

Android labs – 2020/21

Learning resources	1
Lab #1- Introduction to the development workflow and tools	1
Readings & learning resources	1
Lab	2
HW/checkpoint assignment	2
Explore	2
#2- Flexible user interfaces and fragments.....	2
Readings & learning resources	2
Lab	2
Explore	3
#3- Background tasks & reacting to the user context.....	3
Readings & preparation	3
Lab@Home	4
Explore	5
#4- Connecting to remote services & mobile backend	5
Readings and resources	5
Lab	6
Homework/Checkpoint Assignment	6
Explore	6
Common programming cases & receipts	7
Permissions	7

Learning resources

- [Android documentation](#): tutorials, API documentation, tools, best practices,...
- [Android training](#), including [courses by Google](#)
- Android [Courses at Udacity](#) (by Google staff members)
- R. Meirs' "[Professional Android](#)" Book
- [CodePath](#) Android Cliffnotes: very good collection of topics on Android development.

Lab #1- Introduction to the development workflow and tools

Readings & learning resources

- Required: concepts for [Lesson 1 and Lesson 2](#) in [Android Developer Fundamentals course](#)
- Optional: [slides](#) for lesson 1.* and lesson 2.*
- Optional: [Meiers' PA4](#): Chap. 2, 3 & 5.

Lab

Pre-requirements: [install Android Studio](#).

In this lab, we will complete:

- [“Build your first app”](#) – introductory tutorial, from Android documentation.
- [Code labs for “Lesson 2.x” \(from Android Developer Fundamentals course materials\)](#)

HW/checkpoint assignment

CA1: build an app that acts as a dialer, with a “keypad” to enter the calling number. Start with the simplest approach possible.

When you press the dial button, a call should be started (just hand-over to the “real” built-in dialer).

You should add a set of 3 “speed dial” buttons (memories); when the users does a long press on one of these “speed dials”/memories, a secondary activity is offered to allow the user to update the speed dial details (define a label and associate a phone number).



Explore

- [CodePath](#) Android Cliffnotes: very good collection of topics on Android development.

#2- Flexible user interfaces and fragments

Readings & learning resources

- Concepts for [Lesson 4 and Lesson 5](#) in ADF course
- Concepts for [Lesson 1: Fragments](#) in Android Developer Advanced
- Optional: [slides](#) for lesson 4.* and 5.* in ADF
- Optional: [slides](#) for Lesson 1: Fragments in ADAAdv
- The Android Studio visual [Layout Editor](#)
- Optional: [Meiers' PA4](#): Chap. 2, 3 & 5.

Lab

Proposed lab activities: (mostly from the [Android Developer Fundamentals course materials](#)):

- Code lab 4.1 ([Clickable images](#)), from ADF course
- Code lab 4.5 ([RecyclerView](#)), from ADF course. Make sure the RecyclerView AndroidX library.

Suggested adaptation to the codelab, taking into consideration the new the wizard creates the elements:

- @Step 3: in task 1.1, when you select the Basic Activity template, the wizard will include two fragments. You can remove the fragments (fragment_first, fragment_second) and the layouts created for those fragments. Then, adapt the content_main to define the layout of the main activity.
- @Step 3: in task 1.3, instead of creating a new image resource, you can just pick an icon from the drop down list (for attribute srcCompat).
- @Step 4: in task 2.1, be sure to add the androidx.recyclerview.widget.RecyclerView (and not the support library v7). You can easily do it by using the designer (instead of editing the xml).

- c) Code lab for [Lesson 1: Fragments](#) (01.1 + 01.2) from Android Developer Advanced
- d) Code lab 4.4 ([user navigation](#)), from ADF course
- e) Code lab 5.3 ([adaptative layouts](#)), from ADF course. The main objective here is to learn how android makes use of alternative resources.
- f) Code lab 4.3 ([menu and pickers](#)), from ADF course

Note 1: some code labs use the old support library (e.g: android.support.v7.widget.RecyclerView). You should prefer, instead, the new packages under AndroidX (e.g.: **androidx.recyclerview.widget.RecyclerView**). If you need to use the old packages, when creating the project, select the option to use legacy libraries. More info on [migrating to AndroidX](#).

Note 2: be sure to complete the code labs b) and c). Fragments and the RecyclerView will appear very often.

From [CODEPATH](#):

“Within a fragment-heavy app, we need to remember to organize our code according to architectural best practices.

Activities are navigation controllers primarily responsible for:

- Navigation to other activities through intents.
- Presenting navigational components such as the [navigation drawer](#) or the [viewpager](#).
- Hiding and showing relevant fragments using the fragment manager.
- Receiving data from intents and passing data between fragments.

Fragments are content controllers and contain most views, layouts, and event logic including:

- Layouts and views displaying relevant app content.
- Event handling logic associated with relevant views.
- View state management logic such as visibility or error handling.
- Triggering of network request through a client object.
- Retrieval and storage of data from persistence through model objects.

To reiterate, in a fragment-based architecture, the **activities are for navigation** and the **fragments are for views and logic.**”

Explore

- [Material design guidelines](#) for User Experience (UX) and look-and-feel.
- Another guide to [RecyclerView](#).
- Example of [Master-Detail navigation](#), with Fragments.
- The (new) Android JetPack selection of components includes support for a new [Navigation Component](#), which makes it easier to handle the navigation between fragments and activities. There is also a related [codelab available](#).

#3- Background tasks & reacting to the user context

Readings & preparation

- Concepts for [Lesson 7.1, 7.3 in the Background Tasks](#) chapter of ADF. [[Slides](#)]
- Concepts for [Lesson 7.1](#) (Location) in the ADA.

Lab@Home

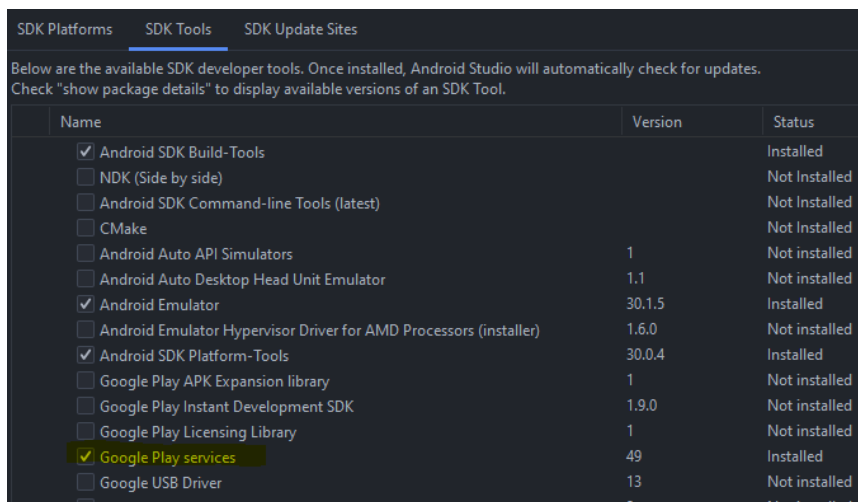
Be sure to complete the following labs:

- A) [Code lab 7.1](#) (Fundamentals): AsyncTask
- B) [Code lab 7.1](#) (Advanced): Get device location and track location updates (this example shows several interesting points, besides the location. Should be completed carefully.)
- C) [Code lab 7.3](#) (Fundamentals): Broadcast receivers

Notes on the code labs:

- A) In some cases, there is a starter project, or you may want to run the solution code. Note that the projects refer to somewhat old configurations. You may want to:
 - Update **compileSdkVersion** and **targetSdkVersion** in build.gradle to a recent version.
 - Use the new AndroidX instead of the old Support Libraries. There is a wizard for this refactoring: Menu Refactoring > Migrate to AndroidX
- B) Android has [two rules](#) concerning the use of threads: do not block the UI thread (→any “heavier” work should be done in a separate thread); do UI work only on the UI thread (→access the UI only from the default thread, also called the UI thread). The **AsyncTask** is a helper class that combine, in the same object, methods that run in a separate thread (can run lengthier tasks), and methods that run in the main thread (can update the UI). Check the [key points](#) about AsyncTask usage.
- C) To get the device **location** and track the location updates, Google offers an optimized API outside the basic Android SDK. The [Fused Location API](#) is included in the [Google Play services](#) and it offer a shared, energy-aware location access. Note that to use Play services you need both to [add the dependency to you project](#) and have the Google Play API in the device.

In the SDK Tools, check the availability of “Google Play Services”:



Name	Version	Status
<input checked="" type="checkbox"/> Android SDK Build-Tools		Installed
<input type="checkbox"/> NDK (Side by side)		Not Installed
<input type="checkbox"/> Android SDK Command-line Tools (latest)		Not Installed
<input type="checkbox"/> CMake		Not Installed
<input type="checkbox"/> Android Auto API Simulators	1	Not installed
<input type="checkbox"/> Android Auto Desktop Head Unit Emulator	1.1	Not installed
<input checked="" type="checkbox"/> Android Emulator	30.1.5	Installed
<input type="checkbox"/> Android Emulator Hypervisor Driver for AMD Processors (installer)	1.6.0	Not installed
<input checked="" type="checkbox"/> Android SDK Platform-Tools	30.0.4	Installed
<input type="checkbox"/> Google Play APK Expansion library	1	Not installed
<input type="checkbox"/> Google Play Instant Development SDK	1.9.0	Not installed
<input type="checkbox"/> Google Play Licensing Library	1	Not installed
<input checked="" type="checkbox"/> Google Play services	49	Installed
<input type="checkbox"/> Google USB Driver	13	Not installed

Confirm the dependency import, using a recent version.

```
dependencies {  
    implementation 'com.google.android.gms:play-services-location:17.1.0'
```

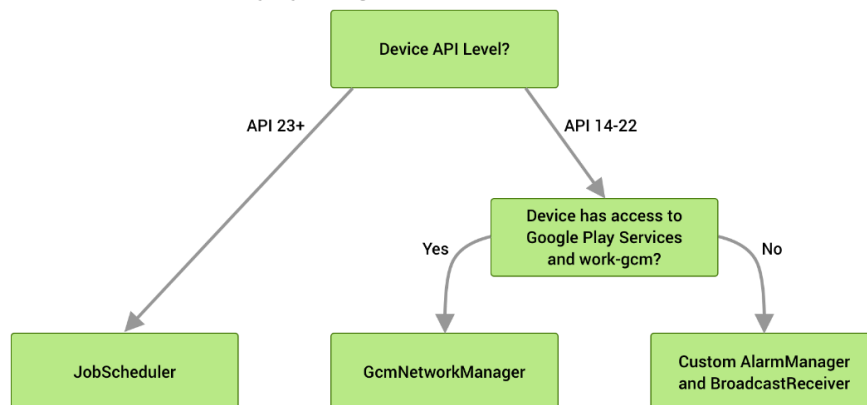
In the Location exercise:

- You will have to handle permissions. The code lab explains how to do it, but there is also [a “receipt”](#) in the last section of this document.
- Be sure that, if using an emulator, it supports Google APIs or Google Play.
- Run Google Maps app (inside the emulator) to (force the) update the location cache (otherwise, location changes may not be assumed in the emulator)

- D) [Broadcast Receivers](#) provide a highly decoupled messaging system in Android, in which one component can subscribe for updates in a topic, while other component would post information to data channel (Publisher/Subscriber pattern). This is useful, for example, to get notifications on system events (just boot, lost connection to WiFi,...). Check the [key points](#) about Broadcast Receivers usage..

Explore

- Codelab: [WorkManager](#) (a more sophisticated job scheduler, with backward compatibility). A comprehensive description of the context and approach is [available here](#). The **WorkManager** library is a good choice for tasks that are useful to complete, even if the user navigates away from a particular screen or your app. Some examples of tasks that are a good use of WorkManager:
 - o Uploading logs
 - o Applying filters to images and saving the image
 - o Periodically syncing local data with the network



- WorkManager provides a convenient API on top of existing, but different, approaches.

#4- Connecting to remote services & mobile backend

Readings and resources

- [Retrofit](#) is a type-safe REST client for Android, Java and Kotlin developed by Square. The library provides a powerful framework for authenticating and interacting with remote APIs and sending network requests with OkHttp. This library makes downloading JSON or XML data from a web API straightforward. Once the data is downloaded it is then parsed into a Java Object.
- [Firebase](#) is the Google's mobile backend platform that [plays very well with Android](#). It offers a convenient way to:
 - Gain insights on user behavior (with Analytics): how/where is the app being used?
 - Set up user authentication (with Authentication).
 - Store structured data with Firestore or blob data with Cloud Storage (central cloud "repository").
 - Send notifications to users (with Cloud Messaging).
 - Find out when and why your app may be crashing (with Crash Reporting).

[Firestore](#) is a redesigned cloud database, that works with Firebase, and offers better scalability. It keeps your data in sync across client apps through **realtime** listeners and offers **offline** support.

Lab

In this lab:

- 1) Study the [example discussed here](#) to access a REST API using Retrofit. [An updated version of the code is available in [project RetrofitSample](#).]
- 2) Follow the [Android Firebase Codelab](#) to develop a simple chat client ("Friendly Chat") and get started with Firebase.

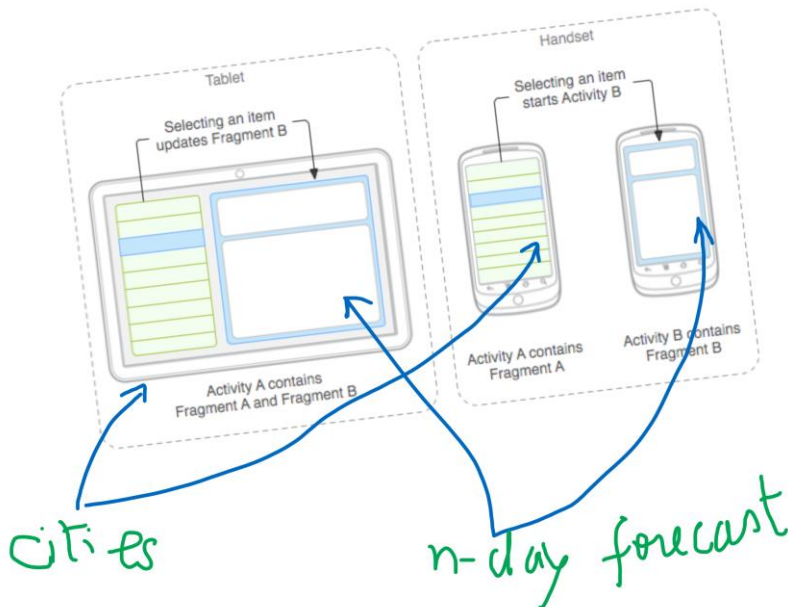
Note1: the **keytool** utility is available in the *bin* directory of your Java installation. You may confirm the Java location: File Menu > Project Structure > SDK location > *JDK Location*.

Note 2: if you get errors related to AndroidX not being enabled, in *gradle.properties*, set `android.useAndroidX=true`

Homework/Checkpoint Assignment

CA2: build an app that offers a weather forecast. The user should, first, select the city of interest (from a limited list of cities) and then get the forecast for the upcoming days (for the selected city). The app should make use of **fragments** (and **RecyclerView**) and offer two **alternative layouts** ("normal" screen, and "large landscape").

The weather forecast content should be obtained by invoking an external API. A [sample project NextWeather](#) is available to demonstrate the use of the IPMA API (using the [Retrofit library](#)).

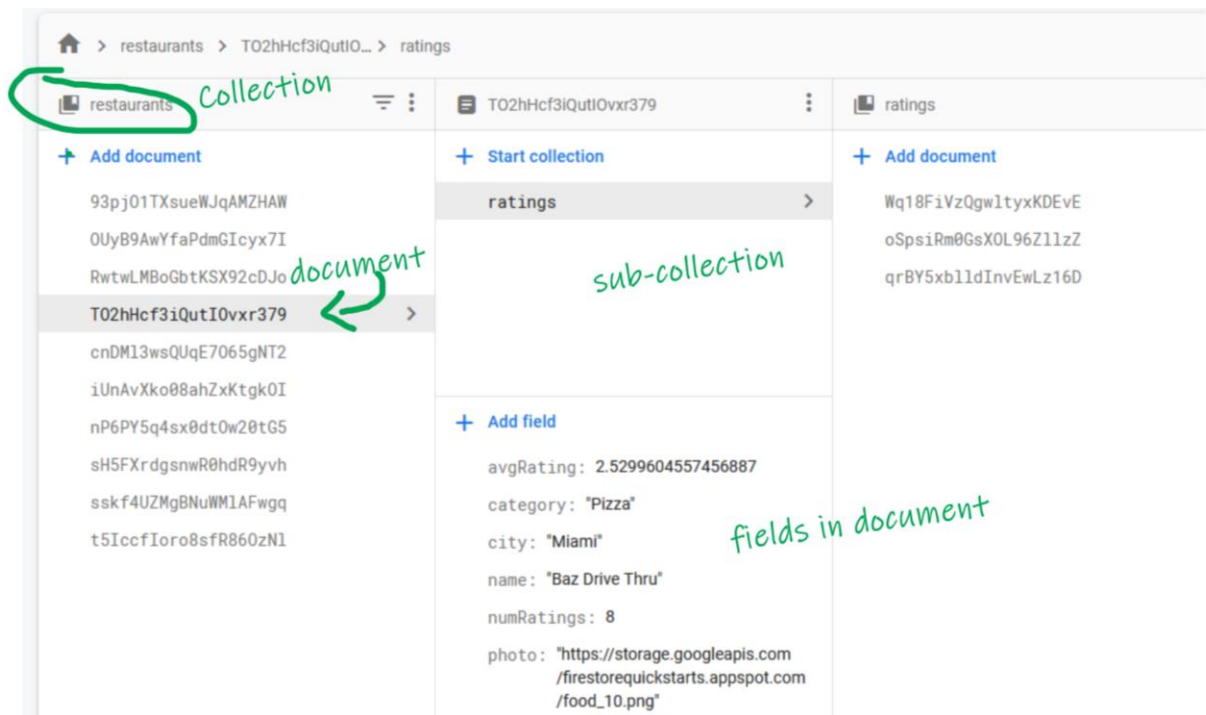


Explore

- 1) [Read and Write data into cloud Firestore](#) (you should, first, have used Firebase).

Note: the project makes use of `FirestoreAdapter`, a `RecyclerView` adapter to get data from Firestore.

Instead of Retrofit, you may use the [Volley library](#), for HTTP requests.



Common programming cases & receipts

Permissions

- Dynamically verify and [ask for App permissions](#) (required from API 23+)