Kotlin for Android Developers (Preview)

Learn Kotlin in an easy way while developing an Android App

Antonio Leiva

Kotlin for Android Developers (Preview)

Learn Kotlin in an easy way while developing an Android App

Antonio Leiva

©2015 Antonio Leiva

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 typical 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 were 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 until the language is stable enough. So feel free to write and tell me what you think about the book, or what could be improved. In the end, it will also be your book. I want this book to be the perfect tool for Android developers, and as such, all the help and ideas will be welcomed.

Thanks for becoming part of this exciting project.

II. Is this book for you?

This book is written to be useful for 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 a 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 typical situations an Android developer finds everyday, 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 the basics of the language, 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. Nice choice! 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 understand 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 could write Android apps using any language that can be compiled to generate a bytecode the 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, as a platform plugin.

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

Kotlin was created with Java developers in mind, and with IntelliJ as its main development IDE. And these are two very interesting characteristics 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 is perfectly capable to 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.
- **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 code of the class.

- **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 that both languages coexist in the same project.

1.2 What do we get with Kotlin?

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

Expresiveness

With Kotlin, it's much easier to avoid boilerplate because most typical 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;
```

```
17      }
18
19      public void setName(String name) {
20          this.name = name;
21      }
22
23      public String getUrl() {
24          return url;
25      }
26
27      public void setUrl(String url) {
28          this.url = url;
29      }
30
31      public String getMbid() {
32          return mbid;
33      }
34
35      public void setMbid(String mbid) {
36          this.mbid = mbid;
37      }
38
39      @Override public String toString() {
40          return "Artist{" +
41                  "id=" + id +
42                  ", name='" + name + '\'' +
43                  ", url='" + url + '\'' +
44                  ", mbid='" + mbid + '\'' +
45                  '}';
46      }
47  }
```

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, most of our code is defensive. We need to check continuously if 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 d\
eal with that
artist.print()

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

// Smart cast. We don't need to use safe call operator \
```

```
15  if we previously
16  // checked nullity
17  if (artist != null) {
18      artist.print()
19  }
20
21  // Only use it when we are sure it's not null. Will thr\
22  ow an exception otherwise.
23  artist!!.print()
24
25  // Use Elvis operator to give an alternative in case th\
26  e object is null.
27  val name = artist?.name ?: "empty"
```

Extension functions

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

```
1  fun Fragment.toast(message: CharSequence, duration: Int\
2   = Toast.LENGTH_SHORT) {
3      Toast.makeText(getActivity(), message, duration).sh\
4  ow()
5  }
```

We can now do:

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

Functional support (Lambdas)

What if, instead of having to write the creation of a new listener every time we need to declare what a click should do, we could just

define what we want to do? We can indeed. This (and many more interesting things) is what we get thanks to lambda usage:

```
1  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.

2 Getting ready

Now that you know some little examples of what you may do with Kotlin, I'm sure you want to start to put it into practice as soon as possible. Don't worry, these first chapters will help you configure your development environment so that you can start writing some code immediately.

2.1 Android Studio

First thing you need is to have Android Studio installed. As you may know, Android Studio is the official Android IDE, which was publicly presented in 2013 as a preview and finally released in 2014.

Android Studio is implemented as a plugin over IntelliJ IDEA[6], a Java IDE created by Jetbrains[7], the company which is also behind Kotlin. So, as you can see, everything is tightly connected.

The adoption of Android Studio was an important change for Android developers. First, because we left behind the buggy Eclipse and moved to a software specifically designed for Java developers, which gives us a perfect interaction with the language. We enjoy awesome features such as a fast and impressively smart code completion, or really powerful analyzing and refactor tools among others.

And second, Gradle[8] became the official build system for Android, which meant a whole bunch of new possibilities related to version building and deploy. Two of the most interesting functions are build systems and flavours, which let you create infinite versions of the

[6] https://www.jetbrains.com/idea
[7] https://www.jetbrains.com
[8] https://gradle.org/

app (or even different apps) really easily while using the same code base.

If you are still using Eclipse, I'm afraid you need to switch to Android Studio if you want to practice many of the parts included in this book. The Kotlin team is creating a plugin for Eclipse, but it will be always far behind from the one in Android Studio, and the integration won't be so perfect. You will also discover what you are missing really soon as you start using it.

I'm not covering the use of Android Studio or Gradle because this is not the focus of the book, but if you haven't used these tools before, don't panic. I'm sure you'll be able to follow the book and learn the basics in the meanwhile.

Download Android Studio from the official page[9] if you don't have it already.

2.2 Install Kotlin plugins

The IDE by itself is not able to understand Kotlin. As I mentioned in the previous section, it was designed to work with Java. But the Kotlin team has created a powerful set of plugins which will make our lives easier. Go to the plugins section inside Android Studio Preferences, and install these two plugins:

- **Kotlin**: This is the basic plugin. It will let you use Android Studio to write and interact with Kotlin code. It's updated every time a new version of the language is released, so that we can make use of the new features and find alternatives and warnings of deprecations. This is the only plugin you need to write Android apps using Kotlin. But we'll be using another one.

[9] https://developer.android.com/sdk/index.html

- **Kotlin Android Extensions**: the Kotlin team has also released another interesting plugin for Android developers. These Android Extensions will let you automatically inject all the views in an XML into an activity, for instance, without the need of using `findViewById()`. You will get an property casted into the proper view type right away. You will need to install this plugin to use this interesting feature. We'll talk deeper about it in next chapters.

Now our environment is ready to understand the language, compile it and execute it just as seamlessly as if we were using Java.

3 Creating a new project

If you are already used to Android Studio and Gradle, this chapter will be quite easy. I don't want to give many details nor screens, because UI changes from time to time and these lines won't be useful anymore.

Our app is consisting on a simple weather app, such as the one used in Google's Beginners Course in Udacity[10]. We'll be probably paying attention to different things, but the idea of the app will be the same, because it includes many of the things you will find in a typical app. If your Android level is low I recommend this course, it's really easy to follow.

3.1 Create the project in Android Studio

First of all, open Android Studio and choose `Create new Project`. It will ask for a name, you can call it whatever you want: `WeatherApp` for instance. Then you need to set a Company Domain. As you are not releasing the app, this field is not very important either, but if you have your own domain, you can use that one. Also choose a location for the project, wherever you want to save it.

In next step, you'll be asked about the minimum API version. We'll select API 15, because one of the libraries we'll be using needs API 15 as minimum. You'll be targeting most Android users anyway. Don't choose any other platform rather than `Phone and Tablet` for now.

Finally, we are required to choose an activity template to start with. We can choose `Add no Activity` and start from scratch (that would

[10] https://www.udacity.com/course/android-development-for-beginners--ud837

be the best idea when starting a Kotlin project), but we'll rather choose `Empty Activity` because I'll show you later an interesting feature in the Kotlin plugin.

Don't worry much about the name of the activities, layouts, etc. that you will find in next screen. We'll change them later if we need to. Press `Finish` and let Android Studio do its work.

3.2 Configure Gradle

The Kotlin plugin includes a tool which does the Gradle configuration for us. But I prefer to keep control of what I'm writing in my Gradle files, otherwise they can get messy rather easily. Anyway, it's a good idea to know how things work before using the automatic tools, so we'll be doing it manually this time.

First, you need to modify the parent `build.gradle` so that it looks like this:

```
1  buildscript {
2      ext.support_version = '23.1.1'
3      ext.kotlin_version = '1.0.0-beta-2423'
4      ext.anko_version = '0.7.3'
5      repositories {
6          jcenter()
7      }
8      dependencies {
9          classpath 'com.android.tools.build:gradle:1.3.1'
10         classpath "org.jetbrains.kotlin:kotlin-gradle-p\
11 lugin:$kotlin_version"
12     }
13 }
14
15 allprojects {
16     repositories {
```

```
        jcenter()
    }
}
```

As you can see, we are creating a variable which saves current Kotlin version. Check which version is available when you're reading these lines, because there's probably a new version. We need that version number in several places, for instance in the new dependency you need to add for the Kotlin plugin. You'll need it again in the module `build.gradle`, where we'll specify that this module uses the **Kotlin plugin**.

We'll do the same for the support library, as well as Anko library. This way, it's easier to modify all the versions in a row, as well as adding new libraries that use the same version without having to change the version everywhere.

We'll add the dependencies to **Kotlin standard library**, **Anko** library and the **Kotlin Android Extensions plugin**.

```
apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'

android {
    ...
}

dependencies {
    compile "com.android.support:appcompat-v7:$support_\
version"
    compile "org.jetbrains.kotlin:kotlin-stdlib:$kotlin\
_version"
    compile "org.jetbrains.anko:anko-sdk15:$anko_versio\
n"
    compile "org.jetbrains.anko:anko-support-v4:$anko_v\
ersion"
```

```
}

buildscript {
    repositories {
        jcenter()
    }
    dependencies {
      classpath "org.jetbrains.kotlin:kotlin-android-ex\
tensions:$kotlin_version"
    }
}
```

Anko library needs a couple of dependencies. The first one refers to the minimum supported SDK. It's important not to be higher than the minimum SDK defined in your `build.gradle`. `sdk19`, `sdk21` and `sdk23` are also available. The second one adds extra features to the `support-v4` library, which we are implicitly using when importing `appcompat-v7`.

3.3 Convert MainActivity to Kotlin code

An interesting feature the Kotlin plugin includes is the ability to convert from Java to Kotlin code. As any automated process, it won't be perfectly optimised, but it will help a lot in your first days until you start getting used to Kotlin language.

So we are using it in our `MainActivity.java` class. Open the file and select `Code -> Convert Java File to Kotlin File`.

3.4 Test that everything works

We're going to add some code to test Kotlin Android Extensions are working. I'm not explaining much about it yet, but I want to be

sure this is working for you. It's probably the trickiest part in this configuration.

First, go to `activity_main.xml` and set an id for the `TextView`:

```
<TextView
    android:id="@+id/message"
    android:text="@string/hello_world"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

Now, add the synthetic import to the activity (don't worry if you don't understand much about it yet):

```
import kotlinx.android.synthetic.activity_main.*
```

At `onCreate`, you can now get access to that `TextView` directly:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    message.text = "Hello Kotlin!"
}
```

Thanks to Kotlin interoperability with Java, we can use setters and getters methods from Java libraries as a property in Kotlin. We'll talk about properties later, but just notice that we can use `message.text` instead of `message.setText` for free. The compiler will use the real Java methods, so there's no performance overhead when using it.

Now run the app and see everything it's working fine. Check that the message `TextView` is showing the new content. If you have any doubts or want to review some code, take a look at Kotlin for

Android Developers repository[11]. I'll be adding a new commit for every chapter, when the chapter implies changes in code, so be sure to review it to check all the changes.

Next chapters will cover some of the new things you are seeing in the converted `MainActivity`. Once you understand the slight differences between Java and Kotlin, you'll be able to create new code by yourself much easier.

[11] https://github.com/antoniolg/Kotlin-for-Android-Developers

4 Classes and functions

Classes in Kotlin follow a really simple structure. However, there are some slight differences from Java that you will want to know before we continue. You can use try.kotlinlang.org[12] to test this and some other simple examples without the need of a real project and deploy to a device.

4.1 How to declare a class

If you want to declare a class, you just need to use the keyword `class`:

```
1  class MainActivity {
2
3  }
```

Classes have a unique default constructor. We'll see in future lessons that we can create extra constructors for some edge cases, but keep in mind that most situations only require a single constructor. Parameters are written just after the name. Brackets are not needed in a class if it doesn't have any content:

```
1  class Person(name: String, surname: String)
```

Where's the body of the constructor then? You can declare an `init` block:

[12] http://try.kotlinlang.org/

```
class Person(name: String, surname: String) {
        init {
        ...
        }
}
```

4.2 Class inheritance

By default a class always extends from `Any` (similar to Java `Object`), but we can extend any other classes. Classes are closed by default (final), so we can only extend a class if it's explicitly declared as `open` or `abstract`:

```
open class Animal(name: String)
class Person(name: String, surname: String) : Animal(na\
me)
```

Note that when using the single constructor nomenclature, we need to specify the parameters we're using for the parent constructor. That's the substitution to `super()` call in Java.

4.3 Functions

Functions (our methods in Java) are declared just using the **`fun`** keyword:

```
fun onCreate(savedInstanceState: Bundle?) {
}
```

It you don't specify a return value, it will return `Unit`, similar to `void` in Java, though this is really an object. You can, of course, specify any type for the return value:

```
fun add(x: Int, y: Int) : Int {
        return x + y
}
```

Tip: Semi-colons are not necessary

As you can see in the previous example, I'm not using semi-colons at the end of the sentences. While you can use them, semi-colons are not necessary and it's a good practice not to use them. When you get used, you'll find that it saves you a lot of time.

However, if the result can be calculated using a single expression, you can get rid of brackets and use equal:

```
fun add(x: Int, y: Int) : Int = x + y
```

4.4 Constructor and functions parameters

Parameters in Kotlin are a bit different from Java. As you can see, we first write the name of the parameter and then its type.

```
fun add(x: Int, y: Int) : Int {
        return x + y
}
```

An extremely useful thing about parameters is that we can make them optional by specifying a **default value**. Here it is an example of a function you could create in an activity, which uses a toast to show a message:

```
fun toast(message: String, length: Int = Toast.LENGTH_S\
HORT) {
    Toast.makeText(this, message, length).show()
}
```

As you can see, the second parameter (`length`) specifies a default value. This means you can write the second value or not, which avoids the need of function overloading:

```
toast("Hello")
toast("Hello", Toast.LENGTH_LONG)
```

This would be equivalent to the next code in Java:

```
void toast(String message){
    toast(message, Toast.LENGTH_SHORT);
}

void toast(String message, int length){
    Toast.makeText(this, message, length).show();
}
```

And this can be as complex as you want. Check this other example:

```
fun niceToast(message: String,
              tag: String = MainActivity::class.java.si\
mpleName,
              length: Int = Toast.LENGTH_SHORT) {
    Toast.makeText(this, "[$tag] $message", length).sho\
w()
}
```

I've added a third parameter that includes a tag which defaults to the class name. The amount of overloads we'd need in Java grows exponentially. You can now make these calls:

```
1 toast("Hello")
2 toast("Hello", "MyTag")
3 toast("Hello", "MyTag", Toast.LENGTH_SHORT)
```

And there is even another option, because you can use **named arguments**, which means you can write the name of the argument preceding the value to specify which one you want:

```
1 toast(message = "Hello", length = Toast.LENGTH_SHORT)
```

Tip: String templates

You can use template expressions directly in your strings. This will help you write complex strings based on fixed and variable parts in a really simple way. In the previous example, I used `"[$className] $message"`.

As you can see, anytime you want to add an expression, you need to use the `$` symbol. If the expression is a bit more complex, you'll need to add a couple of brackets: `"Your name is ${user.name}"`.

5 Writing your first class

We already have our `MainActivity.kt` class. This activity will render a list of daily forecast for the next week, so the layout requires some changes.

5.1 Creating the layout

The main control that will support the forecast list will be a `RecyclerView`, so you need to add a new dependency to the `build.gradle`:

```
1  dependencies {
2      compile fileTree(dir: 'libs', include: ['*.jar'])
3      compile "com.android.support:appcompat-v7:$support_\
4  version"
5      compile "com.android.support:recyclerview-v7:$suppo\
6  rt_version"
7      ...
8  }
```

Now, in `activity_main.xml` :

```
<FrameLayout xmlns:android="http://schemas.android.com/\
apk/res/android"
             android:layout_width="match_parent"
             android:layout_height="match_parent">

    <android.support.v7.widget.RecyclerView
        android:id="@+id/forecast_list"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</FrameLayout>
```

In `MainActivity.kt`, remove the line we added to test everything worked (it will be showing an error now). For the moment, we'll continue using the good old `findViewById()`:

```
val forecastList = findViewById(R.id.forecast_list) as \
RecyclerView
forecastList.layoutManager = LinearLayoutManager(this)
```

As you can see, we define the variable and cast it to `RecyclerView`. It's a bit different from Java, and we'll see those differences in the next chapter. A `LayoutManager` is also specified, using the property naming instead of the setter. A list will be enough for this layout.

Object instantiation

Object instantiation presents some differences from Java too. As you can see, we omit the "`new`" word. The constructor call is still there, but we save four precious characters. `LinearLayoutManager(this)` creates an instance of the object.

5.2 The Recycler Adapter

We need an adapter for the recycler too. I talked about `RecyclerView` in my blog[13] some time ago, so it may help you if your are not used to it.

The views used in the `RecyclerView` will be just `TextViews` for now, and a simple list of texts that we'll create manually. Add a new Kotlin file called `ForecastListAdapter.kt`, and include this code:

```
public class ForecastListAdapter(val items: List<String\
>) :
        RecyclerView.Adapter<ForecastListAdapter.ViewHo\
lder>() {

    override fun onCreateViewHolder(parent: ViewGroup, \
viewType: Int):
            ForecastListAdapter.ViewHolder? {
        return ViewHolder(TextView(parent.context))
    }

    override fun onBindViewHolder(holder: ForecastListA\
dapter.ViewHolder,
                                  position: Int) {
        holder.textView.text = items.get(position)
    }

    override fun getItemCount(): Int = items.size()

    class ViewHolder(val textView: TextView) : Recycler\
View.ViewHolder(textView)
}
```

Again, we can access to the context and the text as properties. You

[13] http://antonioleiva.com/recyclerview/

can keep doing it as usual (using getters and setter), but you'll get a warning from the compiler. This check can be disabled if you prefer to keep using the Java way. Once you get used to properties you will love it anyway, and it saves some amount of extra characters.

Back to the `MainActivity`, now just create the list of strings and then assign the adapter:

```
1  private val items = listOf(
2      "Mon 6/23 - Sunny - 31/17",
3      "Tue 6/24 - Foggy - 21/8",
4      "Wed 6/25 - Cloudy - 22/17",
5      "Thurs 6/26 - Rainy - 18/11",
6      "Fri 6/27 - Foggy - 21/10",
7      "Sat 6/28 - TRAPPED IN WEATHERSTATION - 23/18",
8      "Sun 6/29 - Sunny - 20/7"
9  )
10
11 override fun onCreate(savedInstanceState: Bundle?) {
12     ...
13     val forecastList = findViewById(R.id.forecast_list)\
14  as RecyclerView
15     forecastList.layoutManager = LinearLayoutManager(th\
16 is)
17     forecastList.adapter = ForecastListAdapter(items)
18 }
```

List creation

Though I'll talk about collections later on this book, I just want to explain for now that you can create constant lists (what we will see as immutable soon) by using the helper function `listOf`. It receives a `vararg` of items of any type and infers the type of the result.

There are many other alternative functions, such as `setOf`, `arrayListOf` or `hashSetOf` among others.

I also moved some classes to new packages, in order to improve organisation.

We saw many new things in such a small amount of code, so I'll be covering it in the next chapter. We can't continue until we learn some important concepts about basic types, variables and properties.

Get the book!

Hope you enjoyed this preview of **Kotlin for Android Developers**. As a thank you for being part of the newsletter and reading up to this point, I'm giving you a 15% discount[14] on it, so that you can keep learning about it.

Thanks for becoming part of this. Please contact if you need something I can help with, at contact@antonioleiva.com[15]

Best,

Antonio Leiva.

[14] http://goo.gl/taoYJh

[15] mailto:contact@antonioleiva.com