www.inovex.de

Modern Android app development with Kotlin

Hello,

my name is Johannes and this is the course that I usually give in 4 days. Using this document, will you be able to learn the content by yourself? If not, let us know and we'll arrange a guided training for you.

This training introduces participants to the programming language Kotlin and its use in modern Android app development. During the exercises accompanying the training, participants are building a small application enabling a user to search GitHub projects. Throughout the modules, the app is built from the ground up adding the functionality step by step.

This PDF is interactive – click on the links to navigate through the document and discover supplementary information from diverse resources.

Index

Your Trainer
Johannes Schamburger
Android Developer
+49 721 619 021-0 johannes.schamburger@inovex.de inovex.de

Module 1

Kotlin language features

Kotlin is a statically typed, multi-paradigm programming language which is interoperable with Java. Since 2019, it is the first choice for building Android applications with official support from Google

Content

- Kotlin syntax
- important Kotlin features including nullability, mutability, data / sealed classes, extension functions and higher order functions

Exercises

The Kotlin language concepts are illustrated via live coding and rehearsed in a quiz.

Resources

→ Official documentation
→ Kotlin Playground
→ Kotlin Koans
→ API reference
→ Kotlin Cheat Sheet

Module 2

Tools

This module shows the basic usage of the tools necessary to create and build an Android application and gives an overview over the source code of an app.

Content

- Android Studio
    - Logcat
    - App Inspection
    - Layout Inspector
- Android Emulator
- Android project structure
    - AndroidManifest
    - Activities
- Gradle

Resources

→ Android Studio documentation
→ Official Android documentation
→ API reference

Module 3 UI with Jetpack Compose

For several years, Android app layouts were created using XML layout files in combination with technologies like data binding to modify the UI in the Java or Kotlin code. Jetpack Compose is a newer technology and the recommended toolkit for building native Android UI. It offers a declarative API to define layouts completely in Kotlin. This segment of the training shows how to create Compose layouts and how to make them interactive.

Content

- Composables
- Compose Layouts
- State handling in Compose
- Modifiers

Exercises

Participants create the app's UI showing a search bar and a list of GitHub projects in preparation for the functionality which will be built in the following modules.

Resources

→ Compose documentation
→ Compose Essentials course
→ Compose Basics Codelab

Module 4 App Architecture

To make an Android application maintainable in the long run, several architectural principles such as Separation of concerns, single source of truth and Unidirectional Data Flow should be respected.

Content

- Layered Architecture
- Model View ViewModel (MVVM)
- Repository pattern
- Unit Testing + Mocking

Exercises

Participants implement the layered architecture for the application created in the previous exercise to lay the foundation for adding functionality to the app. Also, the basics of unit testing (including mocking) are practiced by test driven development (TDD).

Resources

→ Architecture Introduction
→ Recommendations
→ Modern Android App Architecture course

Module 5

Dependency Injection

The bigger an app gets the more tedious it gets to manage objects used at several locations throughout the app manually. Dependency Injection aims at separating the construction and sharing of dependencies from the classes using them.

Content

- Dependency Injection frameworks: Koin, Dagger, Hilt
- Constructor injection and field injection
- Android specific challenges

Exercises

To practice usage of Dependency Injection, one or more of the introduced frameworks can be used for the demo application based on the interests of the participants.

Resources

→ Dependency injection in Android
→ Koin
→ Dagger
→ Codelab
→ Hilt

Module 6

Concurrency

Many functionalities within an Android application require concurrency. The main thread, which is responsible for rendering the UI and processing user input, may not be blocked. Therefore, potentially long-running operations like network requests, file/database I/O or complex computations need to be performed on separate threads.

Content

- Coroutines
- Suspending functions
- Scopes
- Dispatchers
- Flows

Exercises

The exercises simulate an asynchronous way of searching GitHub projects and make the necessary changes throughout the app. Also, they introduce a flow representing the UI state which will be rendered by the UI components created in Module 3.

Resources

Coroutines
→ Kotlin Coroutines documentation
→ Coroutines on Android
→ Testing Coroutines
Flows
→ Kotlin Flows documentation
→ Flows on Android
→ Testing Flows on Android
→ Turbine

Module 7

Network

Many Android applications interact with data from backend resources. To retrieve and modify this data, apps typically use HTTP requests.

Content

• Retrofit
• Moshi (JSON Parser)

Exercises

In the exercise for this module, participants add the functionality to search for projects using the Git-Hub API (as opposed to displaying static data as it was before).

Resources

→ Retrofit
→ JSON Parsing library Moshi

Module 8

Local Storage

Android applications typically need to persist data across application restarts in order to enable the user to pick up where they left off. There are several ways depending on the nature of the data to be saved. The easiest option is to save simple key-value pairs.

Content

• DataStore

Exercises

Participants save the user-entered search query and restore it upon app restart.

Resources

→ DataStore
→ Codelab DataStore

Module 9

Databases

Storing data in the DataStore, as discussed in the previous module, is suitable for simple key-value pairs, but not for more complex structured data. To enable storing data in databases, Android offers APIs for SQLite databases. Developers can use these APIs directly or use a library like Room to simplify database interactions.

Content

• Room
• Data Access Objects (DAOs)
• Database relations

Exercises

The example application is extended with a local cache of search results, thus avoiding unnecessary network traffic and adding offline capabilities to the app.

Resources

→ Save data using SQLite
→ Room documentation
→ Codelab Room

Module 10

Modularization

As applications are getting bigger and more complex, developers may think about splitting the source code up into several modules. This can have multiple benefits like scalability, encapsulation and testability but at the same time might introduce challenges such as making the codebase more complex and creating boilerplate code.

Content

- principles for modularization
    - high cohesion
    - low coupling
    - reasonable granularity
- modularization patterns
- technical solutions for modularization in Android apps
- navigation in Android apps
- (optional) Kotlin Multiplatform Mobile (KMM)

Exercises

Participants extract modules from the app's source code and optionally create a Kotlin Multiplatform Module (KMM) which could potentially be shared between Android and iOS apps.

Resources

→ Official documentation

Kotlin Multiplatform Mobile (KMM)

→ Official documentation

→ Tutorial for a KMM app with Ktor and SQLDelight

Navigation in Android apps

→ Official documentation

→ Android navigation codelab

→ Navigation in Compose

→ Jetpack Compose Navigation codelab

Module 11

Performance Testing

Performance issues are a relevant topic for every Android app, especially if it is running on a multitude of devices. There are several ways of identifying and fixing performance issues, either by manual or automated testing.

Content

- Manual Profiling
- Macrobenchmark and Microbenchmark
- Google Play Console

Exercise

Participants write a simple Macrobenchmark for the example application.

Resources

→ Guide to app performance

→ Android Studio profiler

→ Performance Debugging - MAD Skills video playlist

Module 12

UI Testing

Writing tests to ensure the app handles user interactions correctly should be a good practice for every Android application. In contrast to unit tests of components, UI tests actually start the app under test on a device or an emulator and perform actions and assertions on the app.

Content

- UI testing of Compose layouts
- Snapshot testing

Exercises

Participants learn to write UI tests for the app's main screen including reactions on user input events and create simple snapshot tests using the library Paparazzi.

Resources

→ Android UI Testing
→ Compose UI Testing
→ Testing Cheatsheet
→ Codelab Compose UI Testing
→ Snapshot testing library Paparazzi

Module 13

Continuous Integration

There are many options to automatically build, test and deploy an app on a Continuous Integration (CI) platform. This module will take an exemplary look at Gitlab CI.

Content

- typical setup of CI for Android apps

Exercise

Participants extend the CI configuration with a new job and execute it in Gitlab.

Resources

→ Get started with Gitlab CI/CD

Modul 14

Module 14 Android Automotive

The Android Automotive Operating System (AAOS) is an infotainment platform running on a car’s head unit. There are several differences when designing and implementing applications running in the car compared to phone or tablet apps. Regarding the UI/UX, it is especially important to present information in a way that minimizes distraction of the driver. Also, alternative input methods like voice control are more relevant than for phone apps.
In terms of functionality, automotive apps can access various signals from the car like the current gear or the battery level.

Content

- Overview over AAOS
- Android Automotive emulator features

Resources

→ Developer documentation
→ Design documentation
→ Test using the AAOS emulator
→ Introductory video from Android Dev Summit