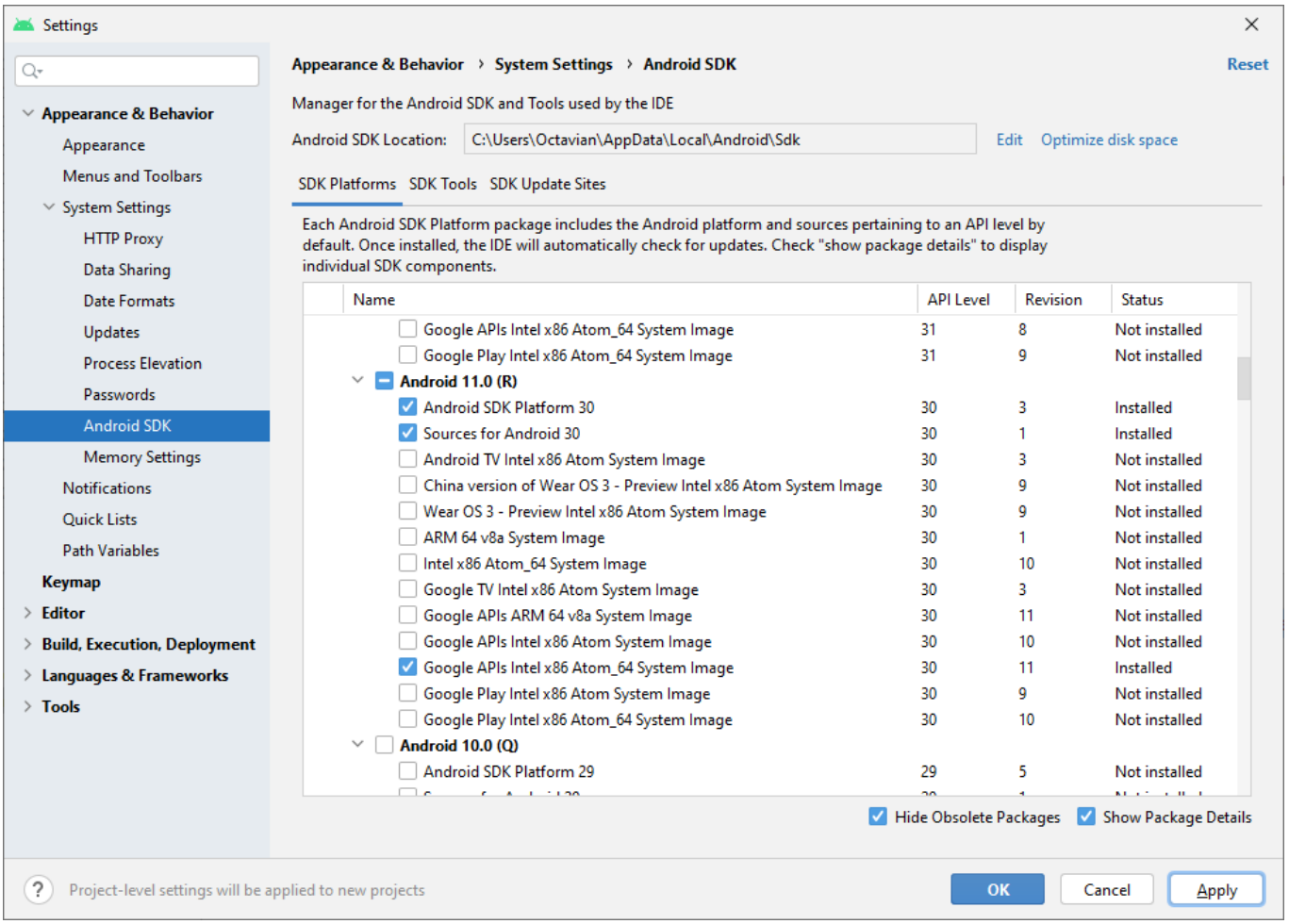
Preparatory Lab - Android Development Tools

Android Studio provides a powerful, but somewhat complex environment for developing mobile apps. In this lab we will get familiar with the environment and basic tools for managing application development and testing.

Android Environment

Open Android Studio and even before you start a new project go to **Configure** (or the three dots icon in the upper right corner of the window) and open **SDK Manager.** Android OS is an evolving project and every once in a while Google will release a new version of the platform. To make things even more confusing, these versions have descriptive (Pie, Q, R, S, etc.), version (9.0, 10.0, 11.0, 12), and integer (API 28, 29, 30, 31 etc.) labels.



Assignment: Show package details and make sure that you have Android 11.0 (R) SDK installed, together with the Google APIs Intel x86 Atom_64 System Image for Android 11.0 (if you are using Apple M1 CPU, or a similar ARM architecture, you may install Google APIs ARM 64 image instead).

Assignment: Next, open the **SDK Tools** tab and make sure that Android SDK build tools, emulator, Android SDK platform-tools and Android SDK tools, Google play services, and HAXM installer are installed.

Project Structure

Start a new Android Studio project (you can select app name and the package name, and **Kotlin** as the coding language) with a single Empty Activity. The compilation will most likely start automatically and might fail.

Before we fix it, let's see the project organization. On the left hand side, in the **Project** window select **Android** view (default). The file tree that you see represents your project. The main part of it is in the "app" folder. "app" is the default *module* of your project. Modules represent independent pieces of code and can be used, for example, if you wish to develop the same app for smartphones and smartwatches. In this case you could create one core module containing the common logic, and then two modules, each for a different device type.

Assignment: Examine "app" module. Locate the MainActivity Kotlin code (in the java folder). Now find the layout file, it is located somewhere in the "res" folder.

For most part, programming an Android application boils down to creating and modifying files in the "app" folder. However, additional information is needed for Android to know how to understand your application. This information is held in **AndroidManifest.XML**

Assignment: Open AndroidManifest.XML and see if you can understand how Android knows which Activity to launch once a user clicks on the app icon?

Build System

What you see in Android Studio are your source code files. We are still a long way from a packaged APK that can be shipped and installed on users' phones. To manage project compilation, Android Studio uses **Gradle**. Gradle is a build automation system (think of it as a building recipe executor). **Android Gradle Plugin** adds features specific to building Android apps.

File->Project Structure->Project gives you information about the Gradle and Gradle Plugin version you have (note that these should be compatible, check the table on the following page to ensure that they indeed are: https://developer.android.com/studio/releases/gradle-plugin.html)

Gradle takes recipes - .gradle files - and compiles your programme. The key recipe is the **build.gradle** file contained in your app folder.

```
plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
}

android {
    compileSdk 30

    defaultConfig {
        applicationId "com.example.myapplication"
        minSdk 29
        targetSdk 30
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
}

dependencies {

    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.appcompat:appcompat:1.3.0'
    implementation 'com.google.android.material:material:1.4.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
}
```

Assignment: Open your module's build.gradle file. In case your application does not compile, fix it, so that the module compiles with the API version you have installed earlier (hint: compileSdkVersion should match the SDK version you downloaded earlier). Recompile the project. Fix any additional errors you may get.

Emulator

Android Studio comes with a device emulator, so you can debug your programme even if you don't have a physical device with you.

Assignment: Open **Android Virtual Device Manager**. Create a new virtual device with an image you have previously downloaded (the same API as the one you are using for the app compilation).

Start the emulator. You now have a very realistic picture of an actual Android device. Feel free to explore a bit, and then test the buttons on the side of the virtual device. Finally, click on the three dots button. This will open additional options that, among other things, let you set the sensor readings (experiment with accelerometer!), battery level, take snapshots, record the video of the interaction with the virtual device, etc.

Go ahead and run your app in the emulator!

Android Debug Bridge (ADB)

How did Android Studio send your application to the emulator? By using **adb**. This powerful command line utility lets you manage applications on a (virtual) device, copy files to the device, send events to it, etc. We will see some of the adb's functionalities in a moment.

Open the command line and run "`adb`". You should see a list of options that can be specified with adb. If the command is not found, please check that 1) you have installed SDK tools, SDK platform tools and SDK command line tools in Android Studio; 2) that adb (most likely in `Android/sdk/platform-tools`) is in your `PATH` environment variable.

Assignment: List devices that adb can connect to. If you have a physical phone, connect it via USB and see if it is shown in the list[1].

With adb we can also access certain Android functionalities directly on the device! "`adb shell`" starts a remote shell from the connected devices. Now you can, for example, traverse the directory structure. You can also list installed apps on the device with "`pm list packages`". You are directly accessing the Package Manager!

Our colleagues from Slovakia create an interesting test application that showcases the possibilities of adb. Download their TryMe application.

Assignment: Using adb install TryMe application on the emulator. Hint: `adb install tryme.apk`

You can now check whether the app is indeed installed. Remember how we listed all the packages using adb - find TryMe.

If you try to find the app in the emulator, you will not see it. This app does not have a launcher. Without adb it would be impossible to launch it. However, adb can also access the Activity Manager's functionality through the "`am`" command run in the shell. To start an activity called ACTIVITYNAME that lives in the application with package name PACKAGENAME, you need to call
"`am start -n PACKAGENAME/.ACTIVITYNAME`"

Once you complete the preparatory lab successfully, go ahead and uninstall the apps you installed on the emulator using `adb uninstall`.

Happy coding!

[1] Note: if this is your first time using this phone for debugging, you will have to enable USB debugging, first. Please read the instructions here: https://developer.android.com/studio/debug/dev-options