Java for Absolute Beginners

Learn to Program the Fundamentals the Java 9+ Way

Iuliana Cosmina

Apress®

Java for Absolute Beginners

Learn to Program the Fundamentals the Java 9+ Way

Iuliana Cosmina

Apress®

Java for Absolute Beginners: Learn to Program the Fundamentals the Java 9+ Way

Iuliana Cosmina
Edinburgh, UK

ISBN-13 (pbk): 978-1-4842-3777-9 ISBN-13 (electronic): 978-1-4842-3778-6
https://doi.org/10.1007/978-1-4842-3778-6

Library of Congress Control Number: 2018964482

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Steve Anglin
Development Editor: Matthew Moodie
Coordinating Editor: Mark Powers

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail editorial@apress.com; for reprint, paperback, or audio rights, please email bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484237779. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

This book is dedicated to all men that told me software engineering is not for women.

And to that one professor that told me I'm not PhD material.

How do ya' like them apples?

Table of Contents

About the Author

Iuliana Cosmina is currently a software engineer for NCR Edinburgh. She has been writing Java code since 2002. She has contributed to various types of applications, such as experimental search engines, ERPs, track and trace, and banking. During her career, she has been a teacher, a team leader, software architect, a DevOps professional, and a software manager.

She is a Spring-certified professional, as defined by Pivotal, the makers of Spring Framework, Boot, and other tools. She considers Spring the best Java framework to work with.

When she is not programming, she spends her time reading, blogging, learning to play piano, travelling, hiking, or biking.

- You can find some of her personal work on her GitHub account at `https://github.com/iuliana`.
- You can find her complete CV on her LinkedIn account at `www.linkedin.com/in/iulianacosmina`.
- You can contact her at `Iuliana.Cosmina@gmail.com`.

About the Technical Reviewer

Wallace Jackson has been writing for leading multimedia publications about his work in new media content development since the advent of *Multimedia Producer Magazine* nearly two decades ago. He has authored a half-dozen Android book titles for Apress, including four titles in the popular Pro Android series. Wallace received his undergraduate degree in business economics from the University of California at Los Angeles and a graduate degree in MIS design and implementation from the University of Southern California. He is currently the CEO of Mind Taffy Design, a new media content production and digital campaign design and development agency.

Acknowledgments

Here I am again, the main author of a technical book for the third time.

This book was quite challenging to write, because I had to quickly adapt to changes made to the Java ecosystem. With the new six months interval release system, modules being introduced, and backward compatibility thrown out the window, I found myself with a project that stopped compiling and had to invest precious time into fixing it, understand why it broke in the first place, and eventually adapt the book.

Writing books for beginners is tricky, because as an experienced developer, it might be difficult to find the right examples and explain them in such a way that even a non-technical person would easily understand them. That is why I am profoundly grateful to Matthew Moodie and Mark Powers for all the support and advice they provided to keep this book at beginner level. We have been working together for four years and it has been a fruitful collaboration so far.

I would like to thank Wallace Jackson; his recommendations and corrections were crucial for the final form of the book.

Apress has published many of the books that I have read and used to improve myself professionally. It is a great honor to publish my fourth book with Apress, and it gives me enormous satisfaction to be able to contribute to the “making” of a new generation of Java developers.

I am grateful to all my friends who had the patience to listen to me complain about sleepless nights and writer’s block. Thank you all for being supportive and making sure I still had some fun while writing this book. You have no idea how dear you are to me.

I am thankful to John Mayer still, as his music provided yet again, a great environment for my working nights.

A special thank you to Achim Wagner, whom I consider both a mentor and a dear friend. He provided me with an environment and support to grow as a professional and as a person, and I will miss working with him.

A special thank you to the Bogza-Vlad family: Monica, Tinel, Cristina, and Stefan. You are all close to my heart and this book might have been released later without your support when I moved to Edinburgh.

And a very special thank-you in advance to all the passionate Java developers who will find mistakes in the book and be so kind to write me about them so I can provide an erratum and make this book even better.

Introduction

Even though I have been writing Java Applications since 2002 I don't think I've ever dived so deeply into the JVM as I did while writing this book. Most companies I've worked for had their own code base when I joined them, and my work was mostly related to designing, improving or maintaining one that already existed. It's like making brownies when you already have brownie mix. Writing this book has given me the opportunity to get down to basics and work with basic ingredients—so, making brownies using eggs, flower, cocoa, milk, and butter.

Java began in 1982 and was created by a handful of people. The most renowned name linked to the beginning of Java is James Gosling, also known as the father of Java, the language that is now used on over three billion devices. When Oracle bought Sun Microsystems, developers were worried about Java's future, especially since its main creator quit the company and went on to create what was thought to be Java's replacement: Scala. That will probably never happen. Java is still here.

Most banking applications are written in Java and because it is definitely dangerous and costly to migrate these applications, Java will be here in 50 years, if not more. Java began by making websites more dynamic and more entertaining, and ended up being the basis for applications run on ATMs, cashier machines, computers, and mobile devices. Sure, this would have been more difficult if Java wasn't cross-platform.

The first Java version was officially released in 1996. Since then, ten more versions have been released, with the latest one, Java 11, being released on 25th September 2018. The work on Java 12 has already begun and the early access build is already available.

This book was written with the intention to cover the fundamental elements of the language and of the JVM, especially the ones introduced in versions 9, 10, and 11. The book provides a complete overview of the most important Java classes in the JVM, all wrapped up in a multimodule project that compiles with Java 11 and Gradle 5.

A group of reviewers has gone over the book, but if you notice any inconsistencies, please send an email to `editorial@apress.com`, or directly to the author, and corrections will be made and published in an erratum that will be uploaded to the official GitHub repository for the book. The example source code for this book can be found on GitHub or downloaded from the official book's product page, located at `www.apress.com/in/book/9781484237779`.

I truly hope you will enjoy using this book to learn Java as much as I enjoyed writing it.

CHAPTER 1

An Introduction to Java and Its History

Java is currently one of the most influential programming languages. It all started in 1990, when an American company that was leading the revolution in the computer industry decided to gather its best engineers together to design and develop a product that would allow them to become an important player in the new emerging Internet world. Among those engineers was James Arthur Gosling, a Canadian computer scientist who is recognized as the "father" of the Java programming language. It would take five years of design, programming, and one rename (from Oak to Java because of trademark issues), but finally in 1996, Java 1.0 was released for Linux, Solaris, Mac, and Windows.

You might have the tendency to skip this chapter altogether. But I think it would be a mistake. I was never much interested in the history of Java. I was using it for work. I knew that James Gosling was the creator and that Oracle bought Sun, and that was pretty much it. I never cared much about how the language evolved, where the inspiration came from, or how one version was different from another. I started learning Java at version 1.5, and I took a lot of things in the language for granted. So, when I was assigned to a project running on Java 1.4, I was quite confused, because I did not know why some of the code I wrote was not compiling. Although the IT industry is moving very fast, there will always be that one client that has a legacy application. And knowing the peculiarities of each Java version is an advantage, because you know the issues when performing a migration.

When I started doing research for this book, I was mesmerized. The history of Java is interesting because it is a tale of incredible growth, success of a technology, and how a clash of egos in management almost killed the company that created it. Because even if Java is the most used technology in software development, it is simply paradoxical that the company that gave birth to it no longer exists.

I. Cosmina, *Java for Absolute Beginners*, https://doi.org/10.1007/978-1-4842-3778-6_1

This chapter covers each version of Java to track the evolution of the language and the Java virtual machine. You can find a timeline for versions 1.0 to 1.8 on the Oracle official site at `http://oracle.com/edgesuite.net/timeline/java./`. But first, I'll introduce the book.

Who This Book Is For

Most Java books for beginners start with the typical *Hello World!* example depicted here:

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

This code, when executed, prints *Hello World!* in the console. But if you have bought this book, it is assumed that you want to develop real applications in Java, and get a real chance when applying for a position as a Java developer. If this is what you want, if this is who you are, a beginner with the wits and the desire to make full use of this language's power, then this book is for you. And that is why to start this book, a complex example is used. We go over it in almost every section, when some part of it is clarified.

Java is a language with a syntax that is readable and based on the English language. So, if you have a logical thinking and a little knowledge of the English language, it should be obvious to you what the following code does without even executing it.

```
package com.apress.ch.one.hw;

import java.util.List;

public class Example01 {

  public static void main(String[] args) {
    List<String> items = List.of("1", "a", "2", "a", "3", "a");

    items.forEach(item -> {
        if (item.equals("a")) {
              System.out.println("A");
        } else {
                System.out.println("Not A");
            }
        });
    }
}
```

In this code example, a list of text values is declared; then the list is traversed, and when a text is equal to "a", the letter "A" is printed in the console; otherwise, "Not A" is printed.

If you are an absolute beginner to programming, this book is for you, especially because the sources attached to this book make use of algorithms and design patterns commonly used in programming. So, if your plan is to get into programming and learn a high-level programming language, read the book, run the examples, write your own code, and you should have a good head start.

If you already know Java, you can use this book too because it covers the specifics of Java versions 9, 10, and 11 (the EAP[1] release).

How This Book Is Structured

The chapter you are reading is an introductory one that covers a little bit of Java history, showing you how the language has evolved and a glimpse into its future. Also, the mechanics of executing a Java application are covered, so that you are prepared for **Chapter** 2. The next chapter shows you how to set up a development environment and introduces you to a simple application. In Chapters 3 to 7, the fundamental parts of the language are covered: packages, modules, classes, objects, operators, data types, statements, streams, lambda expressions, and so forth. Starting with Chapter 8 more advanced features are covered such as: interactions with external data sources: reading writing files, serializing/deserializing objects, testing and creating an interface. Chapter 12 is dedicated fully to the publish-subscribe framework introduced in Java 9. Chapter 13 covers the garbage collector.

The book is completed by the `java-for-absolute-beginners` project. This project is organized in modules (thus it is a multimodule project) that are linked to each other and must be managed by **Gradle**. Gradle is something we developers call a *build tool*, which is used to build projects. To build a project means transforming the code into something that can be executed. I chose to use multimodule projects for the books I write because it is easier to build them, and common elements can be grouped together, keeping the

[1]Early Access Program

configuration of the project simple and non-repetitive. Also, by having all the sources organized in one multimodule project, you get the feedback on whether the sources are working or not as soon as possible, and you can contact the author and ask him or her to update them.

Conventions

This book uses a number of formatting conventions that should make it easier to read. To that end, the following conventions are used within the book:

- code or concept names in paragraphs appear as follows:

  ```
  import java.util.List;
  ```

- code listings appear as follows:

  ```
  public static void main(String[] args) {
          System.out.println("Hello there young developer!");
  }
  ```

- logs in console outputs appear as follows:

  ```
  01:24:07.809 [main] INFO  c.a.Application - Starting Application
  01:24:07.814 [main] DEBUG c.a.p.c.Application - Running in debug mode
  ...
  ```

- ! This symbol appears in front of paragraphs that you should pay specific attention to.
- *Italic* font is used for metaphors, jocular terms and technical terms that the reader should pay special attention to because they are not explained in the current context, but they are covered in the book. Examples: “This was mentioned before at the end of Chapter 4 when *generics* were introduced.” “The **stack** memory is used during execution (also referred to as at *runtime*)” or “Let’s see how this is being done *under the hood*”.
- **Bold** font is used for chapter references and important terms.

As for my style of writing, I like to write my books in the same way I have technical conversations with colleagues and friends: sprinkling jokes, giving production examples, and making analogies to non-programming situations. Because programming is just another way to model the real world.

When Java Was Owned by Sun Microsystems

The first version of Java was released in 1996. Up until that point, there was a small team named the **Green Team** that worked on a prototype language named Oak, which was introduced to the world with a working demo—an interactive handheld home entertainment controller called the Star7. The star of the animated touch-screen user interface was a cartoon character named **Duke**, created by one of the team's graphic artists, Joe Palrang. Over the years, Duke has become the official Java technology mascot, and every JavaOne conference has its own Duke mascot personality and the most simple version is depicted in Figure 1-1.

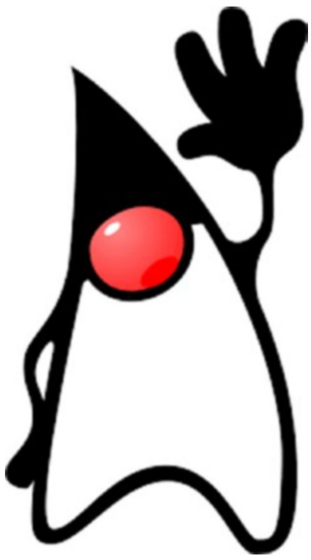


Figure 1-1. *The Duke mascot (image source: `http://oracle.com`)*

The **Green Team** released it to the world via the Internet, because that was the fastest way to create widespread adoptions. You can imagine that they jumped for joy every time somebody downloaded it, because it meant people were interested in it. And there are a few other advantages making software open source, like the fact that contributions and feedback come from a bigger and diverse number of people from all over the world. Thus, for Java, this was the best decision, as it shaped the language a lot of developers are using today. Even after 22 years, Java is still among the top-three most used programming languages.

The American company that started all of this was Sun Microsystems, founded in 1982. It guided the computer revolution by selling computers, computer parts, and software. Among their greatest achievements is the Java programming language. In Figure 1-2,[2] you can see the company logo that was used since Java's birth year until it was acquired by Oracle in 2010.

Figure 1-2. *The Sun Microsystems logo (image source: `https://en.wikipedia.org/wiki/Sun_Microsystems`)*

It is quite difficult to find information about the first version of Java, but dedicated developers that witnessed the birth of Java—when the web was way smaller and full of static pages—did create blogs and shared their experience with the world. It was quite easy for Java to shine with its applets that displayed dynamic content and interacted with the user. But because the development team thought bigger, Java became much more than a web programming language. Because in trying to make applets run in any browser, the team found a solution to a common problem: portability.

Nowadays, developers face a lot of headaches when developing software that should run on any operating system. And with the mobile revolution, things have become really tricky. In Figure 1-3, you see an abstract drawing of what is believed to be the first Java logo.

[2]The story behind the logo can be read here: `https://goodlogo.com/extended.info/sun-microsystems-logo-2385`. You can also read more about Sun Microsystems.

Figure 1-3. *The first Java logo, 1996–2003 (image source:* `http://xahlee.info/`*)*

Java 1.0 was released at the first JavaOne conference—with over 6000 attendees. It started out as a language named Oak[3] that was really similar to C++ and was designed for handheld devices and set-top boxes. It evolved into the first version of Java, which provided developers some advantages that C++ did not.

- **security**: In Java, there is no danger of reading bogus data when accidentally going over the size of an array.
- **automatic memory management**: A Java developer does not have to check if there is enough memory to allocate for an object and then deallocate it explicitly; the operations are automatically handled by the garbage collector. This also means that pointers are not necessary.
- **simplicity**: There are no pointers, unions, templates, structures. Mostly anything in Java can be declared as a class. Also, confusion when using multiple inheritance is avoided by modifying the inheritance model and not allowing multiple class inheritance.
- **support for multithreaded execution**: Java was designed from the start to support development of multithreaded software.
- **portability**: A Java motto is *Write it once, run it everywhere.* This is made possible by the Java virtual machine, which is covered shortly.

[3]The language was named by James Gosling after the oak tree in front of his house.

All this made Java appealing for developers, and by 1997, when Java 1.1 was released, there were already approximatively 400,000 Java developers in the world. JavaOne had 10,000 attendees that year. The path to greatness was set. Before going further in our analysis of each Java version, let's clarify a few things.

Why Is Java Portable?

I mentioned a few times that Java is portable and that Java programs can run on any operating system. It is time to explain how this is possible. Let's start with a simple drawing, like the one in Figure 1-4.

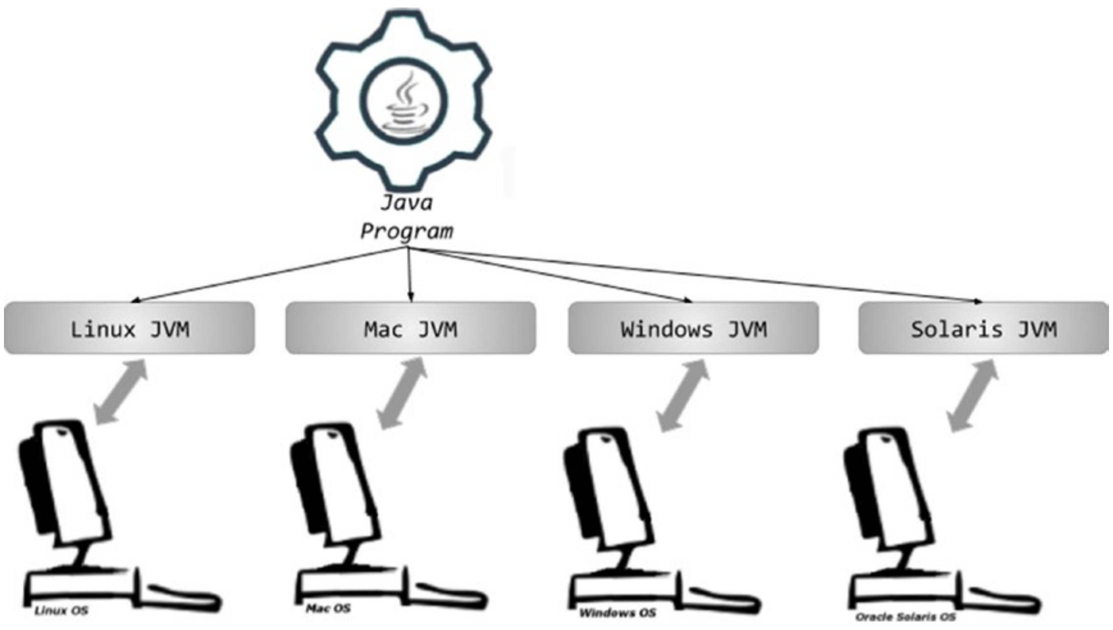


Figure 1-4. *What makes Java portable*

Java is what we call a high-level programming language that allows a developer to write programs that are independent of a particular type of computer. High-level languages are easier to read, write, and maintain. But their code must be translated by a compiler or interpreted into machine language (unreadable by humans because is it made up of numbers) to be executed, because that is the only language that computers understand.

In Figure 1-4, notice that on top of the operating systems, a JVM is needed to execute a Java program. JVM stands for Java virtual machine, which is an abstract computing machine that enables a computer to run a Java program. It is a platform-independent execution environment that converts Java code into machine language and executes it.

So, what is the difference between Java and other high-level languages? Well, other high-level languages compile source code directly into machine code that is designed to run on a specific microprocessor architecture or operating system, such as Windows or UNIX. What JVM does, it that is mimics a Java processor making it possible for a Java program to be interpreted as a sequence of actions or operating system calls on any processor regardless of the operating system.

And because the Java compiler was mentioned, we have to get back to Java 1.1, which was widely used, even as new versions were released. It came with an improved Abstract Window Toolkit (AWT) graphical API (collections of components used for building applets), inner classes, database connectivity classes (JDBC model), classes for remote calls (RMI), a special compiler for Microsoft platforms named JIT,[4] support for internationalization, and Unicode. Also, what made it so widely embraced is that shortly after Java was released, Microsoft licensed it and started creating applications using it. The feedback helped further development of Java, thus Java 1.1 was supported on all browsers of the time, which is why it was so widely deployed.

! A lot of terms used in the introduction of the book might seem foreign to you now, but as you read the book, more information is presented and these words will start to make more sense. For now, just keep in mind, that every new Java version, has something more than the previous version, and at that time, every new component is a novelty.

So, what exactly happens to developer-written Java code until the actual execution? The process is depicted in Figure 1-5.

[4]**J**ust **I**n **T**ime

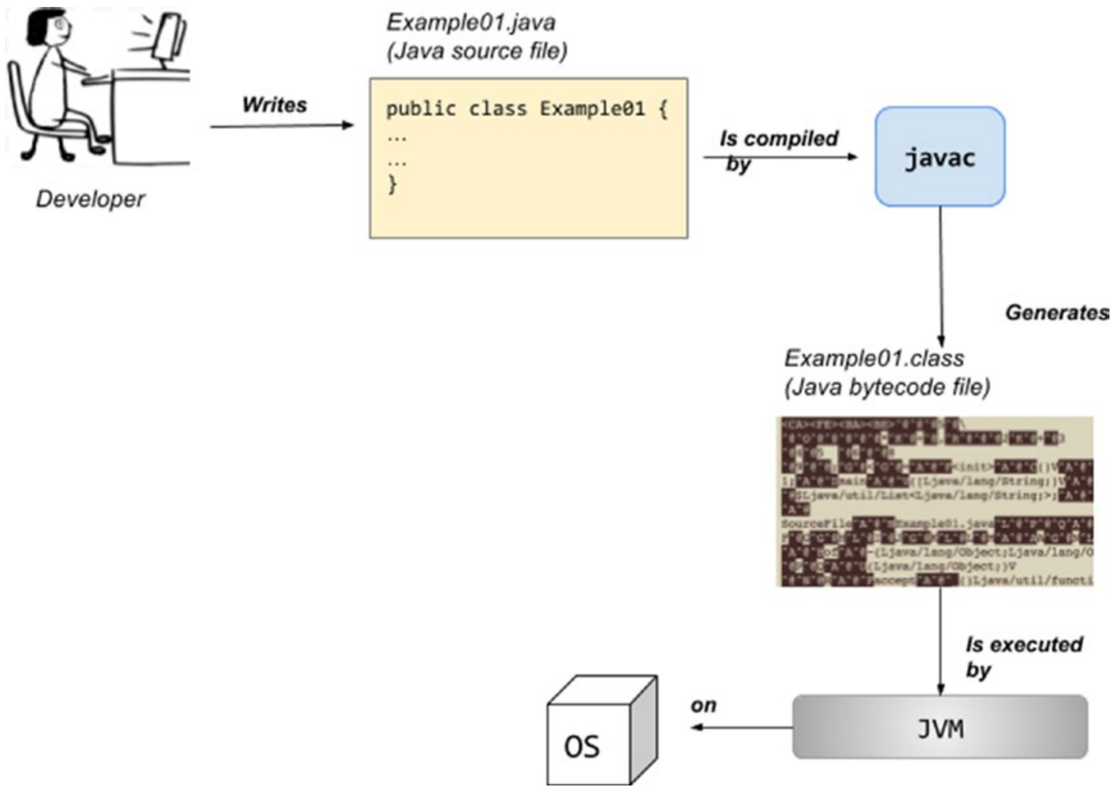


Figure 1-5. *From Java code to machine code*

In Figure 1-5, you see that Java code is compiled and transformed to bytecode that is then interpreted and executed by the Java virtual machine on the underlying operating system. This is what Java is: a compiled and interpreted general-purpose programming language with a large number of features that make it well suited for the web. And now that we've covered how Java code is executed, let's go back to some more history.

Sun Microsystem's Java Versions

The first stable Java version released by Sun Microsystems could be downloaded from the website as an archive named JDK 1.0.2. JDK is an acronym for **J**ava **D**evelopment **K**it. This is the software development environment used for developing Java applications and applets. It includes the **J**ava **R**untime **E**nvironment (JRE), an interpreter (loader), a compiler, an archiver, a documentation generator, and other tools needed for Java development. We will get into this more when I cover how to install the JDK on your computer.

Starting with version 1.2, released in 1998, Java versions were given codenames.[5] The Java version 1.2 codename was **Playground**. It was a massive release and this was the moment when people started talking about **the Java 2 Platform**. Starting with this version, the releases up to J2SE 5.0 were renamed, and J2SE replaced JDK because the Java platform was now composed of three parts:

- J2SE (Java 2 Platform, Standard Edition), which later became JSE, a computing platform for the development and deployment of portable code for desktop and server environments
- J2EE (Java 2 Platform, Enterprise Edition), which later became JEE, a set of specifications extending Java SE with specifications for enterprise features such as distributed computing and web services
- J2ME (Java 2 Platform, Micro Edition), which later became JME, a computing platform for development and deployment of portable code for embedded and mobile devices

With this release, the JIT compiler became part of Sun Microsystem's JVM (which basically means turning code into executable code became a faster operation and the generated executable code was optimized), the Swing graphical API was introduced as a fancy alternative to AWT (new components to create fancy desktop applications were introduced), and the Java collections framework (for working with sets of data) was introduced.

J2SE 1.3 was released in 2000 with the codename **Kestrel** (maybe as a reference to the newly introduced Java sound classes). This release also contained Java XML APIs.

J2SE 1.4 was released in 2002 with the codename **Merlin**. This is the first year that the **J**ava **C**ommunity **P**rocess members were involved in deciding which features the release should contain, and thus, the release was quite consistent. This is the first release of the Java platform developed under the Java Community Process as JSR 59.[6] The following features are among those worth mentioning.

- Support for IPv6 (basically applications that run over a network can now be written to work using networking protocol IPv6).

[5]All codenames, for intermediary releases too, are listed here: `http://www.oracle.com/technetwork/java/javase/codenames-136090.html#close`

[6]If you want to see the contents and the list of Java Specification Requests, follow this URL: `http://www.jcp.org/en/jsr/detail?id=59`

- Non-blocking IO (IO is an acronym for input-output, which refers to reading and writing data— a very slow operation. Making IO non-blocking means to optimize these operations to increase speed of the running application.)
- Logging API (Operations that get executed need to be reported to a file or a resource, which can be read in case of failure to determine the cause and find a solution. This process is called logging and apparently only in this version components to support this operation were introduced.)
- Image processing API (Components developers can use this to manipulate images with Java code.)

Java's coffee cup logo made its entrance in 2003 (between releases 1.4 and 5.0) at the JavaOne conference. You can see it in Figure 1-6.[7]

Figure 1-6. *Java official logo 2003-2006 (image source: `http://oracle.com`)*

J2SE 5.0 was released in 2004 with the codename **Tiger**. Initially, it followed the typical versioning, and was named 1.5, but because this was a major release with a significant number of new features that proved a serious improvement of maturity, stability, scalability, and security of the J2SE, the version was labeled 5.0 and presented like that to the public, even if internally 1.5 was still used. For this version and the next two, it was considered that `1.x = x.0`. Let's list those features because most of them are covered in the book.

[7]The Java language was first named Oak. It was renamed to Java because of copyright issues. There are a few theories that you will find regarding the new name. There is one saying that the JAVA name is actually a collection of the initials of the names being part of the Green team: **J**ames Gosling, **A**rthur **V**an Hoff, and **A**ndy Bechtolsheim, and that the logo is inspired by their love of coffee.

- **Generics** provide compile-time (static) type safety for collections and eliminates the need for most type conversions (which means the type used in a certain context is decided while the application is running, we have a full section about this in **Chapter** 5).
- **Annotations**, also known as **metadata**, are used to tag classes and methods to allow metadata-aware utilities to process them (which means a component is labeled as something another component recognizes and does specific operations with it).
- **Autoboxing/unboxing** are automatic conversion between primitive types and matching object types (wrappers), also covered in **Chapter** 5.
- **Enumerations** define static final ordered sets of values using the `enum` keyword; covered in **Chapter** 5.
- **Varargs** are the last parameter of a method is declared using a type name followed by three dots (`String...`), which implies that any number of arguments of that type can be provided and is placed into an array; covered in **Chapter** 3.
- **Enhanced for each loop** is used to iterate over collections and arrays too; covered in **Chapter** 5.
- Improved semantics for multithreaded Java programs, covered in **Chapter** 7.
- **Static imports** are covered in **Chapter** 5.
- Improvements for RMI (not covered in the book), Swing (**Chapter** 10), concurrency utilities (**Chapter** 7), and introduction to the `Scanner` class; covered in **Chapter** 11.

Java 5 was the first available for Mac OS X (version 10.4) and the default version installed on Mac OS X (version 10.5). There were a lot of updates[8] released for this version to fix issues related to security and performance. It was a pretty buggy release, which is understandable since quite a lot of features were developed in only two years.

[8]Let's call them what they actually are: hotfixes.

In 2006, Java SE 6 was released with a little delay, with the codename **Mustang**. Yes, yet another rename. And yes, yet again a serious number of features were implemented in a short period of time and a lot of updates followed. This was the last major Java release by Sun Microsystems. Oracle acquired the company in January 2010. Let's take a look at the most important features in this release:

- Dramatic performance improvements for the core platform (applications run faster, need less memory or CPU to execute)
- Improved web service support (optimized components that are required for development of web applications)
- JDBC 4.0 (optimized components that are required for development of applications using databases)
- Java Compiler API (basically, from your code you can components that are used to compile code)
- Many GUI improvements, such as integration of SwingWorker in the API, table sorting and filtering, and true Swing double-buffering (eliminating the gray-area effect); basically, improvement of components used to create interfaces for desktop applications

In December 2008, Java FX 1.0 SDK was released. JavaFX is used to create graphical user interfaces for any platform, and the initial version was a scripting language. Until 2008, there were two ways to create a user interface in Java:

- **AWT** (Abstract Window Toolkit) components, which are rendered and controlled by a native peer component specific to the underlying operating system; that is why AWT components are also called *heavyweight components*.
- **Swing** components, which are called *lightweight* because they do not require allocation of native resources in the operating system's windowing toolkit. The Swing API is a complimentary extension of AWT.

In the first versions, it was never really clear if JavaFX would actually have a future and grow up to replace Swing. The management turmoil inside Sun did not help in defining a clear path for the project either.

Oracle Takes Over

Although Sun Microsystems won a lawsuit against Microsoft, in which they agreed to pay $20 million for not implementing the Java 1.1 standard completely, in 2008, the company was in such poor shape that negotiations for a merger with IBM and Hewlett-Packard began. In 2009, Oracle and Sun announced that they agreed on the price: Oracle would acquire Sun for $9.50 a share in cash; this amounted to a $5.6 billion offer. The impact was massive. A lot of engineers quit, including James Gosling, *the father of Java*, which made a lot of developers question the future of the Java platform.

Java SE 7, codename **Dolphin**, was the first Java version released by Oracle in 2011. It was the result of an extensive collaboration between Oracle engineers and members of the worldwide Java communities, like the OpenJDK Community and the Java Community Process (JCP). It contained a lot of changes, but still, a lot fewer than developers expected. Considering the long period between the releases, the expectations were pretty high. Project **Lambda**, which was supposed to allow usage of lambda expressions in Java (this leads to considerable syntax simplification in certain cases), and **Jigsaw** (making JVM and the Java application modular; there is a section in **Chapter** 3 about them) were dropped. Both were released in future versions. The following are the most notable features in Java 7.

- JVM support for dynamic languages with the new `invokedynamic` bytecode (basically, Java code can use code implemented in non-Java languages, such as C)
- Compressed 64-bit pointers (internal optimization of the JVM, so less memory is consumed)
- Small language changes grouped under project **Coin**
  - strings in `switch` (covered in **Chapter** 7)
  - automatic resource management in try-statement (covered in **Chapter** 5)
  - improved type inference for generics—the diamond `<>` operator (covered in **Chapter** 5)
  - binary integer literals (covered in **Chapter** 5)
  - multiple exceptions handling improvements (covered in **Chapter** 5)
- Concurrency improvements

- New I/O library (new classes added to read/write data to/from files, covered in **Chapter** 8)
- `Timsort` to sort collections and arrays of objects instead of `merge sort` (Sets of data that are ordered need to be sorted using an algorithm, basically, in this version, the algorithm was replaced with one that has better performance. Better performance usually means reducing of consumed resources: memory and/or CPU, or reducing the time needed for execution.)

It must have been difficult to pick up a project and update it with almost none of the original development team involved. That can be seen in the 161 updates that followed; most of them needed to fix security issues and vulnerabilities.

JavaFX 2.0 was released with Java 7. This confirmed that the JavaFX project had a future with Oracle. As a major change, JavaFX stopped being a scripting language and became a Java API. This meant that knowledge of the Java language syntax would be enough to start building user graphical interfaces with it. JavaFX started gaining ground over Swing because of its hardware-accelerated graphical engine called **Prism** that did a better job at rendering.

Java SE 8, codename **Spider**, was released in 2014, and included features that were initially intended to be part of Java 7. But, better late than never, right? Three years in the making, Java 8 contained the following key features.

- Language syntax changes
  - Language-level support for lambda expressions (functional programming features)
  - Support for default methods in interfaces (covered in **Chapter** 4)
  - New date and time API (covered in **Chapter** 5)
  - New way to do parallel processing by using streams (covered in **Chapter** 8)
- Improved integration with JavaScript (the Nashorn project). JavaScript is a web scripting language that is quite loved in the development community, so providing support for it in Java probably won Oracle a few new supporters.
- Improvements of the garbage collection process

Starting with Java 8, codenames were dropped to avoid any trademark-law hassles; instead, a semantic versioning that easily distinguishes major, minor, and security-update releases was adopted.[9] The version number matches the following pattern:

```
$MAJOR.$MINOR.$SECURITY
```

When executing `java -version` in a terminal (if you have Java 8 installed), you see something similar to the following log.

```
$ java -version
java version "1.8.0_162"
JavaTM SE Runtime Environment build 1.8.0_162-b12
Java HotSpotTM 64-Bit Server VM build 25.162-b12, mixed mode
```

In this log, the version numbers have the following meaning:

- The `1` represents the major version number, incremented for a major release that contains significant new features as specified in a new edition of the Java SE Platform Specification.
- The `8` represents the minor version number, incremented for a minor update release that may contain compatible bug fixes, revisions to standard APIs and other small features.
- The `0` represents the security level that is incremented for a security-update release that contains critical fixes, including those necessary to improve security. `$SECURITY` is not reset to zero when `$MINOR` is incremented, which lets the users know that this version is a more secure one.
- `162` is the build number.
- `b12` represents additional build information.

This versioning style is quite common for Java applications, thus this versioning style was adopted to align with the general industry practices.

Java SE 9 was released in September 2017. The long-awaited **Jigsaw** project was finally here. The Java platform is finally modular.

[9]Java Enhancement Proposal 223: `http://openjdk.java.net/jeps/223`

! This is a big change for the Java world; it's not a change in syntax and it's not some new feature. It's a change in the design of the platform. Some experienced developers I know, who have used Java since its first years have difficulties adapting. It is supposed to fix some serious problems that Java has been living with for years (covered in **Chapter** 3). You are lucky because, as a beginner, you start from scratch, so you do not need to change the way you develop your applications.

The following are the most important features, aside the introduction of Java modules.[10]

- The Java Shell tool, an interactive command-line interface for evaluation declarations, statements, and expressions written in Java (covered in **Chapter** 3)
- Quite a few security updates
- Improved `try-with-resources`: final variables can now be used as resources (covered in **Chapter** 5)
- `"_"` is removed from the set of legal identifier names (covered in **Chapter** 4)
- Support for private interface methods (covered in **Chapter** 5)
- Enhancements for the Garbage-First (G1) garbage collector; this becomes the default garbage collector (covered in **Chapter** 13)
- Internally, a new more compact String representation is used (covered in **Chapter** 5)
- Concurrency updates (related to parallel execution, mentioned in **Chapter** 5)
- Factory methods for collections (covered in **Chapter** 5)
- Updates of the image processing API optimization of components used to write code that processes images

[10]A detailed description of all JDK 9 features can be found here: `https://docs.oracle.com/javase/9/whatsnew/toc.htm#JSNEW-GUID-983469B6-9BB5-48CA-B71D-8D7012B2F3CA`

Java 9 followed the same versioning scheme as Java 8, with a small change. The Java version number contained in the name of the JDK finally became the `$MAJOR` number in the version scheme. So, if you have Java 9 installed, when executing `java -version` in a terminal, you see something similar to the following log.

```
$ java  -version
java version "9.0.4"
JavaTM SE Runtime Environment build 9.0.4+11
Java HotSpotTM 64-Bit Server VM build 9.0.4+11, mixed mode
```

Java SE 10 (AKA Java 18.3) was released on March 20, 2018. Oracle changed the Java release style, so a new version is released every six months. Also, Java 10 uses the new versioning convention set up by Oracle: the version numbers follow a `$YEAR.$MONTH` format.[11] Apparently, this release versioning style is supposed to make it easier for developers or end users to figure out the age of a release so that they can judge whether to upgrade it to a newer release with the latest security fixes and additional features.

The following are a few features of Java 10.[12]

- A local-variable type inference to enhance the language to extend type inference to local variables (this is the most expected feature and is covered in **Chapter** 5)
- More optimizations for garbage collection (covered in **Chapter** 13)
- Application Class-Data Sharing to reduce the footprint by sharing common class metadata across processes (this is an advanced feature that won't be covered in the book)
- More concurrency updates (related to parallel execution, mentioned in **Chapter** 5)
- Heap allocation on alternative memory devices (The memory needed by JVM to run a Java program—called *heap memory*—can be allocated on an alternative memory device, so the heap can also be split between volatile and non-volatile RAM. More about memory used by Java applications can be read in **Chapter** 5.)

[11]Java Enhancement Proposal 322: `http://openjdk.java.net/jeps/322`

[12]The complete list can be found at `http://openjdk.java.net/projects/jdk/10/` and the release notes containing the detailed list with API and internal changes can be found at `http://www.oracle.com/technetwork/java/javase/10-relnote-issues-4108729.html`10-relnote-issues-4108729.html

And since we've done this before, let's see what running `java -version` in a terminal shows for this Java version.

```
$ java -version
java version "10" 2018-03-20
JavaTM SE Runtime Environment 18.3 build 10+46
Java HotSpotTM 64-Bit Server VM 18.3 build 10+46, mixed mode
```

Java SE 11 (AKA Java 18.9)[13] (released on 25 September 2018) contains the following features:

- Removal of JEE advanced components used to build enterprise Java applications and Corba (really old technology for remote invocation, allowing your application to communicate with applications installed on a different computer) modules
- Local-variable syntax for lambda parameters allow the `var` keyword to be used when declaring the formal parameters of implicitly typed lambda expressions
- Epsilon, a low-overhead garbage collector (is a no-GC, so basically you can run an application without a GC), basically more optimizations to the garbage collection (covered in **Chapter** 13)
- More concurrency updates (related to parallel execution, mentioned in **Chapter** 5)

Aside from these changes, it was also speculated that a new versioning change should be introduced because the `$YEAR.$MONTH` format did not go so well with developers. (Why so many versioning naming changes, right? Is this really so important? Apparently, it is.) The proposed versioning change is similar to the one introduced in Java 9, and if you are curious, you can read a detailed specification for it at `http://openjdk.java.net/jeps/322`.

When this chapter was written, JDK 11 was available only via the early access program, which is why the `"ea"` string is present in the version name; it means *early access*. It is quite difficult to use it, as it is not supported by any editors or other build tools yet. By the time this book is released, Java 11 will be stable and ready to use and the sources for the book are updated accordingly on the GitHub repository.

[13]Details are at `http://openjdk.java.net/projects/jdk/11/`

```
$ java -version
java version "11-ea" 2018-09-18
JavaTM SE Runtime Environment 18.9 build 11-ea+2
Java HotSpotTM 64-Bit Server VM 18.9 build 11-ea+2, mixed mode
```

And this is where the details end. If you want more information on the first 20 years of Java's life, you can find it on Oracle's website.[14]

What the Future Holds

Java has dominated the industry for more than 20 years. It wasn't always at the top of the most-used development technologies, but it never left the top five since its existence. Even with server-side JavaScript smart frameworks, like Node.js, the heavy-lifting is still left to Java. Emerging programming languages like Scala and Kotlin run on the JVM, so maybe the Java programming language will suffer a serious metamorphosis in order to compete, but it will still be here.

The modularization possibility introduced in version 9 opens the gates for Java applications to be installed on smaller devices, because to run a Java application, we no longer need the whole runtime—only its core plus the modules the application was built with.

Also, there are a lot of applications written in Java, especially in the financial domain, so Java will still be here, because of legacy reasons and because migrating these titan applications to another technology is an impossible mission.

Java will probably survive and be on top for the next 10 to 15 years. It does help that it is a very mature technology with a huge community built around it. And the fact that is easy to learn and developer-friendly makes it remain the first choice for most companies. So, you might conclude at this point that learning Java and buying this book is a good investment.

Prerequisites

Before ending this chapter, it is only fair to tell you that to learn Java, you need to know or have a few things....

- Your way around an operating system, such as Windows, Linux or macOS

[14]The first 20 years of Java's life: http://oracle.com.edgesuite.net/timeline/java/

- How to refine your search criteria, because information related to your operating systems is not covered in the book; if you have issues, you must fix them yourself
- An Internet connection

If you already know Java, and you just bought this book out of curiosity or for the modules chapter, knowing about a build tool like Maven or Gradle is helpful, because the source code is organized in a multimodule project that can be fully built with one simple command. I've chosen to use a build tool because in this day and age, learning Java without one makes no sense; any company you apply to most definitely uses one.

Aside from the prerequisites that I listed, nothing else is needed. You do not need to know math, algorithms, or design patterns. Actually, you might end up knowing a few after you read this book.

This being said, let's dig in.

CHAPTER 2

Preparing Your Development Environment

To start learning Java, you need a few things installed on your computer. The following are the requirements:

- **Java** support on your computer (kinda' mandatory).
- An integrated development environment, also known as **IDE**, which is basically an application in which you write your code and that you use to compile and execute it.
  - The recommended IDE for this book is IntelliJ IDEA. You can go to their website to get the free community edition; for the purposes of the book, it will do.
  - Or, you can choose the most popular free IDE for Java development: Eclipse.
  - Or, you can try NetBeans,[1] which is the default choice for most beginners because it was bundled with the JDK until version 8.[2,3]

[1]Get it from here `https://netbeans.org/`

[2]See: `http://www.oracle.com/technetwork/java/javase/downloads/jdk-netbeans-jsp-142931.html`

[3]For Eclipse and NetbeansNetBeans, you will need to install a plugin for Gradle support.

- **Gradle** is a build tool used to organize projects, to easily handle dependencies, and make your work easier as your projects get bigger. (It is mandatory because the projects in this book are organized and built on a Gradle setup.)
- **Git** is a versioning system that you can use to get the sources for the book, and you can experiment with it and create your own version. It is optional because GitHub, which is where the sources for this chapter are hosted, supports direct download.[4]

To write and execute Java programs/applications, you only need the Java Development Kit installed. All other tools that I've listed here are only needed to make your job easier and to familiarize you with a real development job.

! You probably need administrative rights if you install these applications for all users. For Windows 10, you might even need a special application to give your user administrative rights so you can install the necessary tools. This book provides instructions on how to install everything—assuming your user has the necessary rights. If you need more information, the Internet is there to help.

If it seems like a lot, do not get discouraged; this chapter contains instructions on how to install and verify that each of tool is working accordingly. Let's start by making sure your computer supports Java.

Installing Java

Here you are with your computer and you can't wait to start writing Java applications. But first, you need to get yourself a JDK and install it. For this, you need an Internet connection to open `https://developer.oracle.com/java`.

[4]Also, I don't think there is a company that does not use a versioning system these days, so getting comfortable with Git could be a serious advantage when applying for a software developer position.

Scroll down until you see the **Downloads** section. Click the Java SE link. The two links and their contents are depicted in Figure 2-1.

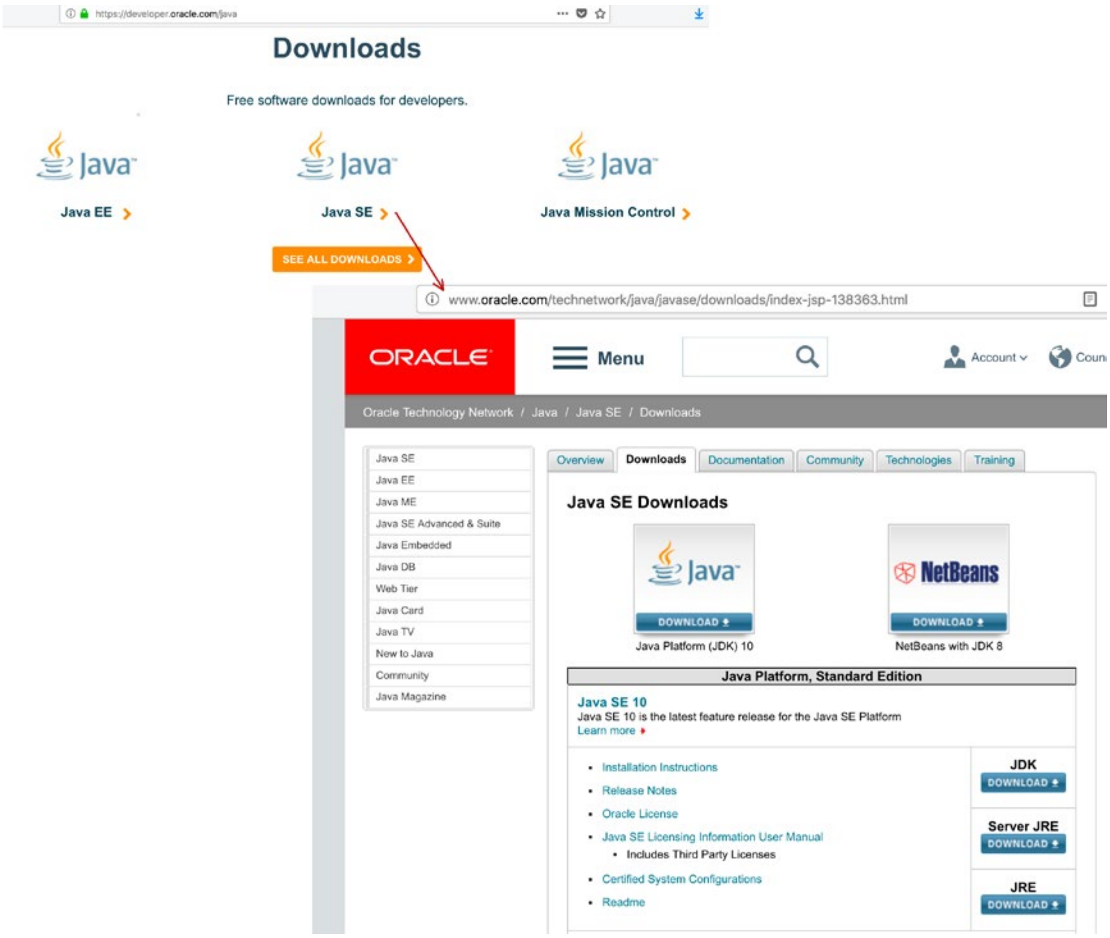
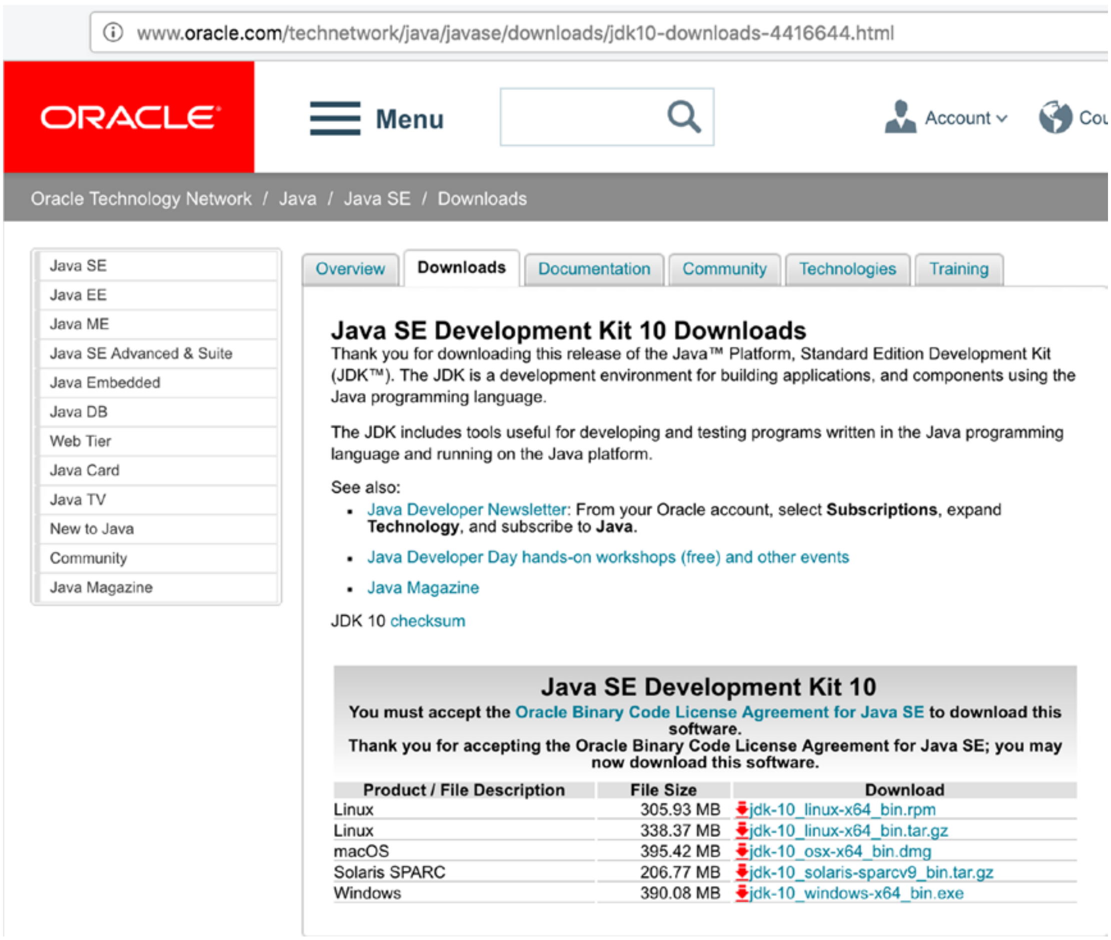


Figure 2-1. *Navigating the Oracle site to find the desired product, JDK in this case*

On the Oracle site, you find the latest stable Java version. Click the **Download JDK** button. You should be redirected to the page depicted in Figure 2-2.



Java SE Development Kit 10 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.

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 10 checksum

Java SE Development Kit 10

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.
Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux	305.93 MB	jdk-10_linux-x64_bin.rpm
Linux	338.37 MB	jdk-10_linux-x64_bin.tar.gz
macOS	395.42 MB	jdk-10_osx-x64_bin.dmg
Solaris SPARC	206.77 MB	jdk-10_solaris-sparcv9_bin.tar.gz
Windows	390.08 MB	jdk-10_windows-x64_bin.exe

***Figure 2-2.** The Oracle page where you can download the desired JDK*

As you can see, JDK is available for a few operating systems. You should download the one matching yours. For writing this book and the source code, I used a macOS computer, which means I download the JDK with the .dmg extension.

You need to accept the license agreement before being allowed to download the desired JDK. You can read it if you are curious, but basically, it tells you that you are allowed to use Java as long as you do not modify its original components. It also tells you that you are responsible for how you use it, so if you use it to write or execute dangerous applications, you are legally responsible.

If you want to get your hands on an early version of JDK that is not officially released yet, go to `http://openjdk.java.net/projects/jdk/`. Under **Releases**, versions 10 and 11, an early access (unstable) JDK 11 is available for download.

! This book covers Java specifics until Java 11, but that version was eight months away when this chapter was written, so some images and details might seem deprecated. Keep in mind that there are common details that remain the same from one version to the next, and those won't be reviewed and changed, as the only thing that is different is the version number. Since this book was planned to be released after Java 11 was released, it is recommended to download that version of the JDK to have full compatibility of the sources.

After you download the JDK, the next step is to install it. Just double-click it and click **Next** until finished. This works for Windows and macOS. The JDK is installed in a specific location.

In Windows, it is `C:\ProgramFiles\Java\jdk-10`.

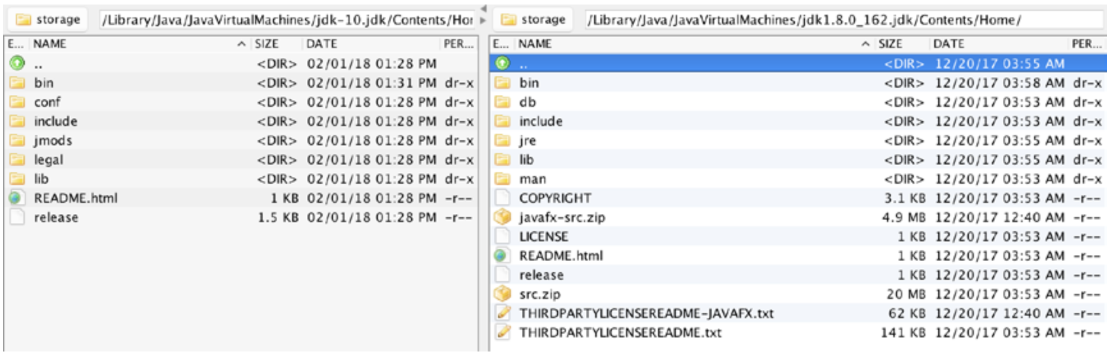
In macOS, it is `/Library/Java/JavaVirtualMachines/jdk-10.jdk/Contents/Home`.

On Linux systems, depending on the distribution, the JDK install location varies. My preferred way is to get the `*.tar.gz` from the Oracle site that contains the full content of the JDK, unpack it, and copy it to a specific location. Also, my preferred location on Linux is `/home/iuliana.cosmina/tools/jdk-10.jdk`.

! Using a PPA (repository)[5] installer on Linux puts the JDK files where they are supposed to go on Linux automatically and updates them automatically when a new version is released using the Linux (Global) updater utility. But if you are using Linux proficiently, you've probably figured this out.

If you go to that location, you can inspect the contents of the JDK. In Figure 2-3, the contents of JDK 10 are on the left; the contents of the JDK 8 are on the right.

[5]Also known as a Package Manager



```
storage  /Library/Java/JavaVirtualMachines/jdk-10.jdk/Contents/Hor
E... NAME                 SIZE   DATE               PER...
     ..                  <DIR>   02/01/18 01:28 PM
     bin                 <DIR>   02/01/18 01:31 PM  dr-x
     conf                <DIR>   02/01/18 01:28 PM  dr-x
     include             <DIR>   02/01/18 01:28 PM  dr-x
     jmods               <DIR>   02/01/18 01:28 PM  dr-x
     legal               <DIR>   02/01/18 01:28 PM  dr-x
     lib                 <DIR>   02/01/18 01:28 PM  dr-x
     README.html          1 KB   02/01/18 01:28 PM  -r--
     release            1.5 KB   02/01/18 01:28 PM  -r--
```

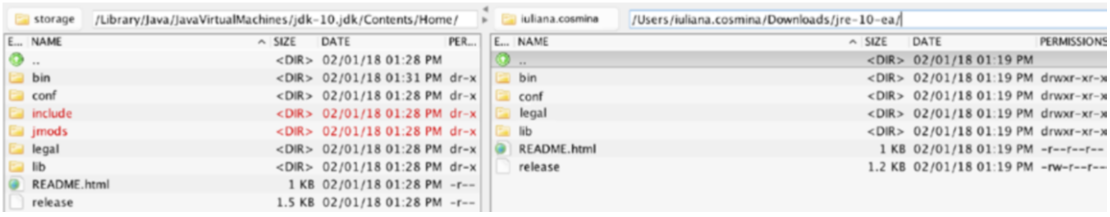
```
storage  /Library/Java/JavaVirtualMachines/jdk1.8.0_162.jdk/Contents/Home/
E... NAME                                  SIZE   DATE               PER...
     ..                                   <DIR>   12/20/17 03:55 AM
     bin                                  <DIR>   12/20/17 03:58 AM  dr-x
     db                                   <DIR>   12/20/17 03:53 AM  dr-x
     include                              <DIR>   12/20/17 03:53 AM  dr-x
     jre                                  <DIR>   12/20/17 03:55 AM  dr-x
     lib                                  <DIR>   12/20/17 03:55 AM  dr-x
     man                                  <DIR>   12/20/17 03:53 AM  dr-x
     COPYRIGHT                           3.1 KB   12/20/17 03:53 AM  -r--
     javafx-src.zip                      4.9 MB   12/20/17 12:40 AM  -r--
     LICENSE                               1 KB   12/20/17 03:53 AM  -r--
     README.html                           1 KB   12/20/17 03:53 AM  -r--
     release                               1 KB   12/20/17 03:53 AM  -r--
     src.zip                              20 MB   12/20/17 03:53 AM  -r--
     THIRDPARTYLICENSEREADME-JAVAFX.txt   62 KB   12/20/17 12:40 AM  -r--
     THIRDPARTYLICENSEREADME.txt         141 KB   12/20/17 03:53 AM  -r--
```

Figure 2-3. *JDK version 8 and ten contents comparison*

I chose to make this comparison because, starting with Java 9, the content of the JDK is organized differently. Until Java 8, the JDK contained a directory called `jre` that contained a Java Runtime Environment (JRE) used by the JDK. The `lib` directory contains Java libraries and support files needed by development tools.

The `bin` contains a set of Java executables for running Java applications.

Starting in Java 9, the JRE was no longer isolated in its own directory. In the Figure 2-4, you see the contents of the JDK 10 on the left, and the contents of the JRE 10 on the right.[6]



```
storage  /Library/Java/JavaVirtualMachines/jdk-10.jdk/Contents/Home/
E... NAME                 SIZE   DATE               PER...
     ..                  <DIR>   02/01/18 01:28 PM
     bin                 <DIR>   02/01/18 01:31 PM  dr-x
     conf                <DIR>   02/01/18 01:28 PM  dr-x
     include             <DIR>   02/01/18 01:28 PM  dr-x
     jmods               <DIR>   02/01/18 01:28 PM  dr-x
     legal               <DIR>   02/01/18 01:28 PM  dr-x
     lib                 <DIR>   02/01/18 01:28 PM  dr-x
     README.html          1 KB   02/01/18 01:28 PM  -r--
     release            1.5 KB   02/01/18 01:28 PM  -r--
```

```
iuliana.cosmina  /Users/iuliana.cosmina/Downloads/jre-10-ea/
E... NAME                 SIZE   DATE               PERMISSIONS
     ..                  <DIR>   02/01/18 01:19 PM
     bin                 <DIR>   02/01/18 01:19 PM  drwxr-xr-x
     conf                <DIR>   02/01/18 01:19 PM  drwxr-xr-x
     legal               <DIR>   02/01/18 01:19 PM  drwxr-xr-x
     lib                 <DIR>   02/01/18 01:19 PM  drwxr-xr-x
     README.html          1 KB   02/01/18 01:19 PM  -r--r--r--
     release            1.2 KB   02/01/18 01:19 PM  -rw-r--r--
```

Figure 2-4. *JDK 10 and JRE contents compared*

[6]JDK and JRE 10 have the same directory structure introduced in version 9.

The directory structure depicted was introduced when Java 9 was released. You can read more about it on the official Oracle site.[7]

The most important thing you need to know about the JDK is that the `bin` directory contains executables and command-line launchers that are defined by the modules linked to the image, thus the JDK has a few of those extra compared to the JRE. The other directories are the `jmods` directory, which contains the compiled module definitions, and the `include` directory, which contains the C-language header files that support native-code programming with the Java Native Interface (JNI) and the Java Virtual Machine (JVM) Debug Interface.

The JAVA_HOME Environment Variable

The most important directory in the JDK is the `bin` directory, because that directory has to be added to the path of your system so you can call the Java executables from anywhere. This allows other applications to call them as well, without extra configurations steps needed. Most IDEs used for handling[8] Java code are written in Java, and they require knowing where the JDK is installed so that they can be run. This is done by declaring an environment variable named `JAVA_HOME` that points to the location of the JDK directory. To make the Java executables callable from any location within a system, you must add the `bin` directory to the system path. The next three sections explain how to do this on the three most common operating systems.

[7]The new directory structure introduced with Java 9 is explained in detail at `https://docs.oracle.com/javase/9/install/installed-directory-structure-jdk-and-jre.htm#JSJIG-GUID-F7178F2F-DC92-47E9-8062-CA6B2612D350`

[8]Includes operations like writing the code, analyzing the code, compiling it, and executing it.

JAVA_HOME on Windows

To declare the `JAVA_HOME` environment variable on a Windows system, you need to open the dialog window for setting up system variables. On Windows systems, click the Start button; in the menu, there is a search box (or right-click the Start button for a context-menu and select Search). Enter the word **environment** in there (the first three letters should suffice) and the option should become available for clicking. These steps are depicted in Figure 2-5.

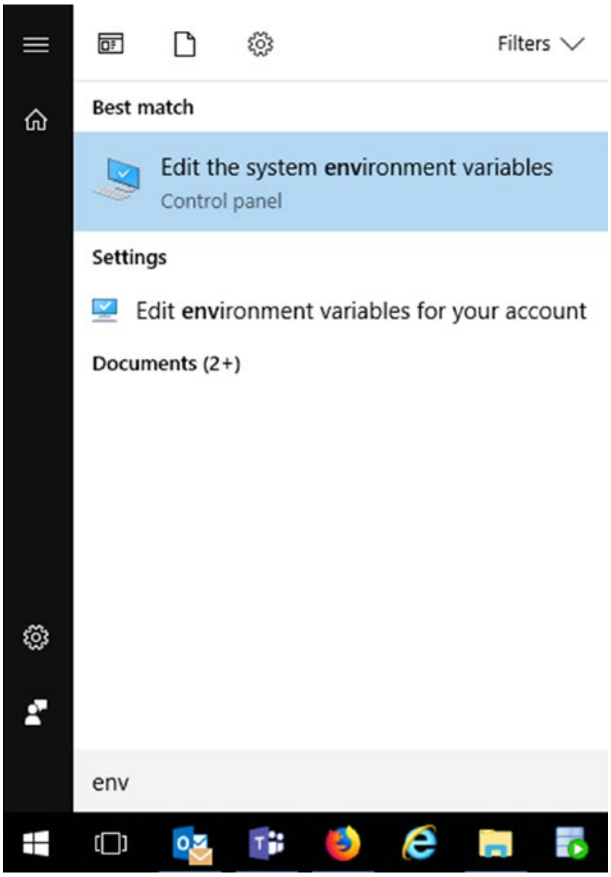
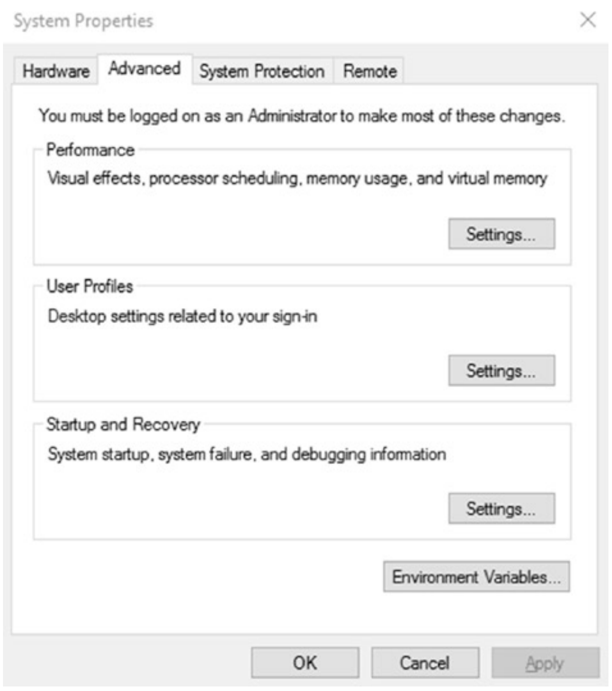

Filters
Best match
Edit the system environment variables
Control panel
Settings
Edit environment variables for your account
Documents (2+)
env

Figure 2-5. *Windows menu item to configure environment variables*

After clicking that menu item, a window like the one shown in Figure 2-6 should open.



You must be logged on as an Administrator to make most of these changes.

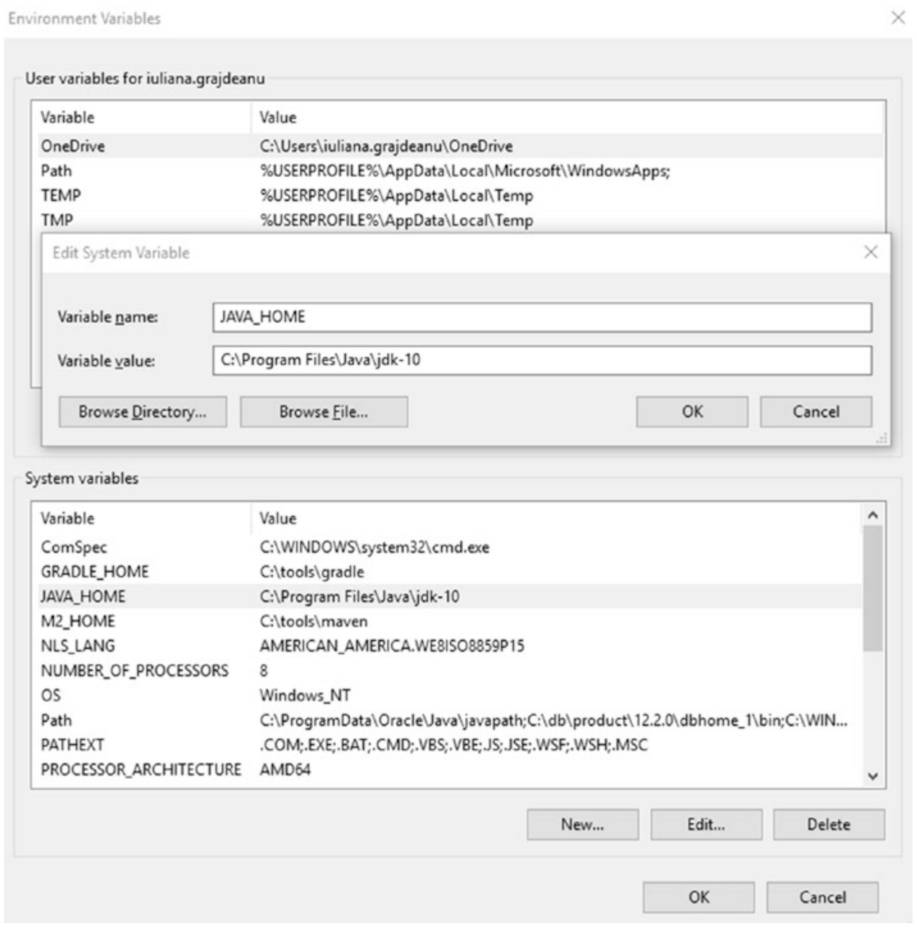
Performance
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles
Desktop settings related to your sign-in

Startup and Recovery
System startup, system failure, and debugging information

Figure 2-6. *First dialog window to set environment variables on Windows*

Click the **Environment Variables** button. Another dialog window opens, which is split into two sections: user variables and system variables. You are interested in **system variables** because that is where we declare `JAVA_HOME`. Just click the **New...** button and a small dialog window appears with two text fields; one requires you to enter the variable name–`JAVA_HOME` in this case, and one requires you to enter the path—to the JDK in this case. The second window and the variable information pop-up dialog window are depicted in Figure 2-7.



User variables for iuliana.grajdeanu

Variable	Value
OneDrive	C:\Users\iuliana.grajdeanu\OneDrive
Path	%USERPROFILE%\AppData\Local\Microsoft\WindowsApps;
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

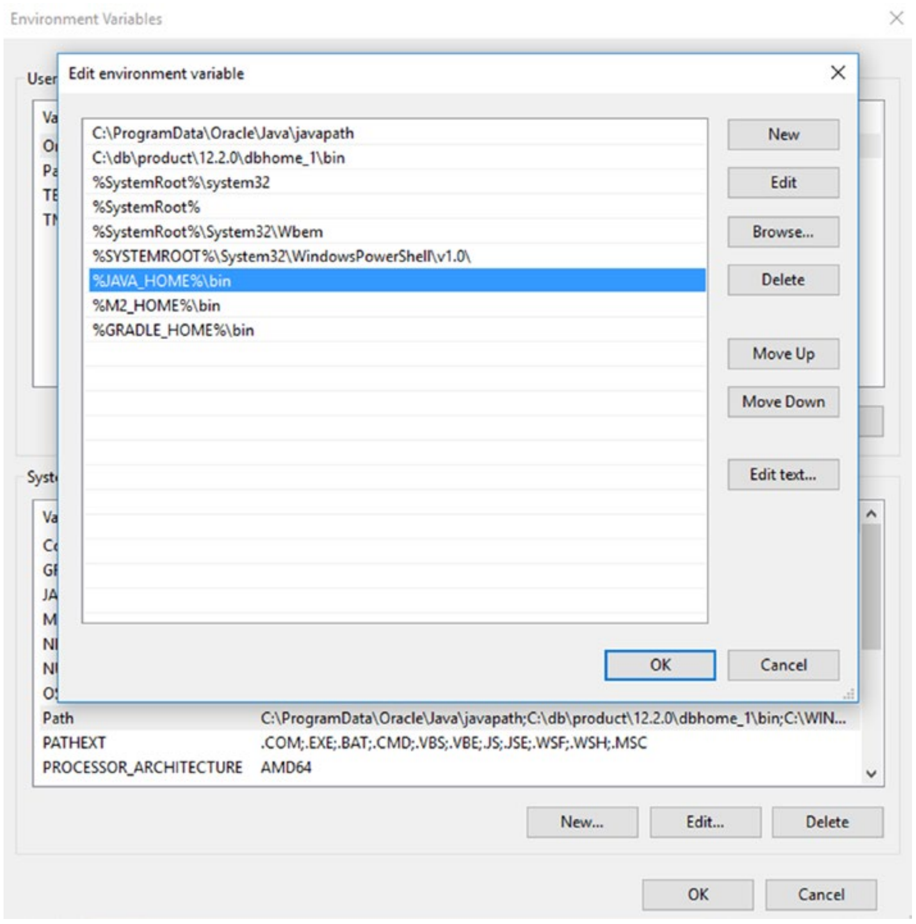
Variable name:	JAVA_HOME
Variable value:	C:\Program Files\Java\jdk-10

System variables

Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
GRADLE_HOME	C:\tools\gradle
JAVA_HOME	C:\Program Files\Java\jdk-10
M2_HOME	C:\tools\maven
NLS_LANG	AMERICAN_AMERICA.WE8ISO8859P15
NUMBER_OF_PROCESSORS	8
OS	Windows_NT
Path	C:\ProgramData\Oracle\Java\javapath;C:\db\product\12.2.0\dbhome_1\bin;C:\WIN...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64

Figure 2-7. *Declaring JAVA_HOME as a system variable on Windows*

After defining the `JAVA_HOME` variable, you need to add the executables to the system path. This can be done by editing the `Path` variable. Just select it from the **System Variables** list and click the **Edit...** button. Starting in Windows 10, each part of the `Path` variable is shown on a different line, so you can add a different line and add `%JAVA_HOME%\bin` on it. This syntax is practical because it takes the location of the `bin` directory from whatever location the `JAVA_HOME` variable contains. The dialog window is depicted in Figure 2-8.



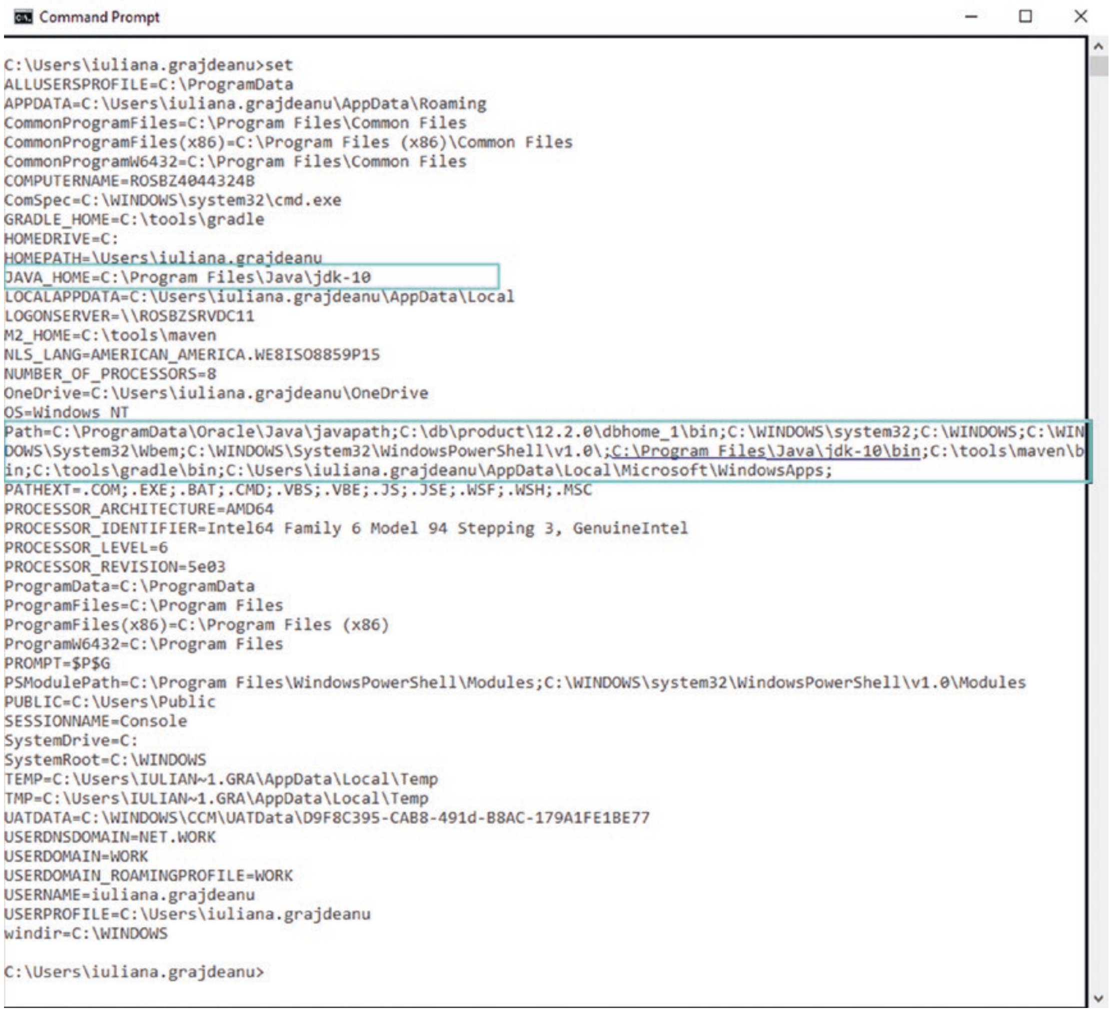
```
C:\ProgramData\Oracle\Java\javapath
C:\db\product\12.2.0\dbhome_1\bin
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%JAVA_HOME%\bin
%M2_HOME%\bin
%GRADLE_HOME%\bin
```

```
Path                    C:\ProgramData\Oracle\Java\javapath;C:\db\product\12.2.0\dbhome_1\bin;C:\WIN...
PATHEXT                 .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE  AMD64
```

Figure 2-8. *Declaring the JDK executables directory as part of the system Path variable on Windows 10*

On older Windows systems, the contents of the `Path` variable are depicted in the dialog box shown in Figure 2-7, so you must add the `%JAVA_HOME%\bin` text in the **Variable value** text field, and separate it from the existing content by using a semicolon (;).

No matter which Windows system you have, you can check that you set everything correctly by opening **Command Prompt** and executing the `set` command. This lists all the system variables and their values. `JAVA_HOME` and `Path` should be there with the desired values. For the setup proposed in this section when executing **set** the output is depicted in Figure 2-9.



```
C:\Users\iuliana.grajdeanu>set
ALLUSERSPROFILE=C:\ProgramData
APPDATA=C:\Users\iuliana.grajdeanu\AppData\Roaming
CommonProgramFiles=C:\Program Files\Common Files
CommonProgramFiles(x86)=C:\Program Files (x86)\Common Files
CommonProgramW6432=C:\Program Files\Common Files
COMPUTERNAME=ROSBZ4044324B
ComSpec=C:\WINDOWS\system32\cmd.exe
GRADLE_HOME=C:\tools\gradle
HOMEDRIVE=C:
HOMEPATH=\Users\iuliana.grajdeanu
JAVA_HOME=C:\Program Files\Java\jdk-10
LOCALAPPDATA=C:\Users\iuliana.grajdeanu\AppData\Local
LOGONSERVER=\\ROSBZSRVDC11
M2_HOME=C:\tools\maven
NLS_LANG=AMERICAN_AMERICA.WE8ISO8859P15
NUMBER_OF_PROCESSORS=8
OneDrive=C:\Users\iuliana.grajdeanu\OneDrive
OS=Windows_NT
Path=C:\ProgramData\Oracle\Java\javapath;C:\db\product\12.2.0\dbhome_1\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\WIN
DOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\Program Files\Java\jdk-10\bin;C:\tools\maven\b
in;C:\tools\gradle\bin;C:\Users\iuliana.grajdeanu\AppData\Local\Microsoft\WindowsApps;
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE=AMD64
PROCESSOR_IDENTIFIER=Intel64 Family 6 Model 94 Stepping 3, GenuineIntel
PROCESSOR_LEVEL=6
PROCESSOR_REVISION=5e03
ProgramData=C:\ProgramData
ProgramFiles=C:\Program Files
ProgramFiles(x86)=C:\Program Files (x86)
ProgramW6432=C:\Program Files
PROMPT=$P$G
PSModulePath=C:\Program Files\WindowsPowerShell\Modules;C:\WINDOWS\system32\WindowsPowerShell\v1.0\Modules
PUBLIC=C:\Users\Public
SESSIONNAME=Console
SystemDrive=C:
SystemRoot=C:\WINDOWS
TEMP=C:\Users\IULIAN~1.GRA\AppData\Local\Temp
TMP=C:\Users\IULIAN~1.GRA\AppData\Local\Temp
UATDATA=C:\WINDOWS\CCM\UATData\D9F8C395-CAB8-491d-B8AC-179A1FE1BE77
USERDNSDOMAIN=NET.WORK
USERDOMAIN=WORK
USERDOMAIN_ROAMINGPROFILE=WORK
USERNAME=iuliana.grajdeanu
USERPROFILE=C:\Users\iuliana.grajdeanu
windir=C:\WINDOWS

C:\Users\iuliana.grajdeanu>
```

***Figure 2-9.** Windows system variables listed with the **set** command*

If you execute the previous command and see the expected output and then execute `java -version` in the **command prompt**, it prints the expected result. You are all set.

```
...> java -version
java version "10-ea" 2018-03-20
Java(TM) SE Runtime Environment 18.3 (build 10-ea+42)
Java HotSpot(TM) 64-Bit Server VM 18.3 (build 10-ea+42, mixed mode)
```

JAVA_HOME on macOS

The location in which JDK is installed is `/Library/Java/JavaVirtualMachines/jdk-10.jdk/Contents/Home`. Your `JAVA_HOME` should point to this location. To do this for the current user, you can do the following:

1. In the `/Users/your.user` directory, create a file named `.bash_profile`.
2. In this file, write the following:

   ```
   export JAVA_HOME=$(/usr/libexec/java_home -v10)

   export PATH=$JAVA_HOME/bin:$PATH
   ```

On macOS, you can simultaneously install multiple Java versions. You can set which version is the one currently used on the system by obtaining the JDK location for the desired version by calling the `/usr/libexec/java_home` command and giving the Java version you are interested in as the argument. The result of executing the command is stored as a value for the `JAVA_HOME` variable.

On my system, I have JDK 8, 9, 10, and 11 installed. If I execute the command, giving an argument to each of the Java versions, look at what happens:

```
$ /usr/libexec/java_home -v11
/Library/Java/JavaVirtualMachines/jdk-11.jdk/Contents/Home

$ /usr/libexec/java_home -v10
/Library/Java/JavaVirtualMachines/jdk-10.jdk/Contents/Home

$ /usr/libexec/java_home -v9
/Library/Java/JavaVirtualMachines/jdk-9.0.4.jdk/Contents/Home

$ /usr/libexec/java_home -v1.8
/Library/Java/JavaVirtualMachines/jdk1.8.0_162.jdk/Contents/Home
```

Depending of the version given as argument, a different JDK location is returned. If you want to test the value of the `JAVA_HOME`, the echo command can help with that.

```
$ echo $JAVA_HOME
/Library/Java/JavaVirtualMachines/jdk-10.jdk/Contents/Home
```

The line `export PATH=$JAVA_HOME/bin:$PATH` adds the contents of the `bin` directory from the JDK location to the system patch. This means that I could open a terminal and execute any of the Java executables under it. For example, I could verify that the Java version set as default for my user is the expected one by executing `java -version`.

```
$ java -version
java version "10-ea" 2018-03-20
Java(TM) SE Runtime Environment 18.3 (build 10-ea+42)
Java HotSpot(TM) 64-Bit Server VM 18.3 (build 10-ea+42, mixed mode)
```

If you do all of this and `java -version` prints the expected result, you are all set.

JAVA_HOME on Linux

! If you are using Linux proficiently, you probably are using a PPA, so you can skip this section. But if you like to control where the JDK is and define your own environment variables, keep reading.

Linux systems are Unix-like operating systems. This is similar to macOS, which is based on Unix. Depending on your Linux distribution, installing Java can be done via the specific package manager or by directly downloading the JDK as a `*.tar.gz` archive from the official Oracle site.

If Java is installed using a package manager, the necessary executables are usually automatically placed in the system path at installation time. That is why in this book, we cover only the cases where you do everything manually, and choose to install Java only for the current user in a location such as `/home/your.user/tools/jdk-10.jdk`,[9] because covering package managers is not the object of the book after all.[10]

[9]Replaces your.user with your actual system username

[10]Linux users do not really need this section anyway.'☺

So, after downloading the JDK archive from the Oracle site and unpacking it at `/home/your.user/tools/jdk-10.jdk`, you need to create a file named either `.bashrc` or `.bash_profile`[11] in your user home directory and add the following to it.

```
export JAVA_HOME=/home/your.user/tools/jdk-10.jdk

export PATH=$JAVA_HOME/bin:$PATH
```

As you can see, the syntax is similar to macOS. To check the location of the JDK and the Java version, same commands mentioned in the macOS section can be used.

Installing Gradle

Gradle 5.x ** The sources attached to this book can be compiled and executed using the Gradle wrapper, which is a batch script on Windows and a shell script for other operating systems. When you start a Gradle build via the wrapper, Gradle automatically downloads and runs the build; thus you do not to really need to install Gradle. Instructions on how to do this can be found by reading the public documentation at `www.gradle.org/docs/current/userguide/gradle_wrapper.html`.

A good practice is to keep code and build tools separate, and for the project attached to this book this is the recommended way to go.

If you decide to use Gradle outside the editor, you can download the binaries only (or if you are curious, you can download the full package, which contains binaries, sources, and documentation) from the official site (`www.gradle.org`), unpack them, and copy the contents somewhere on the hard drive. Create a `GRADLE_HOME` environment variable and point it to the location where you have unpacked Gradle. Also, add `%GRADLE_HOME%\bin` for Windows, or `$GRADLE_HOME/bin` for Unix-based operating systems, to the general path of the system.

Gradle was chosen as a build tool for the sources of this book because of the easy setup, small configuration files, flexibility in defining execution tasks, and because it is practical to learn a build tool—because for medium-sized and large projects, they are a must-have.

[11]On some Linux distributions, the file might already exist, you just need to edit it.

! Verify that the version of Gradle the operating system sees is the one you just installed by opening a terminal (**Command Prompt** in Windows, and any type of terminal you have installed on macOS and Linux) and entering

```
gradle -version
```

You should see something similar to this:

```
------------------------------------------------------------
Gradle 5.0-20180826235923+0000
------------------------------------------------------------
Build time:   2018-08-26 23:59:23 UTC
Revision:     c2edb259761ee18f9a14e271f24ef58530b1300f
Kotlin DSL:   1.0-rc-3
Kotlin:       1.2.60
Groovy:       2.4.15
Ant:          Apache Ant (TM) version 1.9.11 compiled on March 23 2018
JVM:          10 (Oracle Corporation 10+46)
OS:           -- whatever operating system you have --
```

The preceding text is confirmation that Gradle commands can be executed in your terminal; thus, Gradle was installed successfully.

Installing Git

This is an optional section, but as a developer, being familiar with a versioning system is important, so here it is. To install Git on your system, just go to the official page at `https://git-scm.com/downloads` and download the installer. Open the installer and click **Next** until done. This works for Windows and macOS.[12] Yes, it is this easy. You do not need to do anything else.[13] For Linux, you can use your package manager or PPA to install Git.

[12]For macOS, you can use homebrew as well.

[13]Just in case, here is a page with instructions on how to install Git for all operating systems: `https://gist.github.com/derhuerst/1b15ff4652a867391f03`

To test that Git installed successfully on your system, open a terminal (**Command Prompt** in Windows, and any type of terminal you have installed on macOS and Linux) and run `git --version` to see the result that it is printed. It should be the version of Git that you just installed.

```
$ git -version
git version 2.15.1
```

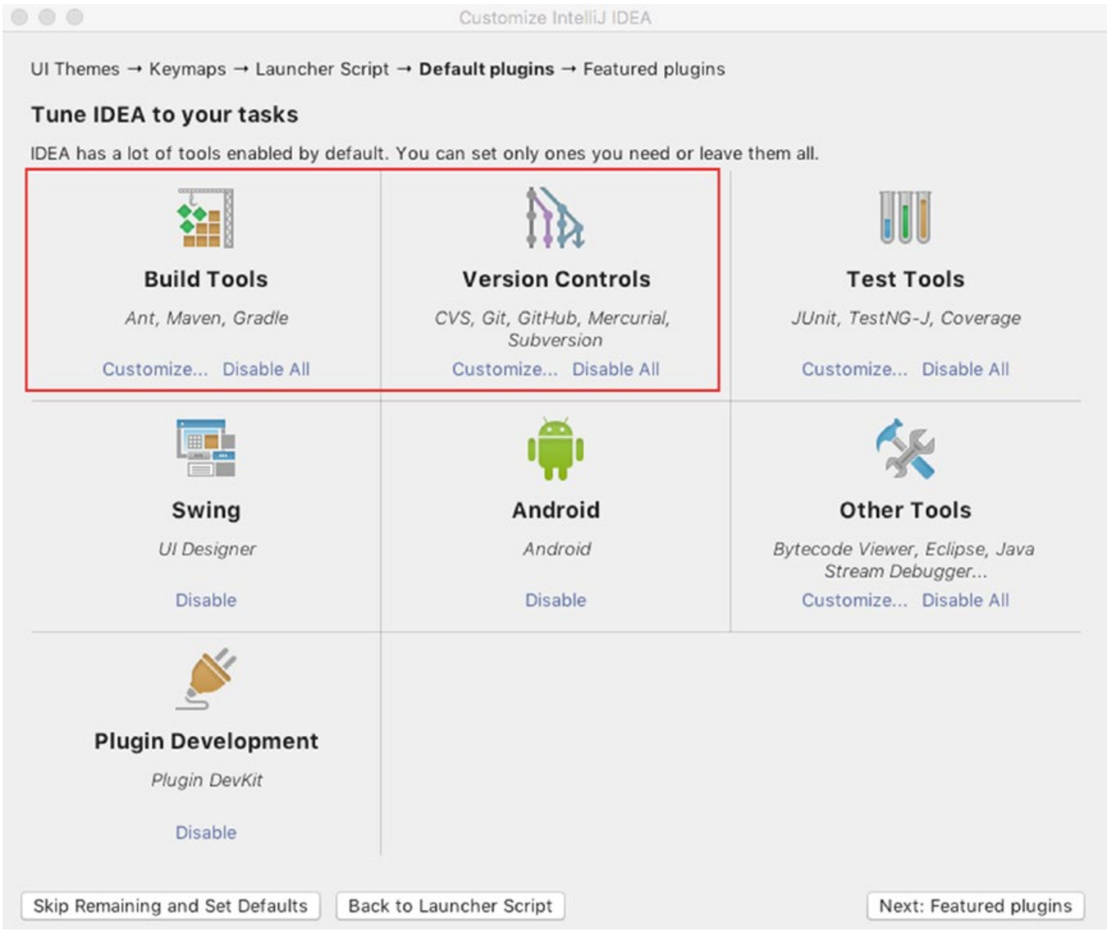
Now that you have Git installed, you can get the sources for this book by cloning the official Git repository in a terminal or directly from the IDE. But more about this a little bit later.

Installing a Java IDE

The editor that I recommend, based on my experience of more than ten years, is IntelliJ IDEA. It is produced by a company called JetBrains. You can download this IDE from their official site at `www.jetbrains.com`. There is an Ultimate Edition available that you can use for free for 30 days; after that, you need to acquire a license. That is why I recommend you download and use the Community Edition,[14] because for the simple development involved in learning Java, this version suffices.

After you download the IntelliJ IDEA archive, double-click it to install it. After that, start it to do a couple of configurations. Just click the **Next** button until you get to the plugin selection step, which should be very similar to the one depicted in Figure 2-10.

[14]The IntelliJ IDEA download page is at `https://www.jetbrains.com/idea/download/`


Customize IntelliJ IDEA
UI Themes → Keymaps → Launcher Script → Default plugins → Featured plugins
Skip Remaining and Set Defaults
Back to Launcher Script
Next: Featured plugins

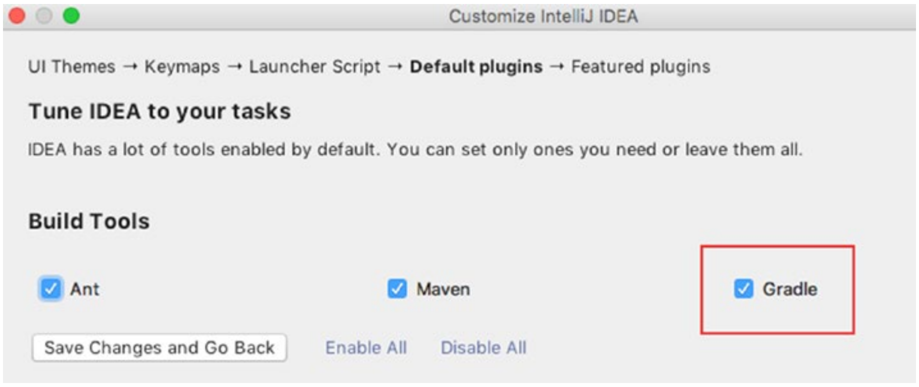
Tune IDEA to your tasks

IDEA has a lot of tools enabled by default. You can set only ones you need or leave them all.

Build Tools *Ant, Maven, Gradle* Customize... Disable All	**Version Controls** *CVS, Git, GitHub, Mercurial, Subversion* Customize... Disable All	**Test Tools** *JUnit, TestNG-J, Coverage* Customize... Disable All
Swing *UI Designer* Disable	**Android** *Android* Disable	**Other Tools** *Bytecode Viewer, Eclipse, Java Stream Debugger...* Customize... Disable All
Plugin Development *Plugin DevKit* Disable		

Figure 2-10. *IntelliJ IDEA Community Edition configure plugins dialog section*

In the previous image, two sections were underlined. The first section configures build tools. If you click **Customize...** button, the window should change to show you the plugins that are available for build tools. Make sure that the option for Gradle is checked, as depicted in Figure 2-11, then click the **Save Changes and Go Back** button.


Customize IntelliJ IDEA

UI Themes → Keymaps → Launcher Script → **Default plugins** → Featured plugins

Tune IDEA to your tasks

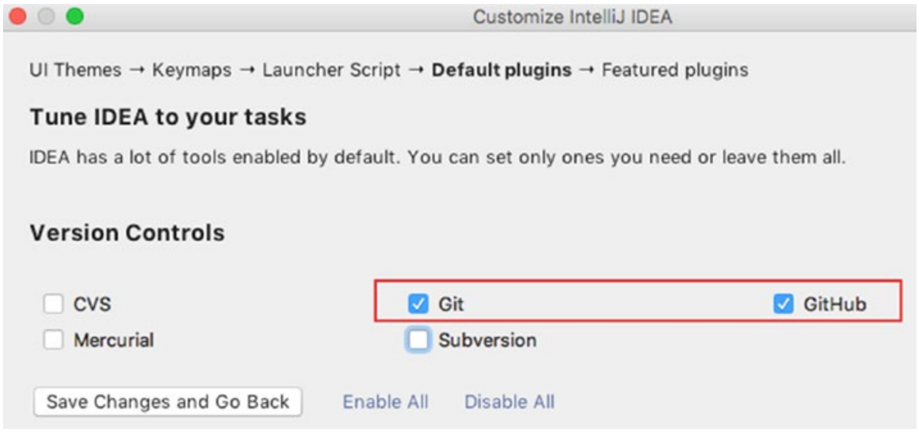
IDEA has a lot of tools enabled by default. You can set only ones you need or leave them all.

Build Tools

☑ Ant ☑ Maven ☑ Gradle

Figure 2-11. *IntelliJ IDEA Community Edition configure Gradle plugin*

The second section configures support for versioning control systems. If you click the **Customize...** button, the window should show you which plugins are available for versioning systems. Make sure that the options for Git and GitHub are checked, as depicted in Figure 2-12, and then click the **Save Changes and Go Back** button. If you go another step forward, you get to another plugin screen that offers you the possibility to install a plugin called **IDE Feature Trainer**. I think if you are a beginner, a plugin might be very useful. The window is depicted in Figure 2-13


Customize IntelliJ IDEA

UI Themes → Keymaps → Launcher Script → **Default plugins** → Featured plugins

Tune IDEA to your tasks

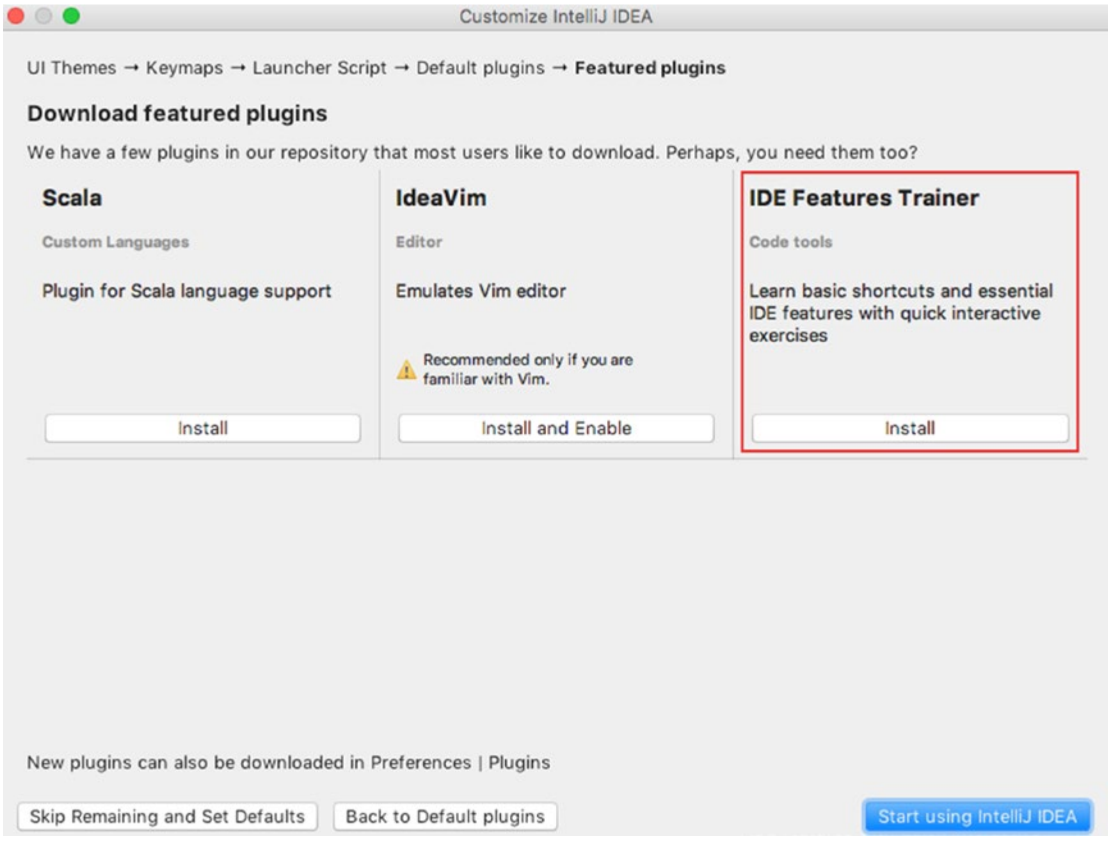
IDEA has a lot of tools enabled by default. You can set only ones you need or leave them all.

Version Controls

☐ CVS ☑ Git ☑ GitHub
☐ Mercurial ☐ Subversion

Figure 2-12. *IntelliJ IDEA Community Edition configure Git plugin*

For the final step, click the **Install** button, and then **Start using IntelliJ IDEA**, and you are all set up and good to go. Your development environment is fully configured and ready for you to write your first Java program.


Customize IntelliJ IDEA

UI Themes → Keymaps → Launcher Script → Default plugins → **Featured plugins**

Download featured plugins

We have a few plugins in our repository that most users like to download. Perhaps, you need them too?

Scala
Custom Languages
Plugin for Scala language support

IdeaVim
Editor
Emulates Vim editor
Recommended only if you are familiar with Vim.

IDE Features Trainer
Code tools
Learn basic shortcuts and essential IDE features with quick interactive exercises

New plugins can also be downloaded in Preferences | Plugins

***Figure 2-13.** IntelliJ IDEA Community Edition configure IDE Feature Trainer plugin*

But before doing that, let's also cover how to retrieve the sources for the book. There are three ways to get the sources for the book:

- Download the zipped package directly from GitHub.
- Clone the repository using a terminal (or Git Bash Shell in Windows) using the following command:

  ```
  git clone git@github.com:Apress/java-for-absolute-
  beginners.git
  ```

- Clone the project using IntelliJ IDEA. For this and cloning from the command line, you need a GitHub user. The following images show all the dialog windows that you see when cloning the project with IntelliJ IDEA. Figure 2-14 shows the window that you see after you start an IntelliJ IDEA instance that was never used. The project is hosted on GitHub, so from the ***Check out from Version Control*** menu, select ***GitHub***. At this point, you to the next dialog window, depicted in Figure 2-15.

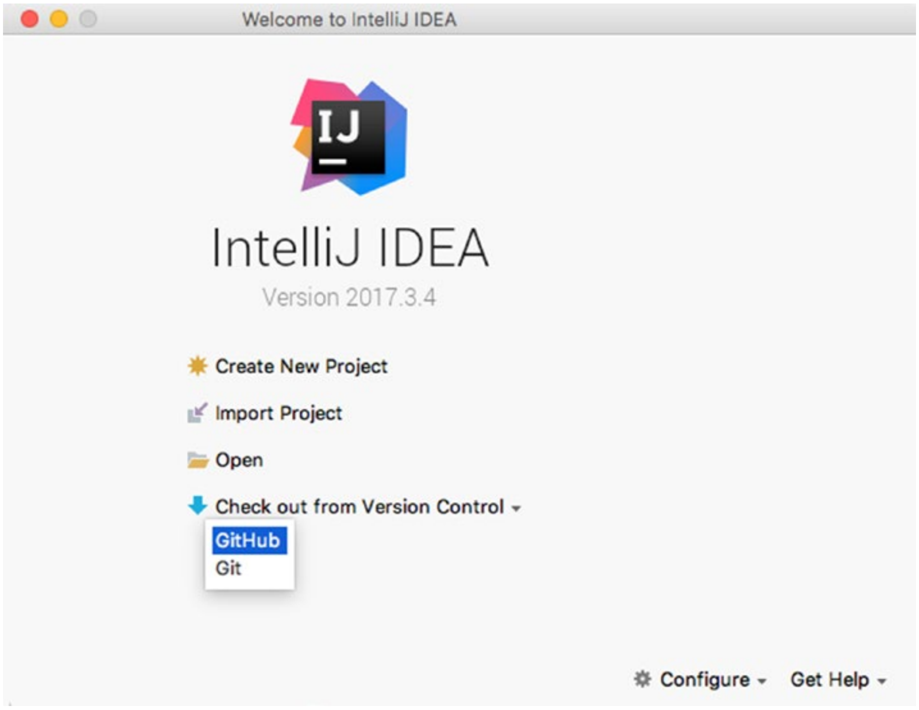
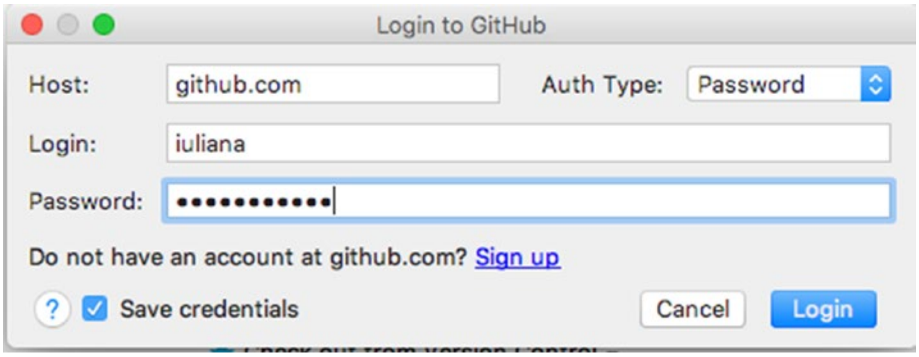


Figure 2-14. *IntelliJ IDEA first dialog window to clone the java-for-absolute-beginners project*


Login to GitHub

Host: github.com Auth Type: Password

Login: iuliana

Password: ●●●●●●●●●●●●

Do not have an account at github.com? Sign up

☑ Save credentials

Figure 2-15. *IntelliJ IDEA second dialog window to clone the java-for-absolute-beginners project*