Pro Android with Kotlin

Developing Modern Mobile Apps with Kotlin and Jetpack

—

Second Edition

—

Peter Späth

Apress®

Pro Android with Kotlin

Developing Modern Mobile Apps with Kotlin and Jetpack

Second Edition

Peter Späth

Apress®

Pro Android with Kotlin: Developing Modern Mobile Apps with Kotlin and Jetpack

Peter Späth
Leipzig, Germany

ISBN-13 (pbk): 978-1-4842-8744-6 ISBN-13 (electronic): 978-1-4842-8745-3
https://doi.org/10.1007/978-1-4842-8745-3

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

Cover designed by eStudioCalamar

Cover image by Shutterstock (shutterstock.com)

Distributed to the book trade worldwide by Springer Science+Business Media 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 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 (github.com/apress). For more detailed information, please visit www.apress.com/source-code.

Printed on acid-free paper

To Margret.

Table of Contents

About the Author

Peter Späth, **PhD,** graduated in 2002 as a physicist and soon afterward became an IT consultant, mainly for Java-related projects. In 2016 he decided to concentrate on writing books on various aspects, but with the main focus set on software development. With a lot of experience in Java-related languages, the upcoming of Kotlin for building Android apps made him enthusiastic about writing books for Kotlin development in the Android environment.

About the Technical Reviewer

Massimo Nardone has more than 25 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 CISO, CSO, security executive, IoT executive, project manager, software engineer, research engineer, chief security architect, PCI/SCADA auditor, and senior lead IT security/cloud/SCADA architect for many years. His 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, and more.

He worked as a 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 is currently working for Cognizant as the head of cybersecurity and CISO to help both internally and externally with clients in areas of information and cybersecurity, like strategy, planning, processes, policies, procedures, governance, awareness, and so forth. In June 2017 he became a permanent member of the ISACA Finland Board.

Massimo has reviewed more than 45 IT books for different publishing companies and is the coauthor of *Pro Spring Security: Securing Spring Framework 5 and Boot 2-based Java Applications* (Apress, 2019), *Beginning EJB in Java EE 8* (Apress, 2018), *Pro JPA 2 in Java EE 8* (Apress, 2018), and *Pro Android Games* (Apress, 2015)

Preface

Pro Android with Kotlin is a successor of the famous Apress series for Android development targeting the Java platform. With Kotlin as an official language in the Android environment, allowing for more elegant programs compared with the Java standard, the new book deals with advanced aspects of a modern Android app. With a thorough description of important parts of Android system internals and professional-level APIs, advanced user interface topics, advanced development topics, in-depth communication surveys, professional-level hardware topics including looking at devices other than the smartphone, a troubleshooting part with guidance on how to fix memory and performance problems, and an introduction to app monetizing, the book is supposed to be an invaluable source for developers willing to build state-of-the-art professional apps for modern Android devices.

This book is not meant to be an introduction to the Kotlin language. For this aim please have a look at the Kotlin website or any introductory-level book about Kotlin. What you will find here however is an attempt to use as many features of Kotlin as necessary to write elegant and stable apps using less code compared with Java.

In 2021 and 2022, Android versions 12 and 12.1 have been introduced. In a professional environment, writing apps that require a corresponding API level of 31 is a bad idea, since the worldwide distribution of devices running a 12.x version is below 10% as of the writing of this book. But you can write code targeting version 12.x and also version 6.0 all the way up to 12.1, thus covering approximately 95% of all Android devices, by introducing branches in your code or by using special compatibility-aware classes, and this is what we will be doing in this book. We still concentrate on modern 12.x development, but if we use modern features not available to lower versions, we will be telling you.

Note that this book does not pay much attention to Android versions below 6.0, which corresponds to an API level of 23. If you look into online tutorials, you will find a lot of constructs targeting API levels below 23. Especially when it comes to support libraries that were introduced to improve backward compatibility, development gets unnecessarily complicated if you look at API versions below 23, for the distribution of such devices is neglectable nowadays. The book will just assume you are not interested in such old versions, making it unnecessary to look at such support libraries in many cases and simplifying development considerably.

Introduction

A couple of years have passed since the first edition of this book. For the first edition, Android 8, and API level 26, was the current version, but since then Android versions 9, 10, 11, and 12 have been published. Also, Android Studio has migrated from versions 3.1 to 4.2 (June 2021) and later to Arctic Fox, Bumblebee, Chipmunk, and Dolphin (all mid-2021 till now). At the same time, the Android Gradle plugin migrated from versions 3.0 to 7.2 and Kotlin from versions 1.2 to 1.5. In addition, android.support packages, which help ensure version compatibility across different API levels, are now deprecated and replaced by Jetpack (`https://developer.android.com/jetpack`).

With respect to sophisticated graphics rendering and game development, while in 2018 the new 3D graphics rendering engine Vulkan was in its infancy and therefore not included in the first edition, it gained more and more influence, and adding a section on it in Chapter 9 today makes a lot of sense.

In mid-2019 Google announced *Android Automotive*, a variant of the Android OS capable of managing infotainment systems of selected cars.

With Jetpack Compose a simplified UI development technology entered the Android world, considerably reducing the amount of code by providing sophisticated APIs and applying an elegant declarative builder-like programming pattern. Jetpack was introduced in 2018.

All these changes reflect in the second edition of the book, making sure you as a developer are up to date with the current Android development APIs and tools.

Brevity vs. Expressiveness

The programs we will be explaining in this book, despite their strong affinity to the Kotlin way of thinking, however are not totally mysterious to a Java developer or developer of any other modern computer language. One of the design goals of Kotlin is expressiveness, so to understand Kotlin programs, you need less effort, even when the programs get shorter. But you have to understand that at some point you have to pay for maximum brevity with a loss of expressiveness and a loss of readability.

When it comes to deciding what is better, the author favors expressiveness over brevity, but be assured that a loquacious programming style is considered a no-go. In the end a professional developer wants to write a concise app, because less code means lower costs when it comes to maintenance.

The Transition from Java to Kotlin

Just to whet your appetite, we will have a look at a really simple app lacking lots of features you will want to see in a more complex and professional app and then compare its Java and Kotlin variants.

If you start *Android Studio* and enter the project creation wizard, you will be asked for a template. Select "Basic Activity," choose 23 as the minimum SDK, and have both legacy support libraries and Kotlin support disabled. After the creation, a `MainActivity` class shows up. Its Java code reads (comments removed)

```
package book.andrkotlpro.frontjava;

import android.os.Bundle;
import com.google.android.material.snackbar.Snackbar;
import androidx.appcompat.app.AppCompatActivity;
import android.view.View;
import androidx.navigation.NavController;
import androidx.navigation.Navigation;
import androidx.navigation.ui.AppBarConfiguration;
import androidx.navigation.ui.NavigationUI;
import book.andrkotlpro.frontjava.databinding.
    ActivityMainBinding;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity extends AppCompatActivity {
    private AppBarConfiguration appBarConfiguration;
    private ActivityMainBinding binding;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);

    binding = ActivityMainBinding.inflate(
        getLayoutInflater());
    setContentView(binding.getRoot());

    setSupportActionBar(binding.toolbar);

    NavController navController =
        Navigation.findNavController(this,
            R.id.nav_host_fragment_content_main);
    appBarConfiguration =
        new AppBarConfiguration.Builder(
            navController.getGraph()).build();
    NavigationUI.setupActionBarWithNavController(
        this, navController, appBarConfiguration);

    binding.fab.setOnClickListener(
            new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Snackbar.make(view,
                "Replace with your own action",
                Snackbar.LENGTH_LONG)
            .setAction("Action", null).show();
        }
    });
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
```

```
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }

    @Override
    public boolean onSupportNavigateUp() {
        NavController navController =
            Navigation.findNavController(this,
                R.id.nav_host_fragment_content_main);
        return NavigationUI.navigateUp(navController,
            appBarConfiguration)
        || super.onSupportNavigateUp();
    }
}
```

A few notes about that Java code: The `public` in front of the class and almost all methods tells that they are visible from everywhere. It cannot be omitted here since otherwise the framework and the rest of the application could not use the class and the methods. The `setContentView()` is, by virtue of the "set," such a common construct that one could think of allowing `contentView = s.th.` to be written for it instead. A couple of competitor languages allow for such a syntax. Also for the `setOnClickListener()`, one might wish to use a `.onClickListener = s.th.` instead. The argument to `setOnClickListener()` is an object of an anonymous inner class – it is already an abbreviation of first declaring and then instantiating and using it. It can even be further shortened to

```
    binding.fab.setOnClickListener(
        view -> {
                Snackbar.make(view,
                    "Replace with your own action",
                    Snackbar.LENGTH_LONG)
                .setAction("Action", null).show();
            }
    );
```

This is because the interface has just a single method. The wizard just kind of forgot this abbreviation.

For the various `.getSomething()`, we could just as well write something like `.something`, which would express the same, but shorter.

A sister project doing the same, but *with* Kotlin support, leads to a transposed code in the Kotlin language as follows:

```
package book.andrkotlpro.frontkotlin

import android.os.Bundle
import com.google.android.material.snackbar.Snackbar
import androidx.appcompat.app.AppCompatActivity
import androidx.navigation.findNavController
import androidx.navigation.ui.AppBarConfiguration
import androidx.navigation.ui.navigateUp
import androidx.navigation.ui.
    setupActionBarWithNavController
import android.view.Menu
import android.view.MenuItem
import book.andrkotlpro.frontkotlin.databinding.
    ActivityMainBinding

class MainActivity : AppCompatActivity() {
    private lateinit var appBarConfiguration:
        AppBarConfiguration
    private lateinit var binding:
        ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = ActivityMainBinding.inflate(
            layoutInflater)
        setContentView(binding.root)

        setSupportActionBar(binding.toolbar)
```

```
        val navController = findNavController(
            R.id.nav_host_fragment_content_main)
        appBarConfiguration = AppBarConfiguration(
            navController.graph)
        setupActionBarWithNavController(navController,
            appBarConfiguration)

        binding.fab.setOnClickListener { view ->
            Snackbar.make(view,
                "Replace with your own action",
                Snackbar.LENGTH_LONG)
            .setAction("Action", null).show()
        }
    }

    override fun onCreateOptionsMenu(menu: Menu): Boolean {
        menuInflater.inflate(R.menu.menu_main, menu)
        return true
    }

    override fun onOptionsItemSelected(item: MenuItem):
            Boolean {
        return when (item.itemId) {
            R.id.action_settings -> true
            else -> super.onOptionsItemSelected(item)
        }
    }

    override fun onSupportNavigateUp(): Boolean {
        val navController = findNavController(
            R.id.nav_host_fragment_content_main)
        return navController.navigateUp(
            appBarConfiguration)
        || super.onSupportNavigateUp()
    }
}
```

Looking at the Kotlin code more thoroughly, a couple of observations emerge:

- We don't need the ";" delimiters – Kotlin checks at line breaks whether the statement is finished or whether the following line needs to be included.
- We don't need that `public` in front of the class and the methods – "public" is standard in Kotlin.
- Instead of `extends` we just write "`:`", improving the readability a little bit.

We don't need to specify a `void` as return type if a function doesn't return anything. Kotlin can infer that.

- Unfortunately, we cannot write `contentView = s.th.` as suggested previously – the Groovy language, for example, allowed for that. The reason this can't be done in Kotlin is that the construct `contentView = s.th.` implies that there must be a class field named `contentView`, which is not the case. The compiler could check for appropriately named methods and then allow for that syntax, but the Kotlin developers decided to impose this restriction and to prohibit the construct if the field doesn't exist. The same holds for the `setOnClickListener`, because a field `onClickListener` doesn't exist either.
- Instead of an anonymous inner class, we can use the functional construct `view -> ...`. This is always possible if the addressed class, the listener in that case, just contains a single method, like `void onClick( View v )` in the base interface used here. The Kotlin compiler knows that it must use that particular single method of the listener class. While for Java such lambda expressions were not available before JDK 8, in Kotlin they were available from the very beginning.

As a résumé of that comparison, the Kotlin code with 1117 characters (imports and spaces omitted) does the same as the Java code with 1329 characters. This is a savings of 17%. Usually, 20–30% is a savings rate you can expect for more complex classes.

Despite the syntax being different from Java, the Kotlin compiler translates its source code to the same virtual machine bytecode as Java, so Kotlin can use the plethora of Java libraries that are out there in the wild, and Java developers switching to or also using Kotlin won't miss them.

The Book's Target Audience

The book is for intermediate to experienced Android developers wishing to use the new Kotlin features to address current Android versions and devices.

The readers will in the end be able to use Android Studio and Kotlin as a language for building advanced and elaborated apps targeting the Android platform.

Being a Kotlin expert is not absolutely necessary for using this book, but having read introductory-level books or studied online resources is surely helpful. The online documentation of Kotlin also provides valuable resources you can use as a reference while reading this book.

Source Code

All source code shown or referred to in this book can be found at `github.com/Apress/pro-android-with-kotlin-2e`.

Online Text Companion

Some lists and tables, as well as some class and interface details, are available to the public as a *text companion* at `github.com/Apress/pro-android-with-kotlin-2e`. References to such online resources are marked appropriately.

How to Read This Book

This book can be read sequentially if you want to get an impression of what can be done on the Android platform. Or you can read chapters independently when need arises while working on your Android projects. Besides, you can use parts of the book as a reference for both finding solutions with respect to particular problems that come up and determining how things can be done using Kotlin instead of Java. This includes the description of special Kotlin language constructs helping you make your code concise and reliable.

The book is split up in chapters. Chapter 1 gives a very short bird's-eye view of the Android system. If you already have some experience with Android, you can skip it or just shortly scan over it.

Chapters 2–6 talk about the Android architecture's corner blocks: an application as a whole, activities, services, broadcasts, and content providers. Talking to you as a pro-level developer, some of the information provided there may seem to be a little bit basic and may be easy to find in the official Android developer documentation or elsewhere on the Web. The reason I nevertheless added them can be seen if looking more thoroughly at it: the information you can find looking at other sources is of varying qualities, sometimes because of historical reasons, sometimes just because it is outdated. So I tried to rectify some of the peculiarities you find there and also provide you a consolidated, fresh, and new view on things, hoping I can save you some time when you try to find out how the deeper-level nuts and bolts of Android work. You can also see them as a reference just to keep under your pillow in case you are in doubt about some development issues coming up while your Android project advances.

Chapter 7 briefly talks about the permission system, something you must of course be acquainted with if you develop pro-level Android apps.

Chapters 8 and 9 deal with APIs you can use in your app, and user interface issues. Because both of them are big issues, it is just not possible to mention everything that refers to these topics. I however tried to give you a selection of useful and interesting solutions for various tasks in that area.

Chapter 10 introduces Firebase, which is a cloud based app development platform providing storage and messaging services.

Chapters 11–12 take a deeper look at development and building strategies and also describe how things can best be done inside Kotlin. While in the previous chapters Kotlin code was presented in a more empirical way, in Chapter 11 I describe how to use Kotlin constructs to produce more elegant and more readable application code.

Chapter 13 on communication describes some methods you can use to communicate between components inside your app or between your app and other apps or the outside world.

Chapter 14 handles different devices from a hardware perspective, including smartphones, wearables like smartwatches, Android TV, and Android Auto. Here we also talk about ways to access the camera and the sensors and how we can interfere with phone calls.

In Chapters 15–18, we deal with testing, troubleshooting, and publishing your app, and the final chapter, Chapter 19, explains how to use the tools provided with the SDK installation (part of Android Studio).

Some Notes About the Code

While in general for all the code presented in this book, I try to follow a *clean code* approach, for brevity and simplicity, I use two anti-patterns you shouldn't follow in your production code:

- I do not use localized string resources, so whenever you see something like

  ```
  android:text = "Some message"
  ```

  inside XML resources, what instead you should do is create a string resource and let the attribute refer to it like in

  ```
  android:text = "@string/message"
  ```

- For logging statements, I always use "LOG" as a tag like in

  ```
  Log.e("LOG", "The message")
  ```

  In your code you instead should create a tag based on the class name

  ```
  companion object {
    val TAG="The class name"
    ...
  }
  ```

  and then use that one:

  ```
  Log.e(TAG, "The message")
  ```

CHAPTER 1

System

The Android OS was born as the child of the Android Inc. company back in 2003 and later acquired by Google LLC in 2005. The first device running Android came into the market in 2008. Since then it ran through numerous updates, with the latest version number by mid-2022 reading 12.1.

Ever since its first build, the market share of the Android OS has been constantly increasing, and by 2021 it is said to stay above 72%. Even though the numbers vary with the sources you use, the success of the Android OS is surely undeniable. This victory partly has its roots in Google LLC being a clever player in the worldwide smartphone market, but it also comes from the Android OS carefully being tailored to match the needs for smartphones and other handheld or handheld-like devices.

Having said that, the majority of computer developers formerly or still working in the PC environment would do a bad job utterly disregarding handheld device development, and this book is the result of giving to you as a developer an aid to understanding the Android OS and mastering the development of programs herein. The book also concentrates on using Kotlin as a language to achieve development demands, but first we will be looking at the Android OS and auxiliary development-related systems to give you an idea about the inner functioning of Android.

The Android Operating System

Android is based on a specially tailored Linux kernel. This kernel provides all the low-level drivers needed to address the hardware and the program execution environment and low-level communication channels.

On top of the kernel, you will find the *Android Runtime (ART)* and a couple of low-level libraries written in C. The latter serve as a glue between application-related libraries and the kernel. The Android Runtime is the execution engine where Android programs run.

P. Späth, *Pro Android with Kotlin*, https://doi.org/10.1007/978-1-4842-8745-3_1