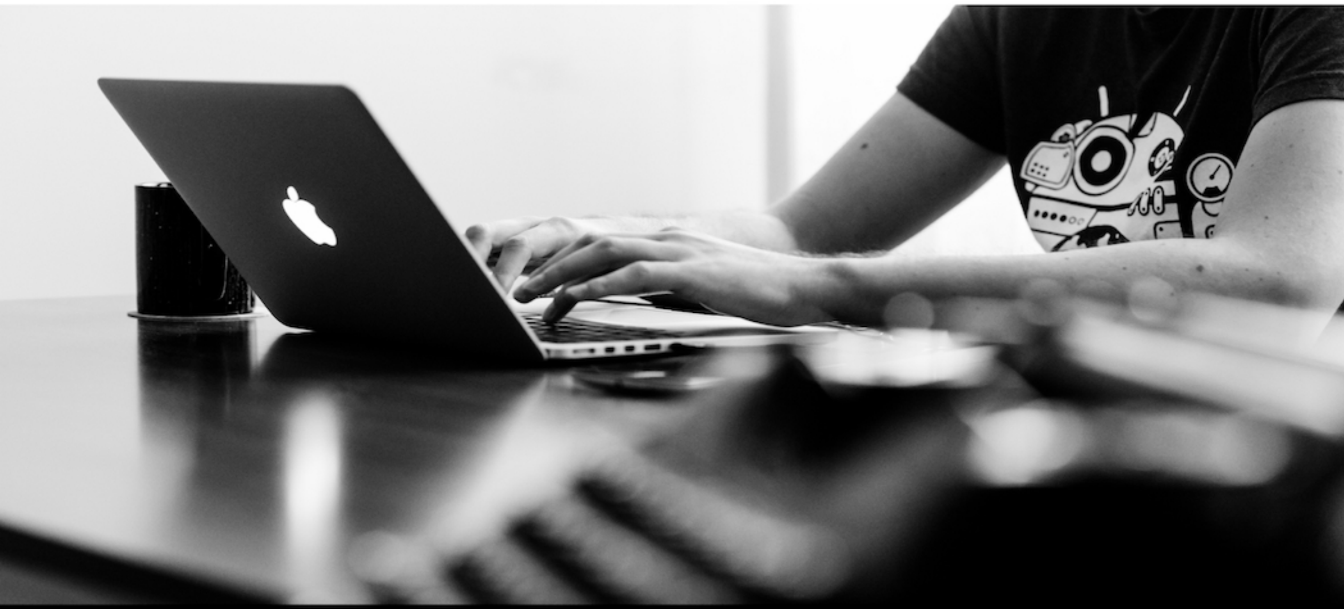
5 TOP GUIDES TO

MASTERING KOTLIN

Learn the tricks that will let you take the most out of the language

ANKO | KOTLIN DSL | COROUTINES
KOTLIN ANDROID EXTENSIONS |JAVA INTEROPERABILITY



antonioleiva.com

5 top guides to mastering Kotlin

Learn the tricks that will let you take the most out of the language

Antonio Leiva

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

This version was published on 2018-09-25

Tweet This Book!

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

The suggested tweet for this book is:

I just got ”5 top guides from mastering kotlin” by @lime_cl

The suggested hashtag for this book is #masteringkotlin.

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

#masteringkotlin

Contents

Introduction

Several years ago I discovered Android by chance. I was no longer enjoying the kind of software I developed at that time and was in search of new adventures.

Android fascinated me since the very beginning: in a few hours, you could have an App running on your mobile and, with little skills, create your first test screens. Everything could be done from your home and with free software.

But there was something in Android development that I didn't quite like much: the language. At work, I was using C#, a language that has always been at the forefront regarding features, and with Java, I was jumping several steps backward.

Concepts such as properties, null treatment or immutability were already present in most languages, but not in Java. Also, in Android we have the added problem that most devices need to compile using an old version of Java, so we couldn't even take advantage of the small language improvements.

That's why, for me, Kotlin was a turning point. The day I decided to try Kotlin, it was love at first sight. A world of possibilities appeared right in front of my eyes, giving me back things that I lost when I switched from C# to Java, but with other new features that made my code much cleaner and more flexible.

But when you learn a new language, not everything is a path of roses. Writing the same code with the new language is easy, but there are new concepts that you have to master and don't know when to use. You need a change of mentality to solve the same problems differently.

For me, that was an essential point in my transition, and that is why since then I dedicated myself to helping other Android developers to follow the same road.

That was 2015, and since then I've led thousands of developers and their companies to boost their Android productivity by switching to Kotlin, in many different ways: hundreds of articles, talks in conferences, the first Kotlin book ever[1], and extremely practical online course[2], live training, 1 to 1 mentoring[3], etc.

[1]https://antonioleiva.com/kotlin-android-developers-book/
[2]https://antonioleiva.com/online-course/
[3]https://antonioleiva.com/mentoring

So if you think I can help you or your company, please don't hesitate to write me to contact@antonioleiva.com[4].

In this ebook, I'll show you some other ways to take the most out of the language. Kotlin is super-powerful, and you will see here that there are little things in Android that you can't do with Kotlin.

So relax and enjoy what I'm bringing for you.

[4]mailto:contact@antonioleiva.com

1 Anko layouts on Android

If you're already developing Android Apps using Kotlin, you've probably heard about Anko layouts and been thinking about using them or at least considered taking a look at them.

The truth is that Anko[5] has been around for really long. In fact, when I started writing my book about Kotlin on Android[6] at the beginning of 2015, this library already existed.

But in case you hadn't heard about it, Anko is a library **developed by the Kotlin** team with the goal of **simplifying the interaction with the Android framework**.

Its most outstanding feature is Anko layouts, which is the main topic of this first guide. But it also has other features, such as small DSL to execute asynchronous tasks[7] in a very simple way, another one to build easy dialogs and alerts[8], a set of functions to deal with SQLite[9], and even an implementation of coroutines[10], among many other things.

I must admit that I'm a huge fan of this library because it has a lot of impressive features and it's been beneficial to me to understand Kotlin and how to apply it to Android development.

But there's something in this library that always pushed me back: **Anko layouts DSL**.

Anko layouts DSL

The idea of Anko layouts is to take the most out of Kotlin to provide a DSL that **makes declaring Android layouts easy and powerful, just by getting rid of the XMLs**.

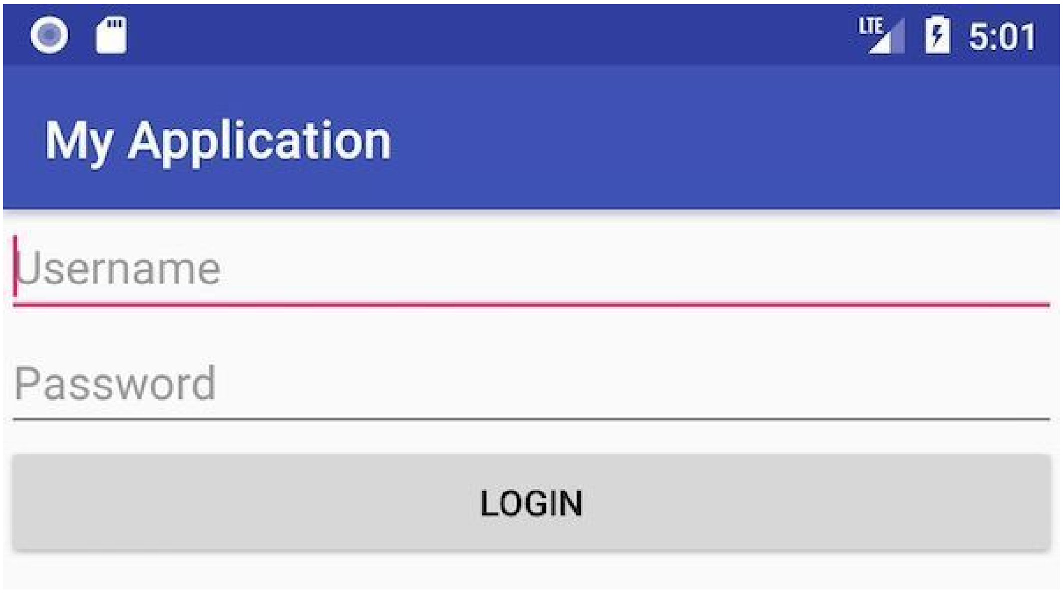
[5]https://github.com/Kotlin/anko/
[6]https://antonioleiva.com/book/
[7]https://antonioleiva.com/anko-background-kotlin-android/
[8]https://antonioleiva.com/dialogs-android-anko-kotlin/
[9]https://antonioleiva.com/databases-anko-kotlin/
[10]https://android.jlelse.eu/a-first-walk-into-kotlin-coroutines-on-android-fe4a6e25f46a

I’ve been developing Android Apps for like 6 years, and I’m really fluent with XML already. So when I tried to do something with Anko layouts, it was like starting from scratch to me.

But for simple examples, it looked really powerful! I could write a small login form in a breeze. With just this code:

```kotlin
UI(true) {
    verticalLayout {
        val user = editText {
            hint = "Username"
        }
        val pass = editText {
            hint = "Password"
        }
        button("Login") {
            setOnClickListener {
                longToast("User: ${user.text}, Pass: ${pass.text}")
            }
        }
    }
}
```

You’d get something like this:



Pretty awesome, right?

But we all know that small samples are never enough to get the whole picture, so I decided to convert my whole Bandhook-Kotlin repo[11] views to Anko layouts. And based on this experience, I decided to write this text.

Should I start using Anko layouts to build my projects?

The answer to this question is not black or white. Based on my experience, it depends a lot on your personal taste. These are the pros and cons I found:

Pros:

- It's ** straightforward to start with it**: the DSL is really intuitive for simple examples, and the code you need to write is clean and nice to read.

[11]https://github.com/antoniolg/Bandhook-Kotlin/

- It skips the inflation step, so the **views written with Anko are much faster to initialize** that those written in XML. There are some articles like this from Simon Vergauwen[12] that show it's even 4 times faster.
- Using Kotlin code means you can do more complex things: for instance, set a dimension that is the sum of two dimensions, ** more straightforward data binding, views are more natural to reuse and compose**…

But these are the cons I found:

- **You need to start learning from scratch**: new API, new rules. So, at the beginning, you'll find some things much more difficult to do that just by using XML. This shouldn't be a stopper though.
- Some things are inherently more complex, like styling your views, setting layout params.
- **You need to write your views blindly**. You don't have a powerful designer to implement the view, or at least see the view you're writing. That's half-true because there's a plugin that should show a preview, but I've never seen it working, it's broken quite often. You cannot use `tools:` attributes either, which are very useful when doing layouts.
- Some things may be even impossible. There are some XML properties that don't have their equivalent in Java, so you can't do much here.
- In general, **I'd encourage you to try it and take your own decisions**. I know people that love Anko layouts and can understand why. I can find an excellent use case for simple layouts like custom views for dialogs or for `RecyclerView` adapters.

How to start using Anko

For a full reference, I recommend you to take a look at the repository[13], but I'll tell you here some steps that will help you follow the code.

[12]https://android.jlelse.eu/400-faster-layouts-with-anko-da17f32c45dd
[13]https://github.com/antoniolg/Bandhook-Kotlin/

Add the corresponding dependencies

Anko is split into several libraries so that you only add what you need. For views, the base one is this:

```
implementation "org.jetbrains.anko:anko-sdk19:$anko_version"
```

The SDK you use should correspond to the minimum version you support (or previous). There's `sdk-15`, `sdk-19`, `sdk-21`, `sdk-23` and `sdk-25`. So let's say your minSDK is `17`, you should use `sdk-15`. Then, depending on what other support libraries you use, you will add the corresponding Anko one: implementation "org.jetbrains.anko:anko-appcompat-v7:$anko_version"

And there are others for design library, CardView, etc. The complete reference is at Anko repository[14].

The base activity

You can just do a `UI` block as I did in the example above, but this can make your activity grow fast. It's better to have the layout in another class.

So for this, I created a base activity, which will force to declare the property that holds the layout:

```
abstract class BaseActivity<out UI : ActivityAnkoComponent<out AppCompatActivity>>
    : AppCompatActivity() {
    abstract val ui: UI

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        (ui as ActivityAnkoComponent<AppCompatActivity>).setContentView(this)
        setSupportActionBar(ui.toolbar)
    }
}
```

In `onCreate`, it sets the view from the property as the view of the activity, and also uses the toolbar from this layout. The generic type of the activity is just an interface that forces implementers to hold a reference to the toolbar:

[14]https://github.com/Kotlin/anko/

```
interface ActivityAnkoComponent<T : AppCompatActivity> : AnkoComponent<T> {
    val toolbar: Toolbar
}
```

That way, the activity can use it in `onCreate`.

Building the layout

Each activity holds an `ActivityAnkoComponent`, that will be the one in charge to build the activity. Here's when we'll use Anko layouts:

```
override fun createView(ui: AnkoContext<MainActivity>) = with(ui) {

    coordinatorLayout {

        appBarLayout {
            toolbar = themedToolbar(R.style.ThemeOverlay_AppCompat_Dark_ActionBar) {
                backgroundResource = R.color.primary
            }.lparams(width = matchParent) {
                scrollFlags = SCROLL_FLAG_SNAP or SCROLL_FLAG_SCROLL or SCROLL_FLAG_ENTER\
_ALWAYS
            }
        }.lparams(width = matchParent)

        recycler = autoFitRecycler()
                .apply(AutofitRecyclerView::style)
                .lparams(matchParent, matchParent) {
                    behavior = AppBarLayout.ScrollingViewBehavior()
                }
    }
}
```

As you can see, the parent class is a `CoordinatorLayout`[15], which holds an `AppBarLayout` with a `Toolbar`, and a `[RecyclerView]`.

An interesting thing is how the layout params are declared. It's a function where you set width and height by argument (default is `WRAP_CONTENT`), and then a block to add the rest of extra parameters:

[15]https://antonioleiva.com/coordinator-layout/

```
.lparams(matchParent, matchParent) {
    behavior = AppBarLayout.ScrollingViewBehavior()
}
```

Use custom views

By default, we only have functions for framework and support libraries views, but you can create you own. For instance, in the example above I created `autoFitRecycler`:

```
fun ViewManager.autoFitRecycler(theme: Int = 0) = autoFitRecycler(theme) {}
inline fun ViewManager.autoFitRecycler(theme: Int = 0, init: AutofitRecyclerView.() -> Un\
it)
    = ankoView(::AutofitRecyclerView, theme, init)
```

You can know more about this at the repository wiki[16]. ### Applying styles For each view, there exists a themed version that allows you to apply a theme, such as the toolbar here:

```
toolbar = themedToolbar(R.style.ThemeOverlay_AppCompat_Dark_ActionBar)
```

But, if what you want is to apply a style, you cannot use an XML one. Instead, you need to use the functions `apply` or `applyRecursively`. This second one, in case you want to apply the style also to the sub-views:

```
.applyRecursively { view ->
    when (view) {
        is EditText -> view.textSize = 18f
    }
}
```

As a way to extract styles, I implemented this extension function for the recycler:

[16]https://github.com/Kotlin/anko/wiki/Anko-Layouts#is-it-extensible

```
fun AutofitRecyclerView.style() {
    clipToPadding = false
    columnWidth = dimen(R.dimen.column_width)
    scrollBarStyle = View.SCROLLBARS_OUTSIDE_OVERLAY
    horizontalPadding = dimen(R.dimen.recycler_spacing)
    verticalPadding = dip(2)
    addItemDecoration(PaddingItemDecoration(dip(2)))
}
```

I can then apply the style this way:

```
recycler = autoFitRecycler()
        .apply(AutofitRecyclerView::style)
```

And these are basically the rough edges you need to know to understand the code.

There are some more complex views, which uses an `AppBarLayout` with a `CollapsingToolbarLayout`[17], which holds an `ImageView`, a `Toolbar` and a `TabLayout`. Then the main area is using a `ViewPager`. I'm leaving the code here for a reference on how it looks:

```
coordinatorLayout {

    themedAppBarLayout(R.style.ThemeOverlay_AppCompat_Dark_ActionBar) {
        fitsSystemWindows = true

        collapsingToolbarLayout = collapsingToolbarLayout {
            fitsSystemWindows = true
            collapsedTitleGravity = Gravity.TOP
            expandedTitleMarginBottom = dip(60)

            image = squareImageView {
                fitsSystemWindows = true
            }.lparamsC(matchParent) {
                collapseMode = COLLAPSE_MODE_PARALLAX
            }

            toolbar = toolbar {
                popupTheme = R.style.ThemeOverlay_AppCompat_Light
                titleMarginTop = dip(16)
```

[17]https://antonioleiva.com/collapsing-toolbar-layout/

```
            }.lparamsC(width = matchParent, height = dip(88)) {
                gravity = Gravity.TOP
                collapseMode = COLLAPSE_MODE_PIN
            }

            tabLayout = tabLayout {
                setSelectedTabIndicatorColor(Color.WHITE)
            }.lparamsC(width = matchParent) {
                gravity = Gravity.BOTTOM
            }

        }.lparams(width = matchParent) {
            scrollFlags = SCROLL_FLAG_SCROLL or SCROLL_FLAG_EXIT_UNTIL_COLLAPSED
        }

    }.lparams(width = matchParent)

    viewPager = viewPager {
        id = View.generateViewId()
    }.lparams {
        behavior = AppBarLayout.ScrollingViewBehavior()
    }
}
```

Kotlin and Anko Layouts, an exciting combination

It's true that working with this DSL is quite fun, and when you overcome the first part of the learning curve, you'll probably enjoy it.

I encourage you to find a view with some complexity and try to implement it using Anko. You will quickly find out whether this library is for you.

Remember there's a full example at Bandhook-Kotlin repository[18].

[18]https://github.com/antoniolg/Bandhook-Kotlin/