Scalable Android Applications in Kotlin

Write and maintain large Android application code bases

Myles Bennett

www.bpbonline.com

First Edition 2025

ISBN: 978-93-65899-276

LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY

The information contained in this book is true to correct and the best of author's and publisher's knowledge. The author has made every effort to ensure the accuracy of these publications, but publisher cannot be held responsible for any loss or damage arising from any information in this book.

All trademarks referred to in the book are acknowledged as properties of their respective owners but BPB Publications cannot guarantee the accuracy of this information.

Dedicated to

My wife and business partner

Heather

and

My daughter ***Aimi***

About the Author

Myles Bennett has been working as an Android developer for more than 13 years, and has worked in mobile development for further 10 years as a Symbian developer. Having graduated with a Bachelor of Engineering degree in 1995, he has worked for many high profile clients such as Samsung, Warner Bros Discovery and Sky, to name a few. In his capacity as a contractor throughout his career, he has been in the unique position to gain exposure to a huge variety of different working environments. He is therefore extremely qualified to say what works and what does not work in terms of large software development projects. As a passionate Kotlin professional, he is currently diversifying into other areas where Kotlin is making an impact. This includes full stack development, serverless provisioning and cross platform implementation.

About the Reviewers

- **Awais Zaka** is a seasoned software developer with more than 20 years of industry experience. He has contributed to various sectors, including media/broadcast, telecommunications, and transport, collaborating with companies like NBCUniversal, Sky, Discovery, O2, BBC, and Samsung. He has also created and released his own apps across multiple platforms. Awais lives in London with his wife and three children. In his free time, he enjoys cycling, playing badminton, and reading.

- **Zaid Kamil** is a Google-certified Android app developer and a professional coding trainer who has been developing and teaching how to make Android apps for over a decade. He is the Training Head at Digipodium, a leading IT training institute in India, where he mentors and guides hundreds of students and professionals in various domains of technology. He has a rich portfolio of projects in both Java and Kotlin, using Android and various third-party libraries and APIs. He is also skilled in the areas of Data Science, AI, Cloud services and Web development, and holds multiple certifications from IBM and Microsoft in Python. He has a keen interest in designing and implementing machine learning models to analyze data and improve business operations. He has worked with cross-functional teams to design and deploy AI solutions for various domains. Zaid Kamil is passionate about learning new skills and technology. He constantly updates himself with the latest trends and developments in the tech industry. He also likes to explore new tools and frameworks that can enhance his coding abilities. He believes that learning is a lifelong process and one should never stop growing and improving.

- **Ayomitide Odunyemi** is currently a Principal Engineer at Rad and CTO of Loger, with 8 years of experience in developing and scaling enterprise applications in domains such as Fintech, Edutech, Blockchain, AI, and Proptech, with a track record delivering 27+ production applications scaling to millions of users.

Acknowledgement

I want to express my deepest gratitude to my family for their unwavering support and encouragement throughout this book's writing, especially my wife Heather and my daughter Aimi.

I am also grateful to BPB Publications for their guidance and expertise in bringing this book to fruition. It was a long journey of revising this book, with valuable participation and collaboration of reviewers, technical experts, and editors.

I would particularly like to acknowledge the valuable contributions of my colleague, Awais Zaka, whose reviews of the last few chapters helped push this book over the line.

Finally, I would like to thank all the readers who have taken an interest in my book and for their support in making it a reality. Your encouragement has been invaluable.

Preface

This book introduces the reader to Kotlin and Jetpack Compose for novice or intermediate Android app developers. It proceeds to build upon this foundation, proposing ideologies and methods valuable to even seasoned professionals.

Modern technology in the mobile space is advancing at an ever increasing rate. Mobile applications in turn are becoming more and more complex with multiple features and user journeys. The subsequent code-bases can quickly become unmanageable if not organized correctly. Typical symptoms of this can be seen when adding or fixing one thing breaks another, or when two developers are unable to work on seperate features without overwriting or conflicting with each others code.

There are many established development paradigms in place to address these issues, such as clean-code architecture, test-driven development, layering, model-view-intent, etc., all of which will be covered here, bridging the gap between the theory and practical application in an Android development environment.

The initial chapters will help all the readers who need to know about Kotlin, Jetpack compose and introduce feature orientated project organization. Continuing chapters chart the history of presentation layer architecture leading to working implementations of MVI and Unit-directional Flow using Kotlin and Jetpack Compose. Further chapters introduce cross platform development as a means of seperation of concerns. The readers will also learn the fine details of unit and automation testing with continous integration.

Chapter 1: Introduction to Kotlin for Android – discusses the finer aspects of Kotlin that makes it stand out from other languages and why it is a great choice for Android development. From nullable and built-in lambda types through to asynchronous implementations with Coroutines and everything in between, this chapter provides the foundation for all the concepts discussed in the entire book.

Chapter 2: Breaking Down App Code with Separation of Concerns - details the break-down of app code by introducing **separation of concerns** (**SoC**). The entire foundation for this book is based on this concept. This chapter also has a brief look at its benefits, examines the concept at a high level and discusses the aspects of the Kotlin language that facilitate its implementation.

Chapter 3: Feature-Oriented Development in Android - continues the theme of SoC, this chapter discusses the high-level method of splitting an app into conceptual features and

how this helps contribute to code quality. It examines the origins of the Feature concept and provides an example in the form of a case study.

Chapter 4: Clean Code Architecture - looks at the recommended way of further subdividing those features into modules representing different layers of CCA. It will describe the original CCA concept in depth and then present a very similar arrangement adapted specifically for Android, combining it with Data-Domain-Presentation layering.

Chapter 5: Cross-Platform App Development - covers the topic of cross-platform development and how it relates to large project development. Over the years, there have been several attempts to unify the development of iOS and Android apps using cross-platform environments. These attempts have largely failed. This chapter looks briefly at those platforms, why they failed and discusses the half-way-house of cross-platform development, **Kotlin Multi-mobile** (**KMM**), and how it can be used in a clean code arrangement for pattern enforcement as well as cross-platform compatibility.

Chapter 6: Dependency Injection - explains the concept and looks at the basic **Dependency Injection** (**DI**) techniques, their benefits, and the popular open-source libraries for implementing it. It also explains why it is vital for clean code and Test-driven Development. Further, this chapter provides some code samples, with and without the libraries.

Chapter 7: Introduction to Jetpack Compose - the modern UI toolkit for building native Android apps. The subsequent chapters rely on some rudimentary knowledge of Jetpack Compose. This chapter provides some basic concepts for those unfamiliar with Compose.

Chapter 8: Presentation Layer Evolution in Compose - presents the Uni-directional Flow presentation architecture suited for the latest development paradigms in Android. In doing so, it charts the journey that led to this arrangement by examining each of the popular architectures that went before.

Chapter 9: Test-Driven Development with Mocking Libraries for Android - Test-Driven Development is a software development methodology that emphasizes writing tests before writing the actual code for a software component. This chapter describes the technique in detail and introduces the popular open-source mocking libraries used in its execution.

Chapter 10: Kotlin DSL and Multimodule Apps - describes how to create a project from scratch using Kotlin DSL, suggests a strategy for a module hierarchy and examines an approach to maintain consistent dependency versioning across modules.

Chapter 11: Creating the Module Hierarchy - introduces a simple method for creating module hierarchies and suggests an approach in line with solutions highlighted elsewhere in this book.

Chapter 12: Networking and APIs in Kotlin - examines use cases for and aspects of networking in Android. By the end of this chapter, the readers will understand the concepts of APIs (in particular, RESTful APIs), caching and authentication. This chapter provides a worked example of a network call using the clean-code architecture and test-driven development concepts introduced elsewhere in the book.

Chapter 13: Creating UI with Jetpack Compose - continuing from *Chapter 7, Introduction to Kotlin*, this chapter examines four important high-level aspects of Jetpack Compose, namely, Themes, The Scaffold, Navigation and Animation, that help structure the code and provides a smooth experience to the user. By the end of this chapter, the user will have a solid foundation in the application of these features and have some ideas for their use in a multiplatform environment.

Chapter 14: Debugging in Kotlin - explores the powerful debugging capabilities integrated within Android Studio. It will demonstrate how to utilize breakpoints, watch variables, and logcat to monitor application behavior and identify issues. This chapter will also cover advanced topics such as memory profiling, analyzing thread performance, and leveraging Kotlin-specific debugging tools.

Chapter 15: Test Automation - focuses on automation testing in Kotlin with Jetpack Compose, providing the essential knowledge and tools to create reliable and maintainable test suites for applications. A range of topics will be covered, from setting up a testing environment and writing basic UI tests to more advanced techniques such as testing state management, handling asynchronous operations, and integrating testing into a continuous integration pipeline.

Chapter 16: Building and Distributing Applications - discusses the process of building and distributing Android apps, exploring the essential steps and best practices to bring ideas to life and share them with the world. By the end of this chapter, the reader will have gained insight into creating and uploading an APK to Google Play Store or Amazon App Store.

Code Bundle and Coloured Images

Please follow the link to download the
Code Bundle and the ***Coloured Images*** of the book:

https://rebrand.ly/rx7hy1h

The code bundle for the book is also hosted on GitHub at
https://github.com/bpbpublications/Scalable-Android-Applications-in-Kotlin.
In case there's an update to the code, it will be updated on the existing GitHub repository.

We have code bundles from our rich catalogue of books and videos available at **https://github.com/bpbpublications**. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Piracy

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at **business@bpbonline.com** with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit **www.bpbonline.com**. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit **www.bpbonline.com**.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

Table of Contents

1. Introduction to Kotlin for Android 1

Introduction 1

Structure 2

Objectives 2

The reason why Kotlin is a great choice for Android development 2

Key differences between Kotlin and Java 3

Null safety *3*

Type inference *5*

Functional programming *6*

Immutability *6*

First-class functions and variables *7*

Lambdas *7*

Extension functions *9*

Scoping functions *10*

Built-in collection class extensions *12*

Getters and setters *13*

One-line functions *15*

Delegation *15*

Lazy initialization *18*

No more type erasure *19*

Named and default parameters *20*

Companion objects *21*

Internal scope *22*

Coroutines *23*

Flows and StateFlows *23*

Flows *24*

StateFlows *24*

Conclusion 25
Points to remember 26
Questions 26

2. Breaking Down App Code with Separation of Concerns 27

Introduction 27
Structure 27
Objectives 28
Benefits of SoC 28
Kotlin code constructs facilitating SoC 29
SOLID principles 30
Single Responsibility Principle 30
Open/Closed Principle 32
Liskov Substitution Principle 34
Interface Segregation Principle 35
Dependency Inversion Principle 37
Conclusion 38
Points to remember 39
Questions 39

3. Feature-Oriented Development in Android 41

Introduction 41
Structure 41
Objectives 42
Understanding feature module 42
Concept origins 42
Granularity of features 43
Identifying features 44
User journeys 45
Case study 45
Conclusion 51
Points to remember 52
Questions 52

4. Clean Code Architecture 53

Introduction 53

Structure 53

Objectives 54

The ubiquitous CCA onion diagram 54

Domain-data-presentation layering 55

Combined CCA/layering for Android 56

Domain layer 57

Domain entities 57

Use cases 59

Data layer 61

Repositories 62

Data sources 63

Presentation layer 64

Flow of control 65

Infrastructure layer 69

Conclusion 70

Points to remember 70

Questions 70

5. Cross-Platform App Development 71

Introduction 71

Structure 71

Objectives 72

Cross-platform development overview 72

Disadvantages of cross-platform development 74

Xamarin 75

Disadvantages of Xamarin 76

Ionic 78

Disadvantages of Ionic 79

React Native 80

Disadvantages of React Native 82

Flutter ... 83
Disadvantages of Flutter ... *84*
Compose Multiplatform ... 85
Disadvantages of Compose Multiplatform ... *86*
Common failures ... 87
Kotlin multi-mobile ... 88
Disadvantages of KMM ... *89*
Conclusion ... 92
Points to remember ... 92
Questions ... 92

6. Dependency Injection ... 93

Introduction ... 93
Structure ... 93
Objectives ... 94
Overview of Dependency Injection ... 94
Setter/method injection ... 95
Constructor injection ... 96
Interface injection ... 96
Field injection ... 98
Injection frameworks ... 99
Hilt injection framework ... *101*
History of Hilt ... *101*
Working with Hilt ... *102*
Setting up Hilt ... *103*
Koin injection framework ... *106*
History of Koin ... *106*
Working with Koin ... *107*
Pros and cons of Hilt and Koin ... 109
Hilt pros ... *109*
Hilt cons ... *109*
Koin pros ... *109*

Koin cons *110*
Conclusion 110
Points to remember 111
Questions 111

7. Introduction to Jetpack Compose 113

Introduction 113
Structure 113
Objectives 114
Overview of Jetpack Compose 114
Advantages over traditional views 115
Getting started with Jetpack Compose 116
Some commonly used composables 119
Modifiers *125*
Scoped custom composables 128
State management primitives 130
Conclusion 134
Points to remember 135
Questions 135

8. Presentation Layer Evolution in Compose 137

Introduction 137
Structure 137
Objectives 138
Summary of presentation architecture evolution 138
Model-View-Controller 139
Model-View-Presenter 141
Model-View-ViewModel 144
LiveData *145*
Data binding *147*
Model-View-Intent 149
Key concepts and summary of MVI *151*
Implementing MVI with Jetpack Compose 152

UDF base ViewModel: First pass ... *152*
Implementing view in Jetpack compose ... *154*
UDF base ViewModel: Second pass ... *156*
Using the side effect in the controller ... *158*
UDF base ViewModel: Third pass ... *162*
Disadvantage of the UDF/MVI pattern ... 165
Conclusion ... 165
Points to remember ... 165
Questions ... 166

9. Test-Driven Development with Mocking Libraries for Android ... 167

Introduction ... 167
Structure ... 167
Objectives ... 168
The TDD cycle ... 168
Advantages of TDD ... 168
Historical obstacles to TDD ... 169
The bowling game example ... 170
Dependency injection and TDD ... 174
Mocking libraries ... 175
Mockito ... *177*
MockK ... *177*
Disadvantages of mocking libraries ... 179
Test driving Kotlin flows and StateFlows ... 179
Problems associated with test-driving flows ... *179*
kotlinx-coroutines-test and dispatchers ... *181*
Turbine mocking library ... *183*
Test driving a UDF ViewModel ... 185
Advanced flow testing scenarios ... 188
Conclusion ... 189
Points to remember ... 190
Questions ... 190

10. Kotlin DSL and Multimodule Apps 191

Introduction 191

Structure 191

Objectives 192

Definition of DSL 192

Advantages of Kotlin DSL for build scripts 192

Multimodule project creation 193

The buildSrc module 195

Steps for creating the buildSrc module 196

Version catalogs 200

Updating and adding to the Version Catalog 203

Recommended IDE settings 206

Optimizing imports on-the-fly 206

Project structure suggestions 207

Conclusion 207

Points to remember 207

Questions 208

11. Creating the Module Hierarchy 209

Introduction 209

Structure 209

Objectives 210

Creating a feature presentation module 210

Creating data and domain modules 213

Binding data to domain 218

Koin DI 219

Hilt DI 222

KAPT 223

KSP 225

Key considerations 227

Splitting out UI from presentation 228

Creating an infrastructure (common) module 230

Conclusion 231
Points to remember 231
Questions 232

12. Networking and APIs in Kotlin 233

Introduction 233
Structure 233
Objectives 234
Networking in Android 234
Definition of an API 235
Key aspects of APIs 236
Common types of APIs 237
RESTful APIs 238
Networking libraries 239
Android framework (java.net) 240
OkHttp 241
Volley 243
Retrofit 244
Ktor 245
Authentication and security 246
API keys 246
OAuth tokens 247
Data module setup 248
Build script updates 248
Data classes 250
Test-driving the network call 251
MockEngine 252
Use-case input port 253
The first test 255
Caching 258
Http caching 259
Custom caching 259

Test-driving the use cases 260
DI module binding 264
Hilt *264*
Koin *265*
Conclusion 266
Points to remember 266
Questions 267

13. Creating UI with Jetpack Compose 269

Introduction 269
Structure 269
Objectives 270
Themes 270
Color scheme *272*
Typography *273*
Fonts *274*
FontFamily *275*
TextStyle *276*
Typography *276*
Shapes *277*
The Scaffold 279
TopAppBar() *282*
BottomAppBar() *284*
DrawerContent() *285*
Fab() *285*
SnackHost() *286*
Scaffold() *287*
Navigation 288
The Navigation component (legacy) *288*
Navigation Compose *290*
Navigation Compose with parameters *292*
Navigation Compose with Scaffold *296*

Navigating in and out of the Scaffold *297*
Animation 300
AnimationSpec *300*
AnimatedVisibility *302*
Transition animations *303*
Multiplatform considerations 305
Conclusion 306
Points to remember 307
Questions 307

14. Debugging in Kotlin 309

Introduction 309
Structure 309
Objectives 310
Android device bridge 310
Breakpoints 311
Conditional breakpoints *313*
Exception breakpoints *313*
Set an exception breakpoint *314*
Watch expressions 315
Evaluate expression 316
Profiling tools 316
Logcat logging 319
Timber 321
Other considerations 323
Conclusion 323
Points to remember 323
Questions 324

15. Test Automation 325

Introduction 325
Structure 325
Objectives 326

Espresso 326
Compose tree 327
ComposeTestRule 329
Finders, Matchers and Asserters 330
Finders *330*
Matchers *330*
Asserters *331*
The Robot Pattern 331
Advantages of the Robot pattern *337*
Code coverage 338
Continuous integration 339
Conclusion 340
Points to remember 340
Questions 341

16. Building and Distributing Applications 343

Introduction 343
Structure 343
Objectives 344
Preparing to release an Android app 344
Setting version information *344*
Android Application Package *345*
Android app bundle *345*
Keystore *346*
Creating a keystore with Android Studio *347*
Creating a keystore on the Command Line *349*
Configuring building and signing in Gradle *351*
Building and signing on CI *352*
Build flavors *356*
Distribution portals 357
Google Play store 358
Create Developer account *358*

Releasing an app on Google Play store 359
Finish setting up your application *359*
Internal/closed testing *361*
Adding countries (closed only) *361*
Adding testers *361*
Creating new release *361*
Send the release to Google for review (Closed only) *364*
Pre-registration *364*
Apply for production *364*
Production application process *365*
Amazon App store 365
Creating Developer Account *365*
Releasing an app on Amazon App Store *366*
Conclusion 373
Points to remember 374
Questions 374

Index 375-386

Chapter 1
Introduction to Kotlin for Android

Introduction

This book describes how to build, from scratch, large, multi-feature apps using Kotlin and Jetpack Compose. It achieves this by combining modern paradigms and techniques, all of which will be described here.

Existing commercial Android code tends not to employ a scalable framework suitable for these types of applications. An important reason for this is that these projects are usually *grown* from one of the Android Studio new project wizards.

All these wizards are designed to showcase certain app features in a **`Hello World`** fashion. They typically produce code with a single module, **`app`**, containing a single Activity and perhaps a Fragment with XML layouts and Android Views. They do not present any kind of scalable structure.

What typically happens with large applications that start in this way is that over time, different developers come and go, usually bolting on the popular paradigm at the time. The resulting code becomes *Frankenstein's* Monster; a tangle of fragile spaghetti code that is very difficult to read, likely to break with any changes and that screams out to be rewritten. Too often though, clients are reluctant to do this as the existing code already represents a significant investment.

Throughout his book, we examine how built-in Kotlin features help to address this by breaking up the code in a recognizable fashion making initial creation, maintenance and updating quicker and easier whilst retaining quality.

Firstly though, this chapter examines the differences between Kotlin and Java and discusses why it is a great choice for Android development.

Structure

This chapter covers the following topics:

- The reason why Kotlin is a great choice for Android development
- Key differences between Kotlin and Java

Objectives

By the end of this chapter, not only will the answer to why Kotlin is a great language for Android be apparent, you will also be introduced to the key concepts of the Kotlin language that make it distinct from other languages. In particular, you will become familiar with functional programming, null safety, extension, and scoping functions, asynchronous programming with coroutines, and much more.

The reason why Kotlin is a great choice for Android development

Kotlin is a modern, open-source programming language that is designed to be concise, expressive, and safe. It has quickly gained popularity in the Android development community, as it offers a number of benefits over Java, the traditional language used for Android development.

Here are some of the reasons why Kotlin is a great choice for Android development:

- **Interoperability with Java**: Kotlin is designed to be fully interoperable with Java, which means that existing Java code can be easily integrated into Kotlin projects and vice versa. This makes it easy for developers who are already familiar with Java to start using Kotlin without having to learn a completely new language.
- **Concise and expressive syntax**: Kotlin has a clean and concise syntax that makes it easy to read and write. It also supports several modern programming features such as lambdas, extension functions, and operator overloading that can make code more expressive and concise.
- **Increased productivity**: Kotlin's concise syntax, powerful features, and strong type system can help increase developer productivity. It can reduce the amount of boilerplate code that developers need to write and can make it easier to refactor code and catch errors early. Kotlin's other language features, such as extension functions and data classes, can help developers write code more quickly and efficiently. This can lead to increased productivity and faster development time.

- **Enhanced performance**: Kotlin's performance is at par with Java, and in some cases, it can even outperform it. This is due to Kotlin's efficient bytecode, which is optimized for performance.
- **Improved code safety**: Kotlin has several features that can help improve code safety, such as null safety, type inference, and data classes. These features can help prevent common runtime errors and make it easier to write code that is more robust and maintainable. One of the most significant problems with Java is the potential for null pointer exceptions, which can cause crashes and other issues in Android apps. Kotlin provides null safety features that help developers avoid these issues and write safer code.
- **Extension functions and properties**: Kotlin allows developers to extend existing classes with new functions and properties without having to create new subclasses. This makes it easy to add new features to existing code without having to modify the original code.
- **Coroutines**: Java has historically relied on third-party solutions and plugins to deal with asynchronous code and background tasks. Android initially provided its own solution in the form of `AsyncTask`. Later, `RxJava` became popular but could be difficult to use due to its chained interface pattern. A value spawned in the first part of the chain would become unavailable in a later part of the chain making complex tasks messy to write. Kotlin provides built-in support for coroutines, which makes it easier to write asynchronous code. This can be especially useful in Android development, where asynchronous operations are common.
- **Android Studio support**: Kotlin is fully supported in Android Studio, which is the primary development environment for Android development. This means that developers can take advantage of Kotlin's features and benefits within a familiar and powerful development environment.

Kotlin is a great choice for Android development due to its concise syntax, interoperability with Java, null safety, enhanced performance, and improved productivity. The adoption rate by developers has already made Kotlin and Android synonymous (despite Kotlin also being picked up for backend development now).

Key differences between Kotlin and Java

Whilst there are many differences between Kotlin and Java, in this section we will describe the features of Kotlin that have been found as the most useful in comparison to Java. It is by no means a comprehensive examination of the Kotlin language. There are plenty of existing texts dedicated to that.

Null safety

This is a key feature of Kotlin that helps prevent null pointer exceptions at runtime. In Kotlin, null safety is achieved through a combination of nullable and non-null types, safe

call operator, and null coalescing, that is, Kotlin provides more advanced type inference capabilities compared to Java. Kotlin's compiler can deduce types based on initializers, expressions, and other context, reducing the need for explicit type declarations. This contributes to more concise and readable code. Java's type inference is more limited, primarily focused on simplifying the usage of generics with the diamond operator..

In Kotlin, every variable has a type, that can either be nullable or non-null. A nullable type is denoted by the **`?`** symbol at the end of the type, while a non-null type does not have the **`?`** symbol. For example, **`String?`** Is a nullable type, while **`String`** is a non-null type.

When a variable is declared as nullable, the compiler forces the developer to handle the possibility of the variable being null. This means that the developer has to use a safe call operator **`?.`** or elvis operator **`?:`** to avoid a **`NullPointerException`** at runtime.

The safe call operator **`?.`** is used to safely access properties or methods on nullable variables. If the variable is null, the expression will return null, instead of throwing a **`NullPointerException`**. For example:

```
1. val str: String? = null
2. val length = str?.length
```

The elvis operator **`?:`** is used to provide a default value for a nullable variable if it is null. For example:

```
1. val str: String? = null
2. val length = str?.length ?: 0 // will be 0 if str is null
```

The equivalent Java code would look like this:

```
1. if (str != null) { // not null
2.     length = str.length();
3. } else {
4.     length = 0;
5. }
```

This becomes extremely powerful when mapping complex data from backend APIs where all fields are nullable. Consider this (somewhat contrived) example given an object received of type **`DetailResponse`**:

```
1. data class DetailResponse(
2.     val website: Website? = null
3. )
4.
5. data class Website(
6.     val uri: Uri? = null
7. )
```

```
8.
9. data class Uri(
10.     val url: String? = null
11. )
```

In the instance where you were only interested in the final URL string, then the mapping would look like this:

```
1. data class DetailDomain (val url: String)
2.
3. fun responseToDomain(detailResponse: DetailResponse?):DetailDomain {
4.     return DetailDomain(detailResponse?.website?.uri?.url ?: "")
5. }
```

In fact, Kotlin provides an extended function to replace the elvis operator just for strings, **`orEmpty()`**, so the return statement could look like this:

```
1.     return DetailDomain(detailResponse?.website?.uri?.url.orEmpty())
```

Kotlin also provides a non-null assertion operator **`!!`** which tells the compiler that a nullable variable is not null. This can be useful in certain situations, but should be used with caution, as it can still result in a **`NullPointerException`** at runtime if the variable is actually null. In fact, we would only recommend its use in unit tests (more on this later). There is a reason it is a double-exclamation mark – to draw attention to it in code reviews.

Type inference

Type inference in Kotlin and Java refers to the ability of the compiler to automatically determine the type of a variable or expression based on the context. However, there are some differences in how type inference is handled in Kotlin compared to Java.

In Java, type inference was introduced in Java 8 with the introduction of the diamond operator (**`<>`**) for generics. The primary purpose of type inference in Java is to simplify the usage of generics. For example:

```
1. List<Integer> numbers = new ArrayList<>();
2. // Type inference with diamond operator
```

In this Java code, the diamond operator (**`<>`**) allows the type **`Integer`** to be inferred based on the declaration of **`numbers`** on the left-hand side.

The Java type inference is more limited compared to Kotlin. Java still requires explicit type declarations in most cases, and the compiler's ability to infer types is more restricted than in Kotlin. In Kotlin, the compiler has more powerful type inference capabilities, which allows it to deduce the type of a variable based on its initializer or its usage. This means that you can omit explicit type declarations in many cases, reducing verbosity and making the code more concise. For example: