KOTLIN

FOR ANDROID DEVELOPERS

Learn Kotlin the easy way while developing an Android App.

Antonio Leiva

Kotlin for Android Developers

Learn Kotlin the easy way while developing an Android App

Antonio Leiva

This book is for sale at http://leanpub.com/kotlin-for-android-developers

This version was published on 2016-03-04

Tweet This Book!

Please help Antonio Leiva by spreading the word about this book on Twitter!

The suggested hashtag for this book is #kotlinandroiddev.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#kotlinandroiddev

This book is dedicated to all the loyal readers of antonioleiva.com, who made me believe that writing about Android development was a powerful tool to help others learn about it. I felt this book as a necessary step forward.

*I also want to do a special mention to **Luis Herrero Jiménez**, who has designed the awesome cover of this book, and to Gautier Mechling for helping me so much by reviewing this book. It's thanks to him that this pages are not full of typos and mistakes.*

And, of course, this is specially dedicated to you. With your support and your help this book is growing, and I hope it will become a reference. So any claim or suggestion you think it will improve the quality of this book will be welcomed. Feel free to write anytime to contact@antonioleiva.com.

Contents

I. About this book

In this book, I'll be creating an Android app from ground up using Kotlin as the main language. The idea is to learn the language by example, instead of following a regular reference book structure. I'll be stopping to explain the most interesting concepts and ideas about Kotlin, comparing it with Java 7. This way, you can see what the differences are and which parts of the language will help you speed up your work.

This book is not meant to be a language reference, but a tool for Android developers to learn Kotlin and be able to continue with their own projects by themselves. I'll be solving many of the typical problems we have to face in our daily lives by making use of the language expressiveness and some other really interesting tools and libraries. However, the text covers most Kotlin features, so by the end of the reading you will have a deep knowledge about the language.

The book is very practical, so it is recommended to follow the examples and the code in front of a computer and try everything it's suggested. You could, however, take a first read to get a broad idea and then dive into practice.

As you could read in previous pages (and probably the site where you downloaded), this is a lean publication. This means that the book has been progressing thanks to the readers comments. Even though it is now finished, I will review it from time to time to keep it up to date with new Kotlin versions. So feel free to write and tell me what you think about the book, or what could be improved. I want this book to be the perfect tool for Android developers, and as such, help and ideas will be welcomed.

Thanks for becoming part of this exciting project.

II. Is this book for you?

This book was written to be useful to Android developers who are interested in learning Kotlin language.

This book is for you if you are in some of the following situations:

- You have some basic knowledge about Android Development and the Android SDK.
- You want to learn how to develop Android apps using Kotlin by following an example.
- You need a guide on how to solve many of the common challenges an Android developer finds every day, by using a cleaner and more expressive language.

On the other hand, this book may not be for you. This is what you won't find in it:

- This is not a Kotlin Bible. I'll explain all language basics, and even more complex ideas when they come out during the process, just when we need them. So you will learn by example and not the other way round.
- I will not explain how to develop an Android app. You won't need a deep knowledge of the platform, but you will need some basics, such as some knowledge of Android Studio, Gradle, Java programming and Android SDK. You may even learn some new Android things in the process!
- This is not a guide to learn functional programming. Of course, I'll explain what you need, as Java 7 is not functional at all, but I won't dive deep in functional topics.

III. About the author

Antonio Leiva is an Android Engineer who spends time learning about new ways to get the most out of Android and then writes about it. He writes a blog at antonioleiva.com[1] about many different topics related to Android development.

Antonio started as a consultant in CRM technologies, but after some time, looking for his real passion, he discovered the Android world. After getting some experience on such an awesome platform, he started a new adventure at a mobile company, where he led several projects for important Spanish companies.

He now works as an Android Engineer at Plex[2], where he also plays an important role in the design and UX of the Android applications.

You can find Antonio on Twitter as @lime_cl[3] or Google+ as +AntonioLeivaGordillo[4].

[1] http://antonioleiva.com

[2] http://plex.tv

[3] https://twitter.com/lime_cl

[4] http://plus.google.com/+AntonioLeivaGordillo‘

1 Introduction

You've decided that Java 7 is obsolete and you deserve a more modern language. Congratulations! As you may know, even with Java 8 out there, which includes many of the improvements we would expect from a modern language, we Android developers are still obliged to use Java 7. This is part because of legal issues. But even without this limitation, if new Android devices today started shipping a virtual machine able to run Java 8, we could't start using it until current Android devices are so obsolete that almost nobody uses them. So I'm afraid we won't see this moment soon.

But not everything is lost. Thanks to the use of the Java Virtual Machine (JVM), we can write Android apps using any language that can be compiled to generate bytecode, which JVM is able to understand.

As you can imagine, there are a lot of options out there, such as Groovy, Scala, Clojure and, of course, Kotlin. In practice, only some of them can be considered real alternatives.

There are pros and cons on any of these languages, and I suggest you to take a look to some of them if you are not really sure which language you should use.

1.1 What is Kotlin?

Kotlin, as described before, is a JVM based language developed by JetBrains[5], a company known for the creation of IntelliJ IDEA, a powerful IDE for Java development. Android Studio, the official Android IDE, is based on IntelliJ.

Kotlin was created with Java developers in mind, and with IntelliJ as its main development IDE. And these are two very interesting features for Android developers:

- **Kotlin is very intuitive and easy to learn for Java developers**. Most parts of the language are very similar to what we already know, and the differences in basic concepts can be learnt in no time.
- **We have total integration with our daily IDE for free**. Android Studio can understand, compile and run Kotlin code. And the support for this language comes from the company who develops the IDE, so we Android developers are first-class citizens.

But this is only related to how the language integrates with our tools. What are the advantages of the language when compared to Java 7?

- **It's more expressive**: this is one of its most important qualities. You can write more with much less code.

[5] https://www.jetbrains.com/

- **It's safer**: Kotlin is null safe, which means that we deal with possible null situations in compile time, to prevent execution time exceptions. We need to explicitly specify if an object can be null, and then check its nullity before using it. You will save a lot of time debugging null pointer exception and fixing nullity bugs.
- **It's functional**: Kotlin is basically an object oriented language, not a pure functional language. However, as many other modern languages, it uses many concepts from functional programming, such as lambda expressions, to resolve some problems in a much easier way. Another nice feature is the way it deals with collections.
- **It makes use of extension functions**: This means we can extend any class with new features even if we don't have access to the its source code.
- **It's highly interoperable**: You can continue using most libraries and code written in Java, because the interoperability between both languages is excellent. It's even possible to create mixed project, with both Kotlin and Java files coexisting.

1.2 What do we get with Kotlin?

Without diving too deep in Kotlin language (we'll learn everything about it throughout this book), these are some interesting features we miss in Java:

Expresiveness

With Kotlin, it's much easier to avoid boilerplate because most common situations are covered by default in the language. For instance, in Java, if we want to create a data class, we'll need to write (or at least generate) this code:

```
1  public class Artist {
2      private long id;
3      private String name;
4      private String url;
5      private String mbid;
6
7      public long getId() {
8          return id;
9      }
10
11     public void setId(long id) {
12         this.id = id;
13     }
14
15     public String getName() {
16         return name;
```

```
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getUrl() {
        return url;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public String getMbid() {
        return mbid;
    }

    public void setMbid(String mbid) {
        this.mbid = mbid;
    }

    @Override public String toString() {
        return "Artist{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", url='" + url + '\'' +
                ", mbid='" + mbid + '\'' +
                '}';
    }
}
```

With Kotlin, you just need to make use of a data class:

```
data class Artist(
    var id: Long,
    var name: String,
    var url: String,
    var mbid: String)
```

This data class auto-generates all the fields and property accessors, as well as some useful methods such as `toString()`.

Null Safety

When we develop using Java, a big part of our code is defensive. We need to check continuously whether something is null before using it if we don't want to find unexpected *NullPointerException*. Kotlin, as many other modern languages, is null safe because we need to explicitly specify if an object can be null by using the safe call operator (written `?`).

We can do things like this:

```
// This won't compile. Artist can't be null
var notNullArtist: Artist = null

// Artist can be null
var artist: Artist? = null

// Won't compile, artist could be null and we need to deal with that
artist.print()

// Will print only if artist != null
artist?.print()

// Smart cast. We don't need to use safe call operator if we previously
// checked nullity
if (artist != null) {
    artist.print()
}

// Only use it when we are sure it's not null. Will throw an exception otherwise.
artist!!.print()

// Use Elvis operator to give an alternative in case the object is null.
val name = artist?.name ?: "empty"
```

Extension functions

We can add new functions to any class. It's a much more readable substitute to the usual utility classes we all have in our projects. We could, for instance, add a new method to fragments to show a toast:

```
fun Fragment.toast(message: CharSequence, duration: Int = Toast.LENGTH_SHORT) {
    Toast.makeText(getActivity(), message, duration).show()
}
```

We can now do:

```
fragment.toast("Hello world!")
```

Functional support (Lambdas)

What if, instead of having to implement an anonymous class every time we need to implement a click listener, we could just define what we want to do? We can indeed. This (and many more interesting things) is what we get thanks to lambdas:

```
view.setOnClickListener { toast("Hello world!") }
```

This is only a small selection of what Kotlin can do to simplify your code. Now that you know some of the many interesting features of the language, you may decide this is not for you. If you continue, we'll start with the practice right away in the next chapter.