elektor books

Getting Started With Java Using Eclipse

Mastering the Language and the Development Platform

Bernhard Steppan

elektor
design > share > earn

[Bernhard Steppan]

[Getting Started With Java Using Eclipse]

[Mastering the Language and the Development Platform]

[Bernhard Steppan]

[Getting Started With Java Using Eclipse]

[Mastering the Language and the Development Platform]

This is an Elektor Publication. Elektor is the media brand of
Elektor International Media B.V.
PO Box 11, NL-6144-ZG, Susteren, The Netherlands.
Telephone: +31 46 4389444

British Cataloguing in Publication Data
A catalogue record for this book is available at the British library.

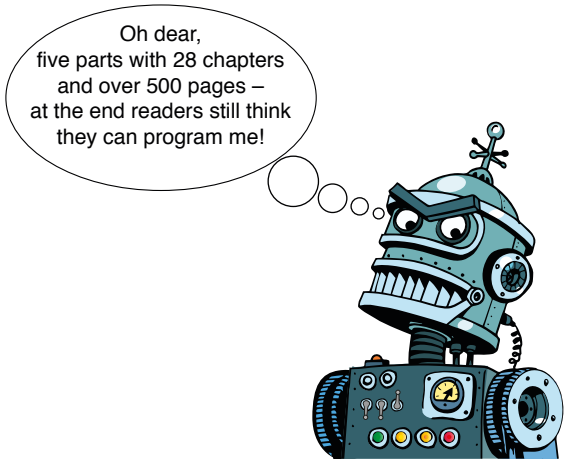
Print: ISBN 978-3-89576-561-2
E-book: ISBN 978-3-89576-562-9

Prepress Production: Bernard Steppan
Editors: Alina Neacsu; Jan Buiting

Contents

Preface

Java is currently undisputedly the most important programming language. Therefore, many would like to learn Java. Unfortunately, getting started is not easy, because programming with Java requires at least two things: mastering the programming language *and* mastering a development environment. This is the reason why this book was written. With the help of many examples, it shows how the language is structured. In addition, the book uses the example of the Eclipse development environment to teach you how to develop Java programs with this tool.



Robert from the machine world accompanies you through the book.

The first part »Basics« gives you the Java and Eclipse basic knowledge. This part lays the programming foundations, gives you an overview of Java technology, and shows you what is special about object-oriented programming. A chapter about the Eclipse development environment completes this part.

In the second part »Java Language« everything revolves around the subtleties of the Java language. This is where the first small Java applications are created. This part offers a mixture of knowledge part and practical exercises. At the end of each chapter, you will find tasks that you can do on your own. With the solutions to the tasks at the end of this book you check the learning success.

Java technology is the focus of the third part »Java Technology«. Additionally, this part introduces you to the rules you must observe when programming, what class libraries are and what advantages they have. In addition, you will learn how to test programs, what algorithms are and how to program them.

A larger Java project is the focus of the fourth part. Here you will apply all the elements from the previous parts on an application with a graphical user interface. The project shows how to develop a larger application piece by piece with the Eclipse development environment.

The fifth part, »Appendix«, concludes this book with solutions to the tasks, with basics of information processing and a chapter on the most common mistakes that can occur when working with Eclipse.

The Plot

As a plot, I based the book on the (fictional) programming course »Java with Eclipse« taught by Professor Roth to four students. The programming course is accompanied by the robot named »Robert« and – among many others – mainly by these five characters throughout the book:

The programming course with Lukas, Anna, Professor Roth, Julia and Florian

Who is the book for?

This book is aimed at active readers. You don't want ready-made solutions, you want to program yourself. Without actively programming yourself and staying on the ball until your self-written program runs, you will not learn Java. The book contains a tempting number of ready programmed examples that you can run at the click of a button. Only reach for the sample solutions if you get stuck. First, try to enter the programs yourself and learn from the mistakes. Only by actively programming will you master Java and the Eclipse development environment.

Bonus Material

The book contains plenty of examples that can be easily imported into the Eclipse environment as solutions. You can easily download them from the Elektor publishing company homepage www.elektor.com/books/programming. Among these downloads, you will also find a bonus chapter, which is not printed due to space limitations. It explains the programming of so-called terminal programs.

Font Conventions

Various parts of the text are highlighted as follows for better readability:

Text Part	Meaning
Data types in body text	*Person*
Data types in headlines	»Person«
Keywords in body text	*implements*
Keywords in headlines	»implements«
Variables in body text	*roland*
Variables in headlines	»roland«
Window (graphical user interface)	ECLIPSE IDE LAUNCHER
GUI element (graphical user interface)	FINISH
Menu (graphical user interface)	FILE
Menu command (graphical user interface)	MENU → FILE → NEW → JAVA PROJECT
Files	*Samples.zip*
Directory paths	*C:/Programs/Eclipse*

Listing (source code from sample programs)	`1 package programmingcourse;`
	`2 public class Robot {`
	`3 (...)`
	`4 }`

Program output	`Result = true`
URL	*http://eclipse.org*
`(...)`	Due to space limitations part of the source code is missing

Acknowledgements

I would like to thank everyone who supported me in writing this book: the Elektor publishing company and my editor Ferdinand te Walvaart for his trust in my work and his great patience. I would like to thank the Hanser publishing company who has given permission to translate the original German manuscript.

As always, my wife Christiane has been very supportive of this project. Many thanks for your help! I would also like to thank Alina Neacsu, who cleaned my book manuscript of spelling mistakes. Many thanks also to Valeriy Kachaev (Studiostoks), from whom the templates of the robot cartoons come.

How to Contact Us

Despite the greatest care, it is not always possible to avoid overlooking one or another error in a book. If you find errors, have suggestions for improvement, or questions, just send me an e-mail at *java-eclipse@steppan.net*. I will answer your questions as soon as possible and try to include your suggestions for improvement in upcoming editions. You can find the most recent additions and further information at *http://www.programmingcourse.net*. Now I hope you enjoy reading and developing your Java programs with Eclipse!

Bernhard Steppan Wiesbaden, March 2023

PART I

Basics

To be able to develop computer programs, you need to master the basics. Chapter »Programming Basics« lays the groundwork for programming Java applications. In chapter 2, »Technology Overview«, you will learn what Java has in common with other programming languages and how Java differs from other languages.

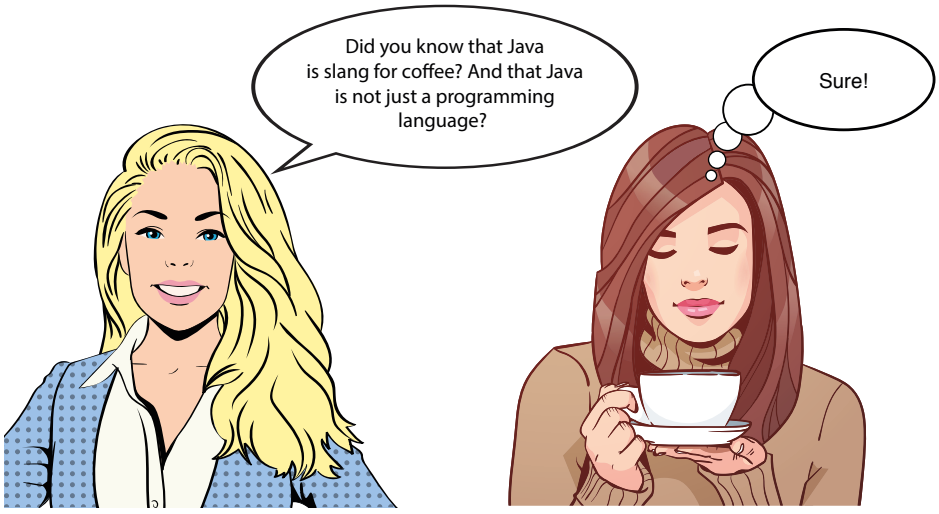


Figure 1: To develop computer programs, you need to master the basics.

Afterwards, it continues with the chapter »Object-Oriented Programming«. It shows what is special about object-oriented programming and how object-oriented programs are structured. The chapter »Development Environment« concludes this part of the book with the installation of the development environment and introduction to »Eclipse«.

1 Programming Basics

1.1 Introduction

Programming means writing computer programs. Computer programs consist of one or more commands in a programming language. The robot named Robert presents a simple Java program to the students in Professor Roth's programming course (Figure 1.1).

```
class Hello {
   public static void main(String[] args) {
      System.out.print("Hello!");
  }
}
```

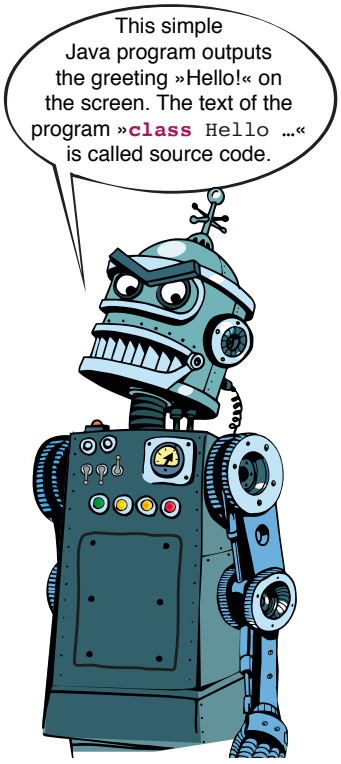


Figure 1.1: Robert from the machine world is *the* expert for machine programs.

The students of the programming course think that there are a lot of instructions for such a simple program. Anna would like to know from Professor Roth whether it could be simpler:

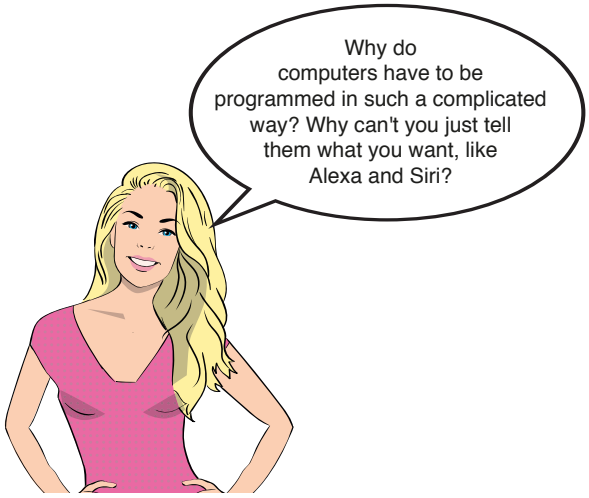


Figure 1.2: Is it really still appropriate today to type in programs?

»Even Alexa, Cortana and Siri«, says Professor Roth, »are just computer programs.« These programs were developed so that humans can control computers via speech. But Alexa & Co. can only do the few tasks for which they were specially programmed. If you want the computer to perform other tasks, such as word processing, you have to write a special program for it. These programs can be developed in Java, for example.

1.2 The Language of the Machine World

When we speak of computers today, we always mean *digital* computers. These machines understand only their digital machine language. Digital means that the computer uses binary code for all information. Therefore, the computer's machine language consists only of a sequence of zeros and ones.

However, the computer's machine language, with its sequences of zeros and ones, is extremely difficult for humans to understand. To program computers directly in machine code would therefore be completely absurd. It would take a very long time and the probability of errors would be high.

If you want to program the computer close to the machine, you use an auxiliary language. This auxiliary language is called assembler language or assembler for short. Professor Roth presents a simple example to his programming class. Like the Java program before, it simply outputs the character string »Hello!« on the screen (Figure 1.3).

Professor Roth's programming course finds the assembly language program quite difficult to understand. How could only programmers learn such a terrible language? The answer is simple, because, in the early days of computers there were no high-level languages like

Java. Programmers had to pay close attention to the computer's processors when they programmed the machine in assembly language.

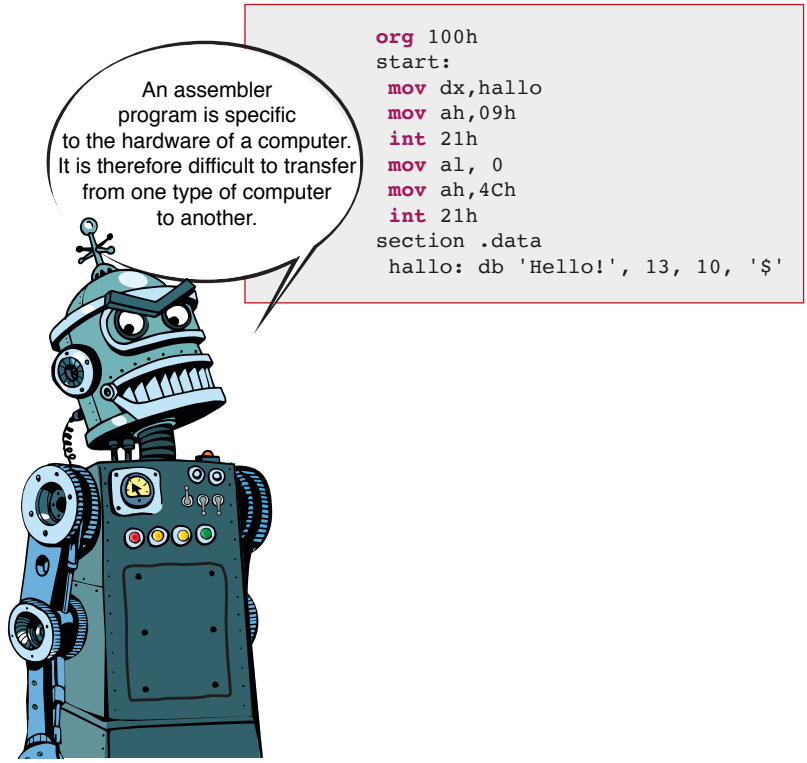


Figure 1.3: This assembler program also outputs »Hello!«.

Assembler programs are much longer compared to functionally equivalent Java programs. They consist of many small-part commands which, taken alone, do little. Therefore, one needs many of these commands to write a larger program. This program is written specifically for one type of computer. It is difficult to transfer to another type of computer.

Besides the high cost of developing such programs, a major disadvantage of the assembly language is that it is difficult to transfer from one type of computer to another. However, the small-part instructions do not have only disadvantages. They have the advantage that a good programmer can use them to create very lean and fast machine programs. They also often require far less main memory than comparable programs written in a high-level language.

1.3 High-Level Programming Languages

It seems to be somehow bewitched: Computers only understand their specific machine language. We, on the other hand, without special programming training, understand only

our native language and maybe one or two foreign languages. How can we bridge this huge gap between the machine world and the human world?

We can either develop even more powerful programs like Alexa, Cortana and Siri, so that the computers execute everything we want. Or we can learn the computer's language if we want to develop special programs for tasks that Alexa & Co. can't do – no, those aren't the only options, because there is, fortunately, a third way.

Programming a complex program in assembler is no longer up-to-date. That's why people started very early to develop programming languages like Java. These languages form a bridge between the (for most humans) difficult to understand machine language and the (for most machines) difficult to understand human language. These languages are called high-level programming languages or high-level languages for short.

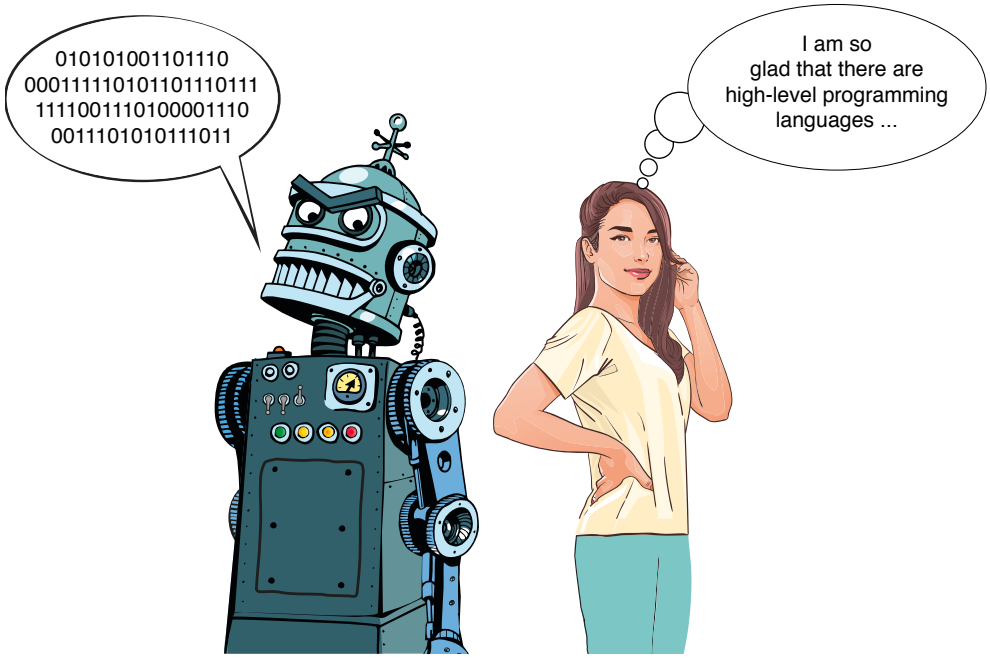


Figure 1.4: High-level programming languages are mediators between humans and machines.

High-level programming languages are much easier for a human to learn and understand than the language of the machine world. But how does it work? How do you translate a high-level language into the language of the machine world? For this purpose, a trick has been thought of. This trick is a special program that translates the source code of a high-level language like Java into the language of the machine world. This program is called a compiler and is part of a development environment.

1.4 Development Environment

1.4.1 Compiler

The compiler is one of the core components of a development environment like Eclipse. It transfers the source code of a Java program into the language of the machine world. The source code is the text that was seen in figures 1.1 and 1.3.

1.4.2 Editor

In the editor you enter the source code of a program as in a word processor. An editor also provides program development support, such as advice on how to fix the errors that are displayed.

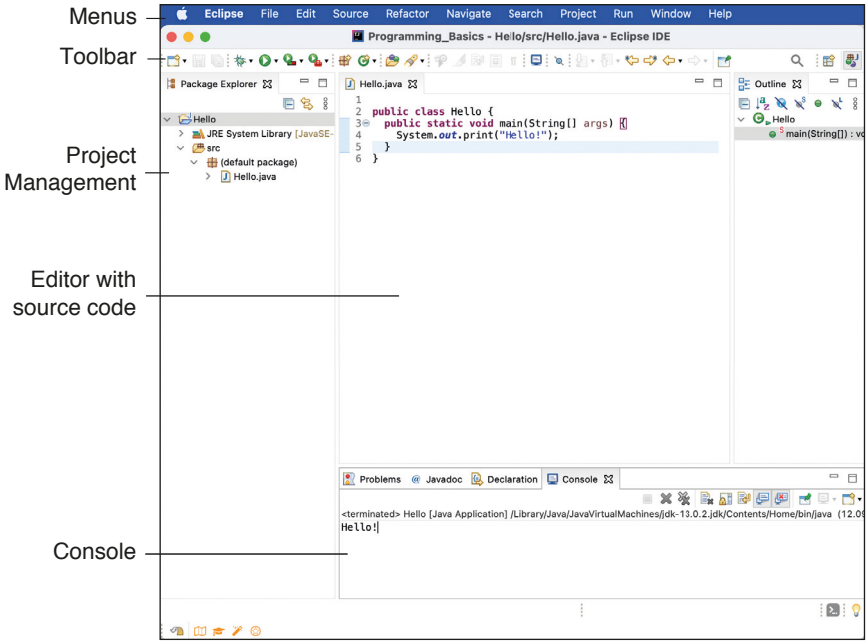


Figure 1.5: Editor, compiler and project management of the Eclipse development environment.

1.4.3 Project Management

Java programs usually consist of a large number of files. So that you do not lose the overview, the Java development environment has a project management. It shows which files belong to a project.

1.5 Runtime Environment

Java programs require a special runtime environment. In other words: Java programs only run with an additional program on your computer. You don't notice this at first, because the Eclipse development environment calls this runtime environment in the background when you run a Java program. To understand what this runtime environment is all about, turn to the next chapter. It shows you how Java has evolved and why a Java runtime environment is necessary at all.

1.6 Summary

Programming means writing computer programs. Computer programs consist of one or more commands in a programming language. These commands are written in the form of a text. In programming, this text is called source code. Computers expect the commands in machine language. We, on the other hand, speak in our human language. To bridge this gap, computer scientists have developed high-level languages. Java is one of these high-level languages.

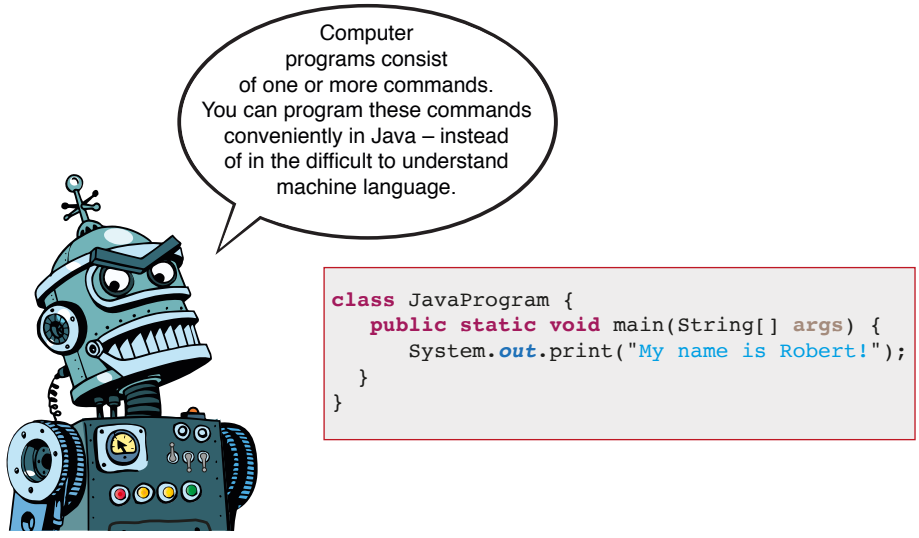


Figure 1.6: Computer programs consist of commands of a programming language.

To translate a high-level language program into machine language so that the computer can execute it, you need an additional program. This translation program is called a compiler. The compiler is part of the development environment. This consists (among other things) still of an editor and project management. With the help of the editor, you write the source code of a program. The project administration administers the different files, which belong to a project.

1.7 Literature

Bernhard Steppan: »A Brief History of Programming Languages«;
https://www.programmingcourse.net/articles/a_brief_history_of_programming_languages

1.8 Exercises

To deepen your knowledge, please go to *https://www.programmingcourse.net/books/java_with_eclipse/programming_basics* and work on the exercises listed there. There you will also find the solutions to the exercises.

2 Technology Overview

2.1 Introduction

Java is more than a successful programming language. Java is a technology. This chapter gives you an overview of the components of this technology and shows you what makes Java stand out from other programming languages and technologies.

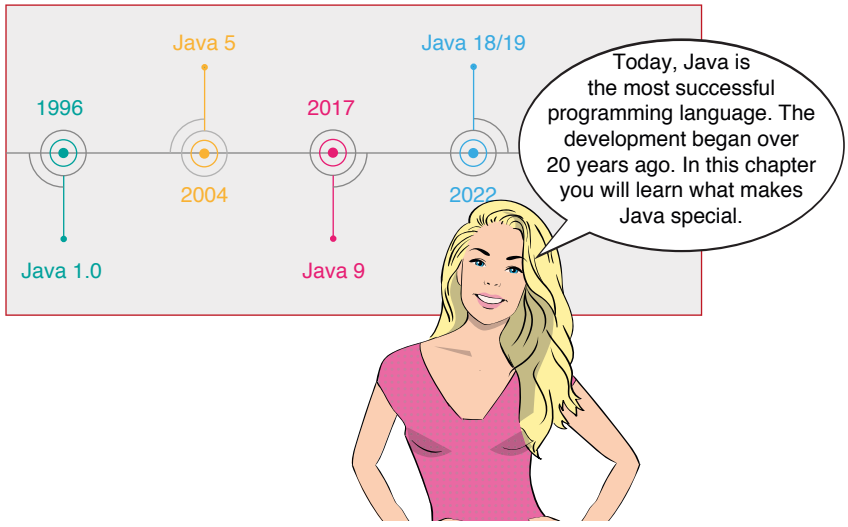


Figure 2.1: Anna gives you an overview of the Java technology.

2.2 Overview

2.2.1 The Early Days of Java

The history of Java began when some programmers from the Californian company Sun Microsystems were asked to develop an object-oriented programming language called Oak (Object Application Kernel) for programming household appliances in 1991. Household appliances are mostly not very powerful. Therefore, Oak programs should be compact so that they can run quickly. Since home appliances use different software and hardware, Oak programs should be as easy as possible to transfer from one appliance to another. It turned out that these requirements were also an ideal fit for implementing Internet programs. Without further ado, the Oak team was given the task of developing the first Internet browser for Oak programs.

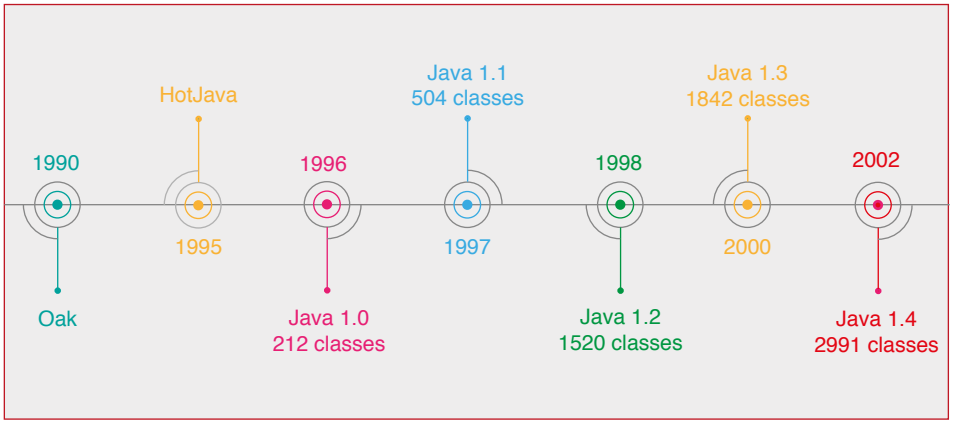


Figure 2.2: The early days of Java.

About a year later, the new browser could display small programs (applets) on HTML pages. It saw the light of day as »HotJava« with the programming language Java[1] in 1995. Java 1.0 followed in 1996 and was replaced by Java 1.1 in the same year. Java programs were still very slow at that time. In addition, programming graphical user interfaces (GUIs) was difficult at the beginning. This only ended with Java 1.2. This version brought a new GUI class library called Swing. The increased performance of Java was also expressed in the size of the Java class libraries. The number of classes increased explosively to over 1500 classes – Java became a technology.

In 2000, Java 1.3 appeared with the so-called hotspot optimization. It translated frequently used parts of a program (hotspots) directly into native machine code. This led to Java programs running significantly faster for the first time. The successor Java 1.4 brought some language extensions and the introduction of Java Webstart. Java Webstart updates a Java program automatically at program start, provided that updates are available for this program.

[1] Java is a slang expression for coffee. According to an anecdote, the team that developed Java named the new programming language after a type of coffee.

2.2.2 The Growth Period of Java

With the successor Java 5, the extent of the class libraries did not increase as strongly as with Java 1.4. The changes to the programming language had it however in itself: Java 5 was characterized by so many serious changes, as there were before only with Java 1.2. Sun Microsystems has thoroughly improved the Java language with version 5. In addition, the Java inventor introduced the product number – the leading one was omitted. Among other things, the product number is supposed to indicate the maturity and stability of the Java version. Internally, Java 5 continues to be referred to as version 1.5. Also all following Java versions carry internally still the old version designations.

Java 6 in the year 2006 brought above all again a further improvement of the execution speed. A further caesura was that Sun Microsystems split off the later OpenJDK from this version (JDK = Java Development Kit).

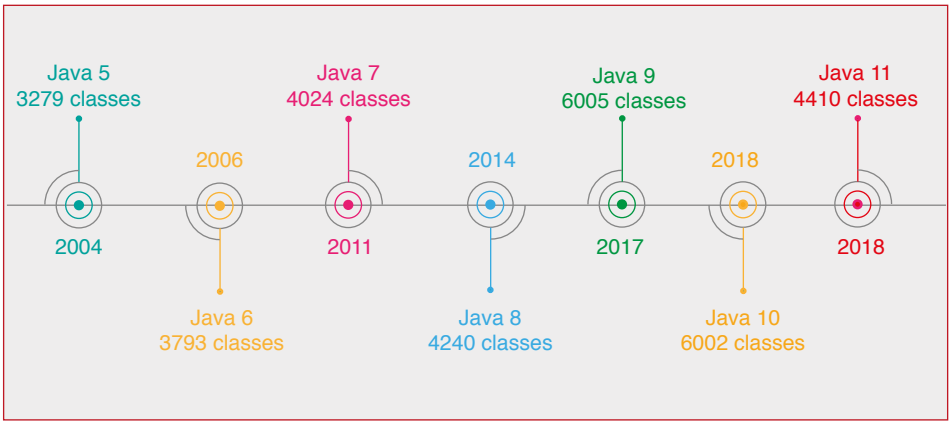


Figure 2.3 The growth period of Java

In 2010 Oracle took over the Java inventor Sun Microsystems. This brought a lot of turmoil to the Californian company, stalling the progress of Java development. It was not until the summer of 2011 that the Java developers reported back with the new version 7. One of the biggest innovations of this version was the integration of the GUI class library JavaFX. Sun Microsystems introduced this class library earlier as an alternative to the GUI class library Swing. The integration of the new library and other classes resulted in the number of classes shipped with Java breaking the sound barrier of 4000 for the first time.

With Java 8 came further language improvements and the so-called LTS version. LTS stands for »Long Term Support«. This is a version with support from the Java manufacturer for a longer period of time. This is especially important for companies that use Java and need support in case of errors for their business-critical Java programs. Another innovation in Java 8 was the integration of a JavaScript Runtime Environment. This made it possible to execute JavaScript directly in Java applications.

The successor Java 9 appeared with considerable delays in 2017. The reason was long-lasting discussions about the new module system with the code name Jigsaw. The module system should allow the same classes to be integrated in different versions in a Java application. Java Webstart was marked as »obsolete« in this version. A class or function that was

marked as »obsolete« could be removed from the Java base at any time. Otherwise, the size of the class libraries increased again by over 1700 classes.

With Java 10 Oracle changed its Java course and began to »clean up« the Java class libraries somewhat. In addition, Oracle began to publish two Java versions every year with version 10. With version 11, Oracle again offers a version with Long Term Support (LTS). The Java manufacturer cleaned up the successor version Java 11 even more extensively. This had the consequence that Applets, Java Webstart and unfortunately also the GUI library JavaFX disappeared from the Java base. These huge changes to the last Java versions meant that many companies did not introduce these newer Java versions.

2.2.3 The Presence And Future of Java

Java 12 brought among other things switch expressions as preview. They extend the previous switch statements and are a fixed part of the language from Java 14. Java 15 introduced, among other things, so-called sealed classes and interfaces as a preview. Sealed classes and interfaces restrict other class to implement or extend them.

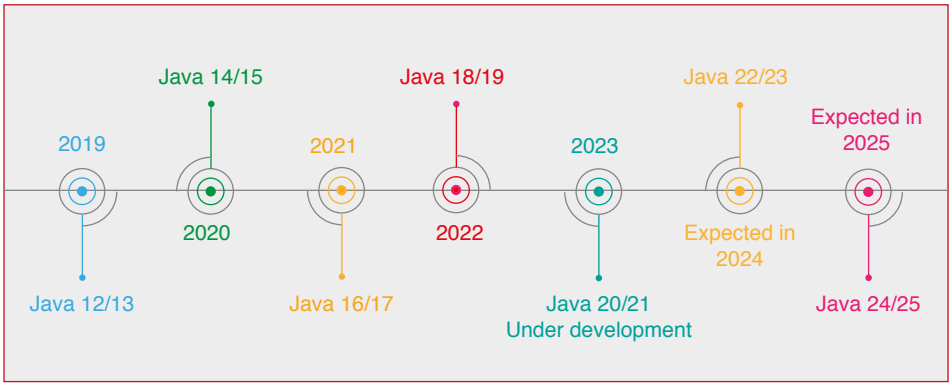


Figure 2.4 The presence and future of Java

Java 16 brought a number of internal changes. This affected for example the source code of the JDK, which is written in C++. With Java 16 it is allowed to use C++ 14 language features. With Java 17, Oracle again offered a version with Long Term Support. Another important innovation of this version was the support for Apple's new computer architecture. For this there is a so-called MacOS/AArch64 port of the JDK.

With Java 18, Oracle introduced UTF-8 as the default character set for the standard Java APIs. At the time this book went to press, Java 19 had been released. With this version a port of the JDK for Linux/RISC-V will be available. Another important innovation of this Java version is virtual threads. They allow a programmer to develop systems that require comparatively few resources from the operating system even under heavy load.

2.3 Why Java?

Java is one of the most successful programming technologies today: it is used at many universities and companies. Many private users also program in Java. The reasons lie in the features of the Java technology.

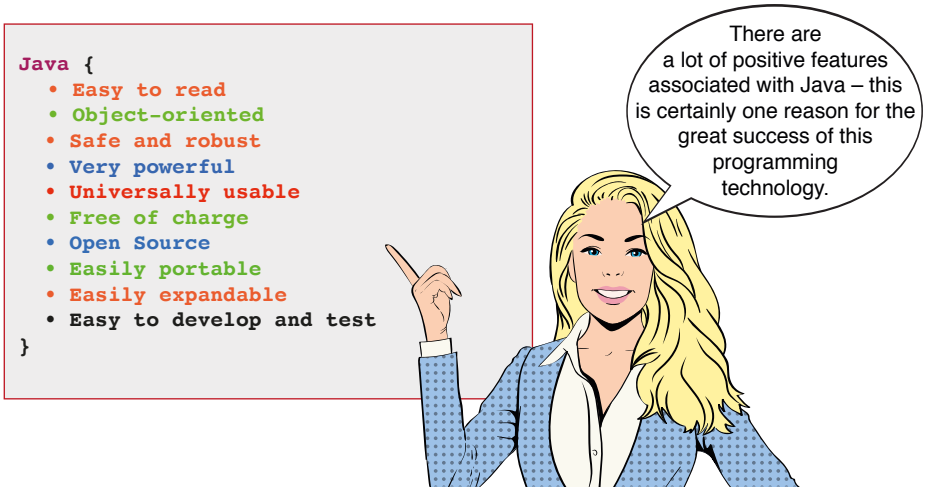


Figure 2.5: The features of Java technology.

2.3.1 Easy to Read

The simple syntax of Java has the advantage that you can learn the language comparatively easily and read Java programs easily. The simple syntax also means that you can more easily understand the structure and meaning of Java programs that you have not developed yourself.

2.3.2 Object-Oriented

The fact that Java programs are easy to read is not only due to the syntax of the programming language, but also because they are structured in an object-oriented way. Object orientation is one of the most important reasons for the clear structure of Java programs. In the chapter 3, »Object-Oriented Programming«, you will learn everything about the topic.

2.3.3 Safe And Robust

One of the most important features of the Java programming language is that it supports the development of safe and robust programs. Java programs owe part of their robustness

to their object orientation. To make Java programs run more stably than object-oriented C++ programs, the developers of the Java language have removed error-prone C++ constructs. This has ensured that Java programs generally run much more safely and do not crash very easily, as some poorly developed and tested C++ programs do.

2.3.4 Very Powerful

The Java programming language helps you develop programs that can meet the most demanding requirements for commercial applications. With such professionally developed Java programs, thousands of users can work efficiently in parallel, safely and without interruption.

2.3.5 Universally Useable

There are programming languages specifically for scientists, others specifically for business people. Some are designed specifically for programming graphical interfaces, others specifically for database queries. Java is so successful because the language can be used universally. The spectrum of Java begins with small programs for private use, continues with tools such as the Eclipse development environment and extends to web applications for large companies. Java can even be used for such exotic programs as controlling Lego robots.

2.3.6 Free of Charge

Another advantage of Java is that the technology has been free of charge since its beginnings. Even professional Java programs can be developed for free.

2.3.7 Open Source

The Java inventor Sun Microsystems has made Java open as OpenJDK (JDK = Java Development Kit). This means that the source code and thus the know-how of the JDK is public. The technical term for this is open source. This ensures that the Java programming platform can be transferred to other operating systems independently of a specific manufacturer. Java programs can therefore be run on (almost) any computer system.

2.3.8 Easily Portable

Forget if you have read or heard somewhere that Java is platform independent. The well-known saying »Write once, run anywhere« is a nice fairy tale that Sun Microsystems came up with to help Java succeed. Large sections of IT management, many specialist authors and journalists still believe this story today.

The truth is: Java programs are comparatively easy to transfer (port) from one operating system to another. This is because Java programs can rely on finding almost the same conditions on other operating systems as on the operating system on which they were developed.

Table 2.1: Operating systems currently supported by Java (64 bit).

Java version	Operating system
`Java 13`	Linux x64, MacOS x64, Windows x64
`Java 14`	Linux x64, MacOS x64, Windows x64
`Java 15`	Linux AArch64, Linux x64, MacOS x64, Windows x64
`Java 16`	Linux AArch64, Linux x64, MacOS x64, Windows x64
`Java 17`	Linux AArch64, Linux x64, MacOS AArch64, MacOS x64, Windows x64
`Java 18`	Linux AArch64, Linux x64, MacOS AArch64, MacOS x64, Windows x64
`Java 19`	Linux AArch64, Linux x64, MacOS AArch64, MacOS x64, Windows x64

Java programs are extremely easy to port under two conditions: The first prerequisite is that the developers of a Java program have not used any specialties of an operating system that are not covered by Java. The second prerequisite is that a Java version suitable for the program exists for the target operating system. The first requirement is very easy to fulfill. The second prerequisite is usually given by the fact that Java is open source.

2.3.9 Easily Expandable

The term Java class library has come up several times before, without me explaining it in more detail: Java class libraries are pre-built program parts that your program can easily use. If you research on the Internet whether a certain problem is solved by a Java class library, you will be amazed how large the offer is.

One of the reasons why Java is so successful is because of the variety of Java class libraries. You do not have to develop all program parts yourself. Instead, you can draw from the pool of prefabricated and proven solutions. And the best thing is: like Java, many of these class libraries are open source and free of charge.

2.3.10 Easy to Develop And Test

In the early days of Java, developing Java programs was painstaking. There were simply no professional development and testing tools. That has changed dramatically. Part of Java's success is certainly due to the fact that there are hardly any other programming languages with as many professional development environments as Eclipse IDE or IntelliJ IDEA. The fact that many professional tools are also free of charge was another plus point for many companies.

2.4 What Belongs to Java?

On the previous pages the term "Java technology" was mentioned several times. I wanted to make clear with it that Java is far more than *only* a programming language in comparison with the predecessor languages C and C++. But what does Java technology consist of? It provides a uniform basis for the development, execution and operation of programs. Java technology is based on the pillars »Java Programming Language«, »Java Virtual Machine«,

»Java Class Libraries« and »Java Development Tools« (Figure 2.6).

If Java is your first programming language, you may be confused by the many new terms and abbreviations in this section. Be patient. You don't need to memorize everything, because throughout the book the terms are repeated with many examples. And after the first program examples, the meaning of the terms will become much clearer to you.

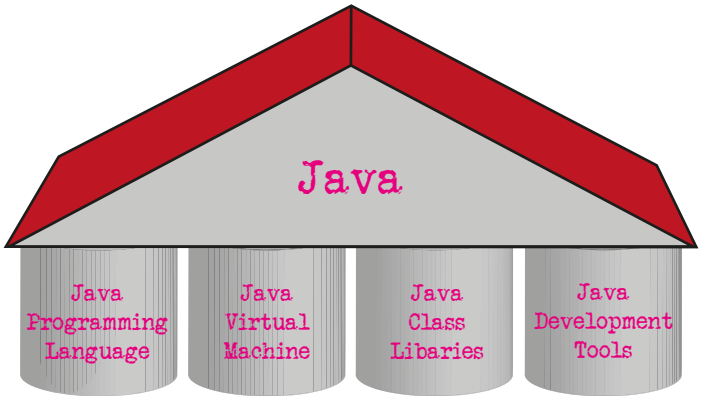


Figure 2.6: The Java technology rests on four pillars.

2.4.1 Java Programming Language

The Java programming language is the core of the Java technology. Like any programming language, Java has a special syntax to write a program. The developers of the Java programming language based their syntax on C++ to make the transition to Java easier. However, some elements, which I will discuss in more detail later, were greatly improved, resulting in the ease of reading and robustness of Java.

2.4.2 Java Virtual Machine

Java programs are not stored as executable files on Windows, for example. Therefore, you (usually) need a virtual machine to run the Java program. Behind the mysterious name »virtual machine« is a special program. This program executes Java programs. In the case of Java, the program is called »Java Virtual Machine«. Since the term is so long, it is usually abbreviated as Java VM or simply JVM.

This Java VM must be specially adapted for operating systems such as Windows, Linux or MacOS. In other words, there is a special Java VM for Windows, one for Linux and one for MacOS. This special Java VM is one of the prerequisites for easily transferring Java programs from one operating system such as Windows to another such as Linux (Figure 2.7).

So if you take it very strictly, Java programs are not platform-independent at all, as often claimed. On the contrary: they are platform-dependent. To be more precise: They are dependent on the Java platform together with the Java VM contained therein. So that Java programs can be transferred easily from one operating system to another, the Java inventor Sun Microsystems resorted to a trick. He simply developed a Java platform for each desired target operating system.

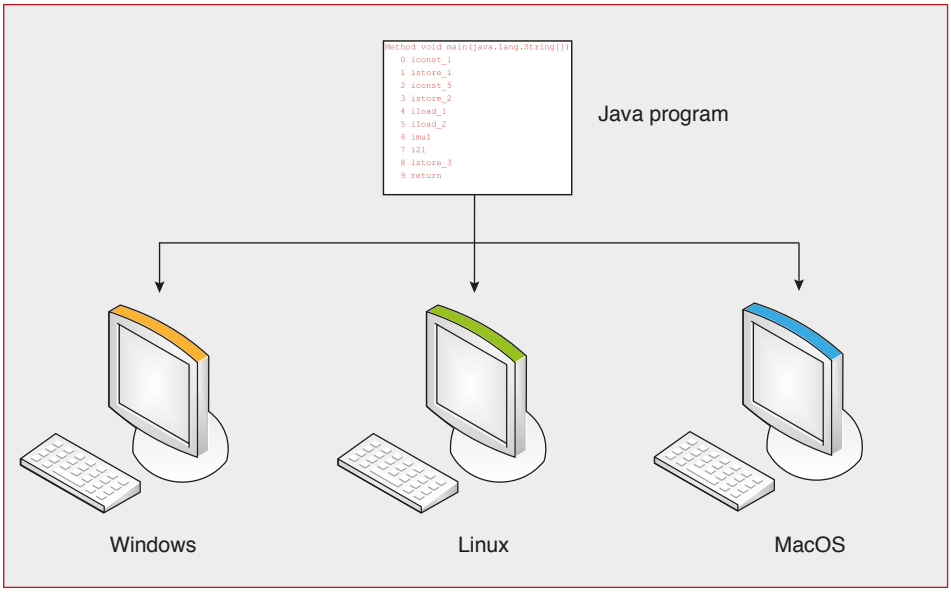


Figure 2.7: Java programs run on various operating systems using the Java VM.

This Java platform protects each Java program from the special peculiarities of each operating system. Now you may ask: Who develops this Java platform? In the early days of Java, the Java inventor Sun Microsystems did it themselves. Since Sun Microsystems was bought by Oracle, primarily Oracle takes care of these different Java ports. Since Java is open source, the further development of Java is increasingly taken care of by the open source community of programmers in addition to Oracle.

Only if a Java platform exists for a certain operating system environment, Java programs can be executed on this environment. This is also the reason why you have to install Java (and therefore a Java VM) on your computer operating system. Chapter 4, section 4.2.2, »Install Java«, shows you exactly how to do this.

2.4.3 Java Class Libraries

This term also came up several times before, without me explaining it in detail. Class libraries are prefabricated programming parts consisting of Java classes. What the term »class« means exactly, I will come back to later. Just this much at this point: class libraries make programming easier for you, because you can fall back on prefabricated program parts.

When you install Java, the most important Java class libraries are already included. These libraries are therefore included in the Java standard. Together with the Java VM, the class libraries form the Java runtime environment (Figure 2.8). Java runtime environment is abbreviated JRE.

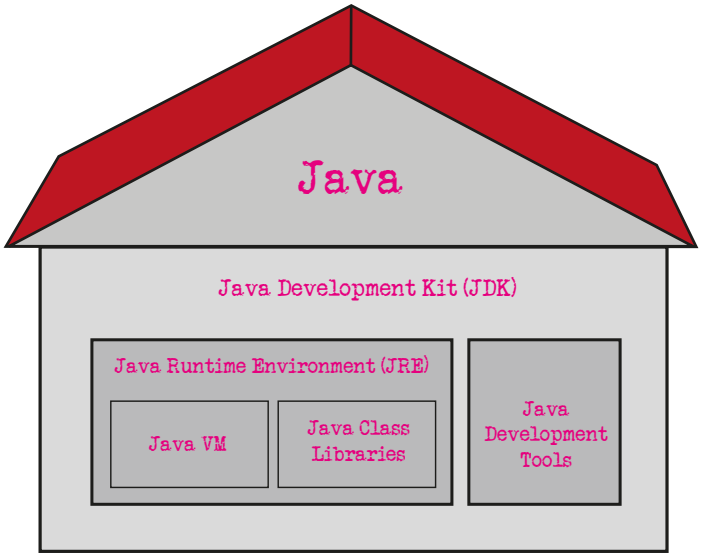


Figure 2.8: The Java Development Kit consists of tools and the runtime environment.

2.4.4 Java Development Tools

Java development tools form the fourth important pillar of the Java technology. The development tools provided with Java can be used to compile Java programs so that they can be executed by the Java Runtime Environment (JRE). In addition, there are other tools to test and deliver programs. Together with the JRE, the development tools form the Java Development Kit (JDK).

2.5 Java Versions

In the early days of Java, it took a relatively long time for a new Java version to be released (section 2.2, »Overview«). Meanwhile, new Java versions are released every six months. At the time this book went to press, Java 17 was just available, so this book is based on that Java version. However, most of the information in this book also applies to older Java versions back to Java 8 and – with high probability – to newer versions as well.

2.6 Java Editions

Java inventor Sun Microsystems originally released three editions of the Java platform. For this book only the Java Standard Edition (Java SE) is important. For the sake of completeness, however, I would like to present all three editions here in a comparative way.

2.6.1 Java Standard Edition

The simplest edition is the Java Standard Edition (Java SE or JSE). With this edition you can develop simple Java programs as presented in this book. You can equate this edition with the Java Development Kit (JDK) in a simplified way.

2.6.2 Java Enterprise Edition

Java Enterprise Edition (Java EE or JEE) is an extension of the Java Standard Edition. Java creator Sun Microsystems released this edition to better support enterprise application development using Java. JEE, however, is not a product like the JDK that can be downloaded anywhere, but rather »just« a set of specifications from the Java inventor. Open source developers can use these specifications to program class libraries for enterprise applications. Of course, you can download them again if you plan to develop such an application.

2.6.3 Java Micro Edition

This Java Micro Edition (Java ME or JME) has its roots in the origins of Java. As you may recall, Java was intended to be developed to better program household appliances. Since these devices are not very powerful, Sun Microsystems developed a particularly small edition of the Java programming platform. Since every smartphone today is so much more powerful than the household appliances of that time, this edition has become more or less obsolete.

2.7 Summary

Java originated in 1995 as the Oak programming language for home appliances. The following year, 1996, Sun Microsystems released the first version of Java. Sun Microsystems designed Java as a technology. This Java technology consists of the Java programming language, the Java class libraries, the Java Runtime Environment (JRE) and the various development tools.

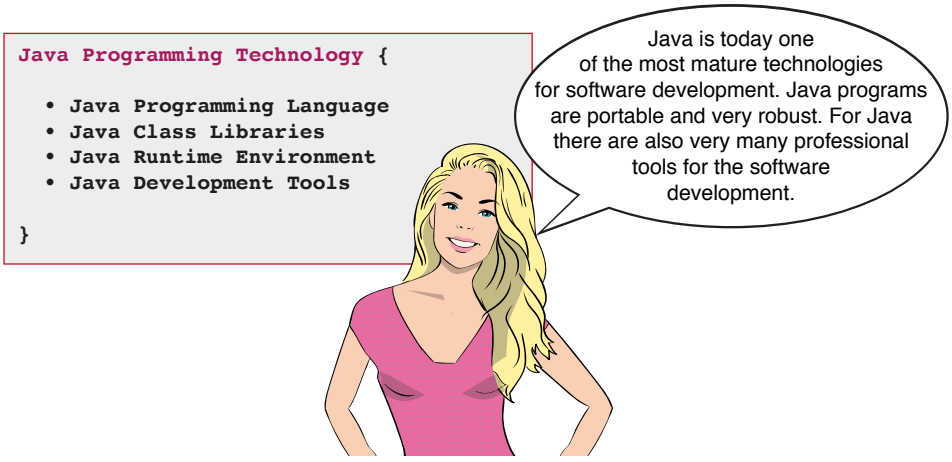


Figure 2.9 Java programs are easily portable and very robust.

The Java Runtime Environment runs Java programs. It consists of the Java Virtual Machine (JVM) and the Java class libraries. Each operating system requires its own runtime environment. Java programs can be developed and tested with the Java development tools. The Java Runtime Environment and the Java development tools together are called Java Development Kit (JDK).

2.8 Literature

Bernhard Steppan: »Oracle Slims Down Java 11«;
https://www.programmingcourse.net/publications/articles/oracle-slims-down-java-11 Supported operating systems and processors: *https://jdk.java.net/archive*

2.9 Exercises

When you have finished this chapter, please go the exercises to deepen your knowledge:
https://www.programmingcourse.net/books/java_with_eclipse/technology_overview
There you will also find the solutions to the exercises.

3 Object-Oriented Programming

3.1 Introduction

Java is an object-oriented programming language. Therefore, it is important to understand exactly how object-oriented programming works. This chapter is all about objects and classes, attributes and methods, and how objects can be protected from encroachment by other hostile objects. You'll also learn why object orientation came into being and what's so special about it.

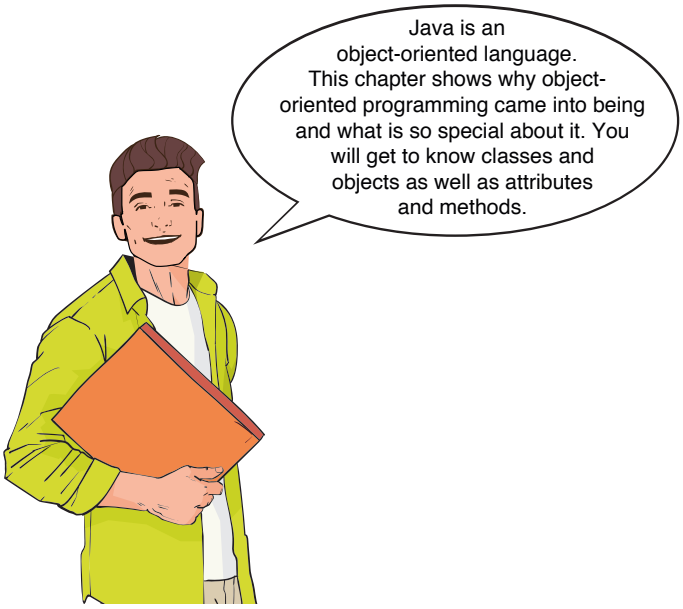


Figure 3.1: Florian explains what is special about object-oriented programming.

3.2 Overview

How did it all come about? It all started in the mid-60s of the 20th century. At that time, there was a software crisis. It was triggered by the increased demands on computer programs. As a result, the software became more complex and more buggy. At congresses, experts discussed the causes of the crisis and the reasons for the increased error rate.

A part of the software experts came to the conclusion that the software crisis could not be mastered with the conventional programming languages. They criticized at the conventional programming languages above all that the natural world could be represented so far only inadequately. They therefore began to develop a generation of new programming languages.

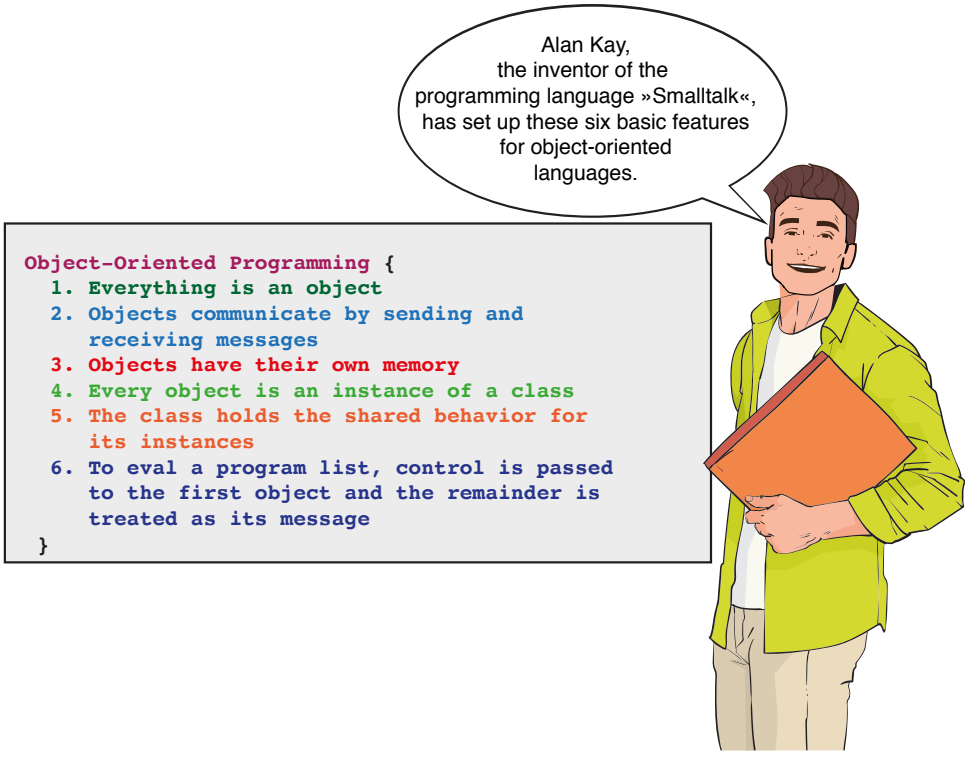


Figure 3.2: Features of object-oriented languages.

The experts began to use natural terms from the form theory of classical Greek philosophy for the new programming languages. They transformed these terms for programming (Figure 3.2). Since everything revolved around the notion of object, they called the new generation of languages »object-oriented«.

3.3 Object

Objects are to a Java program what cells are to an organism: A Java program is composed of these smallest units. If you look at a set of similar objects, you will notice that their basic *appearance* is common. In object-oriented programming, such objects are often referred to as *instances* of a class.

As an example again the programming course of professor Roth is to serve. Anna, Julia, Lukas and Florian take part in the programming course. Let us first pick out only the students Anna and Julia. Both students are objects with many similarities. For example, they have in common that they are women, attend the same programming course and are enrolled at the same university.

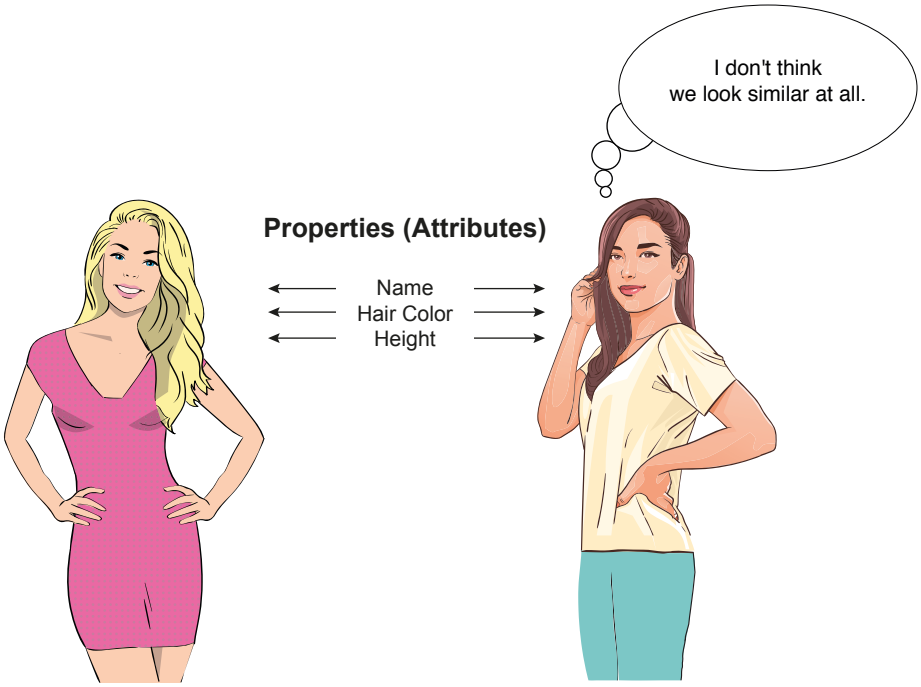


Figure 3.3: The »objects« Anna and Julia are similar.

From the point of view of object-oriented programming, this means that these two objects are *similar*. The differences between these objects result from the different value of their *attributes*. For example, both female students have a different name and different hair color as well as size (Figure 3.3).

Thus, similar objects have only their *principle* shape and certain capabilities in common. Everything else is *individual*. The common shape and the common capabilities of objects are determined by the building plan of the objects. This blueprint is called *class* in object-oriented programming.

3.4 Class

It is the *class* that shapes the principal form and capabilities of objects like the two female students. A class relates to an object as the blueprint of a human being relates to a real human being. The class gives an upper hand to different objects of the same kind. It is also said that a class *classifies* its objects – hence the name.

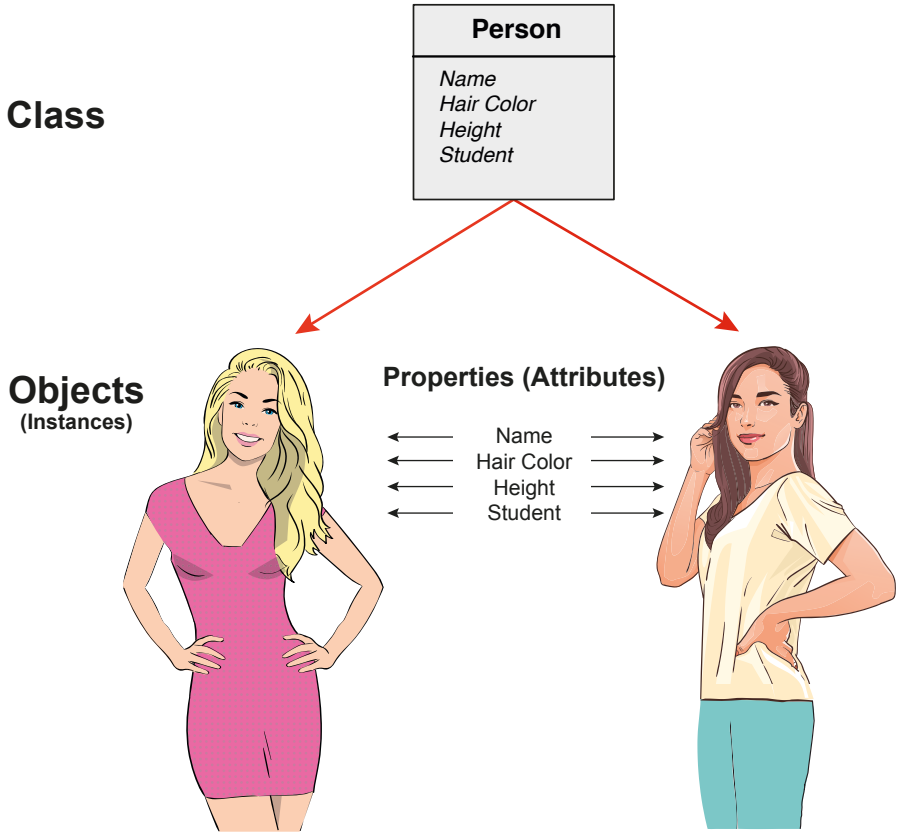


Figure 3.4: The class »Person« provides the blueprint for the »objects« Anna and Julia.

3.4.1 Properties

Our new class *Person* should get the attributes *name, hair color, height* and additionally *student*. Attributes are also called properties. You use these properties to specify the characteristics of an object. When new person objects are created from this class, all instances have an individual *name*, an individual *hair color*, an individual *height*, and an individual value for the *student* property (Figure 3.4).

Constants and Variables

For example, *Anna* should have the following properties: name = Anna, hair color = blonde, height = 1.71 m. Her girl friend from the programming class is named Julia, has hair color = brown, and is 1.72 m tall. Although both persons have been created according to the same blueprint (class), two clearly individual objects have been created: Both have different names, have different hair color, and are both different heights. Some properties, such as height, could be considered invariant, others more mutable. A special form of the variable property of an object is its state.

States

Some properties of the two individuals have been assigned fixed values, while others have been assigned variable values. Unlike the fixed properties, the flexible properties describe the *state* of the object. For example, the *student* property describes whether a person is currently enrolled at a university. The state of an object can change over time.

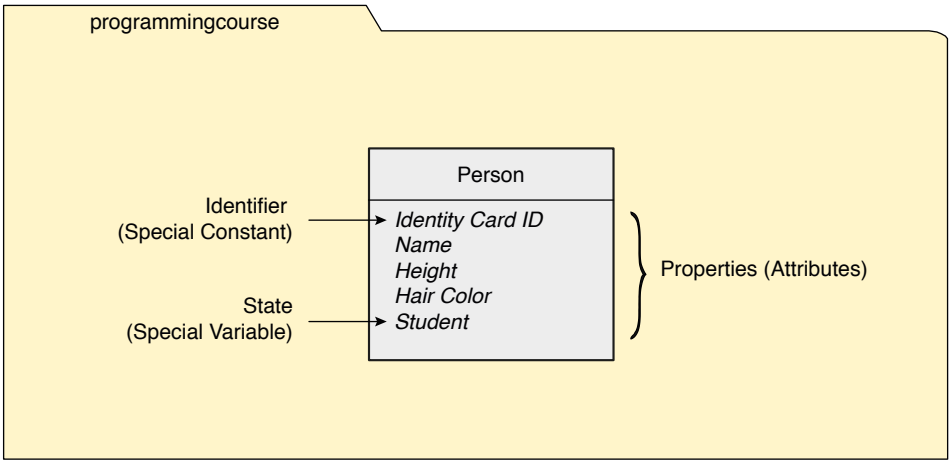


Figure 3.5: Constants, variables, states, and identifiers are properties of a class.

3.4.1.1 Identifier

What would happen if you created the objects *Anna* and *Julia* so that they had the same *size*, the same *hair color*, and the same state *student*? How could they be distinguished then? In this case, both objects have been given individual values for their attributes, but they happen to be the same. Thus, objects in a program are also alike, like identical twins.

So in order to distinguish objects better, you need something like a genetic fingerprint. In programming, the developer assigns a so-called identifier. This identifier is an additional attribute for which care is taken that it is *unique*. Only the identifier of an object ensures that the program can distinguish different copies even if their attributes happen to have the same values.