an absolutely equivalent format to the classic ZIP format. The only difference between a ZIP and a JAR format is that a JAR file must contain a directory called **META-INF**, containing a text file called **MANIFEST.MF**. This file can be used to add properties to the JAR archive in which it is contained. To create a JAR file, we can then use a utility such as WinZIP or WinRar, adding the **MANIFEST.MF** file in a **META-INF** directory. But JDK offers a more convenient alternative: the **jar** tool. With the following command:

```
jar cvf library.jar MyDirectory
```

we will create a file called **library.jar** with the **MyDirectory** directory and all its contents inside.

On the Windows operating system, it is possible to create executable jar files. This means that, once the jar file has been created with our application in it, it will be possible to start it with a typical double click of the mouse, as if it were an .exe file. This will be discussed in Chapter 23.

E.3 CLASSPATH and JAR File

If we want to use, from our application, classes contained within a JAR file that are not directly available in the same directory as where the application is located, it is always possible to use the **CLASSPATH**. In this case, to run our application, we will use a command similar to the following:

```
java -cp .;C:\DirectoryWithJARFile\MyFile.jar mypackage.MyMainClass
```

The JVM will recover the **.class** files within the JAR file automatically. If we want to use multiple JAR files, we will have to execute a command similar to the following:

```
java -cp .;C:\DirectoryWithJARFile\MyFile.jar;C:\DirectoryWithJARFile\MyOtherFile.jar mypackage.MyMainClass
```

If the JAR files that interest us are in the same directory, we can also use the so-called “**CLASSPATH** wildcards”; that is, the “*” symbol can be used to refer to all the JAR files within a certain

directory. This means that the following command is equivalent to the previous one:

```
java -cp .;C:\DirectoryWithFileJARFile\* mypackage.MyMainClass
```

We can use CLASSPATH wildcards only from Version 6 of the language onwards.

Appendix F

Introduction to the Unified Modeling Language

Goals:

At the end of this appendix, the reader should:

- ✓ Know how to define UML, and know its history (Units F.1, F.2, F.3, F.4).

Nowadays we hear about UML very often, but not everyone who talks about UML knows what it really is. Some think it's a programming language and this equivocation is due to the word "language". Others think it is an object-oriented methodology and this is probably due to the misinterpretation of not very in-depth readings. We often hear about UML together with various methodologies. Therefore, to define correctly what UML is, it is preferable to first broadly define what a methodology is.

F.1 What is UML?

An object-oriented **methodology**, in its most general definition, can be understood as a couple constituted by a *process* and a *modeling language*.



The term we are using, for accuracy, is the definition of a "method". A methodology is technically defined as "the science that studies methods". However, these terms are often considered synonymous.

A **modeling language** is, on the other hand, the tool that the methodologies use to describe (possibly in a graphic way) all the static and dynamic characteristics of a project.

Actually, **UML** is nothing more than a modeling language. It is made up, essentially, of a series of graphic diagrams whose elements are simple lines, ovals, rectangles, stylized men and so on. These diagrams have the task of clearly describing everything that may be difficult or too long when it comes to textual documentation during a project.

F.2 When and Where It Was Born

Starting from the early eighties, the global computer scene began to be invaded by object-oriented languages such as SmallTalk and, above all, C++. This is because, with the increasing complexity of the software, and the corresponding design philosophy, the limits of functional programming became evident and it proved insufficient to satisfy the ever-increasing technological pretensions. A new object-oriented mentality was affirmed and new theories emerged which set themselves the ultimate goal of providing more innovative techniques of creating software, obviously exploiting the paradigms of objects. The object-oriented methodologies, in large quantities and all more or less valid, were born one after the other. Initially, they were closely related to a well-defined programming language, but the mentality changed quite early. Beginning in 1988, the first books on object-oriented analysis and design were published. By '93, we had reached a point where there was great confusion: analysts and expert designers such as James Rumbaugh, Jim Odell, Peter Coad, Ivar Jacobson, Grady Booch, Ivar Jacobson and others, all proposed their own methodology, and each of them had their own group of enthusiastic followers. The beginning of the end of the “war of methodologies” coincided with the definition of UML in 1997, but the times had yet to mature.

F.3 Why UML Was Born (Why)

The fundamental problem was that different methodologies proposed not only different processes, which can be positively evaluated, but also different notations. It was clear to everyone that a standard methodology couldn't exist. In fact, the various existing processes had characteristics particularly suited to solving some particular problems. In practice, when starting a project, it is good to be able to choose between different resolute stratagems (processes). The fact that every process is strictly linked to a specific modeling language obviously represents nothing but a hindrance to the various members of a development team. The need for a standard language for methodologies was felt by many, but none of the authors intended to take the first step.

F.4 Who Created UML (Who)?

At that time, **Rational Software Corporation** (now acquired by IBM) thought about it and among its experts was **Grady Booch**, author of a very famous methodology at the time, known as **Booch Methodology**. In 1994, **James Rumbaugh**, creator of the **Object Modeling Technique (OMT)**, one of the most used object-oriented methodology, joined the Booch team at Rational. In October 1995, the 0.8 version of the so-called **Unified Method** was released. So, in the space of a few days, the Swede, **Ivar Jacobson** also joined Booch and Rumbaugh, starting a collaboration that then became historic. In addition to the fundamental concept of “Use Case”, Jacobson brought the famous **Object-Oriented Software Engineering** methodology (**OOSE**), also known as **Objectory** (which in reality was the name of the Jacobson company then incorporated by Rational). Here then the **tres amigos** (as they were nicknamed) began to work on the **Unified Software Development Process (USDP)** which was then renamed, simply **Unified Process (UP)** and, above all, the UML project. The **Object Management Group (OMG, <http://www.omg.org>)**, a non-profit-making consortium that deals with standardization, maintenance and creation of specifications that may be useful in the world of information technology, in the same period, forwarded to all the most important authors a “request for proposal” (RFP) of a standard modeling language. So Rational, together with other big partners such as IBM and Microsoft, proposed version 1.0 of UML in October 1997. OMG responded by setting up a “Revision Task Force” (RTF) led by Cris Kobrin to make improvements to UML. The current version of UML is 2.5.1, but there is a lot of confusion among UML users. In fact, the difficulty of “interpreting the specifications”, the different points of view of the most important authors and the anchoring of some authors to the first versions of the language (for example, the tres amigos), unfortunately make the picture unclear. Indeed, OMG’s ultimate goal is to make UML an ISO standard and that’s why the specifications are intended, rather than for UML users, for the creators of UML development tools. That’s why the specifications are defined in the very complex format: the *UML metamodel*. That is, UML is described by itself. In particular, the UML metamodel is divided into four sections and, just to get an idea of how complex it is, a language is defined (the **Object Constraint Language**, also called **OCL**) only for the purpose of defining unambiguous syntax. All this is in the appreciable future hypothesis of creating applications only by dragging UML elements on top of each other through development tools. Highly recommended (also in the bibliography of Appendix O) for learning UML, is Martin Fowler’s evergreen best seller “UML Distilled”. This book is pragmatic, practical, “sincere” and full of precious references.

The syntax of the most important diagrams of UML will be introduced schematically in the next appendix G.

Appendix G

UML Syntax Reference

Goals:

At the end of this appendix the reader should:

- ✓ Know how to consult the following Syntax Reference to be able to interpret or create simple UML diagrams (Unit G.1).

This appendix presents schematic tables of UML 1.3 syntax. Even if the version is obsolete, it is possible to take advantage of all the following definitions that constitute the fundamental nucleus of the language.

G.1 UML 1.3 Syntax Reference

Unified Modeling Language Syntax Reference		
Diagram Name	Element Names	
Use Case Diagram	Actor	Use Case
	Relationship Link	System Boundary
	Inclusion	Extension
	Generalization	Actor Generalization
Use Case Diagram: shows the interaction between the system and the system users.		

Unified Modeling Language Syntax Reference

Class Diagram	Class / Object	Association / link	Navigability
	Attribute	Aggregation	Multiplicity
	Operation	Composition	Qualified Association
	Member Properties	Extension	Association Class
	Abstract Class / Interface	Implementation	Roles Names
Class Diagram: shows the system classes and the existing relationships among them. N.B.: when it shows the system objects, it's sometimes called as Object Diagram.			
Component Diagram	Component	Dependency	
Component Diagram: shows the software components and their dependencies.			
Deployment Diagram	Node	Link	
Deployment Diagram: shows the physical (hardware) structure of the system.			
Interaction Diagrams: Sequence & Collaboration	Actor	Message	
	Object	Asynchronous Message	
	Creation	Life Line	
	Destruction	Activity Line	
Interaction Diagram: shows how a group of objects collaborate in a fixed lap of time. Sequence Diagram: highlights the messages sequence. Collaboration Diagram: highlights the architectural structure of the objects.			

Unified Modeling Language Syntax Reference

State Transition Diagram	State	Transition
	Start	Action
	End	History

State Transition Diagram (or State Chart Diagram or State Diagram): describes the object behavior showing the states and the events that determinate the transitions from state to state.

Activity Diagram	Activity	Flow
	Branch/Merge	Fork/Join
	Swimlane	Other elements

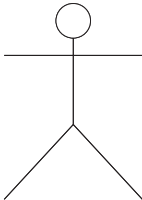
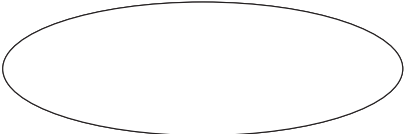
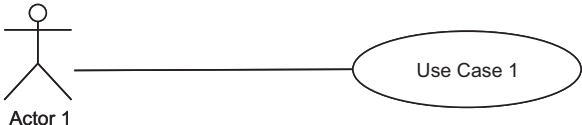
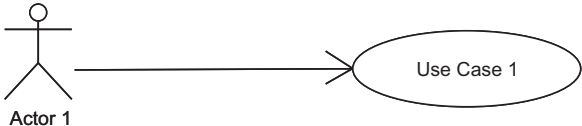
Activity Diagram: describes the system processes with either conditional and parallel activities sequences.

General Purpose Elements & Extension Mechanism	Package	Iteration mark
	Stereotype	Condition
	Constraint	Tagged Value

Generic Elements and Extension Mechanisms: UML elements useful in various diagrams.

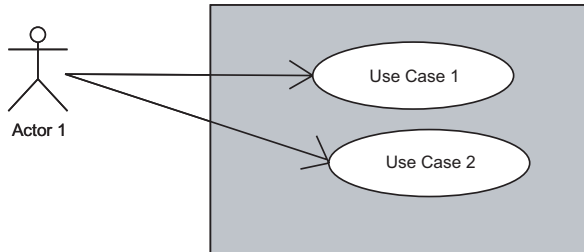
G.1.1 Use Case Diagram

The following table shows the elements of the **use case diagram**.

Use Case Diagram Syntax Reference	
Diagram Name	Element Names
Actor	<div><div>Alternative</div><div><div><<Actor>></div><div>Actor Name</div></div></div> <div>Actor Name</div>
Actor: role played by the user towards the system. N.B.: an actor may not be a person (but for example, a system).	
Use Case	<div><div>Use Case Name</div><div>Alternative</div><div><div>Use Case Name</div></div></div>
Use Case: set of scenarios linked by a common goal for the user. Scenario: is a sequence of steps that describes the interaction between the user and the system. NB: it is possible to describe scenarios using dynamic diagrams.	
Relationship link	<div><div><div>Alternative</div><div></div></div><div>Actor 1</div><div>Use Case 1</div></div>
Relationship Link: logical relationship link between an actor and a use case.	

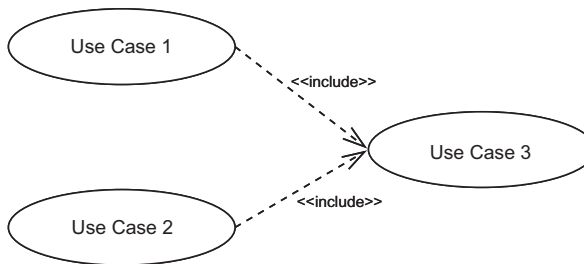
Use Case Diagram Syntax Reference

System Boundary



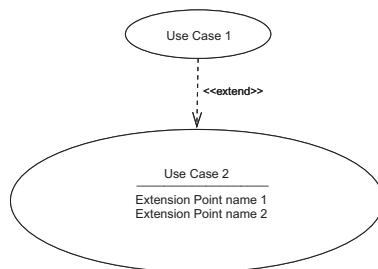
System Boundary: system domain boundary.

Inclusion

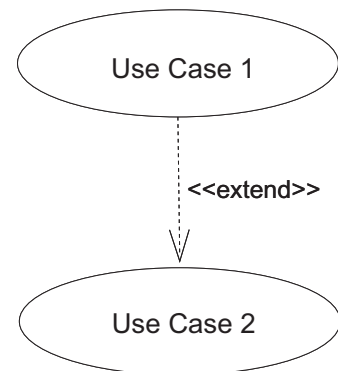


Inclusion: logical relationship link between use cases, that extract a common behavior for more use cases.

Extension



Alternative

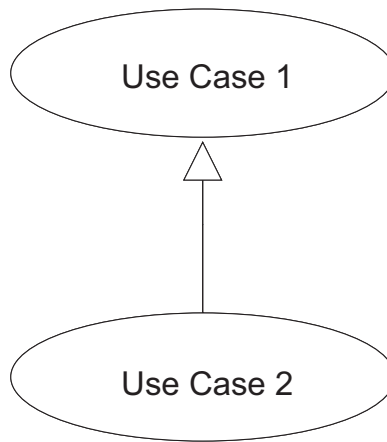


Extension: logical relationship link among use cases with the same semantic target. The specialized use case, reach his target adding extension points, written in the base use case.

Extension Point: describes a specialized use case behavior, not used by the base use case.

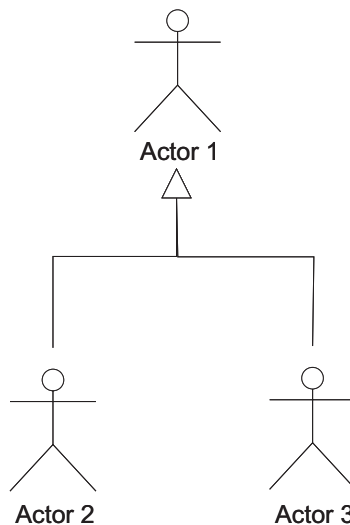
Use Case Diagram Syntax Reference

Generalization



Generalization: logical relationship link between use cases, with the same semantic target. The specialized use case, reaches his target adding new behaviors not used by the base use case, but without syntax formalisms.

Actor generalization



Actor Generalization: logical relationship linking actors. An actor who specializes an actor can relate to any use case related to the extended actor. It can also relate to other use cases, not related to the actor extended.

G.1.2 Class Diagram

The following table shows the elements of the **class diagram**.

Class Diagram Syntax Reference	
Diagram Name	Element Names
Class & Object	<div> <div>ClassName</div> <div>-attributeName: typeName</div> <div>+operationName(): returnType</div> <div>Class</div> </div>
	<div> <div>ObjectName</div> <div>Object</div> </div>
Class: abstraction for a group of objects that share the same functions and features. Object: physical creation (or instance) of a class.	
Attribute	<div> <div>ClassName</div> <div>-attributeName: typeName</div> </div>
Attribute (or instance variable or variable member or class characteristic): characteristic of a class/object.	
Operation	<div> <div>ClassName</div> <div>+operationName(): returnType</div> </div>
Operation (method or function member): functionality of a class/object.	

Class Diagram Syntax Reference

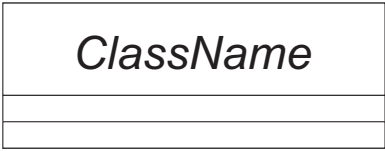
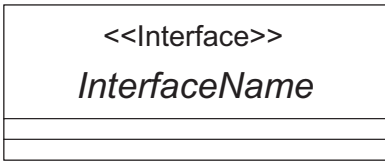
Member Properties	+ memberName “public member”
	# memberName “protected member”
	- memberName “private member”
	<u>memberName</u> o \$memberName “static member”
	operation o operation {abstract} “abstract operation”
	/attributeName “derived attribute”

Public, protected, private: visibility modifiers.

Static Member (or class member): member of the class.

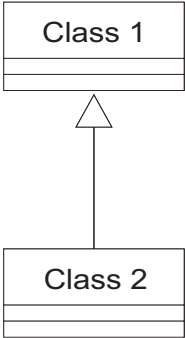
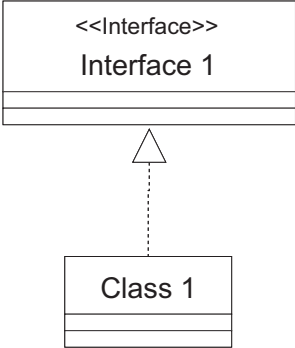
Abstract Operation: method signature (method without implementation).

Derived Attribute: attribute derived from others.

Abstract Class & Interface	 <p>Abstract Class</p>	 <p>Interface</p>
---------------------------------------	---	---

Abstract Class: class that cannot be instantiated that usually declare abstract methods.

Interface: data structure non-instantiable that can declare only abstract methods (and public static constants).

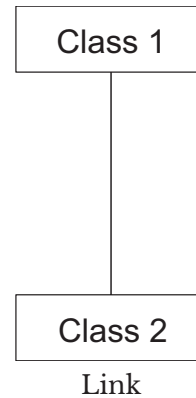
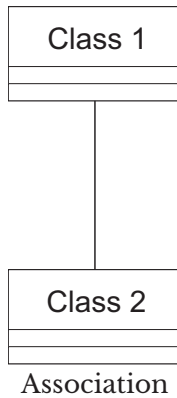
Extension & Implementation	 <p>Extension</p>	 <p>Implementation</p>
---------------------------------------	--	--

Extension: inheritance relationship among classes.

Implementation: extension to implement abstract methods of an interface.

Class Diagram Syntax Reference

Association



Association: relationship among classes where one uses the services (the operations) provided from the other.

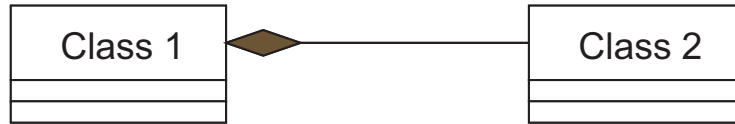
Link: association relationship among objects.

Aggregation



Aggregation: association characterized by containment.

Composition



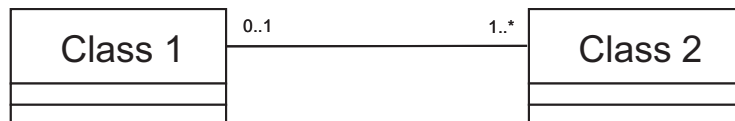
Composition: aggregation among objects with shared life cycle.

Navigability

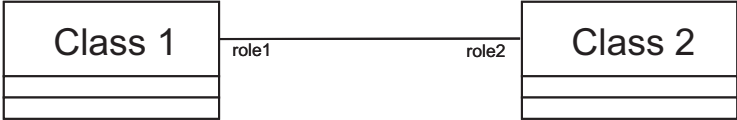
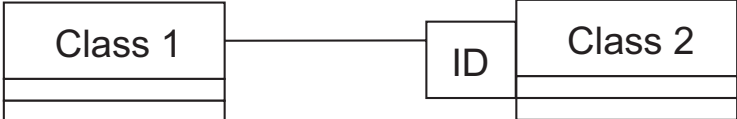
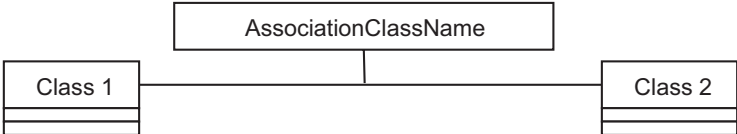


Navigability: direction of an association.

Multiplicity

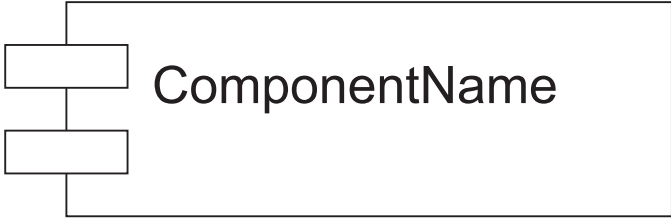


Multiplicity: correspondence among the cardinalities of the objects of the classes involved in the association.

Class Diagram Syntax Reference	
Role Names	
Role Names: description of a class behavior in an association.	
Qualified Association	
Qualified Association: association in which a class object is identified from another class through a sort a “foreign key”.	
Association Class	
Association Class: association coded as a class.	

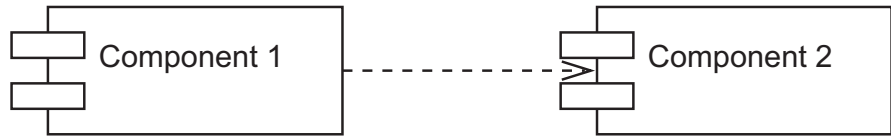
G.1.3 Component & deployment diagram

The following table shows the components of the **component diagram** and of the **deployment diagram**.

Component & Deployment Diagram Syntax Reference	
Diagram Name	Element Names
Component	
Component: executable software module, with identity and with a well specified interface. Usually, it can be developed independently.	

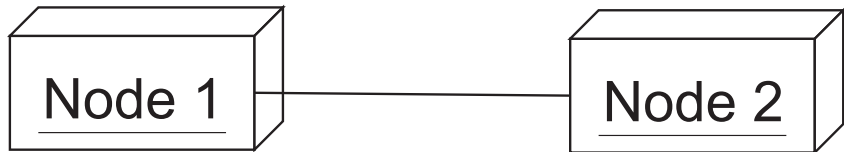
Component & Deployment Diagram Syntax Reference

Dependency



Dependency: relationship among two modelling elements, for which a change in the independent element implies a change in the dependent one. module.

Link



Link: logical relationship in which a participant (a component, a node or a class) uses the other participant services.

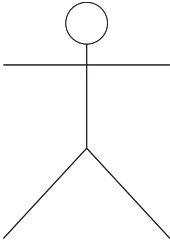

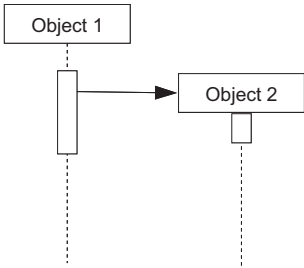
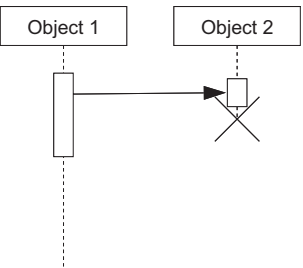
Node



Node: representation of a hardware platform.

G.1.4 Interaction diagram

The following table shows the elements of the **interaction diagrams**, i.e. the **sequence diagram** and the **collaboration diagram**.

Interaction Diagram Syntax Reference	
Diagram Name	Element Names
Actor	 Actor Name
Actor: role played by the user towards the system. N.B.: a user may not be a person (but for example, a system).	
Object	
Object: physical creation (or instance) of a class.	
Creation & Destruction	<div> Creation</div> <div> Destruction</div>
Creation: starting point of an object life line (it can coincide with a call of the created object constructor). Destruction: ending point of an object life line (it can coincide with a call of the created object destructor).	

Interaction Diagram Syntax Reference

Life Line & Activity Line

Object 1

Life Line

Object 1

Object 2

Activity Line

Life Line: object life line.

Activity Line: activity line of an object (coincides with a method execution block).

Message

Object 1

Object 2

Sequence

Object 1

1: →

Object 2

Collaboration

Message: collaboration message among objects (it can coincide with a call of the target object destructor).

Asynchronous Message

Object 1

Object 2

Asynchronuos Message

Asynchronous Message: message that can be executed asynchronously.


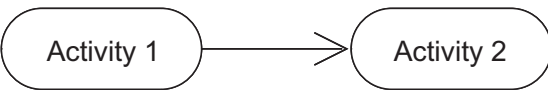
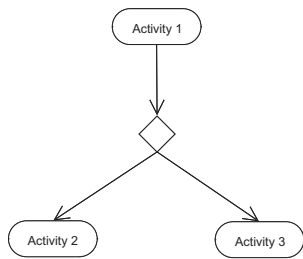
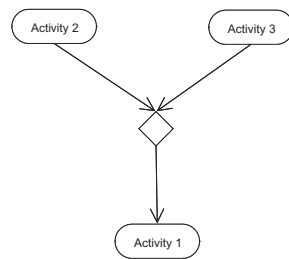
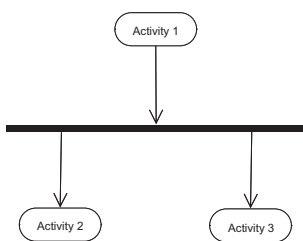
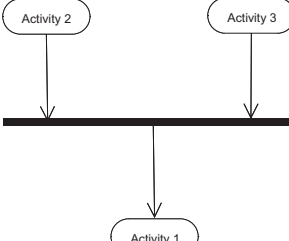
G.1.5 State Diagram

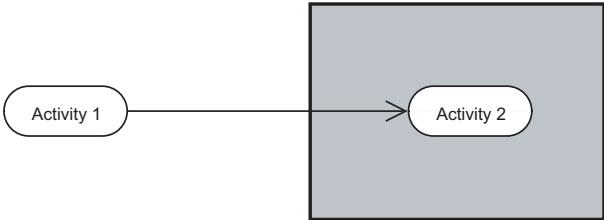
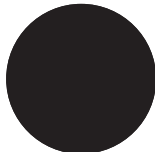
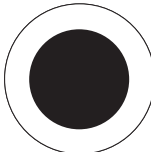


The following table shows the elements of the **state diagram**.

State Transition Diagram Syntax Reference	
Diagram Name	Element Names
State	<div>StateName</div>
State: represents the object state. It can be specialized with actions (internal transitions).	
Transition	<div>State 1eventName(eventsArgs)[condition]/ActionState 2</div>
Transition: activity that ends with a new state for the object.	
Start & End	<div>StartEnd</div>
Start: starting point of a state transition diagram. End: ending point of a state transition diagram.	
Action	<div>StateName action/actionName</div>
Action: activity that characterizes a state.	
History	<div>State 1 HState 2</div>
History: state with memory: it can repristinate precedent states.	

G.1.6 Activity diagram

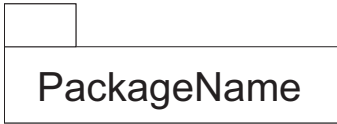
Nella tabella seguente sono riportati gli elementi del **diagramma delle attività** (**activity diagram**).

Activity Diagram Syntax Reference	
Diagram Name	Element Names
Activity	
Activity: system process.	
Flow	
Flow (of activity): set of scenarios linked by a common goal for the user.	
Branch & Merge	<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;">  <p>Branch</p> </div> <div style="text-align: center;">  <p>Merge</p> </div> </div>
Branch: represents a conditional choice. Merge: represents a termination point of a conditional block.	
Fork & Join	<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;">  <p>Fork</p> </div> <div style="text-align: center;">  <p>Join</p> </div> </div>
Fork: represents a starting point for concurrent activities. Join: represents a termination point for concurrent activities.	

Activity Diagram Syntax Reference		
SwimLane		
Swimlane: area of activity competence.		
Start & End	 Start	 End
Start: starting point of a state transition diagram. End: ending point of a state transition diagram.		
Object & State	 Object	 State
Object: physical creation (or instance) of a class. State: represents the object state. It can be specialized with actions (internal transitions).		

G.1.7 General Purpose Elements

The following table shows all the UML elements, which can be used on all the diagrams.

General Purpose Elements Syntax Reference		
Diagram Name	Element Names	
Package & Stereotype	 Package	<<Stereotype>> Stereotype
Package: notation useful to group UML elements. Stereotype: extension mechanism useful to stereotype non-standard constructs in UML.		
Constraint & Tagged value	{constraint} Constraint	{key = value} Tagged Value
Constraint: extension mechanism useful to specify constraints. Tagged value: property constraint.		
Iteration Mark & condition	* Iteration Mark	[condition] Condition
Iteration Mark: represents an iteration. Condition: represents a condition.		

Appendix H

Introduction to XML

Goals:

At the end of this appendix, the reader should:

- ✓ Understand what markup languages are (Unit H.1).
- ✓ Know how to briefly define XML, understand how to structure an XML document, and learn the concepts of a well-formed and valid XML document (Unit H.2).

XML is the acronym of **eXtensible Markup Language**. So, let's start with this definition first.

H.1 Markup Languages

A **markup language** is a language that allows us to annotate textual documents in such a way that they are readable both from the point of view of the machine and from the point of view of the human user. Usually, the *markings* are the so-called **tags**, that surround parts of the text of the document just to label them. Markup languages, more than anything else should be considered *metalanguages*. In fact, the tags of a markup language *annotate* the text, associating information that can be interpreted by both a human and a piece of software.

There are several markup languages, the first was invented by IBM in 1969, and is called **GML** (stands for **Generalized Markup Language**).

GML evolved later into **SGML** (that stands for **Standard Generalized Markup Language**). It was created by ANSI (<https://www.ansi.org>) and was standardized by ISO (<https://www.iso.org/home.htm>) in 1986. It is a metalanguage that allows us to create markup languages (like XML and HTML). The merits of SGML were portability, flexibility, power, standardization (ISO-8879) and the fact that it wasn't a proprietary technology. Its flaws were two-fold: too complex for most developers, and too heavy. In fact, an SGML document necessarily requires a Document Type

Definition (DTD) and a style sheet (Style-Sheet) to be associated with it.

Tim Berners-Lee, was inspired by SGML when he created **HTML**, which stands for **Hyper-Text Markup Language**, the main language for writing hypertexts (web pages).

A “hypertext” is a type of document (for example a web page) that includes, as well as simple text, links to other pages or resources.

HTML is also briefly introduced in Appendix I.

Like HTML, **XML** is also a markup language, and was initially defined in 1996, by an SGML working group, with a view to replacing SGML, but it is currently supported and defined by the W3C consortium (<https://www.w3.org>).

H.2 The eXtensible Markup Language

XML, on the other hand, was designed to be a simple, robust, extensible and generic language that can be used over the Internet. The XML format is text-based, and is therefore readable and easy to document. Its robustness is given by the fact that an XML document passes through two verification phases, including the validation test through another file (DTD or Schema, see section H.2.3). It is *extensible* because we can create custom tags, and new files to validate them and then new technologies. In fact, XML was also the basis on which many technologies were born, such as SOAP Web Services, the format of Microsoft Office documents, RSS feeds and many others.

H.2.1 XML Documents

An XML document is very simple. Below, we create one with a single tag called greeting.

```
<?xml version="1.0"?>
<greeting>
  Hello World!
</greeting>
```

The first line is called a **prologue**, and it's optional (see section H.2.2). Then the greeting tag follows, surrounding the text “Hello World!”. The tag name is always enclosed in sharp brackets, and a tag is optionally composed of an **opening tag** and a **closing tag**. The latter is recognizable by the symbol / which precedes the tag name. There are other tags that are composed of a single element characterized by a / that, in this case, follows the tag name, and which represents an opening and closing tag at the same time. Sometimes this type of tag is called an **empty tag**, since it does not contain any other tags or text. For example, let's consider the following example that abstracts an email with XML:

```
<?xml version="1.0"?>
<email>
  <from>claudio@claudiodesio.com</from>
  <recipients>daxixlx.sxprrx@gmail.com</recipients>
  <subject>New project</subject>
  <urgent/>
  <body format='text'>Ok, let's do it! Bye!</body>
</email>
```

The urgent tag is a tag type consisting of a single opening and closing element (empty tag). The rest of the code is clear enough, within the email tag tags are declared that represent the sender (from tag), recipients (recipients tag), the subject of the mail (subject tag), the urgency of the mail (urgent tag) and the body of the mail (body tag) also accompanied by a format attribute, which indicates the format (in this case, text) of the body of the email.

XML is case sensitive, and free-form.

H.2.2 Structure of an XML Document

The structure of an XML document is always made up of a **prologue** (optional but recommended), followed by the actual tags that make up the element called **document**. For example:

```
<?xml version="1.0"?>      <!-- prologue -->
<greeting>                  <!-- start document -->
  Hello world!
</greeting>                 <!-- end document -->
```

H.2.2.1 Prologue Structure

The prologue has the following structure:

- XML declaration (optional, but recommended)
- Version information
- Stand-alone information
- Encoding information (encoding)
- Document Type Declaration (DTD) (explicit and/or redirected) (optional)
- Comments (optional)

An example of an XML prologue could be:

```
<?xml version="1.0" standalone="yes" encoding="UTF-8"?>
```

Where:

- `<?xml ... ?>`: is the XML declaration
- `version`: defines the version information
- `standalone`: defines how to use the DTD file associated (if any)
- `encoding`: defines the encoding information

Comments are always included within this symbol structure:

```
<!-- This is a comment -->
```

and can be placed anywhere in the XML between one tag and another.

H.2.2.2 Document Structure

The document has the following structure:

- **root** element (root)
- other elements (optional)
- **attributes** (optional)
- text (optional)
- comments (optional)

Let's take again the example of an XML document already seen in the previous section 2.1 which abstracts an email:

```
<email>
  <from>claudio@claudiodesio.com</from>
  <recipients>daxixlx.sxprrx@gmail.com</recipients>
  <subject>New project</subject>
  <urgent/>
  <body format='text'>Ok, let's do it! Bye!</body>
</email>
```

In this case:

- `<email>` is the root element
- `<from>`, `<subject>`, `<recipients>`, `<urgent>` and `<body>` are other elements
- `format` is the attribute of the body tag

- New project is the test of the tag subject, and Ok, let's do it! is the text of the tag body

H.2.3 Characteristics of an XML Document

An XML document must be **well formed**. A well-formed XML document can also be declared **valid** if it satisfies the constraints specified in a file that defines its syntax.

H.2.3.1 Well Formed XML Document

An XML document is well-formed when:

- a root element exists
- it defines elements with opening tags (for example <data>) and closing tags (for example </data>)
- it optionally defines elements with special empty tags (for example <data/>)
- the tags can be inserted but without overlapping:
 - example of NOT valid nesting:

```
<outer>outer tag text<inner>inner tag text</outer></inner>
```

- valid nesting:

```
<outer>outer tag text<inner>inner tag text</inner></outer>
```

- the attribute values can be enclosed in single or double quotes. For example:

```
<font size="12">font test</font>
```

- the names of the elements and attributes are case-sensitive. The following tags are all different: <foo>, <F00> and <Foo>

We can check if an XML file is well formed by opening it with a browser. In Figure H.1, we can see the result of opening the well-formed file that abstracts an email, with the browser Microsoft Edge:

```
<email>
  <from>claudio@claudiodesio.com</from>
  <recipients>daxixlx.spxrx@gmail.com</recipients>
  <subject>New project</subject>
  <urgent/>
  <body format='text'>Ok, let's do it! Bye!</body>
</email>
```

```

    <?xml version="1.0" encoding="ISO-8859-1"?>
-   <email>
        <from>claudio@claudiodesio.com</from>
        <recipients>daxixlx.spxrx@gmail.com</recipients>
        <subject>New project</subject>
        <urgent/>
        <body format="text">Ok, let's do it! Bye!</body>
    </email>

```

Figure H.1 - View a well-formed document in the Microsoft Edge browser.

We can see how tags can also be compacted by clicking on the symbol – next to the email tag. If, instead, we modify the file incorrectly, for example by deleting the closing tag `</subject>`:

```

<email>
  <from>claudio@claudiodesio.com</from>
  <recipients>daxixlx.spxrx@gmail.com</recipients>
  <subject>New project
  <urgent/>
  <body format='text'>Ok, let's do it! Bye!</body>
</email>

```

the browser will no longer be able to analyze the XML and will show the screen shown in Figure H.2.

```

claudio@claudiodesio.com daxixlx.spxrx@gmail.com New project Ok, let's do it! Bye!

```

Figure H.2 - Display of a document NOT well formed in the browser Microsoft Edge.

H.2.3.2 Valid XML Document

XML allows us to create tags as needed. In this way, it allows us to abstract any kind of concept. Then, however, a program that must interact with XML must know the grammar of the document to be able to interpret it. For example, suppose we create a program that turns documents that abstract emails into real emails to be sent. It is important that all XML documents of the email type have the same characteristics and follow the rules, or are **valid** for the purposes of

the program. The validation mechanism is used to define the rules that an XML document must comply with. This mechanism requires:

1. the creation of a **validation file** that defines the rules of one or more XML documents (as we will see, a DTD or XML Schema file);
2. the association of the XML document to be validated with the validation file;
3. the execution of a parser that analyzes the validity of the XML document with respect to its validation file.

There are two types of validation files that can be created. The first type (fallen into dis-use) is called a **DTD** file (acronym of **Data Type Definition**). The second type is called **XML Schema**.

With a DTD document, we can define the structure of an XML document. Consider the following simple XML file:

```
<?xml version="1.0"?>
<user>
  <name>Oscar</name>
  <surname>Wilde</surname>
  <id>8</id>
</user>
```

The following DTD file defines the structure of the previous XML document:

```
<!DOCTYPE user
[
  <!ELEMENT user (name,surname,id)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT surname (#PCDATA)>
  <!ELEMENT id (#PCDATA)>
]>
```

With DOCTYPE, we define the root element (email) and with ELEMENT we define the various tags. With (#PCDATA), we indicate to the parser that it will take care of validating the document, that the tag contains some text, while we would have used EMPTY in case we used an empty tag. If we want to use an XML Schema instead of a DTD file, then the syntax changes:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="user">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="surname" type="xs:string"/>
        <xs:element name="id" type="xs:integer"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```
        </xs:sequence>  
    </xs:complexType>  
</xs:element>  
</xs:schema>
```

XML Schema today represents the standard for the validation of XML documents, since it allows us to specify more constraints than a DTD, as well as having a more intuitive syntax. With Java, it is very easy to validate files as we will see in Chapter 22.

Appendix I

Applet

Goals:

At the end of this appendix, the reader should:

- ✓ Know how to define what an applet is (Unit I.1).
- ✓ Know how to define what HTML is (Unit I.2).
- ✓ Understand how an applet can be distributed (Unit I.3).



The Applet API has been deprecated in Java 9 and officially abandoned from Java 11 onwards. Currently, all the most important browsers no longer support the Java Plugin, which was necessary to run applets. Also other historical technologies such as Java Web Start, based on the Java Plugin, are now definitely dead.



Java Web Start is a technology that allows us to download a Java application, always updated, by clicking on a link on a web page. For information see the following page:

<https://docs.oracle.com/javase/9/deploy/java-web-start-technology.htm>.

For Oracle, the new standard for the distribution of Java applications with a graphical interface, is to create desktop standalone applications, to be distributed together with their own runtime environment created with **jlink** (see Chapter 19).

Although the Applet API can no longer be used in practice, given the historical importance of the technology (see Appendix A), we prefer not to completely eliminate the definition from this book. Those who are not interested can safely avoid reading this appendix.

I.1 Definition of Applet



An applet is an application designed to run directly within a web page; that is, an applet can be directly incorporated into an HTML page using a special tag: the `<APPLET>` tag. We will therefore briefly introduce the HTML language in the next section.

An **applet**, by definition, must extend the `Applet` class of the `java.applet` package. It inherits its methods and can override them. The applet does not have a `main()` method, but the inherited methods are directly invoked by the JVM browser (the Java Plugin) following certain rules. So, if we rewrite the methods properly, we can get the applet to execute the code we want. The following is a trivial applet containing explanatory comments:

```
import java.applet.*;
import java.awt.*;
public class BasicApplet extends Applet {
    public void init() {
        // Called only once by the browser as soon as the applet is run
    }
    public void start () {
        // Called whenever the page it contains the applet becomes visible
    }
    public void paint (Graphics g) {
        // Called whenever the page it contains the applet must be drawn
    }
    public void stop () {
        // Called whenever the page it contains the applet becomes invisible
    }
    public void destroy () {
        // Called only once by the browser when the applet is destroyed
    }
}
```

Taking into account what is written in the comments, the programmer must manage the execution of the applet. It is not mandatory to write all five methods, but at least one should be overridden. For example, the following applet prints a word:

```
import java.applet.*;
import java.awt.*;
public class StringApplet extends Applet {
    public void paint(Graphics g) {
        g.drawString("Applet", 10, 10);
    }
}
```

The `Graphics` object provides many methods for drawing (as mentioned in Chapter 23).

The term “applet” could be derived from “little application”. This is because an applet must be a lightweight application, since it has to be downloaded from the Web along with the HTML pages. Think about the fact that, in 1995, when Java was born (alongside applets), there were no broadband connections. In addition to the size, an applet must also undergo loading, security checks and the interpretation of the JVM. So, it’s good to be small.

To be able to run an applet, however, we also need to create an HTML page that incorporates it. So let’s introduce briefly what HTML is.

I.2 Introduction to HTML

HTML is the acronym for **HyperText Markup Language**, that is, a markup language for hypertexts. A **hypertext** is a type of document that, in addition to including simple text, has links to other pages. This is not a real programming language, but only a language for formatting documents. For example, there are no control structures like `if` or `for`. HTML instructions are called **tags**, as already defined in Appendix H. The tags are simple instructions with the following syntax:

```
<TAG_NAME [ATTRIBUTES_LIST]>
```

where each optional attribute has a syntax like this:

```
KEY=VALUE
```

The value of an attribute could and should be between two single quotes or two double quotes. However, this is necessary if, and only if, the value consists of several separate words. But it is a convention that everyone always uses. Furthermore, almost all HTML tags, with some exceptions, should be closed with a statement like this:

```
</TAG_NAME>
```

For example, the tag

```
<HTML>
```

must be closed with:

```
</HTML>
```

While the tag:

```
<applet code='StringApplet' width='100' height='100'>
```

must be closed with:

```
</applet>
```

A simple HTML page can be written in a text file with a `.htm` or `.html` extension. All you need is a simple text editor like Windows Notepad. A browser like Mozilla Firefox, Google Chrome, etc. then can show us the formatted page. So, let's consider the following `HelloWorld.html` file which contains the following tags:

```
<HTML>
  <HEAD>
    <TITLE>Test HTML</TITLE>
  </HEAD>
  <BODY bgcolor='green'>
    <CENTER>
      <H1>
        Hello
      <BR/>
        HTML World!
      </H1>
    </CENTER>
  </BODY>
</HTML>
```

In this example, the tags `HTML`, `HEAD`, `TITLE`, `BODY` and `CENTER` are declared. The name of the tag is always enclosed within “acute brackets”, and a tag is optionally composed of an opening tag and a closing tag. The latter is recognizable by the symbol `/` which precedes the name of the tag. The only tag that is not declared with an opening and a closure in our example is the `BR` tag, which is declared with an optional symbol `/` which this time follows the name of the tag, and which wants to indicate that the tag represents an opening and closing tag. When we open it in a browser, then we will see a web page entitled “Test HTML”, with the word “Hello World!” centred at the top, on a green background (see Figure L.1). The browser, in fact, knows how to interpret the HTML tags and formats the page accordingly.

In fact, the tags of the example have a meaning that browsers know how to interpret. In particular, the `HTML` tag indicates that we are declaring an HTML file, and must contain all the HTML code. The `HEAD` tag contains other tags that represent information about the document. In our case, it contains the `TITLE` tag, which contains the title of the document. When the `HEAD` tag is closed, the `BODY` tag opens, which contains the visible body of the document. Let's note that `BODY` also declares an **attribute** called `BGCOLOR` (which stands for “background color”) that specifies to the browser that the background color must be green. The `CENTER` tag will lead to

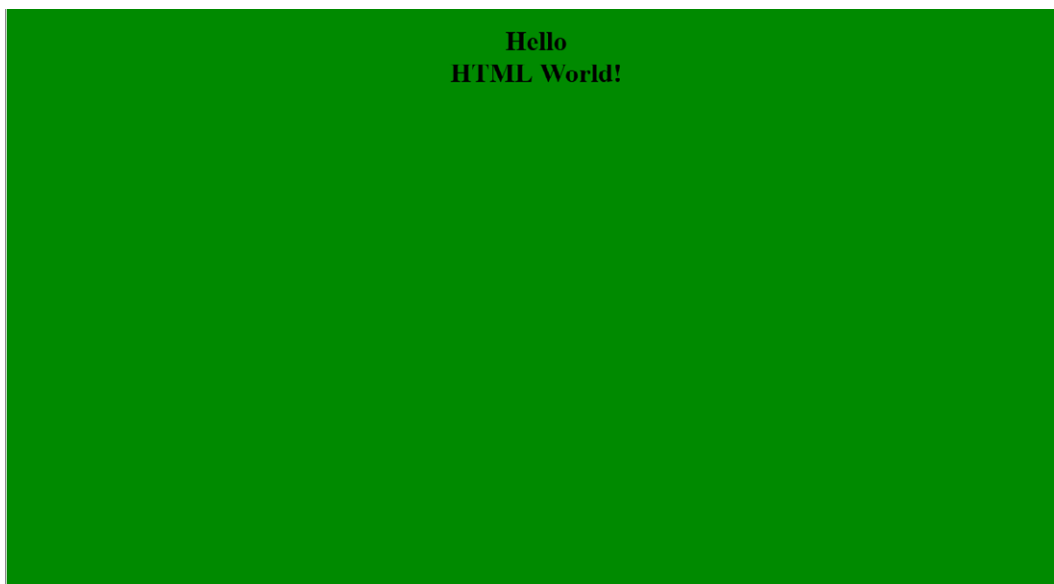


Figure I.1 - HTML page opened in the Mozilla Firefox browser.

an alignment centred on the page. The text contained in the H1 tag will have a size of type H1 (there are also H2, H3, H4, H5, and H6 tags) and represents the maximum text size. Finally, the BR tag which, as we have noted, does not consist of an opening and closing tag, it only represents the command to wrap the next text (it wouldn't be sufficient to simply write the word "HTML WORLD" on the next line).

HTML is not case-sensitive.

All that remains is for us to get an HTML manual (for example <http://www.w3schools.com/html>).

I.3 Installing Applet

Returning to our applet, it will be enough to create a simple file with the suffix .htm or .html, which contains the following tag:

```
<applet code='StringApplet' width='100' height='100'>
</applet>
```



As the width and height attributes change, the size of the area that the HTML page will dedicate to the applet will vary. It is also possible to pass to the HTML page parameters that will be read at runtime by the applet using the mechanism explained in the following example.

By adding the following override of the `init()` method to the previous example, you can parameterize, for example, the sentence to be printed:

```
import java.applet.*;
import java.awt.*;
public class ParameterApplet extends Applet {
    String s;
    @Override
    public void init() {
        String parameterName = "p";
        s = getParameter(parameterName);
    }
    @Override
    public void paint(Graphics g) {
        g.drawString(s, 10, 10);
    }
}
```

The code of the HTML page that must load the previous applet will slightly change:

```
<applet code='ParameterApplet' width='100' height='100'>
  <param name='p' value='Java' />
</applet>
```

Appendix J

Compiling Past Versions of Java Code

Goals:

At the end of this appendix, the reader should:

- ✓ Understand and be able to handle the problems of compiling code written with different versions of Java (Units J.1, J.2, J.3, J.4).

Java 5 introduced new rules in syntax (such as the foreach loop, generics, static imports and more) and even a new keyword: `enum` (as well as `@interface`). Also, Java 1.4 introduced a new keyword: `assert`. Then successive versions of Java like 7, with the construct `try-with-resources`, and Java 8 with `lambda`, `type annotation`, `reference to methods`, etc. have further altered the syntax. Java 10 again, introduced the special word `var`, Java 13 the word `yield`, while the introduction of the reserved words introduced in Java 9 (`exports`, `modules`, `open`, `opens`, `provide`, `require`, `to`, `transitive`, `uses` and `with`), however, does not create problems in compiling obsolete code, from the moment they are relevant to inside the declaration of a module. For example, we can use the `module` keyword as a reference to a string in the following way:

```
public class RestrictedWords {  
    public static void main(String args[]) {  
        String module = "this string use the reference 'module'";  
        System.out.println(module);  
    }  
}
```

But then we have to ask ourselves: what happens when we compile code that was written before the version we are using?

Usually nothing, the code is compiled quietly without problems. The code can be compiled with a command like:

```
javac [options] FileName.java
```

However, there are cases where problems can arise.

J.1 Warning

You can come across some warnings (see unit 9.3.5.5) for several reasons. For example, in pre-Java 5 code, there were no generics and annotations. Thus, all collections were used as raw type (without specifying generics) and all methods that overrode an inherited method were not annotated with the `@Override` standard annotation. This type of code causes a warning if compiled with post-Java 1.4 compilers. But the warnings do not prevent the compiler from creating valid bytecode, the compilation is successful anyway. Where it is not possible to update the old code, it is still possible to disable warnings, or ultimately, to simply ignore them.

J.2 Keywords Used as Identifiers

In particular, since Versions 10, 1.4 and 1.5 (better known as Version 5) have introduced new keywords, we might come across an obsolete code that uses the keywords `assert` or `enum` as variable identifiers. In fact, when these keywords did not exist, it was not uncommon to come across classes that declared `Enumeration` references (see Chapter 18) with the `enum` identifier. If we tried to compile such a class with a 1.5 (or higher) compiler, it would not be able to understand that the word `enum` should be considered as an identifier instead of a keyword, and we would get a compile-time error. The same applies to the code written before Java 1.4, which could make use of the `assert` identifier which, from Java 1.4, has become a keyword. If it is not possible to modify the obsolete code, then we just have to use an option at compilation time to specify which version of the source code you want to use

```
javac -source 1.4 FileName.java
```

In this way, we will inform the compiler to compile the file as if it were a version 1.4 compiler, i.e. without considering the new features introduced in versions subsequent to 1.4 (and therefore without the new keywords and new syntax features).

J.3 Current Syntax vs Previous Syntax

If we compile code that contains Java 5 syntax (static import, foreach loop, generics, etc.) using the above flag, we will get compile-time errors. The same is repeated if we use syntax from

newer versions such as Version 7 (try-with-resources) or 8 (lambda expressions, etc.) or Java 10 (var keyword) or 13 (yield keyword).

So, if we are forced to compile with a `-source` flag, specifying a version prior to the one we are using, then we will have to limit ourselves to using the syntax of the specified version.



J.4 The target Option

As for the execution of the code compiled with the `-source 1.4` option, if you plan to execute the file with a specific version of the JVM, you have to specify it with the `-target` flag, as in the following example:

```
javac -target 1.4 -source 1.4 FileName.java
```

In fact, while the `-source` option declares that the code contained in the file (or files) to be compiled contains a valid syntax for Version 1.4, the `-target` option declares the version of the Java Virtual Machine on which it will be possible to execute the bytecode compiled.

If these options are not specified, the default value for both `-source` and for `-target` will be that of the compiler version used.

Appendix K

Introduction to Apache Derby

Goals:

At the end of this appendix, the reader should:

- ✓ Be able to install and use the basic features of Apache Derby (Units K.1, K.2).

Up to Version 8 of Java, the JDK incorporated the *Java DB* database. It was nothing but the Apache Derby database, but Oracle offered support only for Java DB. With JDK 9, the Java DB was no longer incorporated into the JDK, so to get a copy of this RDBMS, we need to download it at: <https://db.apache.org/derby>. It is a database written completely in Java and is platform independent. It can be used in two different ways:

- as an **embedded database** in our Java application (i.e. that can be incorporated directly into our applications). Being written in Java, it can be used as part of the distributed application, completely transparent to the user. In this mode, the database will be accessible to only one client (the one that incorporates it);
- or it can be started in **Server mode**, like an ordinary database engine. In this case, the database can be accessed from multiple clients.

At the time of writing, the most updated version of Apache Derby is 10.15.1.3.

K.1 Apache Derby Installation

Up to Version 8 of Java, Java DB was distributed along with the Java Development Kit, and was therefore automatically installed on our machine. Now, we need to install Apache Derby independently of the JDK by following the steps below.

K.1.1 Software Download

At the address https://db.apache.org/derby/derby_downloads.html click on the first link with the version number. In our case, we have to click on the link “10.15.1.3” (see Figure K.1).

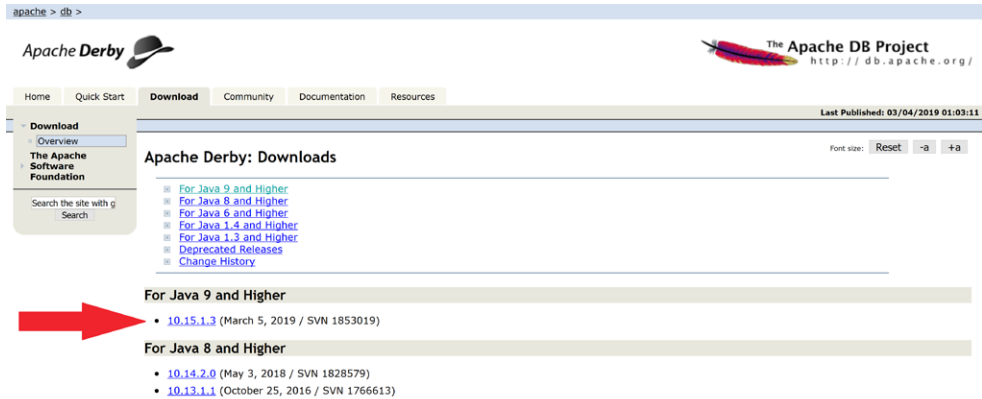


Figure K.1 - Choosing the version to download (the most updated).

On the next page, use the binary file link with the most convenient format. We have chosen the .zip format, as we can see in Figure K.2.

On this page we can also download other file types such as the source code of the project.

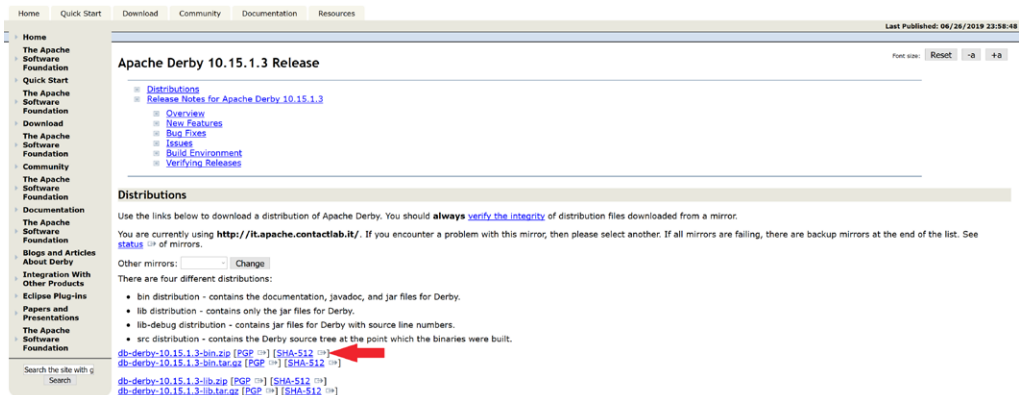
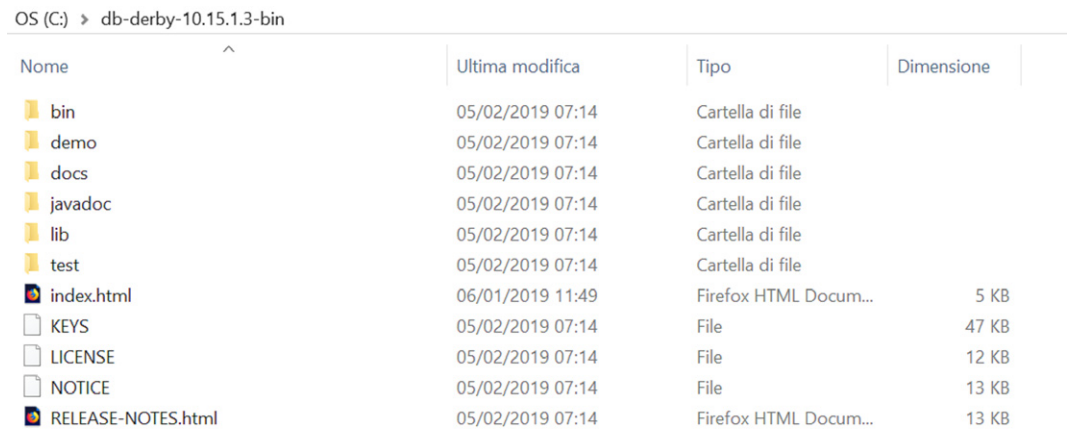


Figure K.2 - Choosing the file to download (the file that contains the binaries with the most convenient format).

At this point, the download of the zip file will start.

K.1.2 Installation

The software does not need any particular type of installation, as it is written in Java. We simply need to download the file and decompress it in an appropriate folder. For example, we have decompressed it in the root folder: **C:\db-derby-10.15.1.3-bin**. Nothing prevents us from placing the folder in any other location, and/or renaming the project folder. We can see the contents of the Apache Derby folder in Figure K.3.



Nome	Ultima modifica	Tipo	Dimensione
bin	05/02/2019 07:14	Cartella di file	
demo	05/02/2019 07:14	Cartella di file	
docs	05/02/2019 07:14	Cartella di file	
javadoc	05/02/2019 07:14	Cartella di file	
lib	05/02/2019 07:14	Cartella di file	
test	05/02/2019 07:14	Cartella di file	
index.html	06/01/2019 11:49	Firefox HTML Docum...	5 KB
KEYS	05/02/2019 07:14	File	47 KB
LICENSE	05/02/2019 07:14	File	12 KB
NOTICE	05/02/2019 07:14	File	13 KB
RELEASE-NOTES.html	05/02/2019 07:14	Firefox HTML Docum...	13 KB

Figure K.3 - Apache Derby installation folder.

In particular, note the following subdirectories:

- **bin**: which contains the execution and configuration scripts.
- **lib**: which contains the JAR files that represent the application itself.

K.1.3 Configuration

To make the installation fully functional, we need to set the **DERBY_HOME** environment variable with the Apache Derby installation directory, which in our case is **C:\db-derby-10.15.1.3-bin**. The procedure for installing the environment variable is identical to that described for setting the **PATH** variable in Appendix B. In Figure K.4, we can observe the setting of the **DERBY_HOME** variable with the Apache Derby installation folder on Windows 10.

It is also advisable to add the **C:\db-derby-10.15.1.3-bin\bin** folder to the **PATH** variable to be able to run Apache Derby from any location on the filesystem, without having to move to its installation folder. In Figure K.5, we can observe the setting of the **PATH** variable, adding the Apache Derby **bin** folder on Windows 10.

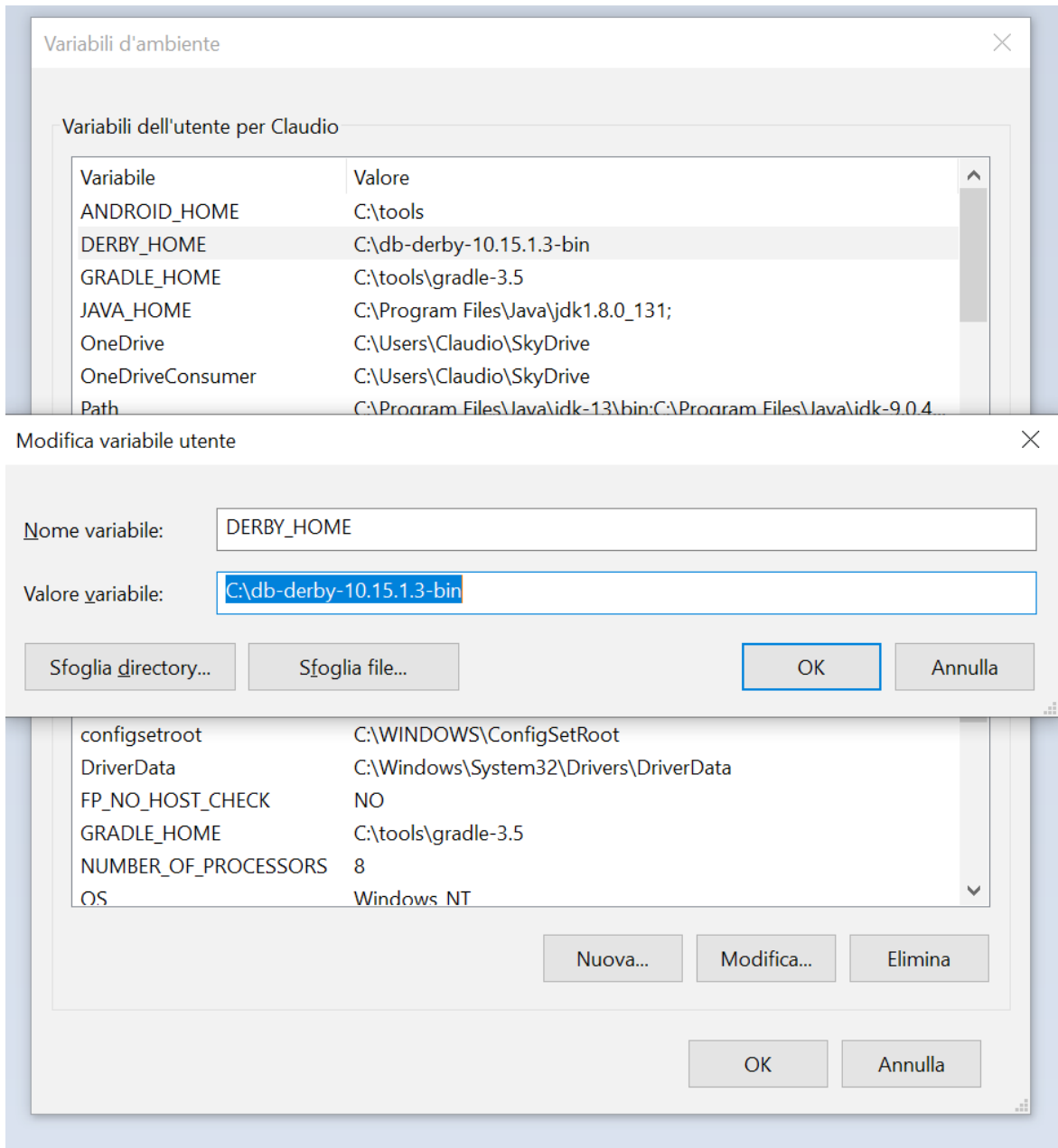


Figure K.4 - Configuration of the DERBY_HOME instance variable.

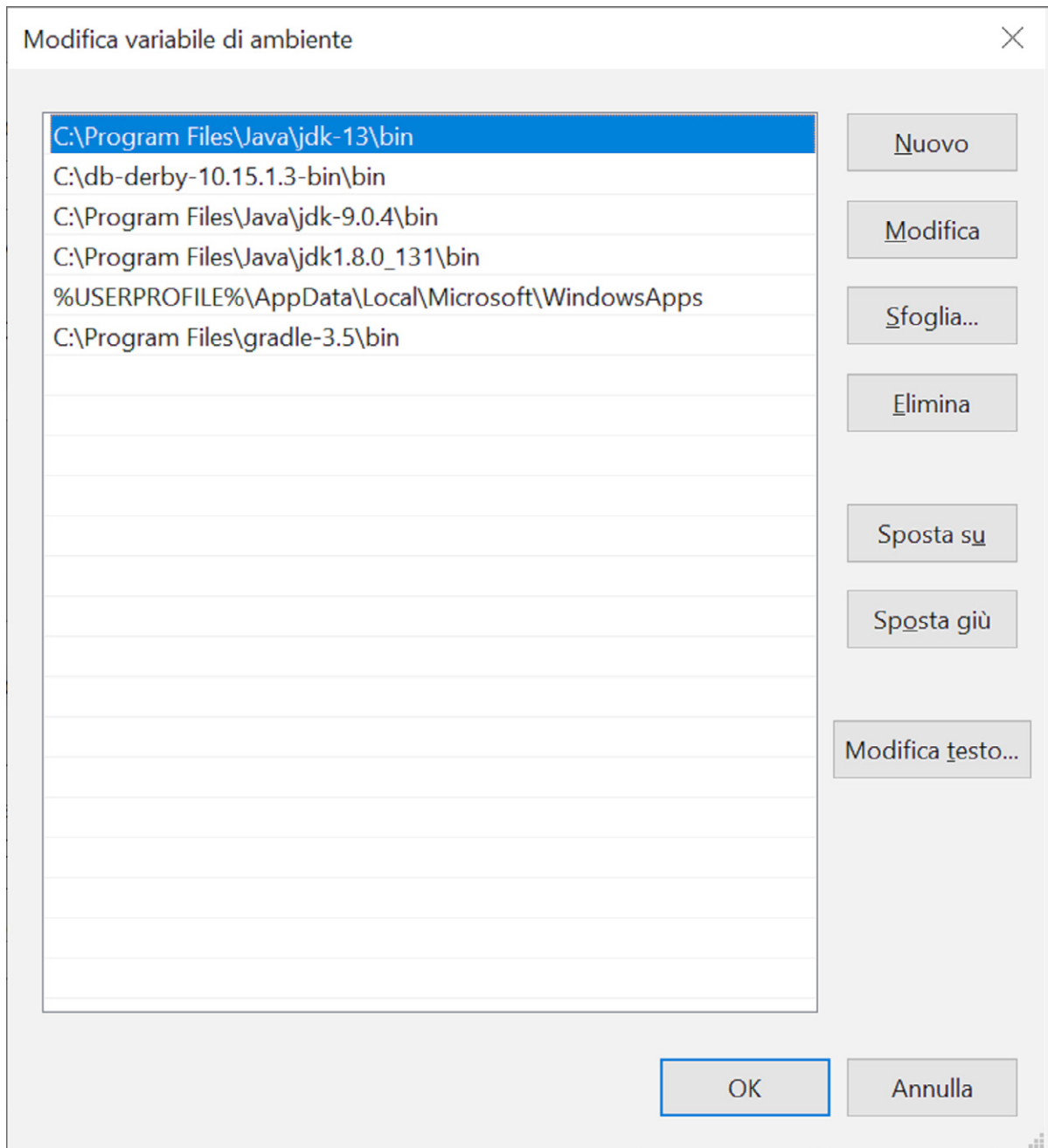


Figure K.5 - Configuration of the PATH instance variable.

K.2 Execution and Use of Apache Derby

As we have already stated, Derby can be run in two modes: **embedded** and **server**.

K.2.1 Server Mode

To run Apache Derby in **server mode**, use the **startNetworkServer.bat** file located in the **bin** folder. We can double-click on the file to run it directly from the **bin** folder (we can also create a desktop shortcut to avoid browsing the various directories). By executing the file with a double click, a command prompt will be opened automatically. The disadvantage is that, if something goes wrong, that prompt will close immediately, without giving us time to read the problem. But if we set the **PATH** variable as indicated in the previous section, we can also run the server from the command line. Just open the command prompt, and run the command **startNetworkServer.bat** from any folder (remember that from a DOS prompt the commands are case insensitive, and that we can also avoid specifying the **.bat** extensions):

```
startNetworkServer
```

The server will be started in a few seconds and will output the following information:

```
Security manager installed using the Basic server security policy.  
Apache Derby Network Server - 10.5.1.3 - (648739) started and ready  
to accept connections on port 1527 at 2019-05-28 22:17:17.921 GMT
```

At this point, as can be read from the RDBMS response, Apache Derby is ready to accept connections on port 1527.

In order to stop the database server, in the same folder there is also the stopNetworkServer.bat script.

In order for our application to be able to access the database, we must also set the **CLASSPATH** variable (see Appendix E) by pointing to the file **derbyclient.jar** (and if needed also to **derbytools.jar** and **derbyoptionaltools.jar**) found in the **lib** folder of the database installation. For example, if we want to execute the **JDBCApp** file that we saw in the examples in Chapter 21, we will have to execute the following instruction from the command line:

```
java -cp .;%DERBY_HOME%\lib\derbyclient.jar JDBCApp
```

Obviously, we must first create the database schema to which we want to connect (see section K.2.3).

K.2.2 Embedded Mode

To use Apache Derby in **embedded mode**, no server can be started. The database resides in a specific folder that is automatically created, and that can be installed with the application itself. In order to run our application, we have to add to the **CLASSPATH** variable (see Appendix E) a value that points to the **derby.jar** file (and **derbytools.jar** and **derbyoptionaltools.jar** if needed) that we can find in the **lib** folder of the database installation.

For example, if we want to execute the **JDBCApp** file that we saw in the examples in Chapter 21, we will have to execute the following instruction from the command line:

```
java -cp .;%DERBY_HOME%\lib\derby.jar JDBCApp
```

In this example, the database folder resides in the same folder as the **JDBCApp** file, and therefore the **CLASSPATH** also points to it.

K.2.3 Interactive Console

To start an interactive console for running SQL scripts, run the interactive console using the command:

```
ij.bat
```

which resides in the **bin** folder of the database installation. At this point, for example, we can create and modify a database with the following commands:

```
CONNECT 'jdbc:derby:Music;create=true';
CREATE TABLE Album ( AlbumId INT, Title VARCHAR(20),
  Artist VARCHAR(255), Release_Year INT, PRIMARY KEY (AlbumId));
INSERT INTO Album (AlbumId, Title, Artist, Release_Year)
  VALUES (1, 'Made In Japan', 'Deep Purple', 1972);
INSERT INTO Album (AlbumId, Title, Artist, Release_Year)
  VALUES (2, 'Be', 'Pain Of Salvation', 2004);
INSERT INTO Album (AlbumId, Title, Artist, Release_Year)
  VALUES (3, 'Images And Words', 'Dream Theater', 1992);
INSERT INTO Album (AlbumId, Title, Artist, Release_Year)
  VALUES (4, 'The Human Equation', 'Ayreon', 2004);
```

We created the database by populating a table we talk about in Chapter 21. Note that each instruction must be followed by a “;” for termination before pressing the **ENTER** key.

This should be enough to do some simple exercises.

If you want to learn more about the topic, you can read the documentation at the link: https://db.apache.org/derby/quick_start.html.

Appendix L

JavaFX Environment Configuration

Goals:

At the end of this appendix, the reader should:

- ✓ Be able to configure the programming environment to use JavaFX technology on Windows operating systems (Units L.1, L.2, L.3, L.4).

✓



In Chapter 24, JavaFX technology is introduced. From version 11 of Java onwards it is necessary to configure the JavaFX programming environment, in addition to the Java programming environment. Before being able to configure the environment for JavaFX, it is obviously necessary that the Java environment is already configured. In the unlikely event you have not yet installed the Java environment, you can consult Appendix B. Below you will find the steps to be taken to download and correctly configure the environment to program with JavaFX on Windows 10 operating systems.

Java and JavaFX versions progress at the same time. This means that if you have installed the Java version 13 environment, then you should install the JavaFX 13 environment.

L.1 JAVA_HOME Variable Setting

The first step is to configure the `JAVA_HOME` environment variable, so that it points to the JDK installation folder.

At the time of writing, the most recent version is 12.0.1, but the same processes that we'll explain in this appendix, apply to later versions. So, if you have downloaded a more recent JDK, where in the following chapter we name paths or strings that contain the string 12.0.1, you will need to update the version number with the one you are using.

For Windows 10 systems, perform the following steps:

1. open the **Control Panel**, then click on **System**, then on **Advanced System Settings** in the new window;
2. in the new window that opens, select the **Advanced** tab and click on **Environment Variables**;
3. among the **System Variables** (or if you prefer among the **User Variables**) add the `JAVA_HOME` variable, by clicking on the **New...** button;
4. Enter the string "`JAVA_HOME`" in the new window as a **Variable name**, and as a **variable value** the string "`C:\Program Files\Java\jdk-12.0.1`".
5. click **OK** on all the open windows, and the configuration of the variable is complete.

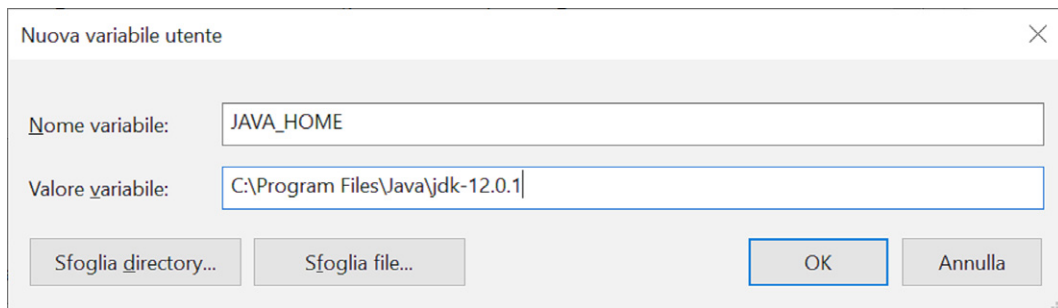


Figure L.8 - Setting `JAVA_HOME` variable on Windows 10 (Italian Version).

L.2 Download JavaFX SDK and Runtime

Go to <http://gluonhq.com/products/javafx>, and choose the version of the SDK (Standard Development Kit) for JavaFX to download. In general, you should choose the most recent, which in the

image L.2 is the 12.0.1 version, but obviously the choice is yours, as long as it is compatible with the version of the JDK you intend to use.

[Home](#) » [Products](#) » JavaFX

JavaFX

Long Term Support

JavaFX 11 is the first long term support release of JavaFX by Gluon. For commercial, long term support of JavaFX 11, please review our [JavaFX Long Term Support options](#).

The JavaFX 11 runtime is available as a platform-specific SDK, as a number of jmods, and as a set of artifacts in maven central.

The OpenJFX page at openjfx.io is a great starting place to learn more about JavaFX 11.

The Release Notes for JavaFX 11 are available in the OpenJFX GitHub repository: [Release Notes](#).

This software is licensed under GPL v2 + Classpath (see <http://openjdk.java.net/legal/gplv2+ce.html>).

Product	Public version	LTS version	Platform	Download
JavaFX Windows SDK	11.0.2	11.0.3 More info	Windows	Download SHA256
JavaFX Windows jmods	11.0.2	11.0.3 More info	Windows	Download SHA256
JavaFX Mac OS X SDK	11.0.2	11.0.3 More info	Mac	Download SHA256
JavaFX Mac OS X jmods	11.0.2	11.0.3 More info	Mac	Download SHA256
JavaFX Linux SDK	11.0.2	11.0.3 More info	Linux	Download SHA256
JavaFX Linux jmods	11.0.2	11.0.3 More info	Linux	Download SHA256
JavaFX armv8hf SDK	11.0.2	11.0.3 More info	Embedded armv8hf	Download SHA256

Latest Release

JavaFX 12 is the latest release of JavaFX. We will support it until the release of JavaFX 13

The JavaFX 12 runtime is available as a platform-specific SDK, as a number of jmods, and as a set of artifacts in maven central.

The Release Notes for JavaFX 12 are available in the OpenJFX GitHub repository: [Release Notes](#).

This software is licensed under GPL v2 + Classpath (see <http://openjdk.java.net/legal/gplv2+ce.html>).

Product	Version	Platform	Download
JavaFX Windows SDK	12.0.1	Windows	Download SHA256
JavaFX Windows jmods	12.0.1	Windows	Download SHA256
JavaFX Mac OS X SDK	12.0.1	Mac	Download SHA256
JavaFX Mac OS X jmods	12.0.1	Mac	Download SHA256
JavaFX Linux SDK	12.0.1	Linux	Download SHA256
JavaFX Linux jmods	12.0.1	Linux	Download SHA256

Early-Access Builds

Early-Access builds for JavaFX 13 are available for download.

These early-access builds are licensed under GPL v2 + Classpath (see <http://openjdk.java.net/legal/gplv2+ce.html>).

Figure L.2 - Screenshot of <http://gluonhq.com/products/javafx>. The arrow highlights the recommended links to download.

Furthermore, we can also download a JMOD files folder (see Figure L.2) which represents the collection of modules needed at runtime for the applications we will create. These modules must be distributed together with the application.

Once you have the zip files, unzip them both inside the `C:\Program Files\Java` folder (same folder as the JDK).

Actually, it is possible to decompress the content of the zip file in any position of your hard disk, but by convention we should prefer the `C:\Program Files\Java` directory, because it is the same one used by the Oracle JDK installer. In particular, the folder containing the JMOD files will eventually be distributed along with the program itself.

L.3 PATH_TO_FX Variable Setting

To use the compiler (`javac`), the interpreter (`java`) with the JavaFX technology, we need to set the `PATH_TO_FX` environment variable, pointing it to the `lib` folder of the Java FX SDK. The procedure is identical to that seen in section L.1. For Windows 10 systems perform the following steps:

1. open the **Control panel**, then click on **System**, then on **Advanced System Settings** in the new window;
2. in the new window that opens, select the **Advanced** tab and click on **Environment variables**;
3. among the **System Variables** (or if you prefer among the **User Variables**) add the `PATH_TO_FX` variable, by clicking on the **New...** button;
4. Enter the string "`PATH_TO_FX` " in the new window as a **variable name**, and as a **variable value** the string "`C:\Program Files\Java\javafx-sdk-12.0.1\lib`".
5. click **OK** on all the open windows, and the configuration of the variable is complete.

Optionally it is also possible to set the `PATH_TO_FX_MODS` variable (which we could use as the value of the `MODULEPATH` variable at runtime), making it point to the JMODS file folder with the usual process:

1. open the control panel, then click on **System**, then on **Advanced System Settings** in the new window;
2. in the new window that opens, select the **Advanced** tab and click on **Environment variables**;

3. among the **System Variables** (or if you prefer among the **User Variables**) add the **PATH_TO_FX_MODS** variable, by clicking on the **New...** button;
4. Enter the string “**PATH_TO_FX_MODS**” in the new window as a **variable name**, and as a **variable value** the string “**C:\Program Files\Java\javafx-jmods-12.0.1**”.
5. click **OK** on the open windows, and the configuration of the variable is complete.

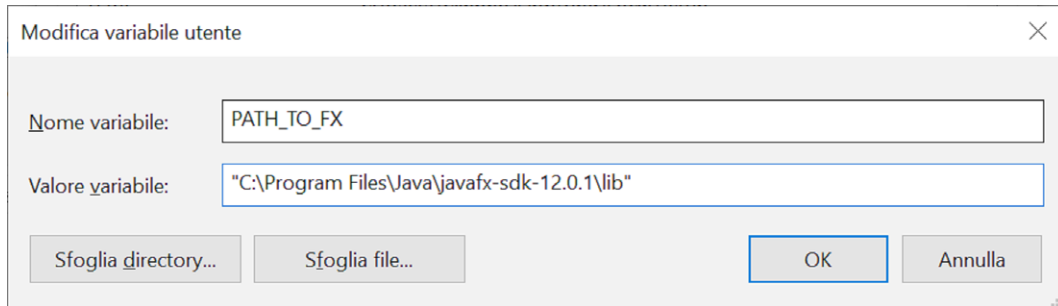


Figure L.3 - PATH_TO_FX variable setting on Windows 10.

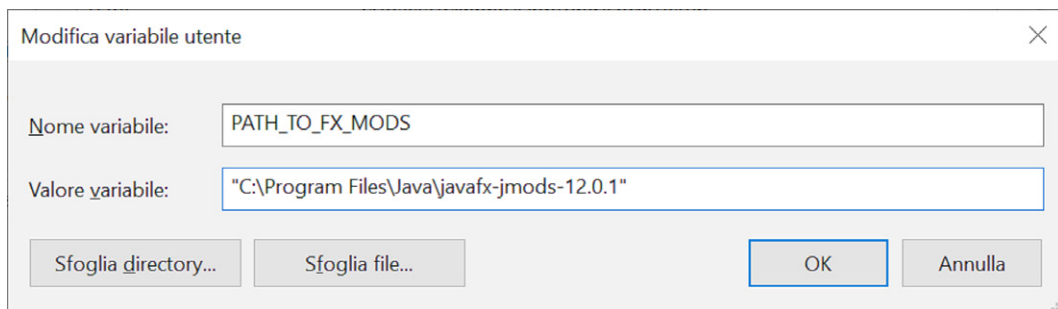


Figure L.4 - Setting the PATH_TO_FX_MODS variable on Windows 10 (Italian version).

L.4 Configuration Check

To verify that everything went well, try to compile this simple file (it is present in the downloadable code of the book if you don't want to copy it by hand):

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class HelloJavaFXWorld extends Application {
```

```
@Override
public void start(Stage stage) {
    Label label = new Label("Hello JavaFX World!");
    label.setFont(new Font("Book Antiqua", 120));
    stage.setScene(new Scene(label));
    stage.setTitle("HelloWorld with JavaFX version " +
        System.getProperty("javafx.version"));
    stage.sizeToScene();
    stage.show();
}

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

with the following command:

```
javac --module-path %PATH_TO_FX% --add-modules javafx.controls HelloJavaFXWorld.java
```

and then launch it with the following command:

```
java --module-path %PATH_TO_FX% --add-modules javafx.controls HelloJavaFXWorld
```

if everything is in place, the interface visible in figure L.5 should be launched.



Figure L.5 - Visualization of the HelloWorldJavaFX program.

Appendice M

EJE (Everyone's Java Editor)

Goals:

At the end of this appendix, the reader should:

- ✓ Be able to install EJE (Units M.1, M.2).
- ✓ Be able to use EJE (Units M.3, M.4).
- ✓ **EJE** (acronym for **Everyone's Java Editor**) is a lightweight and user-friendly editor for the Java programming language. It offers basic features that are perfectly suited to those who must begin to take their first steps with Java programming.

✓



The EJE development started above all to gain experience on graphical user interfaces starting from 2001. Initially it was called “Color Editor” and was written with AWT API. Then, together with the creation of the first Java manual I wrote in 2002 (commissioned by an IT company) it was rewritten with the Swing library. Until 2008 EJE was developed only by itself! Then the project evolved very slowly due to lack of time, just to support new versions of Java. Meanwhile, since it is hosted on SourceForge at the address <http://sourceforge.net/projects/eje> has been downloaded over 80000 times by users from all continents. It is used as an educational editor in many universities around the world. We certainly know that it was used in the universities of Bergen (Sweden), Adelaide (Australia), Beijing (China), Cape Town (South Africa), Buenos Aires (Argentina) and even Carnegie Mellon University (university where James Gosling graduated!), which also made a request for collaboration to improve EJE, which unfortunately did not come true.

M.1 System Requirements

EJE needs some system requirements to run. As for the hardware, the minimum choice must be the following:

- RAM memory, minimum 512 MB (2 GB recommended)
- Hard disk space: approximately 888 KB

Software necessary to run EJE:

- Java platform: Java Development Kit Standard Edition (Open JDK), v1.8.x or higher. Refer to Appendix B to install it.
- Operating system: independent of the operating system. Tested on Microsoft Windows and Linux (different distributions). On Mac OS X the correct functioning was reported by multiple users.

M.2 EJE Installation and Execution

EJE is a multi-platform program, but requires two different scripts to run on Windows or Linux operating systems. Two different execution scripts were created:

- **eje.bat** for Windows systems
- **eje.sh** for Linux (and similar) systems.

M.2.1 For Windows Users

Once the **eje.zip** file is downloaded to your computer:

1. unzip the **eje.zip** file using a decompression utility such as WinRar or WinZip;
2. run the **eje.bat** file (**eje_win9x.bat** for Windows 95/98) with a double click.

M.2.2 For Unix-like Operating Systems (Linux, Solaris, etc.) Users

Once the **eje.zip** file is downloaded to your computer:

1. unzip, via gzip or other utility, the **eje.zip** file. A directory named EJE will be created;
2. change the permissions of the **eje.sh** file with the **chmod** command. From terminal type:

```
chmod a+x eje.sh
```

3. run the **eje.sh** file from the EJE directory or from terminal.

M.2.3 For Users Who Have Problems with These Scripts (Windows 9x/NT/ME/2000/XP & Linux, Solaris)

From the command line (DOS prompt or Unix shell) type the following command:

```
java -classpath . com.cdsc.eje.gui.EJE
```

If the operating system does not recognize the command, it means that you have not installed the Java Development Kit (1.8 or higher), or you have not set the PATH environment variable with the bin directory of the JDK (see Appendix B or “installation notes” of the JDK).

M.3 User Manual

EJE is a lightweight and user-friendly editor for the Java programming language. Many other tools exist that allow to write Java code, but they are often heavyweight IDE that need a relatively long time to learn, for a good usage. Besides, those tools need large system resources, that could be not necessary for your code or could not be present on your system. EJE does not aspire to replace the aforementioned tools, but could help you to write code quickly and in a profitable way. It's been created for who wants to learn the Java language but not a complex IDE, and is perfect as support for this book.

The main features of EJE are the following:

1. can compile and execute files (also with arguments) directly from the editor.
2. Java syntax highlighting.
3. File system fast explorer with a tree panel (this version supports a new enhanced version).
4. Fast file system browsing through directory tree-tree and possibility to organize work-directory as a tree.
5. Full keyboard navigability of all features.
6. Allows to cancel and reconfirm the last action (undo & redo).
7. Search, replace and go-to-line utilities.
8. Dynamic code templates insertions life properties, loops, constructors, types, comments and many others. You can also select some text and use these insertions, the construct will surround the selected text.

- 9.** Look and feel customization.
- 10.** Support line numbering.
- 11.** You can set timers to be notified with specified time and messages.
- 12.** Automatic class introspection pop-up to display members of declared objects.
- 13.** You can open the standard API documentation in an external browser.
- 14.** You can generate documentation of your own Java source files automatically with the *javadoc* utility.
- 15.** Automatic code formatting with C or Java style.
- 16.** Quick navigation between open files.
- 17.** You can set many options: font type, style and size, activate-deactivate introspection pop-up, braces style, java version target and source compilation, enable-disable assertions, enable-disable warnings, language, look & feel (Nimbus style included), enable-disable line numbering, Java Development Kit, Documentation output directory, classpath, etc...
- 18.** Source files can be printed.
- 19.** Internationalization (English, Italian, German and Spanish languages are actually supported).
- 20.** You can choose your preferred theme (Standard, Dark, Dusk, BrighterDusk).
- 21.** Supports for Java version 14!

The EJE user interface is very simple and user-friendly. Figure M.1 gives you a snapshot of an older version of EJE in action. EJE will use by default the English language if executed on a non-Italian, non-Spanish or non-German operating system.

The panel 1 shows the directory tree of your file system. The contents will not be visible if there's no Java source files. Just one click to open Java source file in panel 2. You can create your work-directories in this tree as shortcuts.

The panel 2 shows the current open files.

The panel 3 shows the messages that your processes (as compilation or execution) will send you.

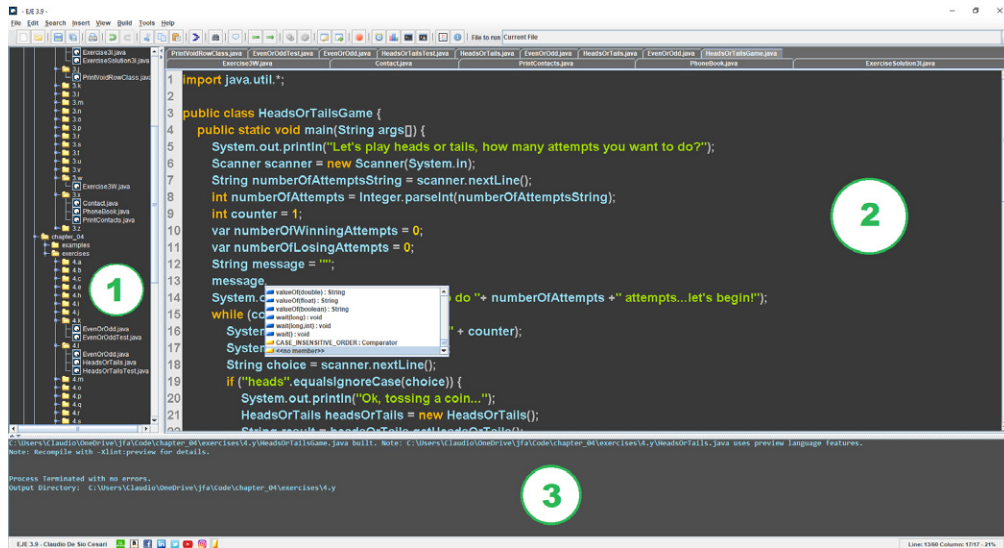




































Figure M.1 - EJE in action.











M.4 Descriptive table of the main EJE commands













Command	Icon	Where Is	Shortcut	Synopsis
New...		File Menu, Toolbar	CTRL-N	Create a new file
Open...		File Menu, Toolbar	CTRL-O	Open a file from file system
Recent Files...		File Menu		Open a recently opened file
Save		File Menu, Toolbar, Text Area Popup	CTRL-S	Save current file
Save All		File Menu, Toolbar	CTRL-SHIFT-S	Save all open files






Save As...		File Menu		Save a file with a different name from the original
Print...		File Menu, Toolbar	CTRL-P	Print the current file
Options		File Menu	F12	Opens the options dialog
Close File		File Menu, Text Area Popup	CTRL-W	Close the current file
Quit		File Menu	CTRL-Q	Quit EJE
Undo		Edit Menu, Toolbar	CTRL-Z	Undo the last action
Redo		Edit Menu, Toolbar	CTRL-Y	Redo the last action
Cut		Edit Menu, Toolbar, Text Area Popup	CTRL-X	Move selection to the clipboard
Copy		Edit Menu, Toolbar, Text Area Popup	CTRL-C	Copy selection to clipboard
Paste		Edit Menu, Toolbar, Text Area Popup	CTRL-V	Paste clipboard
Delete		Edit Menu, Text Area Popup		Delete selection
Select All		Edit Menu, Text Area Popup	CTRL-A	Select all text

To Upper Case		Edit Menu, Text Area Popup	CTRL-U	Uppercase selection text
To Lower Case		Edit Menu, Text Area Popup	CTRL-L	Lowercase selection text
Invert Upper-Lower		Edit Menu, Text Area Popup	CTRL-J	Invert lower and upper case in selected text
Find		Search Menu, Toolbar	CTRL-F	Search for expressions in the text
Find Next		Search Menu	F3	Search for the next expression in the text
Replace		Search Menu	CTRL-H	Search and replace expressions in the text
Go To...		Search Menu	CTRL-G	Move the cursor to the line
Class		Insert Menu	CTRL-0	Inserts (or surrounds selected text with) a class template
Interface		Insert Menu	CTRL-1	Inserts (or surrounds selected text with) an interface template
Enumeration		Insert Menu	CTRL-2	Inserts (or surrounds selected text with) an enumeration template
Annotation		Insert Menu	CTRL-3	Inserts (or surrounds selected text with) an annotation template
Constructor		Insert Menu	CTRL-4	Inserts (or surrounds selected text with) a constructor template

Constructor With Args		Insert Menu	CTRL-5	Inserts (or surrounds selected text with) a constructor with arguments template (opens a wizard for specifying arguments)
Main Method		Insert Menu	CTRL-6	Inserts (or surrounds selected text with) a main() method template
JavaBean Property		Insert Menu	CTRL-7	Opens a wizard to create a JavaBean property
Constant		Insert Menu	CTRL-8	Opens a wizard to create a constant
Singleton		Insert Menu	CTRL-9	Creates the structure for a singleton class
If		Insert Menu	CTRL-F1	Inserts (or surround selected text with) an if construct template
Switch		Insert Menu	CTRL-F2	Inserts (or surround selected text with) a switch construct template
For		Insert Menu	CTRL-F3	Inserts (or surround selected text with) a for construct template
While		Insert Menu	CTRL-F4	Inserts (or surround selected text with) a while construct template
Do While		Insert Menu	CTRL-F5	Inserts (or surround selected text with) a do while construct template

For-Each		Insert Menu	CTRL-F6	Inserts (or surround selected text with) a for-each construct template
Try/catch		Insert Menu	CTRL-F7	Inserts (or surround selected text with) a try/catch construct template
(...)		Insert Menu	CTRL-F8	Inserts (or surround the selected text with) a pair of round brackets
[...]		Insert Menu	CTRL-F9	Inserts (or surrounds the selected text with) a pair of square brackets
{...}		Insert Menu	CTRL-F10	Inserts (or surrounds the selected text with) a pair of braces
System.out.println()		Insert Menu	CTRL-F11	Insert (or surround selected text with) a System.out.println() template
Comment-Out Selection		Insert Menu	CTRL-F12	Insert (or surround selected text with) a comment template
Next File		View Menu, Toolbar	F5	Select the next file
Previous File		View Menu, Toolbar	F4	Select the previous file
Toolbar		View Menu		Hide/show toolbar
Status bar		View Menu		Hide/show status bar
Choose Work Directory		Tools Menu		Allows you to choose a work-directory that will be opened in the panel 1 tree

Build		Build Menu, Toolbar, Text Area Popup	F7	Compile the current file
Build Project		Build Menu, Toolbar	SHIFT-F7	Compile all open files
Execute		Build Menu, Toolbar, Text Area Popup	F9	Runs the current file
Execute With Args		Build Menu	SHIFT-F9	Runs the current file using the specified arguments
Stop Processing...		Build Menu, Toolbar	ESC	Stop the current process
Alarm Clock		Tools Menu		Allows you to set a timeout to show you a message
Show Memory		Tools Menu		Shows the allocated and used memory by EJE
Generate Documentation		Tools Menu		Generates the javadoc documentation of the current file
Format Code		Tools Menu, Toolbar, Text Area Popup	CTRL-SHIFT-F	Format the code
Application Help Contents		Help Menu	F1	Show this user manual
Java Documentation		Help Menu	F2	Shows the documentation of the standard Java library
About...		Help Menu	F11	View information about EJE

Open all Java Files		Tree popup Menu		Open all Java files in the selected directory
Open Window		Tree popup Menu		Open a window in the selected directory
Close Folder		Tree popup Menu		Close the selected directory
Command Line		Build menu, tools menu, tool bar	F8	Open a command line session (only on Windows platforms)
JShell		Tools menu, tool bar	SHIFT-F8	Open a JShell session (only on Windows platforms with JDK version 9 or higher platform)

Appendix N

Easter Eggs, References and Quotes

Goals:

At the end of this appendix the reader should:

- ✓ Understand some quotes, references and Easter eggs included in this text (Unit N.1).

N.1 Creativity

Despite having had a predominantly scientific education, actually I have always been a fan of humanistic arts such as literature, painting, cinema and above all music. In general, I am a creative person, and I get excited about creations in any field, from a top made by crochet, to the realization of a complex enterprise application, from a pencil drawing on a paper-sheet, to the score of a symphony, from the collecting of stamps, to the structure of a literary, theatrical or cinematographic work.

When I combine programming with analysis, design and architecture, I can also *create*. I think it's the same for any programmer, we like this job because we are creatives. Even when I write a book I apply, sometimes unconsciously, analysis, design and architecture techniques, and that's how I finally succeed into completing the work. In my spare time I also write fictional stories, novels, and above all I write and record songs. I try to create something every day, a little piece that one day will perhaps be part of something bigger, and that will somehow survive me. Creativity is an important part of my life, I think many people are not aware of how important it is to be creative.

Within this text, I celebrate creativity (especially of the works, authors and characters that move me) as I have always done in the past, through quotations, hidden meanings, references and Easter eggs. For example, years ago I wrote a free book and called it "OO && Java 5"

(<http://www.claudiodesio.com>). The name of the book, which represents a Java expression, contained a hidden meaning. The term “OO” is the acronym of Object Orientation, and the short-circuit operator && (see Chapter 3) implied that in some way Java 5 should always be used in an object-oriented manner. In practice, we could read the title like this: “if *Object Orientation* is not true, then it is useless to check whether *Java 5* is true”. This book has been downloaded nearly half a million times, but I don’t know how many people caught the message...

Another example is the EJE logo, which can be considered an Easter egg. We can note that the name “EJE” differs from the word “eye” for a single character. The EJE logo in fact, is nothing but an eye, but not just any one, but that of [Homer Simpson](#)! It’s just the result of a screenshot of an image of Homer, to which I changed the color of the outline of Homer’s eye from yellow to blue. For me it was a way to honour the genius of what was at the time one of my favourite television programs.

Figure 1 shows the EJE splash screen, at the center of which, the logo can be seen.



Figure N.1 – EJE Splash screen.

A quote is also hidden in the splash screen. The phrase “take the time” is very suitable for the Java programming context, also because EJE is designed for those who want to start learning the language. Actually, “Take the Time” is also the title of a [song](#) by [Dream Theater](#), one of my favourite bands. It’s not an easy song to listen to, but I don’t like simple things very much. The “refrain” of the song says “you ‘ll find all you need in your mind, if you take the time”, which is like my philosophy for life.

In other words, in the next section, if you are interested, you can find out more about my passions.

N.1.1 References, Easter Eggs and Quotes



I will avoid mentioning all the names, dates, and references to personal items, friends and relatives...

N.1.1.1 Volume 1

- In section 3.3.5, the examples show two sentences in Latin. The first “Java melius semper quam latinam linguam est”, should mean (if I am correct) “Java is always better than the Latin language”, in the sense that it is always better to study Java than Latin. The second sentence “Meum filium maxime amo, sed ille me latinam linguam studere compellit”, means “I love my son very much, but he forces me to study Latin”.
- In section 3.3.5.3 on escape characters, there are examples that print the string “IQ”. Again, this is the name of another amazing progressive music **band**. As usual it is not commercial music, but not as extreme as that of Dream Theater’s “Take the time”.
- In section 3.5.2, the great poet **Salvatore Quasimodo**, one of the most important figures of Italian Hermeticism, and winner of the Nobel Prize for Literature in 1959 is quoted in a code example. At first, instead of the name of the poet I wanted to insert the last verse of the wonderful poem “To the branches of the willows” (“Alle fronde dei salici”), but then I realized that it was not suitable for the purpose of the example.
- In section 3.7.2, the string “Fokus” is named in the first example code box. This is the name of the building of the “Quartieri Spagnoli Foundation” (<http://www.foqusnapoli.it>), a beautiful reality in a difficult neighborhood in Naples, where I had the opportunity to deliver some training courses at the end of 2018 for Oracle, which gave me a lot from a personal point of view. I learned a lot from that experience, and so I wanted somehow to make this memory of mine even more persistent.
- In section 4.1.6, I quote James Gosling, the main architect of Java creation ... a person I owe a lot to!
- In one of the examples in section 4.3.1, an array of strings is created with the names of four of my favourite classical composers: **Antonio Vivaldi**, **Ludwig van Beethoven**, **Johann Sebastian Bach**, **Piotr Tchaikovsky**. In the example **VarForTest.java** you find online, the array is enriched by other names like **Fryderyk Chopin**, **Frank Joseph Haydn**, **Claude Debussy**, **Giacomo Puccini** and **Maurice Ravel**.

- In section 5.1, in the example `GreetingsFromAliens.java`, there is an Easter egg. By default, the program prints the string “Greetings humans!”. This is a quote taken from the opening track of the album “[Ziltoid The Omniscient](#)” by [Devin Townsend](#) (another artist close to extreme progressive music). In fact, at the end of the exercise the output is changed to “Greetings from Ziltoid!”
- In the output of the example in section 6.8.1, there’s the quote of one of the greatest lyric poems ever written: “[L’infinito](#)” by [Giacomo Leopardi](#).
- In section 8.3.7, in the last box of code, the `ControlTower` class declares two methods whose parameter is `Flying v`. Actually, even here there is an Easter egg. In fact, “Flying v”, is also the name of an [electric guitar](#) made by the famous brand [Gibson](#), which I dreamed of having as a boy (today I have several guitars, but still no Gibson guitars).
- Towards the end of section 8.3.3 there is another Easter egg. In fact, in a complex section (for aliens), the string “reH Dunmo ‘law’ tlhIngan” is defined. This is the [Klingon language](#) translation (an alien language invented in the [Star Trek](#) series) of the phrase “always better than the Klingon” ... implying that Java, although difficult to learn, is still preferable to Klingon!

N.1.1.2 Volume 2

- In section 10.2.3 two songs are named: “[The Road of Bones](#)” by the aforementioned IQ, and “[In the Passing Light Of Day](#)” by the awesome band [Pain of Salvation](#).
- In the example in section 13.5.1, the “KK” string is used. It is a tribute to one of my sporting idols, the footballer (and above all the man) [Kalidou Koulibali](#).
- In the example in section 13.5.3.1, the [sentence](#) “It is not logical, but it is often true.”, represents a tribute to the legendary [Dr. Spock](#) character of the [Star Trek](#) series, who utters it in the first episode of the second season.
- In the same section the name “Ligeia” is used twice. It is the name of a [short story](#) that you can find in various collections of another of my favourite authors: [Edgar Allan Poe](#). The story struck me so much that so many years (and lots of hair) ago, I sang in a hard rock band that I called “Lady Ligeia”.
- In section 13.5.3.3 the word “McGuffin” appears, and this time it is a tribute to the genius of one of my favourite directors: [Alfred Hitchcock](#). You can find the definition of a McGuffin’s on [Wikipedia](#).
- In the same section the string “Aomame” is used. This is the unforgettable protagonist of the [trilogy](#) novel “1Q84” by [Haruki Murakami](#), another author that I love.

- In section 13.5.3.4 there is a clear reference to one of the most important bands in the history of rock, and one which has greatly influenced my musical culture: [Deep Purple](#). Also reported are the surnames of the golden age musicians of the Deep Purple, that played on what is considered by many to be, the best live album in the history of rock: [Made in Japan](#). You have probably already heard the “[Smoke on the water](#)” guitar riff.
- It is also difficult not to recognize the first verse of the first *canto* of the [Divine Comedy](#) by [Dante Alighieri](#) in the same section. Apart from the value of a such work of art, I also find the symbolism of the numbers with which it has been structured, irresistible... an absolute masterpiece even from this point of view.
- In the last example of the section, the name [Salvo D'Acquisto](#) is used, a real great hero (read his history).
- In section 17.4.1, there is another tribute to a great author [Michail Bulgàkov](#), and to his [masterpiece novel](#) “The Master and Margarita”.
- In section 18.2.2, a HashSet is instantiated containing the names of my three favourite progressive groups: [Dream Theater](#), [Ayreon](#) and [Pain of Salvation](#).
- The famous band that is described in the example in section 18.5.3 is that of [The Ramones](#), while in section 18.9 the song “Take the Time” by Dream Theater is quoted again.
- The “Divine Comedy” is mentioned again in section 18.9.4.
- The example “Music” database schema explained in section 21.2.4, contains a single table that defines music albums. In addition to the already mentioned “Made in Japan” by Deep Purple, there are three other masterpiece albums of the progressive rock genre: “[Images and Words](#)” by Dream Theater, “[The Human Equation](#)” by Ayreon, and “[Be](#)” by Pain of Salvation.
- In section 21.4.4, a show called “JCS” is mentioned, which is obviously the acronym for “Jesus Christ Superstar”. It is a [rock opera](#) written by [Andrew Lloyd Webber](#) with lyrics by [Tim Rice](#), which since 1970, has continued to be successful in theatres around the world. The 1973 [movie](#) is a true masterpiece, as is, obviously, the [soundtrack](#).
- In section 21.5, other albums that are important to me are added to the “Music” database: “[The Real Thing](#)” by [Faith No More](#), “[Mack the Knife: Ella in Berlin](#)” by [Ella Fitzgerald](#) and “[OK Computer](#)” by [Radiohead](#). Following that, in section 21.5.2, other albums are named as “[I Am the Blues](#)” by [Willie Dixon](#) and “[Tapestry](#)” by [Carole King](#). Then in section 21.6.2.1 the [Steve Vai](#) album “[Passion and Warfare](#)” is added and in section 21.6.2.2 “[London Calling](#)” by [The Clash](#).

- In section 23.5.3, Dream Theater’s “Take the Time” is mentioned for the third time.
- The example in section 23.5.6.5, is entirely dedicated to the brilliant sitcom “[The Big Bang Theory](#)”, of which I am a big fan.
- In the example in section 23.6.2, a very famous sentence from the [novel](#) “The Leopard” (“Il Gattopardo”) by [Giuseppe Tomasi di Lampedusa](#) is cited: “ If we want everything to remain as it is, everything must change.”. In the book the sentence is about politics, but all in all it is also valid if we talk about refactoring...

N.1.2 Exercises References

Also among the exercises there are references:

- In the solution of exercise 1.w, the great musician and my idol [Stevie Wonder](#) are mentioned, and the very young unfortunate icon of the resistance of [the four days of Naples Gennarino Capuozzo](#).
- In exercise 1.z, you are asked to create a class called SayMyName. This is a quote from the TV series [Breaking Bad](#). For me, as for millions of other people, it is a true masterpiece.
- In exercise 2.q, the string “Quelo” is used. In this case it is a tribute to one of the best and most appreciated Italian actors and comedians: [Corrado Guzzanti](#). “Quelo” was the name of a divinity to which one of the most famous characters played by him was devoted (an improbable holy man who sought followers for a new religion).
- In exercise 14.f a string is used that corresponds to the first verse of the song “[Metropolis Part I](#)” by Dream Theater (from the album “Images and Words”).
- In exercise 18.t, the beautiful poem by [Jacques Prévert](#) “Les enfants qui s’aiment” is reported, and in the solution of exercise 18.z its translation into Italian.
- Exercises 17.b and 17.o show the names of some of my early musical instruments (the family has recently expanded).
- The most famous components of the [Ramones](#) are also mentioned in exercise 6.b.
- In exercise 7.x, Mario Ruoppolo, a character played in his latest film “Il Postino” (“[The Postman](#)”) by the great [Massimo Troisi](#) (nominated for an Oscar after his untimely death), Vincenzo Malinconico, mythical character of a series of novels (including “[I hadn’t understood](#)”) by [Diego De Silva](#), and the real name of the Marvel [Wolverine](#) superhero, are mentioned.

- In the solution of exercise 8.z, there are two musical references: [Molly Malone](#), the legendary song by [The Dubliners](#), and [Phil Lynott](#), the late leader of [Thin Lizzy](#). For both these quotes there are two sculptures in the city of Dublin (in Ireland) that I visited in the summer of 2019.
- In the solution of exercise 10.n there is a small easter egg that recalls the novel of “[Fahrenheit 451](#)” by [Ray Bradbury](#).
- In the track of exercise 20.f the names Ross and Phoebe Buffay are used. It is a reference to the cult TV series “[Friends](#)” broadcast from 1994 to 2004.

N.1.3 Cover References

The cover was created by some collaborators, starting from some of my ideas. It’s not the typical IT book cover, it was conceived and designed for months! You can see “among the stars” various references and easter eggs.

N.1.3.1 Volume 1

- On the front cover, at the top left above the “J” of the title, you can see an oak leaf, this is an Easter egg. If you read Appendix A on the history of Java, you already know that the first name given to the Java language was “Oak”.
- Three constellations can be seen: the “W” of [Cassiopeia](#) near the subtitle, and near the left arm of the model there is the [constellation of Leo](#), while near the right arm there is the [constellation of Ursa Major](#). These three constellations have meanings that only people close to me can understand.
- Below the right elbow of the model is the Lambda symbol, which recalls the Java lambda expressions.
- Next to the right elbow, however, there is a “bug” with a “J” of Java on the back.
- There is another reference to the Java language on the near the head of the model. The word “CAFEBABE”, is present at the beginning of each compiled Java file (in the bytecode). Just open any `.class` file, with a hex editor to check. It is in fact a hexadecimal number, which identifies the `.class` format. You can get more information at this [address](#) on Wikipedia. In effect, the writing on the cover is an Easter egg of an Easter egg!
- Finally, the binary number “01011001” (which is equivalent to the letter “Y” in Unicode decoding), is an Easter egg for another fantastic Ayreon [album](#), whose title is “01011001”.

- On the back of the cover, at the top left there is another reference to my passion for music: the treble clef.
- At the top right there are the icon of two gears, which symbolize complexity (it is the same icon used in the book that characterizes complex topics).
- In the lower left corner, you can recognize the [Millennium Falcon](#), the famous spaceship from the [Star Wars](#) saga. Even the description of the book recalls the [titles that introduce the films of the saga](#), being inclined they should give the idea of sliding downwards.
- At the bottom right there is a moon, and also this symbol has a meaning that only a person close to me can understand.
- Finally, under the description there is an icon that should symbolize the Factory design pattern.
- Curiosity: the “alien” model on the cover of Volume I is my second son Simone.

N.1.3.2 Volume 2

- On the front cover at the top left above the “J” of the title, the atom symbol can be seen. It is another tribute to one of my favorite TV series: [The Big Bang Theory](#).
- Under the title on the right, there is a sun, which represents a double easter egg: it recalls the logo of an album special for me like “[The passing light of day](#)” by Pain of Salvation, and recalls the [Sun Microsystems](#), the company that created Java, and for which I worked many years at the beginning of my career.
- To the left of the model’s head, there is a word written with the octal system, which can only understand who is near me.
- On the right of the model’s head, on the other hand, there is the name of the Dream Theater album “Images and Words” (see section N.1.1), written with the base64 encoding!
- Near the left shoulder of the model there is the [constellation of the Cygnus](#), while near the right arm there is the [constellation of the Canis Mayor](#), also in this case, the meaning is understandable only by people close to me.
- Under the model right elbow there is the short circuit operator “&&”. This is an Easter egg that refers to the book that many years ago I wrote and uploaded on my website www.claudiodesio.com: “Object Oriented && Java 5” (which I have already mentioned in paragraph N.1).

- In the lower right corner, there is a padlock with a Java “J”, which should represent the encapsulation paradigm.
- On the back of the cover, at the top left, there is a key that should represent the key to open the encapsulation padlock.
- At the top right, there is the stylized little man who represents an actor in the UML use case diagram.
- On the bottom left and right of the description there are two icons used in the book to label the very complex parts (alien icon), and the rarely used topics (the dinosaur).
- Below the description, it says Java for Aliens using the hexadecimal system!
- Curiosity: the “alien” model on the cover of Volume II is my firstborn Andrea.

N.1.4 References in the analytical index

- There is a single Easter egg under “recursion” of the Volume 2 index.

Appendix O

Bibliography

Goals:

At the end of this appendix, the reader should:

- ✓ Be able to consult a list of books, sites and articles that the author deemed useful for writing this book (Unit O.1).
- ✓ Be able to consult a list of books, sites and articles that the author recommends reading, in order to continue training in Java with a view to specializing in Java technologies (Unit O.2).
- ✓ Be able to consult a list of books, sites and articles that the author recommends reading in order to continue training in Java with a view to specializing in topics relating to Object Orientation (Unit O.3).

Below is a list of books, sites or articles that may interest the reader who wants to continue his training. We have divided the bibliography into three sections.

1. Section O.1 lists all the books that were used to write this book.
2. In section O.2, on the other hand, some resources are recommended with a view to supporting further training aimed at the specialization of the most requested Java technologies.
3. Finally, in section O.3, several books on Object Orientation are listed in order to support further training aimed at the specialization of Java technologies.

Almost all references are in English.

0.1 Resources for Java

Below is a series of books and links, which were useful for the preparation of this book:

0.1.1 Books

- Cay S. Horstman, “Core Java Eleventh Edition Volume I & II”, Pearson (2019): is a book now in its eleventh edition which has been a best seller for many years, and has sold millions of copies. Horstman is a very famous and highly respected author.
- Paul Deitel, Harvey Deitel “Java 9 for Programmers”, Prentice Hall (2017): A complete book on Java 9. One of the best books ever. Deitel & Deitel are always very clear and precise.
- Sander Mak, Paul Bakker “Java 9 Modularity”, O’Reilly (2017): Very well done, impeccably explains all the aspects of modularization in over three hundred pages.
- Josh Juneau “Java 9 Recipes”, APress (2017): An introduction to Java 9 based on examples. Great for people who already know Java.
- Kishori Sharan “Beginning Java 9 Fundamentals”, APress (2017): An excellent introductory book on Java 9.
- Kishori Sharan “Java 9 Revealed, for Early Adoption and Migration”, APress (2017): A great book but only for the latest Java 9.
- Joshua Bloch, “Effective Java, Third Edition”, Addison-Wesley (2018): One of the best books on Java advanced by one of the historical Java developers. A highly anticipated new edition after ten years.
- Cay S. Horstman, “Java SE 8 for the Really Impatient”, Addison-Wesley (2014): An advanced and synthetic book, which only features news on Java 8. This book was published a few months before the official release of the Java 8 release.
- Jeanne Boyarsky and Scott Selikoff, “OCA, Oracle Certified Associate Java SE 8 programmer I”, Sybex (2015): a text with excellent explanations on the most complex situations to manage, that can occur when developing in Java. Very good for passing the OCA certification exam on Java 8.
- Jeanne Boyarsky and Scott Selikoff, “OCP, Oracle Certified Professional Java SE 8 programmer II”, Sybex (2016): still a great text to pass the OCP certification exam on Java 8, where more advanced topics and libraries are addressed.

- Khalid A. Mughal and Rolf W. Rasmussen, “A Programmer’s Guide to Java SE 8 Oracle Certified Associate (OCA)”, Addison Wesley (2017): I didn’t need to read the whole book, but for what little I read, I found it very well done. Some of the most complex parts of this book inspired me.
- Mala Gupta, “OCA, Java SE 7 Programmer I Certification Guide”, Manning (2013): A good book to prepare for the Oracle Java 7 Programmer I certification.
- S. G. Ganesh and Tushar Sharma, “Oracle Certified Professional Java SE 7 Programmer Exams 1z0-804 and 1z0-805”, APress (2013): Another good book on Java certification, which also covers the Programmer II exam.
- Katie Sierra and Bert Bates, “OCP Java SE 7 Programmer Study Guide”, Oracle Press (2013): like the previous one, it covers both exams Programmer I & II (OCA & OCP). By studying this book (and especially by practicing their tests) you have an excellent chance to pass the exams.
- Philip Heller and Simon Roberts, “The Complete Java Certification guide”, Sybex (2006): This book is very dated, but it was very important for understanding some basic features of Java.
- Robert C. Martin, “Clean Code: A Handbook of Agile Software Craftsmanship”, Prentice Hall (2009): Known as “Uncle Bob”, the author, Robert C. Martin, is an important exponent of the Agile movement and, in this book, he explains the “Clean Code” technique for optimal programming.
- Doug Lea, “Concurrent Programming in Java: Design Principles and Pattern (2nd Edition)”, Addison Wesley Publishing (1999): This book is very old, but the author was the main architect of the original competition architecture in Java. Recommended for those who are fond of concurrent programming.
- Brian Goetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, Doug Lea “Java Concurrency in Practice”, Addison Wesley Publishing (2006): A practical guide on threads accessible even to newbies, but without the latest news from the library.
- Claudio De Sio Cesari, “Manuale di Java 8”, Hoepli Editore (2014) [Italian language]: Compared to “Java 9 Manual”, the appendices on JDBC and Java FX were the last two chapters of the book. It is currently the most reviewed programming book on Amazon.it, and certainly one of the best sellers in Italy.

- Claudio De Sio Cesari, “Manuale di Java 7”, Hoepli Editore (2011) [Italian language]: Organized differently than “Java 8 Manual”. The final part was dedicated to investigating some topics. This allowed the neophytes to approach the language more gradually. This structure was also used with “Java 6 Manual”. With Java 8, a structure with a steeper learning curve was chosen, as the language has been enriched with many new features.
- Claudio De Sio Cesari, “Manuale di Java 6”, Hoepli Editore (2006) [Italian language]: The first book in the series. The last book has the same structure as the first four chapters.
- Claudio De Sio Cesari, “Object Oriented & Java 5” (2005) [Italian language]: You can download this book for free from the author’s personal website: <http://www.claudiodesio.com>. It has been downloaded over 450,000 times, and consists of 721 pages, and is also the basis of all the manuals published later. It is a complete manual, but updated for Version 5.

0.1.2 Online Resources

- <http://openjdk.java.net>: Every new study started from this address. Version by version I studied all the news that are discussed there.
- <http://www.azul.com>: as for the previous link, the Azul site contains many new features of each version, and has been very valuable.
- <http://docs.oracle.com/javase/specs/>: The official specifications of the language, to remove any doubt.
- <http://docs.oracle.com/javase/9/docs/api/overview-summary.html>: The official documentation for Java 9.
- <http://docs.oracle.com/javase/tutorial/>: The fundamental Oracle tutorials.
- <http://docs.oracle.com/javase/8/javase-clienttechnologies.htm>: Tutorials on JavaFX, simple and full of information.
- <http://www.claudiodesio.com>: The author’s personal website. Unfortunately, it is not updated often ... but sooner or later, it will feature some surprises. From it I have extracted some appendices of this book.
- javarevisited.blogspot.com: blog full of interesting posts, short and with excellent examples.
- <http://www.javacodegeeks.com>: so many interesting articles on Java.
- <https://blog.codefx.org>: Nicolai Parlog’s blog is always very updated to the latest news, precious.

0.2 Resources for Java Technologies

Now that we know the Java language, it's time to start exploring the technologies that populate the Java universe. Today, the most important Java-based technologies are certainly Java Enterprise Edition and the development on Google Android platform which, even though it's not an official Oracle technology, still uses Java as a programming language (although it is possible to use other languages to program on this platform). Here is a list of interesting resources:

0.2.1 Resources for Java EE

- Antonio Goncalves, “Beginning Java EE, Third Edition”, Apress (2013): a great introduction to Java Enterprise Edition version 7. Although the version is not the last one, I think it's the best book to start with.
- Arun Gupta, “Java EE 7 Essentials”, O'Reilly (2013): an excellent alternative to the book by Antonio Goncalves, by a great developer like Arun Gupta.
- Ram Kulkarni, “Java EE 8 Development with Eclipse”, Packt (2018): a short and practical book on Java Enterprise Edition version 8.
- David Heffelfinger, “Java EE 8 Application Development”, Packt (2017): a quick overview of Java Enterprise Edition version 8. It already requires knowledge of the platform, not recommended for inexperienced developers.
- Kamalmeet Singh, Mert Caliskan, “Java EE 8 Microservices”, Packt (2018): covers interesting topics such as microservices, Docker and the cloud in general, with introduction to the latest version of Java Enterprise Edition.
- Craig Walls, Ryan Breidenbach, “Spring in Action”, Manning (2014): One of the best-selling books for learning the Spring framework.
- Don Brown, Chad Michael Davis, Scott Stanlick, “Struts 2 in action”, Manning (2008): One of the best-selling books for learning Struts 2.
- <http://spring.io/docs>: Official documentation for the Spring framework.
- <http://www.javaworld.com>: Many interesting articles for all technologies.
- <http://www.theserverside.com>: Many interesting articles for enterprise technologies.

0.2.2 Resources for Android programming

- Dawn Griffiths, “Head First Android Development”, Apress (2017): the perfect book to start studying programming for Android.

- Grant Allen, “Beginning Android”, APress (2014): Great introductory book on Java programming for Android.
- Dave MacLean, Satya Komatineni “Pro Android”, APress (2014): Book on Android, well-enough written but not always impeccable.
- Vladimir Silva “Pro Android Games”, APress (2014): Explains how to create games with Android.

0.3 Resources for Object Orientation

- These books are not for programming but talk about Object-Oriented or UML methodologies. Some of them focus on areas of pure philosophy, so we are talking about very demanding subjects, and relatively far from programming.
- Martin Fowler, “UML Distilled”, Addison-Wesley (2010): An indispensable, concise, practical book full of references and very direct.
- Ivar Jacobson, Grady Booch, James Rumbaugh “The Unified Software Development Process” Addison-Wesley Publishing (1999): How UP works, an illuminating but also dispersive book.
- Ivar Jacobson, Grady Booch, James Rumbaugh “The Unified Modeling Language Reference Manual” Addison-Wesley Publishing (2010): UML from the source.
- Ivar Jacobson, Grady Booch, James Rumbaugh “The Unified Modeling Language User Guide” Addison-Wesley Publishing (1999): As above.
- Simon Bennet, John Skelton, Ken Lunn “Introduction to UML” McGraw-Hill - Schaum’s (2010): Good book with a meaningful case study.
- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, “Design Patterns” Addison-Wesley Publishing (2002): A classic, but interesting, above all from the historical point of view. For collectors.
- Mark Grand, “Patterns in Java”, Wiley & Sons (2002): Essential for those who are fond of patterns.
- Robert C. Martin, “Clean Architecture: A Craftsman’s Guide To Software Structure and Design”, Prentice Hall (2018): Uncle Bob’s last spectacular book, always original and interesting.
- <http://hillside.net/patterns>: Official portal of the pattern community.

- Meyer Bertrand, “Object-Oriented Software Construction” Prentice Hall (1997): A little dated, but absolutely enlightening.
- Alistair Cockburn, “Responsibility Based Model”, (RBM) <http://alistair.cockburn.us>.
- Kent Beck, “Extreme Programming, an introduction” <http://www.extremeprogramming.org>: An innovative and original methodology. Kent Beck, is one of the founders of the Agile movement.
- Jeff Sutherland, “Scrum: The Art of Doing Twice the Work in Half the Time” Random House Business (2015): a good compendium on the most famous agile methodology by one of its authors.
- Martin Fowler, “Refactoring”, Addison-Wesley (1999): A very useful book on refactoring.
- Craig Larman, “Applying UML and Patterns”, Prentice Hall (2008): The best book ever read!

