Learn Android Studio 3 with Kotlin

Efficient Android App Development

Ted Hagos

Apress®

Learn Android Studio 3 with Kotlin

Efficient Android App Development

Ted Hagos

Apress®

Learn Android Studio 3 with Kotlin: Efficient Android App Development

Ted Hagos
Manila, National Capital Region, Philippines

ISBN-13 (pbk): 978-1-4842-3906-3 ISBN-13 (electronic): 978-1-4842-3907-0
https://doi.org/10.1007/978-1-4842-3907-0

Library of Congress Control Number: 2018962941

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

Cover designed by eStudioCalamar

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 rights@apress.com, or visit http://www.apress.com/rights-permissions.

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/9781484239063. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

For Adrianne and Stephanie.

Table of Contents

About the Author

Ted Hagos is the CTO and Data Protection Officer of RenditionDigital International, a software development company based out of Dublin, Ireland. Before he joined RDI, he had various software development roles and also spent time as trainer at IBM Advanced Career Education, Ateneo ITI, and Asia Pacific College. He spent many years in software development dating back to Turbo C, Clipper, dBase IV, and Visual Basic. Eventually, he found Java and spent many years there. Nowadays, he's busy with full-stack JavaScript and Android.

About the Technical Reviewers

Massimo Nardone has more than 24 years of experience in Security, Web/Mobile development, Cloud, and IT Architecture. His true IT passions are Security and Android.

He has been programming and teaching how to program with Android, Perl, PHP, Java, VB, Python, C/C++, and MySQL for more than 20 years.

He holds a Master of Science degree in Computing Science from the University of Salerno, Italy.

He has worked as a Project Manager, Software Engineer, Research Engineer, Chief Security Architect, Information Security Manager, PCI/SCADA Auditor and Senior Lead IT Security/Cloud/SCADA Architect for many years.

Technical skills include: Security, Android, Cloud, Java, MySQL, Drupal, Cobol, Perl, Web and Mobile development, MongoDB, D3, Joomla, Couchbase, C/C++, WebGL, Python, Pro Rails, Django CMS, Jekyll, Scratch, etc.

He worked as visiting lecturer and supervisor for exercises at the Networking Laboratory of the Helsinki University of Technology (Aalto University). He holds four international patents (PKI, SIP, SAML, and Proxy areas).

He currently works as Chief Information Security Officer (CISO) for Cargotec Oyj, and he is a member of ISACA Finland Chapter Board.

Massimo has reviewed more than 45 IT books for different publishers and has coauthored *Pro JPA in Java EE 8* (Apress, 2018), *Beginning EJB in Java EE 8* (Apress, 2018), and *Pro Android Games* (Apress, 2015).

Val Okafor is a software architect with expertise in Android development and resides in sunny San Diego, California. He has over 12 years of industry experience and has worked for corporations such as Sony Electronics, The Home Depot, San Diego County, and American Council on Exercise. Val earned his BSc in IT from National University, San Diego and his Masters in Software Engineering from Regis University, Colorado. He is the creator and principal engineer of Pronto line of mobile apps including Pronto Diary, Pronto Invoice, and Pronto Quotes.

His passion for software development goes beyond his skill and training; he also enjoys sharing his knowledge with other developers. He has taught Android development to over 5,000 students through Udemy, and his blog valokafor.com is considered an essential reading for Android developers. Val was also recently named among the first cohort of Realm MVP program because of his active participation in the Realm database community.

Acknowledgments

To Stephanie and Adrianne, for bearing with me for the past 9 months while I wrote this book. My thanks and my love.

To Mark Powers, for his understanding when I missed some of the writing deadlines and for keeping the schedule straight.

To Steve Anglin, for bringing me to Apress.

To everyone who made this book possible, Thank you. It truly feels great to hold one's printed book in one's hands. It's even more awesome the second time around.

Introduction

Welcome to the Kotlin edition of *Learn Android Studio 3*, This book will help you get started in your programming journey with the little green robot. You already bought the book, so you don't need to be convinced that programming for the mobile platform offers a lot of opportunity for software developers. Thank you for buying it, by the way.

Who This Book Is For

The book is aimed at beginning Android programmers, but it isn't for people who are completely new to programming. Ideally, you already are a Java programmer trying to get your feet wet in Android, and you wanna try the Kotlin language (coz all your dev friends told you it was cool). But in case you're not a Java developer or you don't have Android programming experience, don't sweat it. The book is friendly enough—I tried hard to write it that way—and approachable enough such that anyone with a passing knowledge of either C#, JavaScript, C, or C++ will be able to follow the code samples and the concepts presented in this book.

What's Different in the Kotlin Edition

All the code examples and the demo projects are mostly new. They're not a plain Kotlin port of the first edition's examples. I've also added new chapters; here they are:

- Collections
- Generics
- Higher Order Functions
- Broadcast Receivers

Some chapters in the first edition have been split into two or more chapters. I split them so that I can treat the subjects with more depth—for example, "Intents," "SharedPreferences," "Internal Storage," and "Fragments."

Organization and Treatment

The book is divided into two major parts. Chapters 1 to 7 are all about the Kotlin language, and Chapters 8 to 20 are about Android programming.

While you can use it as a reference book, I didn't write it that way. It's not meant as a substitute for the docs in `https://kotlinglang.org` or the Android developer guides `https://developer.android.com`. It's also not meant to be a "Definitive Guide" type of book where you can spend hours or days exploring every nook and cranny. Quite the contrary—I wanted it to be a "get started quick" type of book, like a recipe book, but without losing our grasp on the fundamental concepts.

Android and Kotlin are big subjects; I don't think there exists a "single best way" to present the materials for either of these two. So, I made certain bets on the instructional design. Here they are:

- **Bite-sized concepts**. The troublesome topics are broken down into a series of small steps so that you can solve them in isolation. When you can solve small problems, it gives you confidence to solve bigger ones. This approach helps a beginning programmer to grow in the direction of skill.
- **Conciseness**. I tried to keep each chapter as short as possible, so you can finish it in one sitting. Originally, I wanted each chapter to be a "20-minute read"; that was too ambitious, so, I gave up on it—but still, the chapters are short.
- **Multiple Learning Curves**. The book is about three topics: Android Studio, Android Programming, and Kotlin. Although Kotlin and Android programming may seem to have dedicated chapters for them, techniques on how to use Android Studio (and IntelliJ) are scattered throughout the book.
- **Balance between concept and code**. Admittedly, the treatment is biased (just a little bit) toward code. Programming is not a spectator sport; we learn by doing. Nonetheless, in every chapter, I tried to explain what the fundamental concepts are, what we're trying to do, what problems are we trying to solve, how we might solve those problems, and what does the solution look like—in code. Almost all of the chapters have one or more demo projects in them.

- **Verbose and complete code presentations**. Sometimes (most of the time actually), I presented the full source example, but only one or two lines of it are relevant. I erred on the side of caution (and verbosity) because it's easier for a beginner to understand the relevant codes if he can see it in relation to the whole program. So, you don't have to worry about, "Where do I put this code? Does this go inside function main or inside a class?"
- **Immediacy and coherence**. Like I said, I wanted this to be a "get started quick" or a "recipe" kind of book. So, instead of covering everything, including the kitchen sink, I chose to cover some topics and ignore others. I chose use-cases whose complexities are easy or moderate and covered topics that are only relevant for those use-cases. For example, in the BroadcastReceiver and Intent chapters, I didn't cover LocalBroadcastManager and PendingIntent. Cool as these topics are, they weren't relevant for the use-cases I chose. If I added more use-cases or demo-projects, that would have stretched the length of the chapter. It's a balancing act, you see.
- **Independent demo projects**. I designed them as such so that the demo project could be started (and followed) from scratch. There is no "putting it all together" project in the end. This way, the book can be conveniently used as a reference. If you pick a topic, it's almost self-contained, including the demo project.

In the end, I can only hope that the bets I made will pay off and that you will walk away as a slightly better programmer after reading the book.

Chapter Overviews

Chapter 1: "Getting into Kotlin" introduces the language. It tells you how to setup Kotlin in various ways on the three major platforms: macOS, Linux, and Windows. It also contains instructions on how to create, configure, and run a project in IntelliJ—this is the IDE I used to create all the Kotlin code samples for Chapters 1 through 7.

Chapter 2: "Kotlin Basics" dives into the language fundamentals of Kotlin. You'll learn the basic building blocks of a Kotlin program (e.g., Strings, control structures,

exception handling, basic data types). You'll also see some of Kotlin's features that are very different from Java, like its treatment of nullable and non-nullable types.

Chapter 3: "Functions." There's a whole chapter dedicated to functions because Kotlin's functions have something new up their sleeves. It has all the trimmings of a modern language like default and named parameters, infix functions, and operators; and with Kotlin, we can also create extension functions. Extension functions lets you add behavior to an existing class, without inheriting from it and without changing its source.

Chapter 4: "Working with Types." This chapter deals with object-oriented topics. You'll learn how Kotlin treats interfaces, classes, and access modifiers. We'll also learn about the new *data classes* in Kotlin. It also talks about *object declarations*—it's the replacement for Java's *static* keyword.

Chapter 5: "Lambdas and Higher Order Functions." Now we go to Kotlins's functional programming capabilities. It discusses how to create and use higher-order functions, lambdas, and closures.

Chapter 6: "Collections" walks through the classic collection classes of Java and how to use them in Kotlin.

Chapter 7: "Generics." Using generics in Kotlin isn't that much different from Java. If generics is old hat for you, then most of this chapter will be a review. But try to read through it still because it talks about *reified generics*, which Java doesn't have.

Chapter 8: "Android Studio Introduction and Setup." This chapter talks a bit about Android's history, its technical make-up, and the OS. It also walks you through the installation and setup of Android Studio.

Chapter 9: "Getting Started" gets you grounded on the fundamental concepts about Android programming. It talks about components, what they are, how they are organized, and how they come together in an Android app. In this chapter, you'll learn how the basic workflow of an Android project—how to create a project and run it on an emulator

Chapter 10: "Activities and Layouts." Here, we'll learn how to build a UI. Activity, Layout, and View objects are the building blocks for an Android UI.

Chapter 11: "Event Handling." You'll learn how to react to user-generated events like clicks and longclicks. We'll use some concepts that we learned in Chapters 4 and 5 (inner objects and lambdas) to help us write more compact and succinct event-handling code.

Chapter 12: "Intents." This chapter reviews some fundamental concepts on Android programming, specifically the concept of components, which dovetails to the topic of Intents. You'll learn how to use Intents to launch another Activity and pass data in-and-around Activities.

Chapter 13: "Themes and Menus." This is a short chapter. You'll learn how to add styles/themes to your app. We'll also work with some menus and the ActionBar.

Chapter 14: "Fragments." You'll learn how to use Android Fragments as a more granular composition unit for UI. We'll also see how to use Fragments to address changes in device orientation.

Chapter 15: "Running in the Background." Any non-trivial app will do something substantial like read from a file, write to a file, download something from the network, etc. These activities will likely take more than 16 ms to execute (you'll learn why 16 ms should be the upper limit and why you should not exceed it). When that happens, the user will see and feel "jank." This chapter discusses the various ways on how to run our code in a background thread.

Chapter 16: "Debugging" shows some of the things you can do to debug your apps in Android Studio 3. It goes through a list of the kinds of errors you might encounter while coding and what you can do in Android Studio to respond them.

Chapter 17: "SharedPreferences." When you need to save simple data, you can use the SharedPreferences API. This chapter walks you through detailed examples on how to do that.

Chapter 18: "Internal Storage." Just like in SharedPreferences, you can also store data using the Internal Storage API of Android. This chapter discusses internal and external storage.

Chapter 19: "BroadcastReceivers." Android has a way to make highly decoupled components talk to each other. This chapter talks about how BroadcastReceivers can facilitate messaging for Android components.

Chapter 20: "App Distribution." When you're ready to distribute your app, you'll need to sign it and list it in a marketplace like Google Play. This chapter walks you through the steps on how to do it.

How to Get the Most From This Book

I designed it like a workbook; it's best to use it like that. Most chapters have a "Demo Project" section. There are details on how to create a project—for example, what name should you use for the project, the minimum SDK to target, etc. The reason I included this information is so you can follow the coding exercise.

I used three kinds of blocks in the book: *Examples*, *Listings,* and *Figures*.

- **Examples** are commands that you would type in a terminal window.
- **Listings** contains program or code listing; it's something that you would type in a program file.
- **Figures** could be screenshots or diagrams. Some of the screenshots are annotated to point out a sequence of steps and how to do them on the IDE. I used Android Studio 3.1 and IntelliJ 2018.2 for the examples in this book; it's possible that by the time you read this book, you'll be using a different or higher version of these tools.

Programmers (mostly) learn by doing. If you work your way through the demo projects, I think the lessons will stick better. Remember that coding is like swimming or driving, you can read as many books as you want on the subjects, but if you don't go in the water or behind the wheel, you won't get anywhere.

Source Code

Source Code for this book can be accessed by clicking the **Download Source Code** button at `www.apress.com/9781484239063`.

PART I

The Kotlin Language

CHAPTER 1

Getting into Kotlin

What we'll cover:

- An introduction to the Kotlin language
- How to get Kotlin
- Installing Kotlin on macOS, Windows, and Linux
- Running a Kotlin program in the command line
- Creating and running a project in IntelliJ IDEA

This chapter introduces the Kotlin language and goes into some details on how to set up a development environment. You will find instructions on how to install Kotlin on macOS, Windows, and Linux. You'll also find instructions on how to install a Kotlin environment using just bare-bones command line. Each developer gravitates to certain kind of setup, and yours truly is not an exception. Here's the setup that I've used throughout the book:

- IntelliJ 2018 running on macOS (High Sierra). I used this throughout chapters 1 to 7
- Android Studio 3 on macOS (High Siera). I used this for the rest of the book

You don't need to follow my exact setup. We've taken pains to ensure that the instructions in this book works in Linux and Windows just as well as they do in macOS. Also, when I say Linux, I don't mean all the distributions of Linux. The fact is, I tested these codes only in Lubuntu 17. Why? Because that's the Linux distro that I'm most familiar with. I believe that most readers of this book (who use Linux) will also be familiar with this Linux distro (or any of its close cousins).

T. Hagos, *Learn Android Studio 3 with Kotlin*, https://doi.org/10.1007/978-1-4842-3907-0_1

Android Studio 3 and IntelliJ works on Windows 7, 8, and 10 (32- and 64-bit), but I only tested the exercises on Windows 10 64-bit—this is the only machine I have access to; and I believe that most readers who use Windows use this setup as well.

Lastly, let's discuss the JDK version. At the time of writing, JDK 10 is in early access. So the choices for JDK version was 8 or 9 (since JDK 7 ended its life sometime in 2015). I went with 9—no special reason, I think 8 would have worked just as well.

About Kotlin

Kotlin is a new language that targets the Java platform; its programs run on the JVM (Java Virtual Machine), which puts it in the company of languages like Groovy, Scala, Jython, and Clojure, to name a few.

Kotlin is from JetBrains, the creators of IntelliJ, PyCharm, WebStorm, ReSharper, and other great development tools. In 2011, JetBrains unveiled Kotlin; the following year, they open-sourced Kotlin under the Apache 2 license. At Google I/O 2017, Google announced first-class support for Kotlin on the Android platform. If you're wondering where the name Kotlin came from, it's the name of an island near St. Petersburg, where most of the Kotlin team members are located. According to Andrey Breslav of JetBrains, Kotlin was named after an island, just like Java was named after the Indonesian island of Java. However, you might remember that the history of the Java language contains references that it was named after the coffee, rather than the island.

Kotlin has many characteristics and capabilities as a language, and we have the whole first part of this book to explore those, but here are a few things that makes it interesting.

- **Like Java, it's object-oriented**. So, all those long hours you've invested in Java's OOP and design pattern won't go to waste. Kotlin classes, interfaces, and generics look and behave quite a lot like those of Java. This is definitely a strength because, unlike other JVM languages (e.g., Scala), Kotlin doesn't look too foreign. It doesn't alienate Java programmers; instead, it allows them to build on their strengths.
- **Statically and strongly typed**. Another area that Kotlin shares with Java is the type system. It also uses static and strong typing. However, unlike in Java, you don't have to always declare the type of the variable before you use it. Kotlin uses *type inference*.

- **Less ceremonious than Java**. We don't (always) have to write a class; top-level functions are OK. We don't need to explicitly write getters and setters for data objects; there are language features in Kotlin, which allows us to do away with such boiler-plate codes. Also, the natural way of writing codes in Kotlin prevents us from ever assigning *null* to a variable. If you want to explicitly allow a value to be *null*, you have to do so in a deliberate way.
- **It's a functional language**. Functions are not just a named collection of statements; you can use them anywhere you might use a variable. You can pass functions from a parameter input to other functions, and you can even return functions from other functions. This way coding allows for a different way of abstraction.
- **Interoperability with Java**. Kotlin can use Java libraries, and you can use it from Java programs as well. This lowers the barrier to entry in Kotlin; the interoperability with Java makes the decision to start a new project using Kotlin a less daunting enterprise.

There are many reasons to use Kotlin in your next project, but there are also counter-arguments to it. We won't list the pros and cons of why you should or why you shouldn't use Kotlin in your next project; but I'll discuss one reason why I would advise you to slow down and pause before you get all gung-ho about it.

It's still relatively new. Some people are convinced that it's approaching its "peak of inflated expectation" and will soon enter the "trough of disillusionment." Their main argument is that if you bet on Kotlin right now, you'll be saddled with learning curve problems and you'll be obligated to maintain that codebase—even if Kotlin disappears in a puff of smoke. In other words, you might carry it as a technical debt.

Kotlin's adoption will also come at some cost. You'll have to train your team on how to use it. No matter how experienced your team is, they will definitely lose some speed along the way—and that's a project management concern. Also, because Kotlin is new, there is no "Effective Kotlin" guide post yet, while Java programmers will always have their "Effective Java."

It will all boil down to your bet. If you bet that Kotlin will go the distance instead of quietly disappearing in the dark, then the bet would have paid off. If you're wrong about the bet, then you go down the arduous road of maintaining the codebase of a defunct language—a technical debt. Either that or you rework it back to Java.

Google has officially supported the language in Android Studio, and more and more developers are getting on the bandwagon. Adoption is growing. These are good signs that Kotlin won't go down quietly and might actually go the distance. Plus, it's a cool language.

Note "Peak of inflated expectation" and "Trough of disillusionment" are part of the the "Hype cycle." The **hype cycle** is a branded graphical presentation developed and used by the American research, advisory, and information technology firm Gartner, for representing the maturity, adoption, and social application of specific technologies. You can read more about it at `https://gtnr.it/cycleofhype`.

Let's continue and build ourselves a dev environment.

Installing the Java SDK

Before we can use Kotlin, we need to install the JDK. If you already have an existing setup of the Java development kit, you can skip this section and jump to the next one (Installing Kotlin). The JDK installer is available for Windows, Linux, and macOS. You can download the currently stable version from the Oracle site, `http://bit.ly/java9download`.[1]

Figure 1-1 shows the download page for Oracle JDK. Choose the installer appropriate for your platform, then click the "Accept License Agreement" to proceed.

[1]Available from `http://www.oracle.com/technetwork/java/javase/downloads/jdk9-downloads-3848520.html`

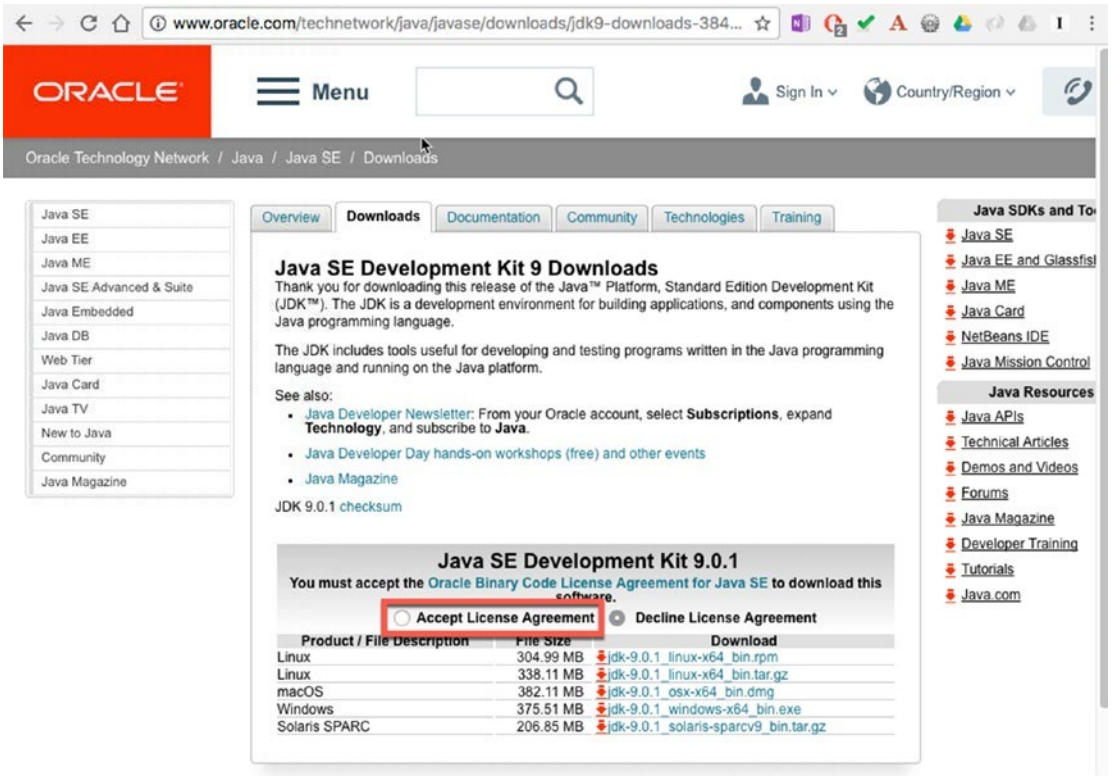

Figure 1-1. *Oracle JDK download page*

Installing on macOS

To install the JDK on macOS, double-click the downloaded **dmg** file and follow the prompts. The installer takes care of updating the system path, so you don't need to perform any further action after the installation.

When you're done with the installation, you can test if the JDK has been installed by launching the "Terminal.app" and trying out the `Java` command (*see* Listing 1-1).

Listing 1-1. Test the JDK tools on a macOS Terminal

```
$ java -version
$ javac -version
```

You'll know that you've installed the JDK without problems if the terminal outputs the version of `java` and `javac` as shown in Figure 1-2.

```
 ted in ~
  java -version
java version "9.0.1"
Java(TM) SE Runtime Environment (build 9.0.1+11)
Java HotSpot(TM) 64-Bit Server VM (build 9.0.1+11, mixed mode)

 ted in ~
  javac -version
javac 9.0.1

 ted in ~
```

Figure 1-2. *java and javac on the Terminal.app*

Installing on Windows 10

You can install Android Studio 3 in Windows 7/8/10 (32- and 64-bit); but for the purpose of this book, I only used Windows 10 64-bit.

To install the JDK on Windows, double-click the downloaded zipped file, and follow the prompts. Unlike in macOS, you must perform extra configuration after the setup. You need to (1) include java/bin in your system path and (2) include a *CLASSPATH* definition in the *Environment Variables* of Windows. Table 1-1 walks you through the steps on how to do this.

Table 1-1. *JDK Configuration in Windows*

1 Include JAVA_HOME/bin to the system path	1. Click **Start ➤ Control Panel ➤ System** 2. Click **Advanced ➤ Environment Variables.** There are two boxes for variables, the upper box reads "User variables" and the lower box reads "System variables," the system `PATH` will be in the "System variables" box. 3. Add the location of the bin folder to the system `PATH` variable. 4. It is typical for the PATH variable to look like this: `C:\WINDOWS\system32;C:\WINDOWS;C:\Program Files\Java\jdk-9\bin;`
2 Create a *CLASSPATH* definition in Windows *Environment Variables*	While the **Environment Variables** window is still open, click the "New" button on the "User variables" section. Another dialog window will pop up with two text boxes that will allow you to add a new variable. Use the values below to populate the textboxes. 1. Name ➤ `CLASSPATH` 2. Value ➤ `C:\WINDOWS\system32;C:\WINDOWS;C:\Program Files\Java\jdk-9\jre\lib\rt.jar;`

Close the Environment Variables window and get a *cmd* window so we can test whether our changes have taken effect. When the cmd window is open, type the commands as shown in Listing 1-2.

Listing 1-2. Test the JDK tools on a Windows cmd shell

```
C:\Users\yourname>java -version
C:\Users\yourname>javac -version
```

If the cmd shell shows you the version of java and javac, then you have successfully installed and configured the JDK. If, on the other hand, you saw an error message (e.g., "Bad command or file name"), it means that JAVA_HOME\bin is still not part of the system path. You should revisit Table 1-1 and recheck your entries, then retest.

Installing on Linux

If you are a Linux user, you may have seen the tar ball and rpm options on the download, you may use that and install it like you would install any other software on your Linux platform or you may install the JDK from the repositories (see Listing 1-3). This instruction applies to Debian and its derivatives (e.g., Ubuntu, Mint, etc.).

Listing 1-3. Installing the JDK in Ubuntu Using a PPA

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java9-installer
sudo update-alternatives --config java
```

When the download finishes, you can test the installation by trying out the `java` and `javac` tools from the command line (see Listing 1-4). Open your favorite terminal emulator (e.g., *xterm, terminator, gnome-terminal, lxterminal*, etc.).

Listing 1-4. Test the JDK Tools on Linux

```
$ java -version
$ javac -version
```

If the install was successful, you should be able to see the version of java and javac in your system. Once the JDK is up and running, we can now get Kotlin.

Installing Kotlin

There are a couple of ways to get started in Kotlin coding. You can use the online IDE, which is the quickest because it won't require you to install anything. You may also try to download an IDE that has a plug-in for Kotlin (e.g., IntelliJ, Android Studio, or Eclipse). Finally, you can download the command line tools for Kotlin. If you don't want to install a full-blown IDE and simply use your trusty favorite editor, you can certainly do that with the command line tools. We won't explore each and every one of these options, but we'll take a look at the command line tools and IntelliJ.

Note This book is about Android Studio, so you might be wondering why we won't use Android Studio to try out Kotlin. That's because this part of the book is about Kotlin only and not about Android programming (yet). I thought it best to focus more on the language and not be hampered by Android-specific topics when we do some coding exercises. Android Studio is based on IntelliJ anyway, so any IDE techniques we learn in this part of the book should carry over nicely when we get to part 2.

Installing the Command Line Tools

Even if you opt for the command line tools, there are a couple of choices for installation method. We can install it by (1) downloading a zipped file; (2) using SDKMAN if your OS and tooling supports it; or (3) using HomeBrew or MacPorts if you are on macOS. You only need to pick which one of these methods you are most comfortable with and go with that.

HomeBrew or MacPort

If you are on macOS and already using either `brew` or `port`, see either Listing 1-5 or 1-6 for the terminal commands to get Kotlin.

Listing 1-5. Install Kotlin Using HomeBrew

```
$ brew update
$ brew install kotlin
```