UA/DETI • 45424 - Introdução à Computação Móvel (ICM)
Ilídio C Oliveira (ico@ua.pt)

Last revision:

Android labs – 2024/24

Introduction **1**
Classes plan 1
Learning resources 1
Lab #1- Introduction to the development workflow and tools **2**
A) Kotlin basics 2
B) Introduction to Android native apps development 2
Lab #2: Composable and managing state **3**
C) UI, events and state 3
Lab #3: Activity lifecycle and architecture components (UI) **4**
D) App architecture and ViewModel 4
E) Accessing (REST) data from the internet 5
Lab #4: Data access (local) & location updates **6**
F) Data layer using ROOM and recommended architecture 6
G) Location updates 7
Week #5- Mobile Backend-as-a-Service (Firebase) **8**
H) Integrate Mobile Backend-as-a-Service (Firebase) 8
Week #6- Deferrable work and ML kit **8**
I) The WorkManager API 8
J) Extending the app with AI 10
Week #7: UI with Views **10**
Android project **11**

Introduction

Classes plan

See the overall course schedule, in Moodle.

Learning resources

Selected Android learning resources:

- **Primary reference**: official Android developers' documentation: tutorials, API documentation, tools, best practices,...
- **Main resource** for classes:
    - "Android Basics with Compose (ABwC)": introductory course by Google.
    - Parts from "Android Development with Kotlin: Classroom course" → with lecture slides
- Other Android training resources:
    - More Courses by Google with codelabs.
    - YouTube: Android Developers channel ; Philipp Lackner channel.
- Books
    - "Programming Android with Kotlin" [Available from O'Reilly]
    - "Professional Android" (2018) by Reto Meier [Available from O'Reilly]

Lab #1- Introduction to the development workflow and tools

A) Kotlin basics

Slides for Kotlin concepts:

- lesson 1 Kotlin Basics → Lesson 2 Functions → Lesson 3 Classes and Objects

See also:

- Kotlin style guide
- Kotlin online playground (run Kotlin code directly in the browser)

Hands-on exercises (with code labs):

- Unit 1 > Introduction to Kotlin : take the codelabs for basic Kotlin as needed. [suggested: activity #5]

B) Introduction to Android native apps development

Slides for supporting concepts:

- Compose tutorial (essential concepts)

See also:

- Android API levels

Hands-on exercises & code labs:

- Unit 1 > "Setup Android Studio" [if needed]. Includes: "how to connect you [physical] Android device".

- Unit 1 > "Build a basic layout" [activities #3 and #4. Suggested for homework → #5]
- Unit 2 > "Kotlin fundamentals" [activities #3, #4 and #5]
- Unit 2 > "Add a button to an app" [activities #2 and #3. Suggested for homework → #4]

Lab #2: Composable and managing state

Resources and readings:

- Thinking in Compose: theory and principles [must read]

C) UI, events and state

Hands-on exercises & code labs

☞ Hands-on: Unit 2 > Interacting with UI and State: Tip calculator [Activities → #3, #4]

Homework 1:

- **study this example** before (it is a hands-on with code-along video support).
- create a simple application to manage your movies and series "watch list".
- you should be able to add entries to the Watch List (stuff you want to watch) and, later, you should be able to mark entries as already watched (e.g.: tick a box). For convenience, start the app with a few entries pre-filled.
- you should use Composables, stateless UI, a list and the ViewModel pattern. Try to rotate the device while using the app (the list should not be lost with configuration changes...).
- for now, it is not needed to save the information in a persistent way (i.e., save to a database).

Notes and key ideias:

- **Composition**: a description of the UI built by Jetpack Compose when it executes composables. Initial composition: creation of a Composition by running composables the first time. **Recomposition**: re-running composables to update the Composition when data changes. To be able to do this, Compose needs to know what state to track.
- Use Compose's State and MutableState types to make state observable by Compose, i.e., wrap regular data into an observable/tracker object.
- Pattern "**state hoistitng**": Composables should be stateless, moving the state to "upper" scopes. The "upper" scope passes the required state (data) as parameter to the Composable as well as the callback functions to receive any events of interest unidirectional data flow.
- The ViewModel pattern allows to move the UI-state outside the UI definition. The ViewModel holds the data and the UI observes the (state-wrapped) data, reacting to

changes in state.

Lab #3: Activity lifecycle and architecture components (UI)

D) App architecture and ViewModel

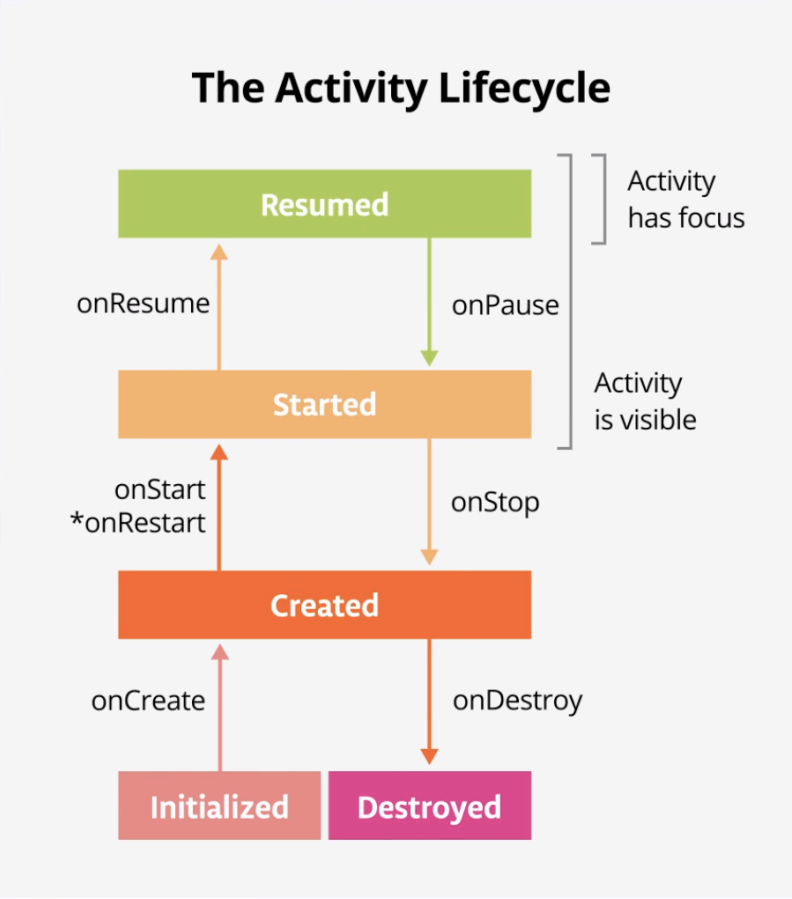
As Android apps grow in size, it's important to define an architecture that allows the app to scale, increases the app's robustness, and makes the app easier to test. An app architecture defines the boundaries between parts of the app and the responsibilities each part should have.

Must read:

- Architecture guidelines and recommended architecture
- Unidirectional data flow explained [video]

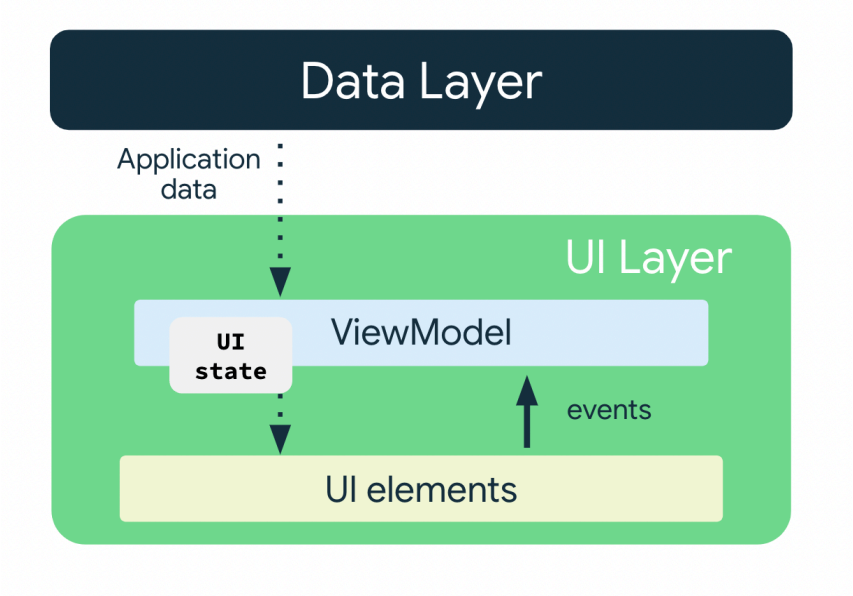
Hands-on exercises & code labs:

- ☞ Activity lifecycle and Logging → Unit 4 / Architecture Components / Activity #2 - Stages of Activity lifecycle. Suggestion: use the solution code project and change the logging messages for the onCreate, onPause, onResume events, using your own.
- ☞ ViewModel and State → Unit 4 / Architecture Components / Activity #5 - ViewModel and State in Compose.



The UI update loop for an app using **unidirectional data flow** (UDF) looks like the following:

- Event: Part of the UI generates an event and passes it upward—such as a button click passed to the ViewModel to handle—or an event that is passed from other layers of your app, such as an indication that the user session has expired.
- Update state: An event handler might change the state.
- Display state: The state holder passes down the state, and the UI displays it.



E) Accessing (REST) data from the internet

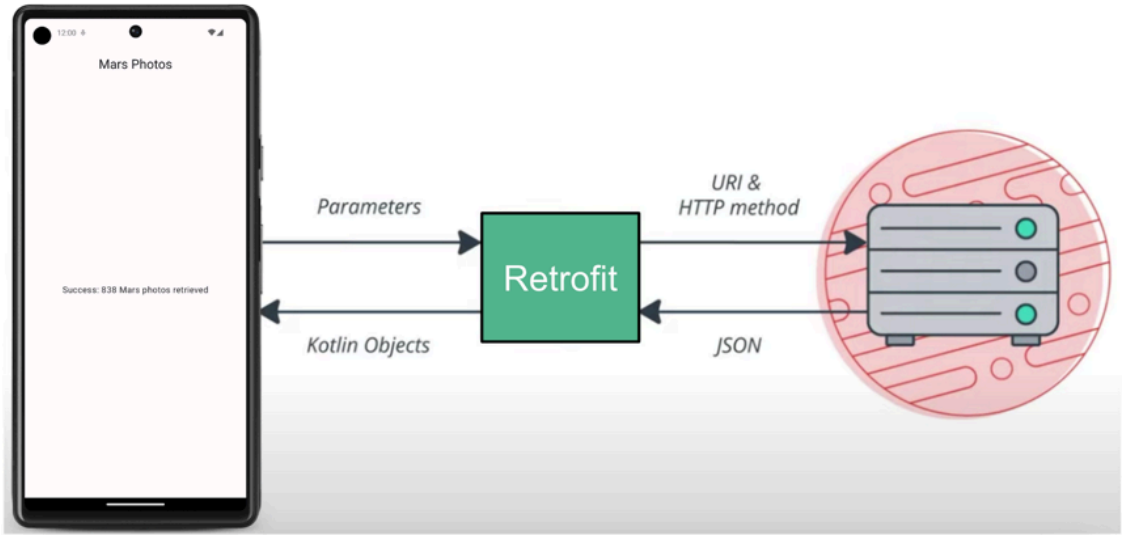
Certain operations can be slow or affected by latency, negatively influencing a fluid user experience. "Heavy" or "slow" tasks should occur outside the default "**main thread**". Kotlin provides a convenient programming abstraction, the coroutine, to execute asynchronous code in "suspendable" functions. Getting data from the internet is an example of a programming case that can benefit from coroutines.

Readings:

- Slides for coroutines (lesson 9, slide 29+)
- Slides for Connect to the internet (lesson 11)
- Requesting "permissions" guidelines

Hands-on exercises & code labs:

☞ Unit 5 → Get data from the internet / Activity 5: Get data from the internet



Homework:

- Customer Journey Map/Experience map for the main scenario(s)/personna(s).
- Prototype the user experience for your project assignment app. You may choose a rapid prototyping environment (e.g.: Figma) or start with the UI of your implementation.

Lab #4: Data access (local) & location updates

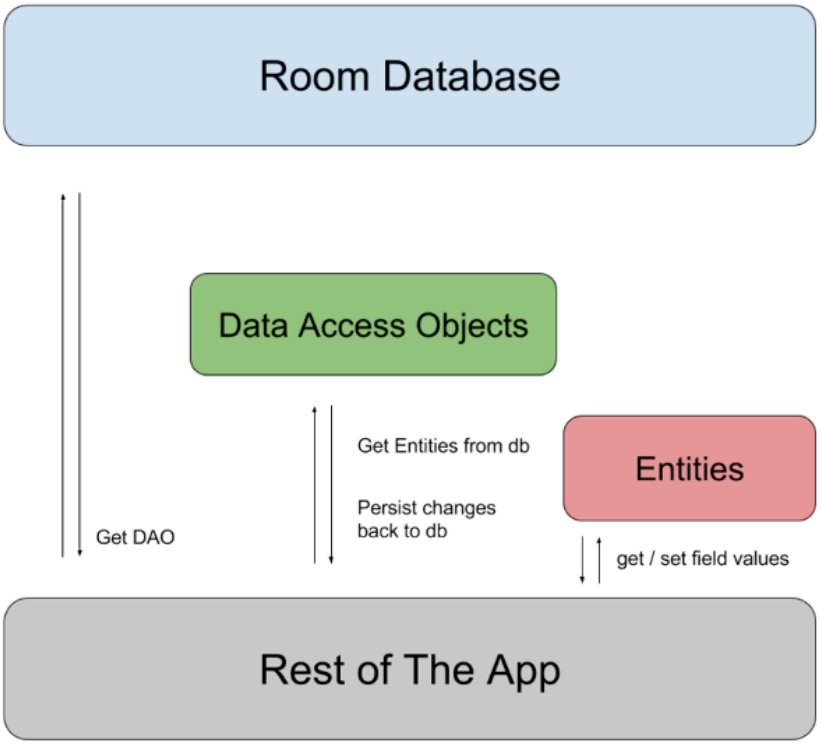
F) Data layer using ROOM and recommended architecture

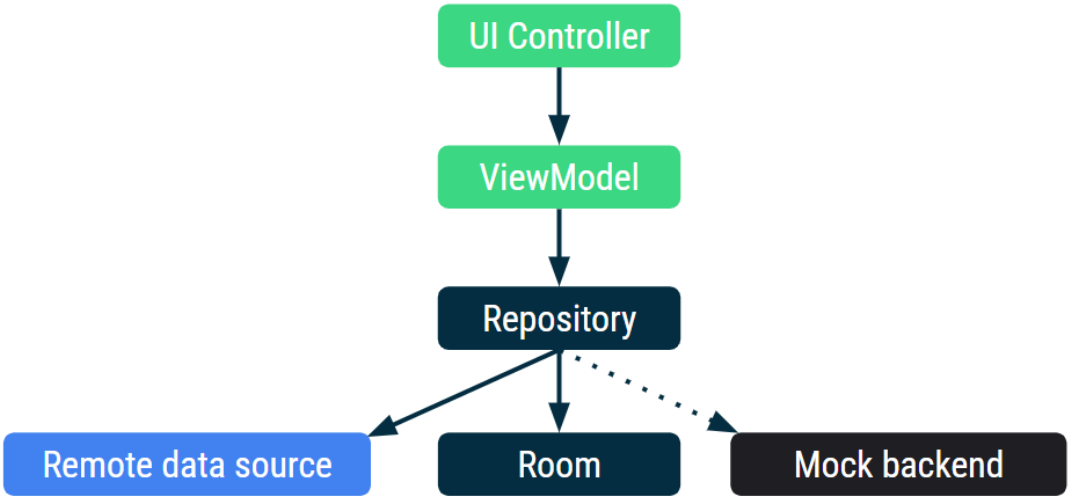
Readings:

- Slides for Room and app architecture (Persistence, lesson 9)
- Simple key-value pair storage: Preferences DataStore. (does not require a predefined schema, and it does not provide type safety or referencial integrity).
- Slides for the Repository pattern (lesson 12, initial slides only)

Hands-on exercises & code labs:

☞ Unit 6 → Get Use Room for data persistence (activities #4 Persist Data with Room, #5 Read and update data with Room)





G) Location updates

Readings:

- Overview of the "Location and Context API" services, by Google. Requires Google Play Services on the target devices.
- Concepts, API, and best practices to build location-aware applications
- Documentation on the Service application component (specially: foreground Services).

Hands-on exercises & code labs:

- Complete the "Receive location updates in Kotlin"

Suggested hands-on:

- Add a map to your Android app

Week #5- Mobile Backend-as-a-Service (Firebase)

H) Integrate Mobile Backend-as-a-Service (Firebase)

Readings:

- Firebase services
- Starting with Friebase in Android: setup an project to use Firebase.
- Understand Firestore data model. Cloud Firestore is a NoSQL, document-oriented database. Unlike a SQL database, there are no tables or rows. Instead, you store data in documents, which are organized into collections.

Frequent Firebase use cases for mobile applications:

- Set up a user authentication flow with Authentication.
- Create a central database with Cloud Firestore (generally preferred) or Realtime Database.
- Store files (media), like photos and videos, with Cloud Storage.
- Trigger backend code that runs in a secure environment with Cloud Functions.
- Send notifications with Cloud Messaging.
- Gain insights on user behavior with Google Analytics.

Hands-on exercises & code labs:

☞ Hands-on: complete Build Friendly Chat code lab. (shows how to configure the Android project and interact with the Firebase backend.)

Week #6- Deferrable work and ML kit

I) The WorkManager API

Readings:

WorkManager is an Android library that runs deferrable background work when the work's constraints are satisfied. WorkManager is intended for tasks that require a guarantee that the system will run them (remains scheduled through app restarts and system reboots). WorkManager is the primary recommended API for background processing. Supersseeds other previous API (Job Scheduler,...).

WorkManager supports chaining (e.g.: a pipeline of tasks to execute in sequence) and flexible scheduling constraints (e.g.: "run only when WiFi present").

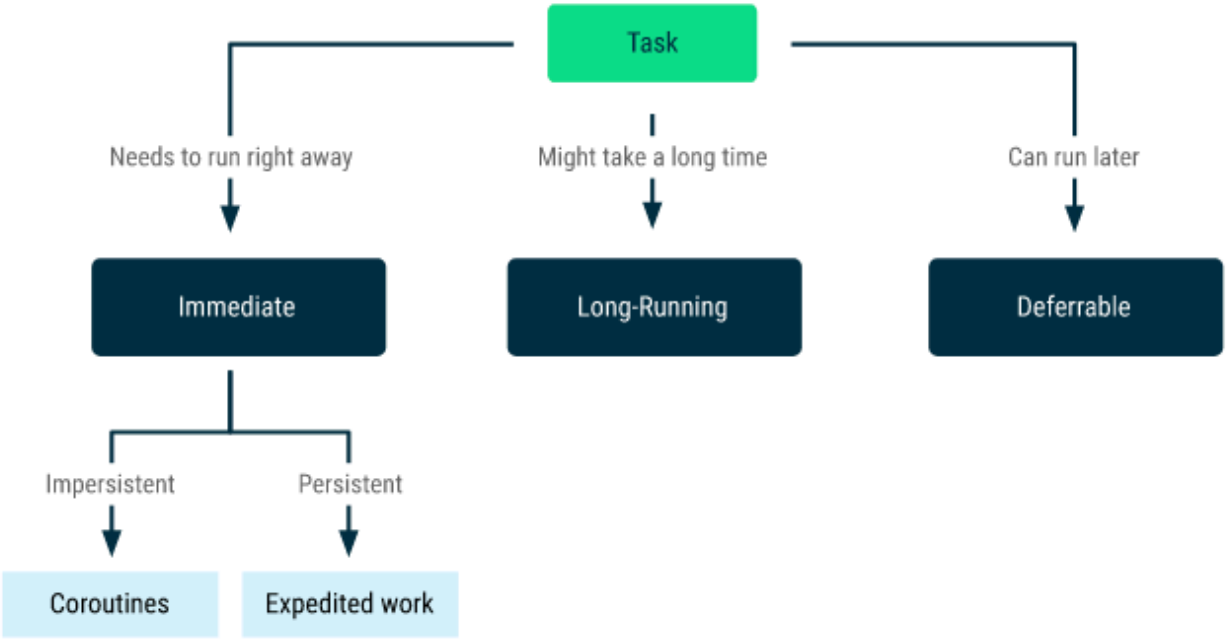
→ Slides for lesson #12 (slide 10+)

→ Guide to "Schedule tasks with WorkManager"

→ Blog post on WorkManager

Hands-on exercises & code labs:

☞ Unit 7 → Schedule tasks with WorkManager: #3 - Background work with WorkManager



Use Case	Examples	Solution
Guaranteed execution of deferrable work	• Upload logs to your server • Encrypt/Decrypt content to upload/download	WorkManager
A task initiated in response to an external event	• Syncing new online content like email	FCM + WorkManager
Continue user-initiated work that needs to run immediately even if the user leaves the app	• Music player • Tracking activity • Transit navigation	Foreground Service
Trigger actions that involve user interactions, like notifications at an exact time.	• Alarm clock • Medicine reminder • Notification about a TV show that is about to start	AlarmManager

J) Extending the app with AI

You can extend your mobile application with AI. A common use case is taking advantage of the device camera for image recognition/classification.

Google offers the ML Kit mobile SDK that brings Google's machine learning expertise to Android and iOS apps. Includes **Vision + Natural Language** APIs to solve common challenges in your apps, using models available from Google. The ML Kit APIs run on-device, allowing for real-time use cases (some updates may depend on the cloud).

For some use cases, instead of the generic ready-to-use models, you may want to develop your own classification model (with the required training). The TensorFlow Lite framework makes it easy to apply ML in your app. Works offline and you do not need to exchange data with the cloud.

- There is a gallery of sample applications that illustrate the use of TensorFlow Lite in Android
- There is an open collaborative space to share pre-trained Tensor Flow models
- Tensor Flow models can be prepared in friendly environments, such as TeachableMachine (web based).

Hands-on exercises & code labs:

- ☞ Code lab: Detect objects in images with ML Kit
- ☞ [Suggested] "play" with ML-Kit samples available (demonstrator for Vision use cases

Week #7: UI with Views

The Views system is available from the early versions of Android, using XML files to write layout descriptions. The current guidelines, however, favor the use of Compose.

K) Using the Android Views system

Readings:

- Slides: lesson #5 - Layouts (with Views)
- Mixing Views and Compose side-by-side: check Unit #8.

Hands-on exercises & code labs

☞ Codelab: build your first app in Kotlin (using the Views system)

☞ Codelab: Room with a View (integration project: demonstrates Android Architecture in a project that uses the Views system). (1)

(1) Notes: the solution project, if needed, is somewhat old. Consider using JDK 11 and Gradle plugin 3.6.x.

Android project

Assessment criteria	Main Topics
A) Goals achievement: mobility use cases	The use cases are not natural in a web/desktop application Camera-intensive, maps and/or location-intensive,... Proactive notifications Sensors (built-in or connected sensors) and adaptation to (sensed) user context UX coherent with mobility use cases and language (e.g.: content first,...) Others depending on the nature of the project
B) Complexity (of the implementation)	Adopts architeture guidelines/patterns and best practices Implements a backend strategy Data layer (strategy for); still useful if offline User-readiness (is UX ready for users?) UX continues across devices Others depending on the nature of the project
C) Effort	Fair effort for the team (taking into consideration the starting baseline for each profile)