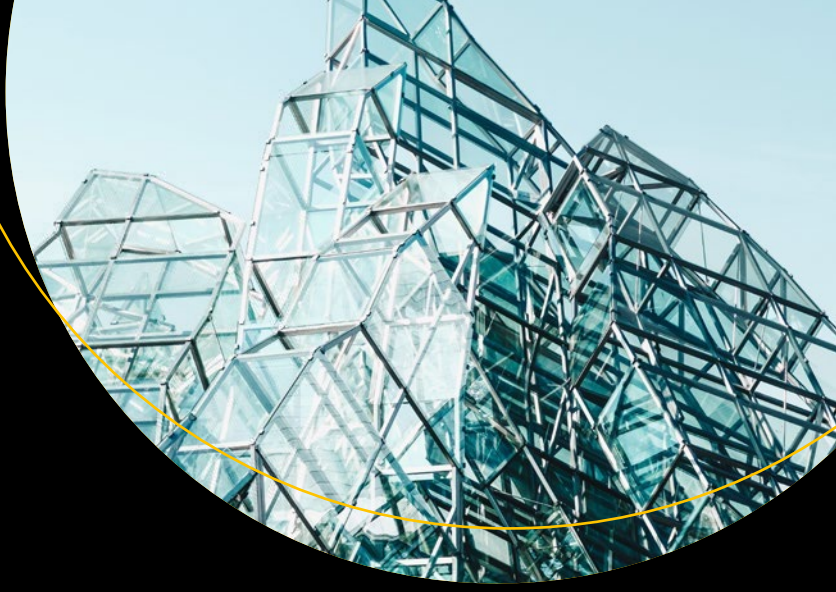

Beginning Kotlin

Build Applications with Better Code, Productivity, and Performance

Ted Hagos

Apress®

Beginning Kotlin

Build Applications with Better Code, Productivity, and Performance

Ted Hagos

Apress®

Beginning Kotlin: Build Applications with Better Code, Productivity, and Performance

Ted Hagos
Makati, Philippines

ISBN-13 (pbk): 978-1-4842-8697-5 ISBN-13 (electronic): 978-1-4842-8698-2
https://doi.org/10.1007/978-1-4842-8698-2

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Steve Anglin
Development Editor: Laura Berendson
Coordinating Editor: Jill Balzano

Cover designed by eStudioCalamar

Cover image by Akira Hojo on Unsplash (www.unsplash.com)

Distributed to the book trade worldwide by Apress Media, LLC, 1 New York Plaza, New York, NY 10004, U.S.A. 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 booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail 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 (https://github.com/Apress). For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

For Steph and Adrianne; and also, Wayne, Amy, plus Bagel.

Table of Contents

About the Author

Ted Hagos is the CTO and Data Protection Officer of RenditionDigital International (RDI), a software development company based out of Dublin. Before he joined RDI, he had various software development roles and also spent time as a 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 working with it. Nowadays, he's busy with Kotlin, full-stack JavaScript, Android, and Spring applications.

About the Technical Reviewer

Manuel Jordan Elera is an autodidactic developer and researcher who enjoys learning new technologies for his own experiments and creating new integrations. Manuel won the Springy Award 2013 Community Champion and Spring Champion. In his little free time, he reads the Bible and composes music on his guitar. Manuel is known as dr_pompeii. He has tech-reviewed numerous books, including *Pro Spring MVC with WebFlux* (Apress, 2020), *Pro Spring Boot 2* (Apress, 2019), *Rapid Java Persistence and Microservices* (Apress, 2019), *Java Language Features* (Apress, 2018), *Spring Boot 2 Recipes* (Apress, 2018), and *Java APIs, Extensions and Libraries* (Apress, 2018). You can read his detailed tutorials on Spring technologies and contact him through his blog at `www.manueljordanelera.blogspot.com`. You can follow Manuel on his `Twitter account, @dr_pompeii`.

Acknowledgments

Thanks to Steve Anglin, Mark Powers, Shonmirin, and Manuel Jordan Elera for making this book possible.

Introduction

Welcome to *Beginning Kotlin*.

Kotlin may have (initially) gained popularity as a drop-in replacement for Java for building Android apps, but it's ubiquitous now. You can use it to build web apps—both on the front and back end. Kotlin is quickly gaining popularity in areas that used to be Java strongholds. If you want to get a head start on Kotlin, this book is an excellent place to start.

Chapter Overviews

Chapter 1 (Introduction to Kotlin) – Brief introduction to the Kotlin language. You'll learn a bit about the language's history and some of its well-known capabilities. The chapter also walks you through the steps on how to install Kotlin and get started with a quick project.

Chapter 2 (A Quick Tour of Kotlin) – Discusses the language elements of Kotlin. You'll learn the syntax with brisk examples. You'll know how to declare strings, variables, and constants. You'll also learn how to move around using Kotlin's control structures for looping, branching, and exception handling.

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 let you add behavior to an existing class without inheriting from it and without changing its source.

Chapter 4 (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 (Higher Order Functions and Lambdas) – 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 are 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 (Debugging) – Kotlin has plenty of language improvements that help us avoid errors. Still, errors are a big part of a programmer's life. This chapter walks you through the steps to do a debugging session in IntelliJ.

Chapter 9 (Unit Testing) – If you're a fan of unit testing—why shouldn't you be?—this is for you. We'll walk through the steps to write and run unit tests using JUnit5.

Chapter 10 (Writing Idiomatic Kotlin) – Kotlin was designed to be very similar to Java, but it also has plenty of features that make it stand out. This chapter points out those unique Kotlin features and shows examples of how to use them.

Chapter 11 (Creating a Spring Boot Project) – A quick introduction to how to create a Spring Boot project using Kotlin.

Source Code and Supplementary Material

All source code used in this book can be downloaded from `github.com/apress/beginning-kotlin`, where you will also find the following online-only extras:

Appendix A (Java Interoperability) – You can use Kotlin classes from Java codes and vice versa. This chapter explores Java-Kotlin interoperability.

Appendix B (Building a Microservices App Part 1) – Spring Boot is an excellent choice for building microservices. This appendix details how to build a sample microservices project.

Appendix C (Building a Microservices App Part 2) – Continues from Appendix B.

Appendix D (Cloud Deployment) – It's the age of cloud-native apps. This chapter discusses the workflow for bringing Spring Boot apps into the cloud.

CHAPTER 1

Introduction to 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.

What we'll cover:

- Overview of the Kotlin language
- How to get Kotlin
- Getting Kotlin running in IntelliJ

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; since then, Kotlin's ubiquity has spread. You can use Kotlin for web development (server-side and/or client-side), desktop development, and of course, mobile development. With Kotlin/Native in the works (still in beta at the time of writing), you should soon be able to write Kotlin codes that compile to native code that can run without a VM. In this book, we will focus on Kotlin for server-side web development—using Spring Boot.

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 is 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

T. Hagos, *Beginning Kotlin*, https://doi.org/10.1007/978-1-4842-8698-2_1

might remember that the history of the Java language contains references that it was named after the coffee, rather than the island.

About Kotlin

It's Simple

```
fun main() {
    val name = "stranger"          // Declare your first variable
    println("Hi, $name!")          // ...and use it!
    print("Current count:")
    for (i in 0..10) {             // Loop over a range
                                      from 0 to 10
        print(" $i")
    }
}
```

As you can see from the preceding code—which I lifted from https://kotlinlang.org—we don't (always) have to write a class. Top-level functions can run without wrapping them inside classes.

It's Asynchronous

```
import kotlinx.coroutines.*

suspend fun main() {                                    // #1
    val start = System.currentTimeMillis()
    coroutineScope {                                    // #2
        for (i in 1..10) {
            launch {                                    // #3
                delay(3000L - i * 300)                  // #4
                log(start, "Countdown: $i")
```

```
            }
        }
    }
                                                    // #5

    log(start, "Liftoff!")
}

fun log(start: Long, msg: String) {
    println("$msg " +
            "(on ${Thread.currentThread().name}) " +
            "after ${(System.currentTimeMillis() -
            start)/1000F}s")
}
```

#1 This is a function that can be suspended and resumed later.

#2 This statement creates a scope for starting coroutines.

#3 Start ten concurrent tasks.

#4 Pause the execution.

#5 Execution continues when all coroutines in the scope have finished.

It's Object-Oriented

```
abstract class Person(val name: String) {
    abstract fun greet()
}

interface FoodConsumer {
    fun eat()
    fun pay(amount: Int) = println("Delicious! Here's $amount
    bucks!")
}
```

```
class RestaurantCustomer(name: String, val dish: String) :
Person(name), FoodConsumer {
    fun order() = println("$dish, please!")
    override fun eat() = println("*Eats $dish*")
    override fun greet() = println("It's me, $name.")
}

fun main() {
    val sam = RestaurantCustomer("Sam", "Mixed salad")
    sam.greet() // An implementation of an abstract function
    sam.order() // A member function
    sam.eat() // An implementation of an interface function
    sam.pay(10) // A default implementation in an interface
}
```

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 Java's. This is definitely a strength because, unlike other JVM languages like Scala, Kotlin doesn't look too foreign. It doesn't alienate Java programmers. Instead, it allows them to build on their strengths.

It's Functional

Kotlin is a functional language. Higher order functions are functions that operate on other functions, either by taking them in as parameters or by returning them.

In a functional language (like Kotlin), functions are not just a named collection of statements. You can use them anywhere you might use a variable. You can pass functions as an argument to other functions, and you can even return functions from other functions. This way, coding allows for a different way of abstraction. Functions are treated as first-class citizens.

```
fun main() {
    // Who sent the most messages?
    val frequentSender = messages
        .groupBy(Message::sender)
        .maxByOrNull { (_, messages) -> messages.size }
        ?.key                                                    // #1
    println(frequentSender) // [Ma]

    // Who are the senders?
    val senders = messages
        .asSequence()                                            // #2
        .filter { it.body.isNotBlank() && !it.isRead }           // #3
        .map(Message::sender)                                    // #4
        .distinct()
        .sorted()
        .toList()                                                // #5
    println(senders) // [Adam, Ma]
}

data class Message(                                              // #6
    val sender: String,
    val body: String,
    val isRead: Boolean = false                                 // #7
)

val messages = listOf(                                           // #8
    Message("Ma", "Hey! Where are you?"),
    Message("Adam", "Everything going according to plan
    today?"),
    Message("Ma", "Please reply. I've lost you!"),
)
```

#1 Get their names.
#2 Make operations lazy (for a long call chain).
#3 Use lambdas ...
#4 ... or member references.
#5 Convert sequence back to a list to get a result.
#6 Create a data class.
#7 Provide a default.
#8 Create a list.

Ideal for Tests

```
// Tests
// The following example works for JVM only
import org.junit.Test
import kotlin.test.*

class SampleTest {
    @Test
    fun `test sum`() {                  // #1
        val a = 1
        val b = 41
        assertEquals(42, sum(a, b), "Wrong result for
        sum($a, $b)")
    }

    @Test
    fun `test computation`() {
        assertTrue("Computation failed") {
            setup()                     // #2
            compute()
        }
    }
}
```

```
// Sources
fun sum(a: Int, b: Int) = a + b
fun setup() {}
fun compute() = true
```

#1 Write test names with whitespaces in backticks
#2 Use lambda returning the test subject
Other things we can say about Kotlin are the following:

- **It's less ceremonious than Java**. We don't need to explicitly write getters and setters for data objects; there are language features in Kotlin which allow 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 statically and strongly typed**. Another area that Kotlin shares with Java are the type system. It also uses static and strong typing. However, unlike in Java, you don't always have to declare the type of the variable before you use it. Kotlin uses *type inference*.
- **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, and the interoperability with Java makes the decision to start a new project using Kotlin a less daunting enterprise.

All the sample codes above are from kotlinlang.org. If you'd like to try the codes without investing time (yet) on IDE setup, go to `https://kotlinlang.org`. You should see the code samples I used above (Figure 1-1).

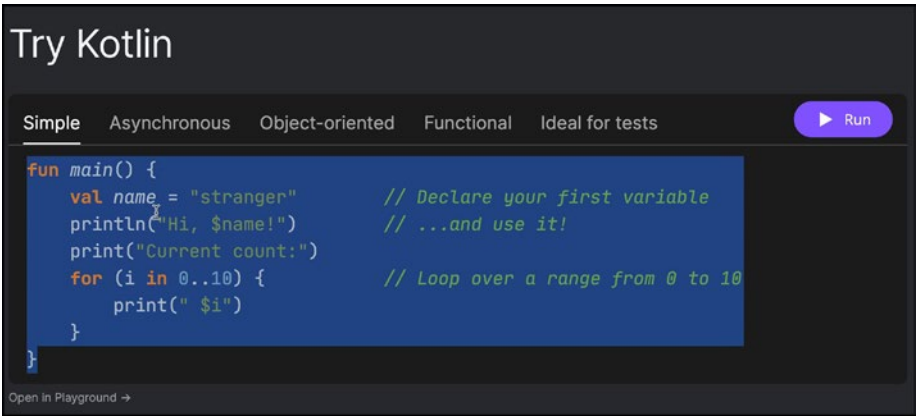

Figure 1-1. *Try Kotlin*

If you click the "Run" button or the "Open in Playground" link, you should be able to run the codes, see the result, and even edit the codes.

The Kotlin playground is good for short snippets and for experimenting. If you intend to do non-trivial coding work in Kotlin, you'll need an IDE.

It is possible to use Kotlin without IntelliJ. You should be able to find Kotlin plugins for Eclipse or NetBeans—you can even use the Kotlin command line SDK if you feel really up to it, but the simplest way to get started is to just use JetBrain's IntelliJ. After all, Kotlin was a brainchild of JetBrains.

Setup and Configuration of IntelliJ

With IntelliJ, you don't need to download the JDK (Java Development Kit) separately; you also don't need to install the Kotlin plugin manually. You can manage both the JDK and Kotlin plugins within IntelliJ.

To download IntelliJ, go to `https://jetbrains.com`, click "Developer Tools", then "IntelliJ IDEA", as shown in Figure 1-2.

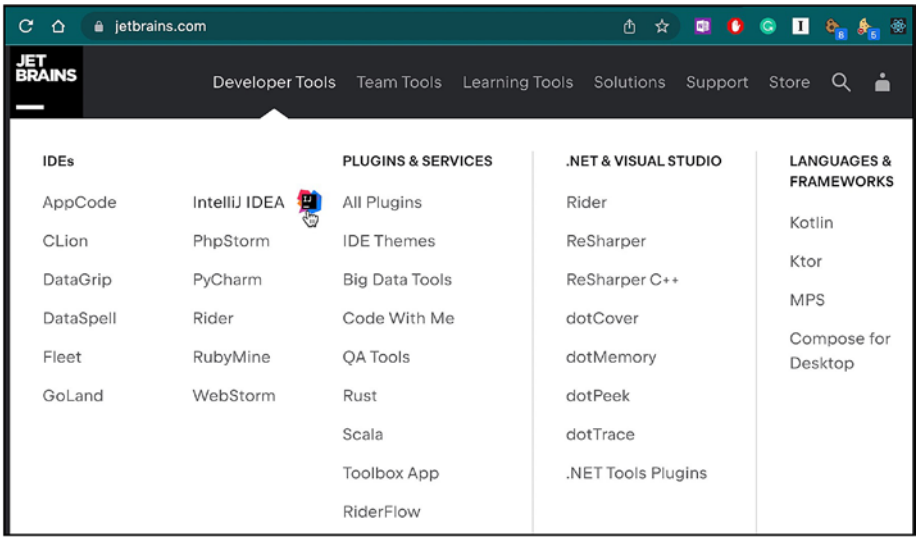

Figure 1-2. *Developer Tools section on the JetBrains site*

Then, in the screen that follows (Figure 1-3), choose either the Community or the Ultimate edition.

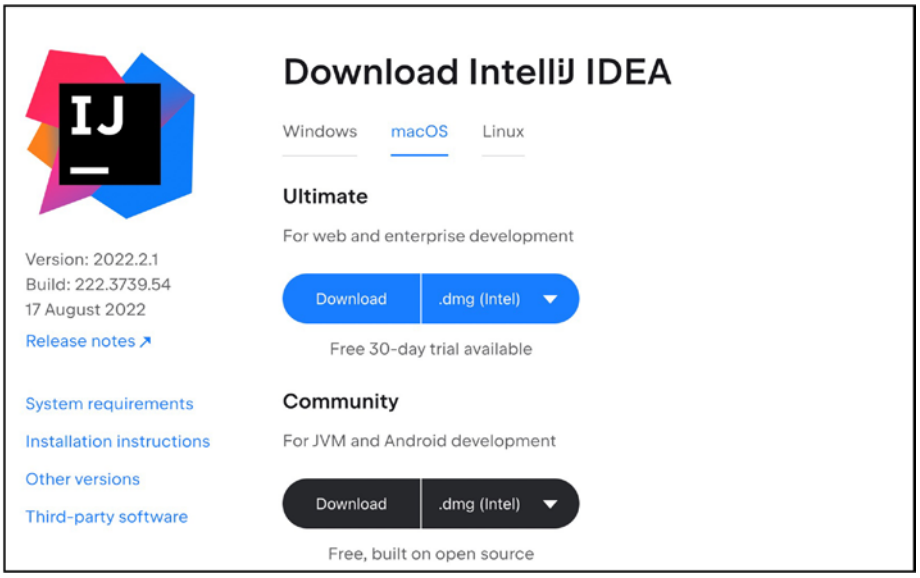

Figure 1-3. *Download IntelliJ*