Lab 1 – Getting Started

Introduction

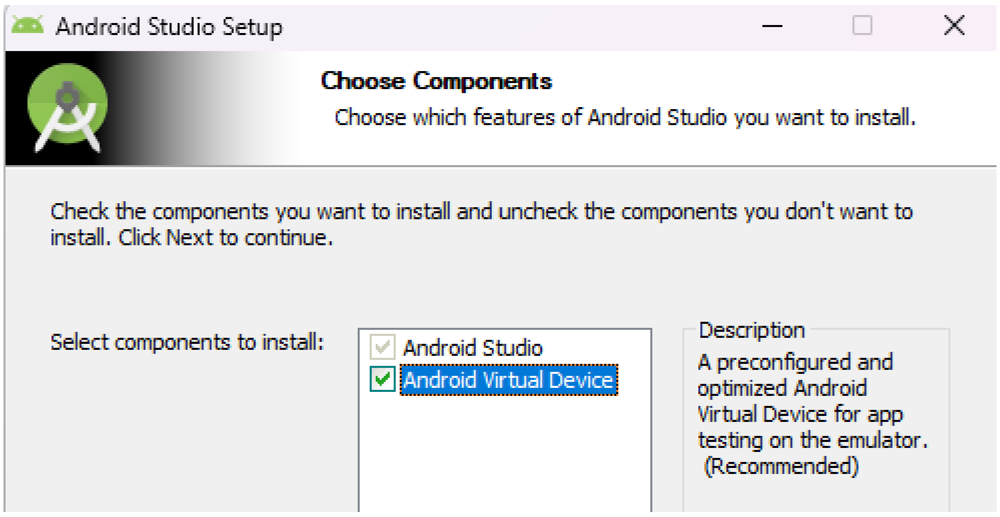
Kia Ora! Welcome to our first lab session on Mobile App Development! Over this semester, we'll be taking a deep dive into many different areas of mobile application development. Our main focus will be Android applications, but you'll be able to take the knowledge learned in this class and use it to help you create other applications such as iOS apps, web apps and even fully featured desktop applications. App development has come a long way since the early days – now is a better time than ever to get head-first into developing your first apps!

Our app development journey is going to be split into two major parts. For this semester, we're going to be focused on traditional app development with Android Studio. We'll be using Java which is the native programming language for Android in order to learn the ins and outs of creating apps from scratch. If you decide to continue with this paper in level 7, we'll then move into learning a newer technology called Flutter. Flutter is a framework that transforms the app development process, by allowing you to write code from a single codebase and deploy it to many different platforms. As part of learning Flutter, we'll be learning the basics of a new programming language called Dart.

In this lab specifically, we're going to be downloading, configuring and installing everything that we're going to need to work on our projects this semester. We'll then get head-first into creating our first Android App! If at any point you have any questions during this session or future sessions, please feel free to call me over to help or send me [an email](). I'm always happy to answer any queries you have.

Getting started

To begin with, let's get our development environment all up and running. If you are using your own personal device, I'd recommend downloading and installing Android Studio locally before proceeding. When installing Android Studio, ensure that the **Android Virtual Device** component is checked.


Android Studio Setup

Choose Components

Choose which features of Android Studio you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:

- [✓] Android Studio
- [✓] Android Virtual Device

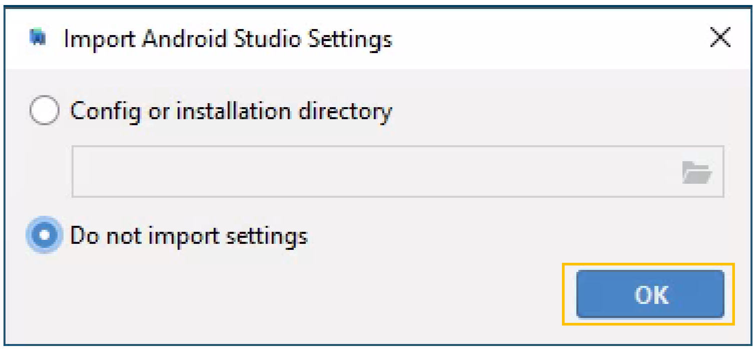
Description

A preconfigured and optimized Android Virtual Device for app testing on the emulator. (Recommended)

If you're using a Te Pūkenga provided computer in one of our specialty labs, this software should already be installed locally. Unlike other software used in other papers, Android Studio does not work using WVD.

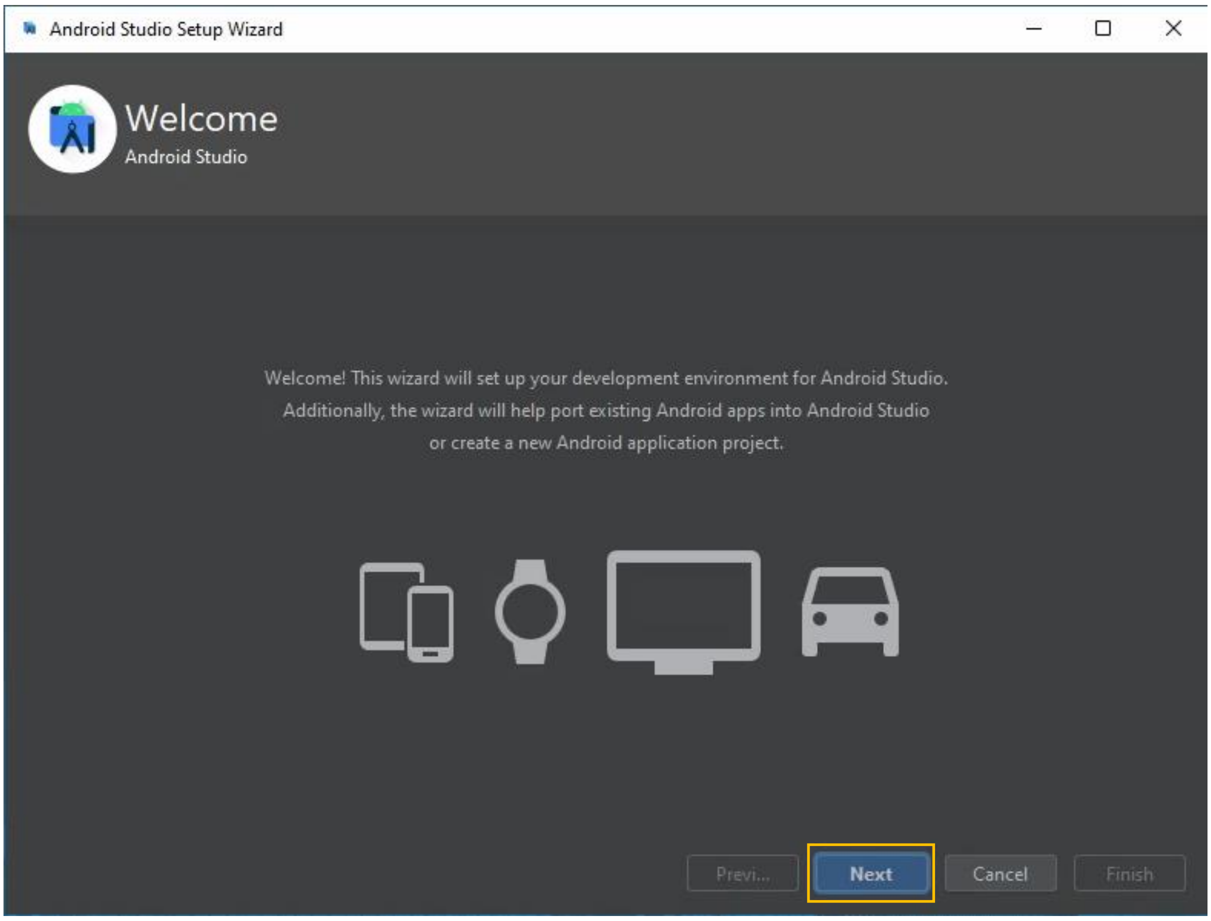
Configuring Android Studio

Upon first launch, you'll be asked to import launch settings. Just leave **Do not import settings** checked and click **OK**.


Import Android Studio Settings

Config or installation directory [✓ Do not import settings]

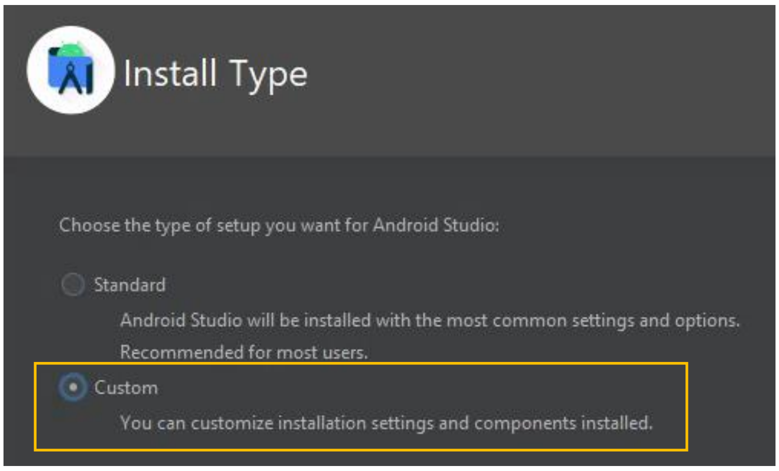
If this is your first time opening up Android Studio on this device, you'll first be greeted with the setup wizard. It'll look a bit like this:


Android Studio Setup Wizard

Welcome
Android Studio

Welcome! This wizard will set up your development environment for Android Studio.
Additionally, the wizard will help port existing Android apps into Android Studio
or create a new Android application project.

Click **Next**, select **Custom** and then click **Next** again.

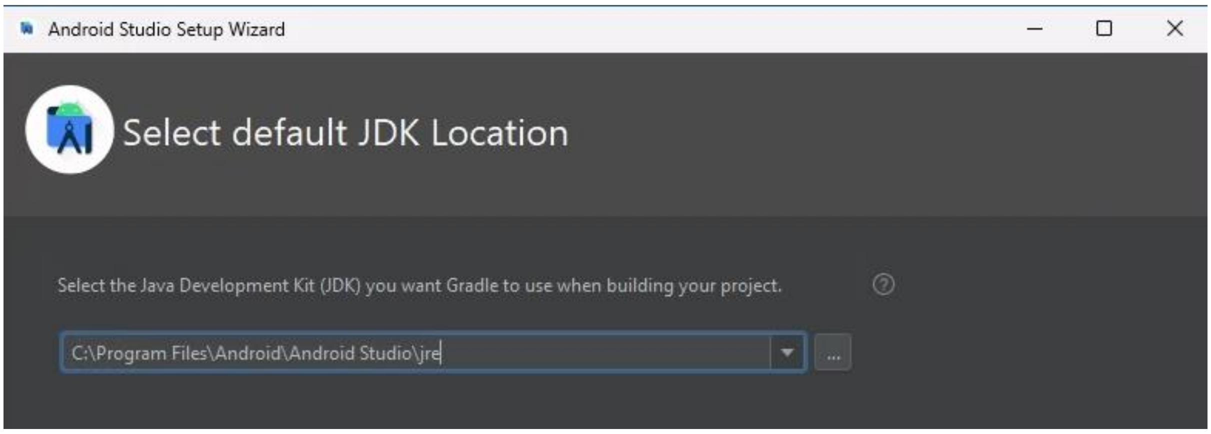

Install Type

Choose the type of setup you want for Android Studio:

Standard
Android Studio will be installed with the most common settings and options.
Recommended for most users.

[✓ Custom]
You can customize installation settings and components installed.

Leave the default development kit and click **Next**.

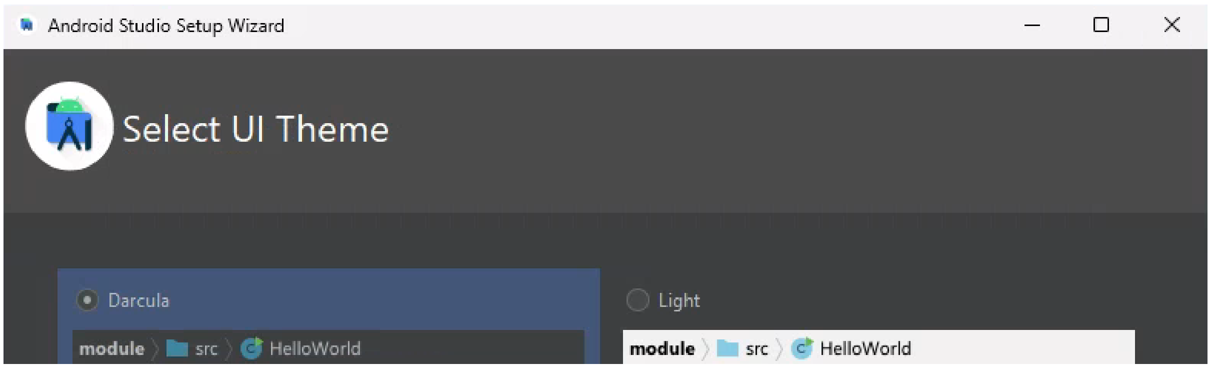

Android Studio Setup Wizard

Select default JDK Location

Select the Java Development Kit (JDK) you want Gradle to use when building your project.

C:\Program Files\Android\Android Studio\jre

Now select the UI theme you want to use and click **Next** again.

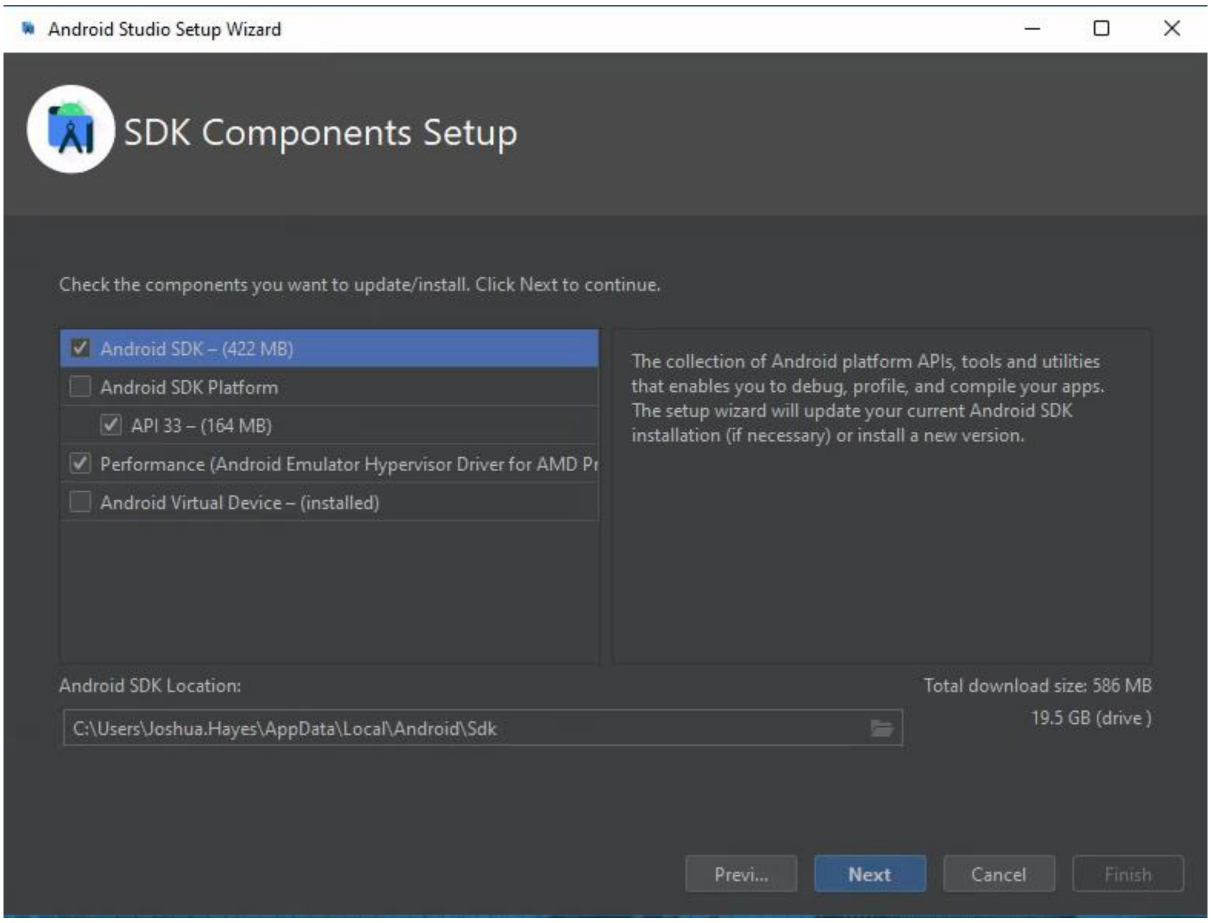

Android Studio Setup Wizard

Select UI Theme

● Darcula ○ Light

For the SDK Components setup, those on a Te Pūkenga provided computer in a specialty lab can just hit **next**. It will then tell you that SDK Components are already installed and complete the setup wizard automatically.

If you are on your own personal device, ensure that you have the Android ASK, API and Performance options selected. If **Android Virtual Device** is unchecked and allows you to check it, make sure you have it selected. (Some devices may say it is unavailable). It should look a bit like this:


Android Studio Setup Wizard

SDK Components Setup

Check the components you want to update/install. Click Next to continue.

- ✓ Android SDK – (422 MB)
- Android SDK Platform
  - ✓ API 33 – (164 MB)
- ✓ Performance (Android Emulator Hypervisor Driver for AMD Pr
- Android Virtual Device – (installed)

The collection of Android platform APIs, tools and utilities that enables you to debug, profile, and compile your apps. The setup wizard will update your current Android SDK installation (if necessary) or install a new version.

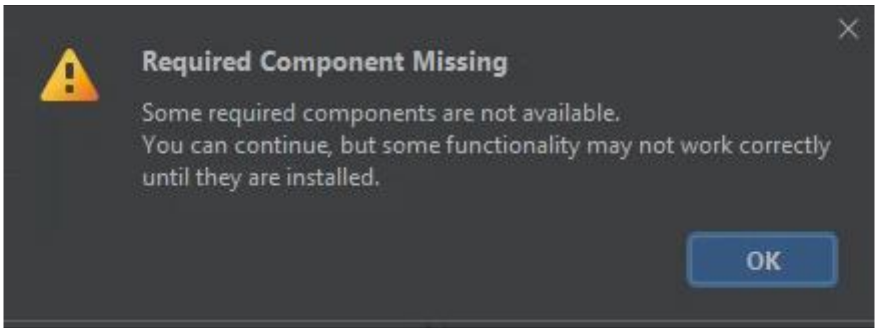
Android SDK Location:

C:\Users\Joshua.Hayes\AppData\Local\Android\Sdk

Total download size: 586 MB

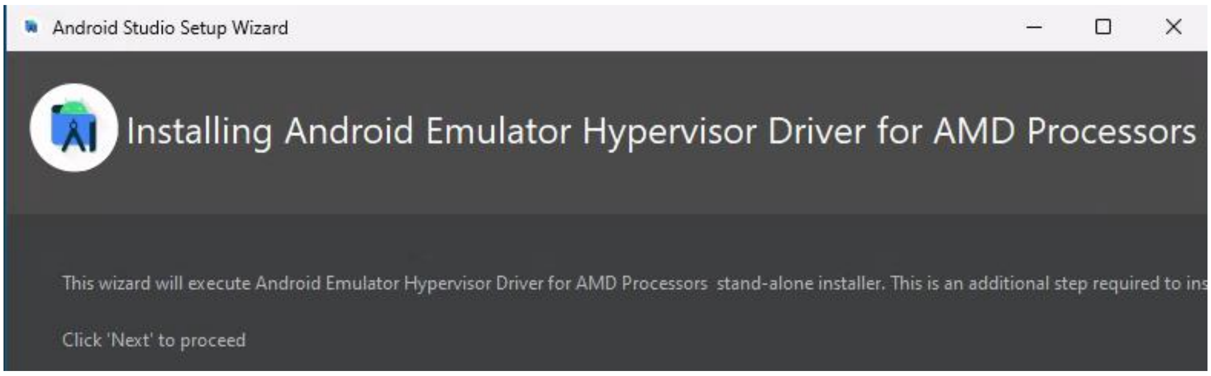
19.5 GB (drive)

Now click **Next**. You can dismiss this dialog if you get it – we can sort that out later.


Required Component Missing

Some required components are not available.
You can continue, but some functionality may not work correctly until they are installed.

If you're on an AMD processor, you may get a screen a bit like this. Just hit **next**.

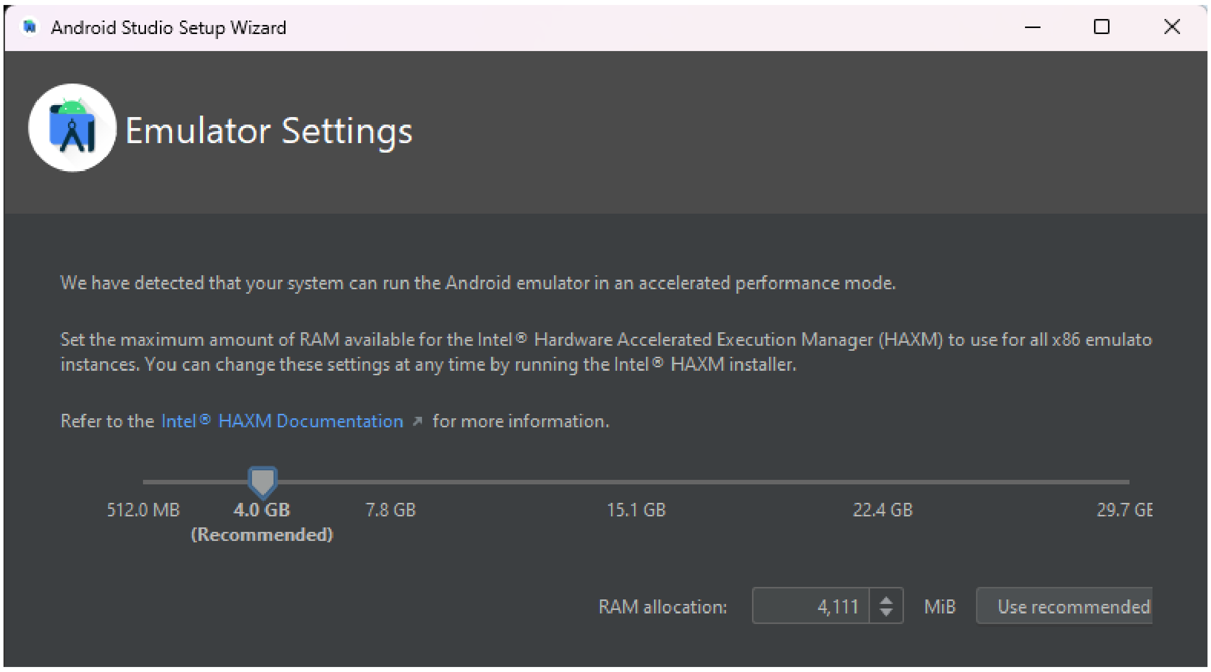

Android Studio Setup Wizard

Installing Android Emulator Hypervisor Driver for AMD Processors

This wizard will execute Android Emulator Hypervisor Driver for AMD Processors stand-alone installer. This is an additional step required to ins

Click 'Next' to proceed

If you're on an Intel processor, you might get some emulator settings like this. Just ensure that you've clicked **Use recommend** and hit **Next**.



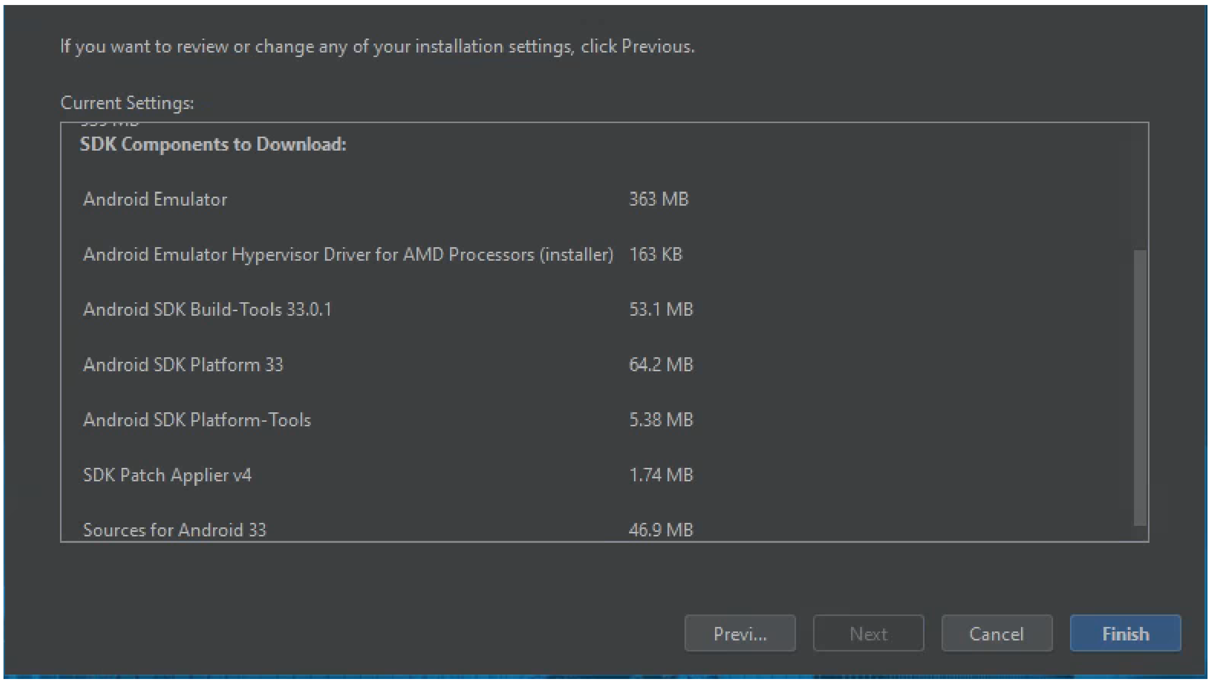
We have detected that your system can run the Android emulator in an accelerated performance mode.

Set the maximum amount of RAM available for the Intel® Hardware Accelerated Execution Manager (HAXM) to use for all x86 emulato instances. You can change these settings at any time by running the Intel® HAXM installer.

Refer to the Intel® HAXM Documentation for more information.

RAM allocation: 4,111 MiB

Finally, click **Finish**.



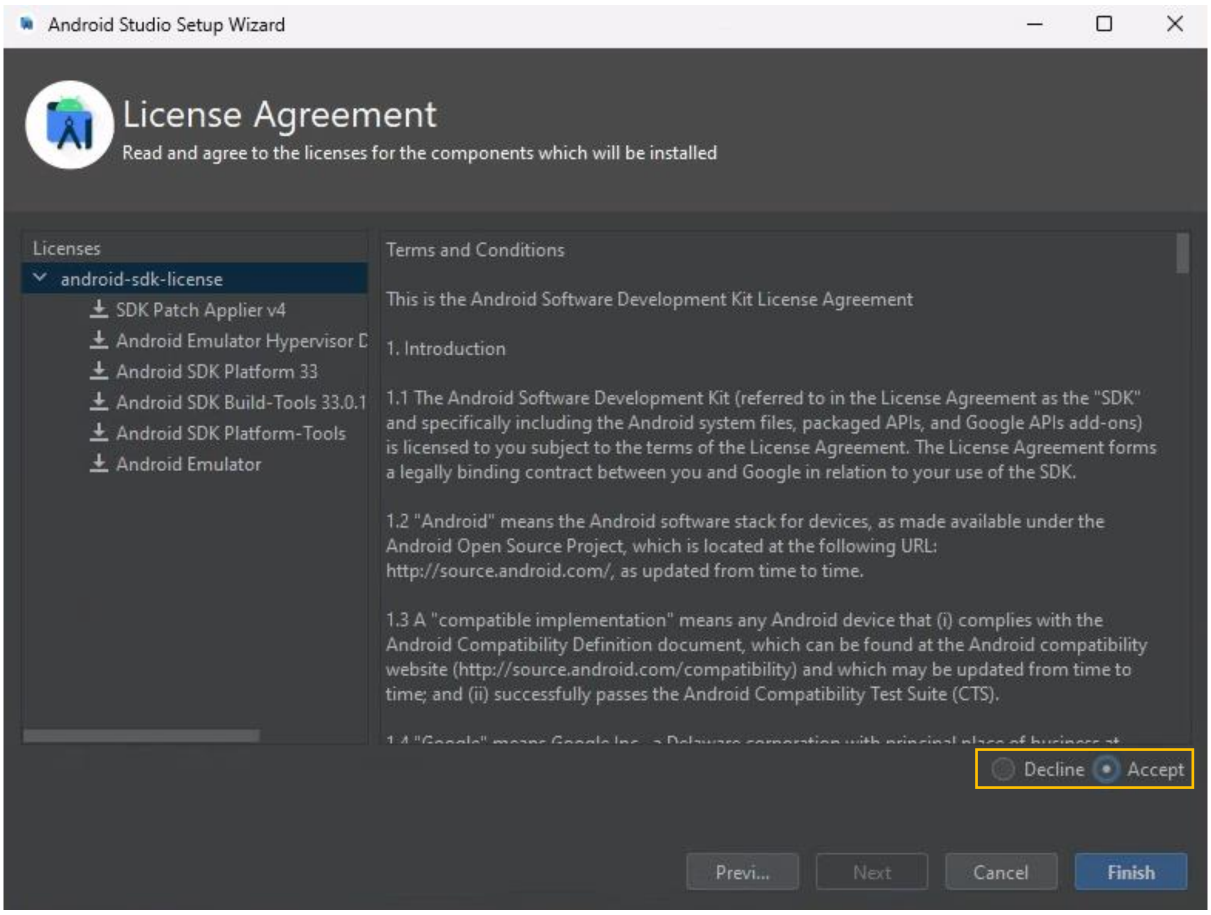
If you want to review or change any of your installation settings, click Previous.

Current Settings:

SDK Components to Download:

Android Emulator	363 MB
Android Emulator Hypervisor Driver for AMD Processors (installer)	163 KB
Android SDK Build-Tools 33.0.1	53.1 MB
Android SDK Platform 33	64.2 MB
Android SDK Platform-Tools	5.38 MB
SDK Patch Applier v4	1.74 MB
Sources for Android 33	46.9 MB

If you are given the Licence Agreement screen, ensure that **Accept** is checked for each of the options on the left and then click **Finish** again.


Android Studio Setup Wizard

License Agreement

Read and agree to the licenses for the components which will be installed

Licenses
- android-sdk-license
  - SDK Patch Applier v4
  - Android Emulator Hypervisor D
  - Android SDK Platform 33
  - Android SDK Build-Tools 33.0.1
  - Android SDK Platform-Tools
  - Android Emulator

Terms and Conditions

This is the Android Software Development Kit License Agreement

1. Introduction

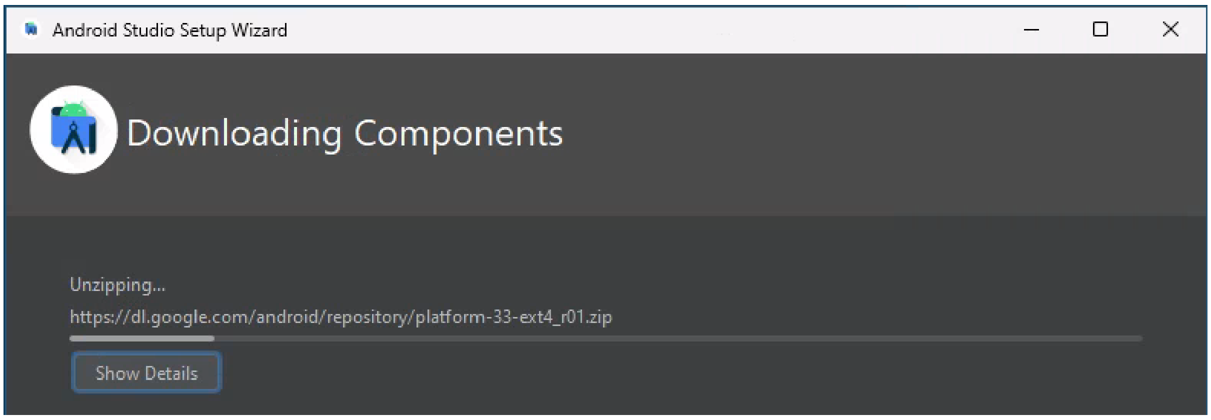
1.1 The Android Software Development Kit (referred to in the License Agreement as the "SDK" and specifically including the Android system files, packaged APIs, and Google APIs add-ons) is licensed to you subject to the terms of the License Agreement. The License Agreement forms a legally binding contract between you and Google in relation to your use of the SDK.

1.2 "Android" means the Android software stack for devices, as made available under the Android Open Source Project, which is located at the following URL: http://source.android.com/, as updated from time to time.

1.3 A "compatible implementation" means any Android device that (i) complies with the Android Compatibility Definition document, which can be found at the Android compatibility website (http://source.android.com/compatibility) and which may be updated from time to time; and (ii) successfully passes the Android Compatibility Test Suite (CTS).

Decline [✓ Accept]

It'll take a bit to download, but once it's complete you'll be able to click **Finish** again to finish the setup wizard.

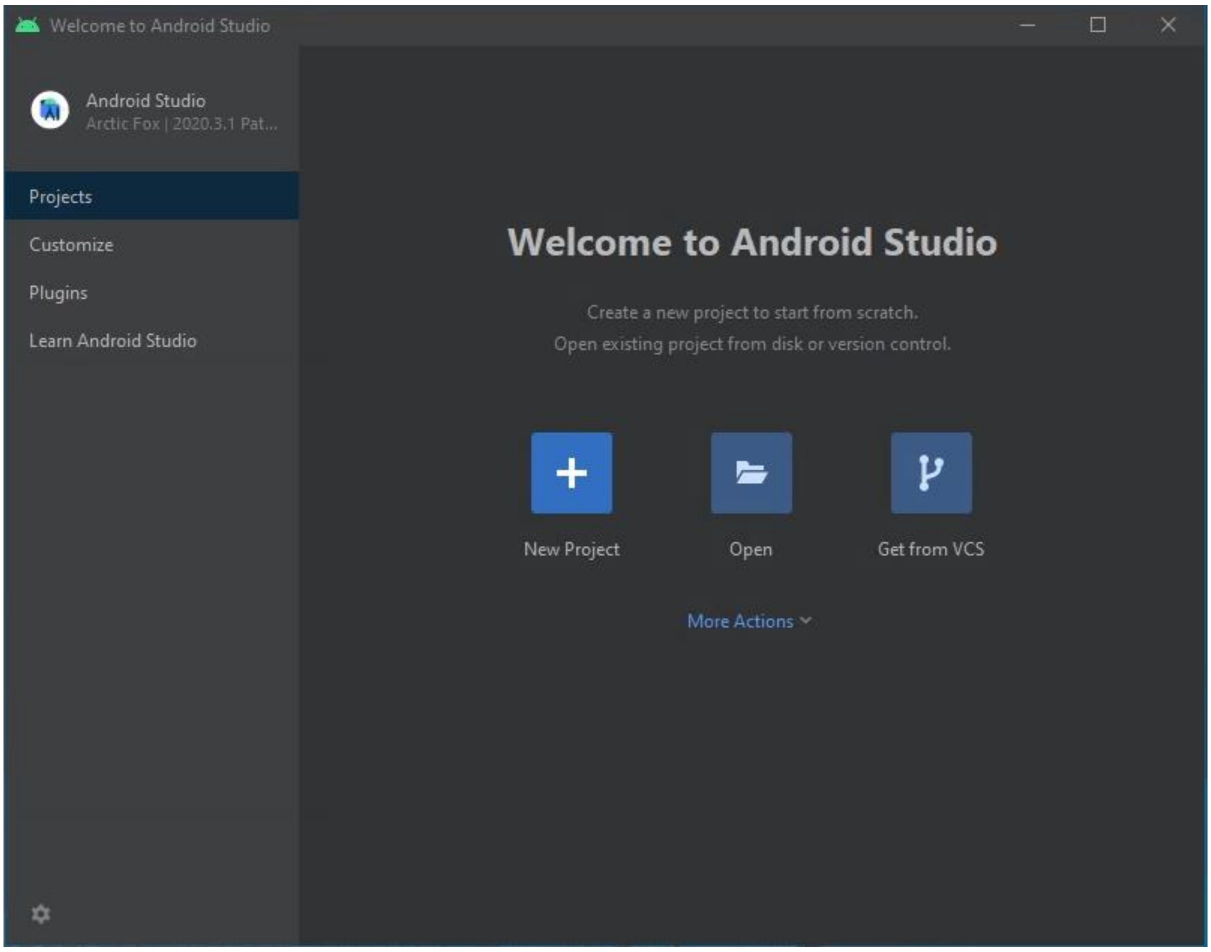

Android Studio Setup Wizard

Downloading Components

Unzipping...

https://dl.google.com/android/repository/platform-33-ext4_r01.zip

Now you should be greeted by the Android Studio home-screen, which should look a bit like this.

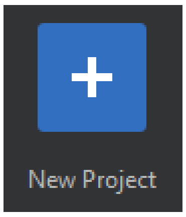

Welcome to Android Studio
Android Studio
Arctic Fox | 2020.3.1 Pat...
Projects
Customize
Plugins
Learn Android Studio
Welcome to Android Studio
Create a new project to start from scratch.
Open existing project from disk or version control.
New Project
Open
Get from VCS
More Actions

This is where you'll be able to manage your plugins, create/open projects or learn a bit more about Android Studio itself. While you can use other editors to develop android apps, we'll be using Android Studio as it is a lot easier to learn this way with Java and Kotlin.

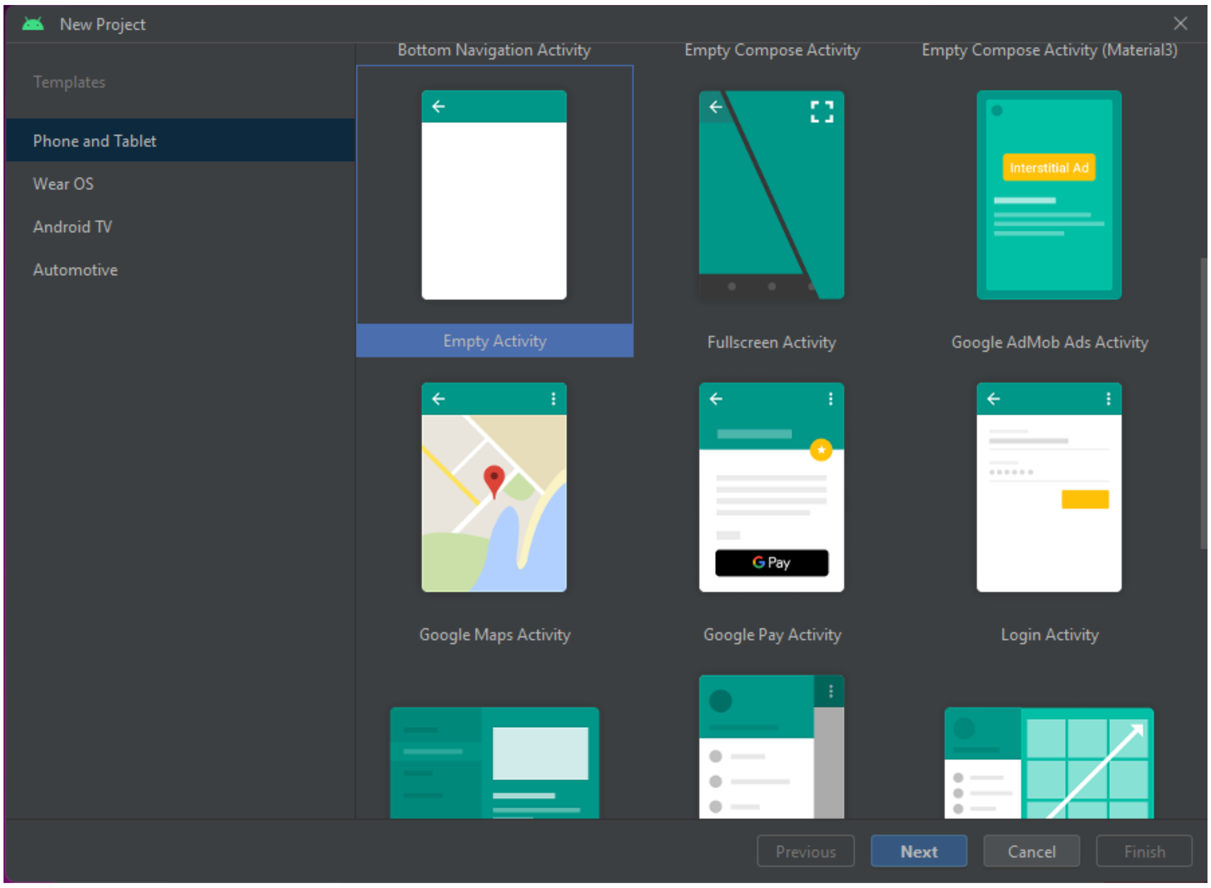
Creating our first app

It's time to start creating our first Android App.

On the **Projects** tab of Android Studio, select **New Project**.

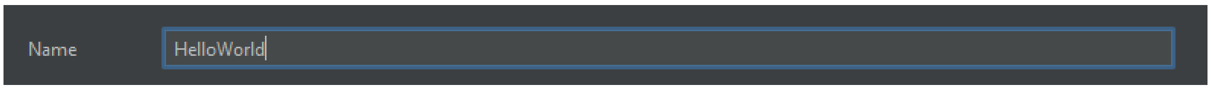


Whenever you create a new project in Android Studio, you'll be presented with a bunch of templates to help get you started. Ensure that you're in the Phone and Tablet tab, and select **Empty Activity**.



Once you select **Next**, you'll be presented with a bunch of application settings you can configure. Let's break down each one of them.

The **Name** just represents your project name. Call your project **HelloWorld**.

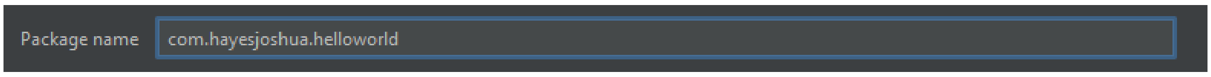


Name: HelloWorld

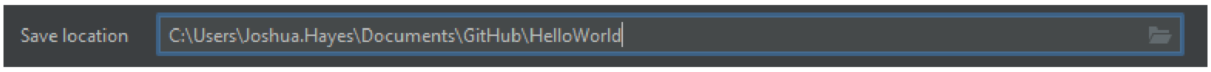
The **Package Name** is a bit more important, as it's the way Android uniquely identifies each application. The best way to think of it, is as a reverse domain name. **com** is the Top-Level

Domain (TLD), the middle part is your company name (or domain name), and the last part is your application name (or sub-domain).

For the purposes of this class, I'd like you to name your apps **com.[yourname].[appname]**. This would help you (and me) identify who made what apps.


Package name com.hayesjoshua.helloworld

The **Save Location** is pretty self-explanatory. I'd either recommend saving your projects in a **OneDrive** folder, or alternately saving them in a folder to back up against a repository such as GitHub.


Save location C:\Users\Joshua.Hayes\Documents\GitHub\HelloWorld

As mentioned in the introduction, the **Language** we'll be using this semester is **Java**.

Language Java

The **Minimum API level** represents the oldest version of Android that your app will be able to run on. The lower this API level, the more devices you will be able to run your app on. For this class, we'll be targeting **API 26: Android 8.0 (Oreo)** which should run on around 90% of all devices.

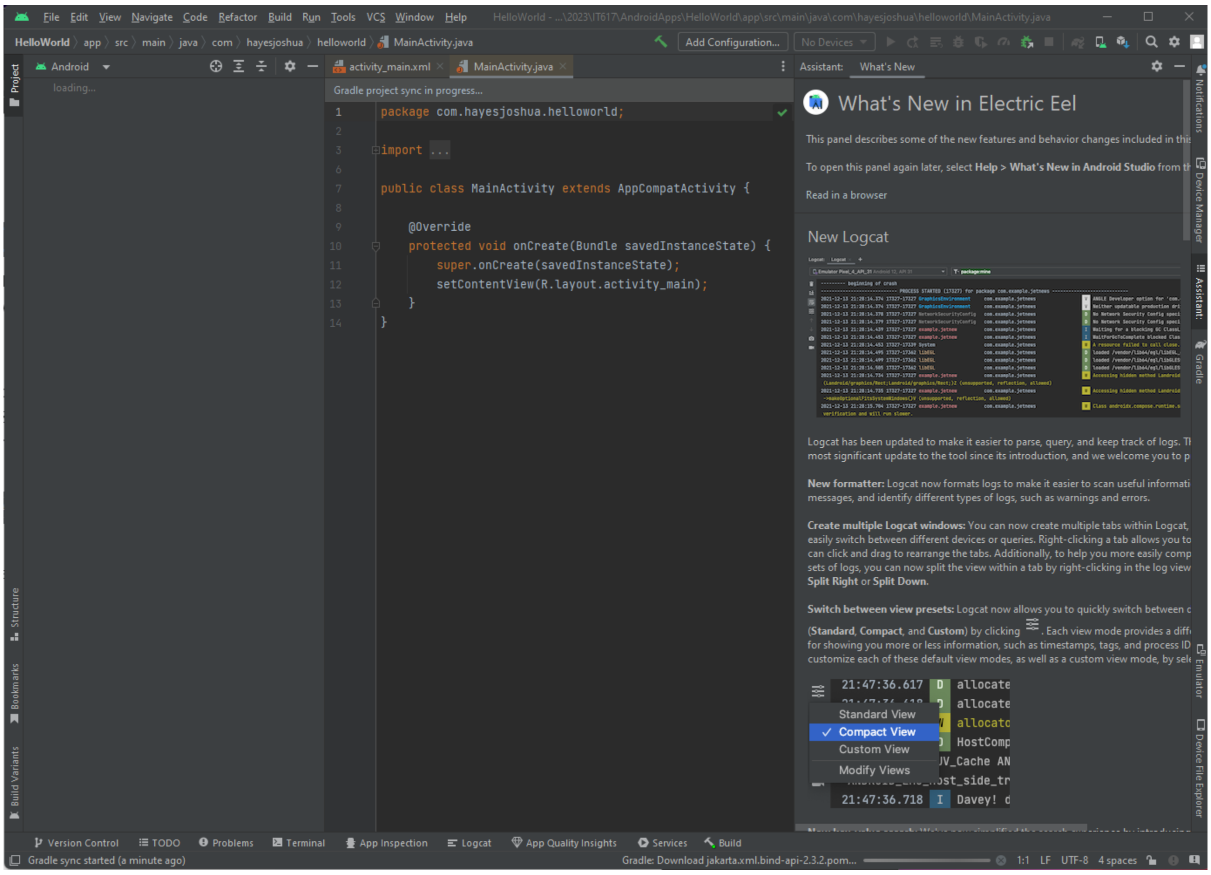
If you have an Android phone that you'd like to use for testing your apps, chances are high that it is Android 8.0 or newer. If you're using an older version than 8.0, I'd suggest sticking to an emulator for the purpose of debugging. (You can select a lower SDK version if you'd like, but all of our labs will assume you're targeting API 26).

Minimum SDK API 26: Android 8.0 (Oreo)

Finally, hit **Finish**. If this is the first time you've created an app, it'll take a bit to download. Once it's completed, just click **Finish** again to head into Android Studio.

Tour of Android Studio

This is the homepage of Android Studio, where you'll be spending most of your time. Let's take some time to explore it.



```
package com.hayesjoshua.helloworld;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

What's New in Electric Eel

This panel describes some of the new features and behavior changes included in this

To open this panel again later, select **Help > What's New in Android Studio** from th

Read in a browser

New Logcat

Logcat has been updated to make it easier to parse, query, and keep track of logs. Th most significant update to the tool since its introduction, and we welcome you to p

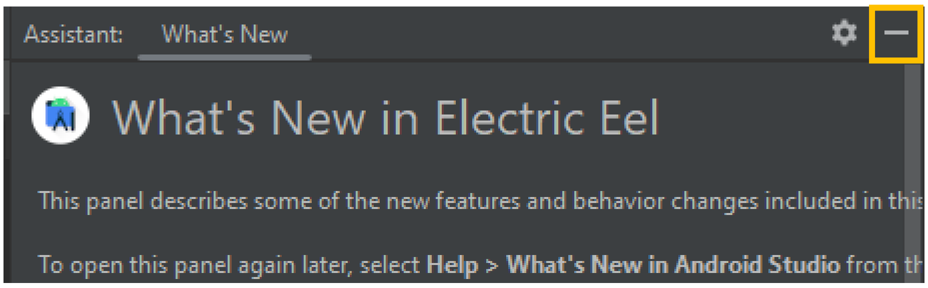
New formatter: Logcat now formats logs to make it easier to scan useful informati messages, and identify different types of logs, such as warnings and errors.

Create multiple Logcat windows: You can now create multiple tabs within Logcat, easily switch between different devices or queries. Right-clicking a tab allows you to can click and drag to rearrange the tabs. Additionally, to help you more easily comp sets of logs, you can now split the view within a tab by right-clicking in the log view **Split Right** or **Split Down**.

Switch between view presets: Logcat now allows you to quickly switch between d (**Standard**, **Compact**, and **Custom**) by clicking . Each view mode provides a diff for showing you more or less information, such as timestamps, tags, and process ID customize each of these default view modes, as well as a custom view mode, by sele

Standard View
Compact View
Custom View
Modify Views
Gradle project sync in progress...
Gradle sync started (a minute ago)

The **What's New** section takes up a bit of space, so press the minimise button in the corner to hide it.

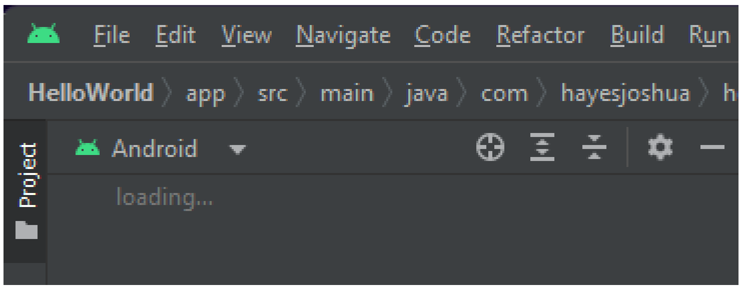


What's New in Electric Eel

This panel describes some of the new features and behavior changes included in this

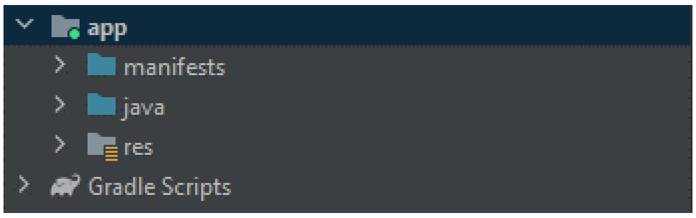
To open this panel again later, select **Help > What's New in Android Studio** from th

The top-left of Android Studio is where you find the structure of your application. If it says **loading** and does not display any files, just wait a few minutes while **Gradle** automatically downloads all the required files.



File Edit View Navigate Code Refactor Build Run
HelloWorld › app › src › main › java › com › hayesjoshua › h
Project
Android
loading...

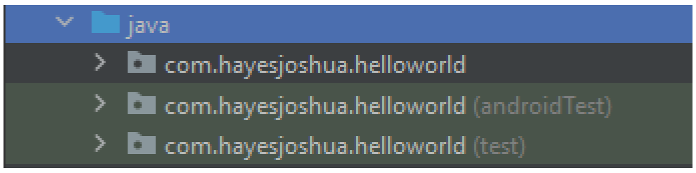
Once it's all loaded, it should look like this:



Your mobile app is generally split into three main folders.

- Inside the **java** folder, you'll find all the **source code** of your application. This is where the core of the project's Java code sits during development.
- The **res** folder contains source files and resources for your app go, such as images, videos and **XML** files describing how your app should look.
- The **manifests** folder contains information about your app package, including what activities, services etc. that your application contains. Whenever we create a new activity in our app, or want to define our application icon or theme, we'll do that here.

If you head inside the **java** folder, you'll notice that there is three different **com.[name].[helloworld] folders**.

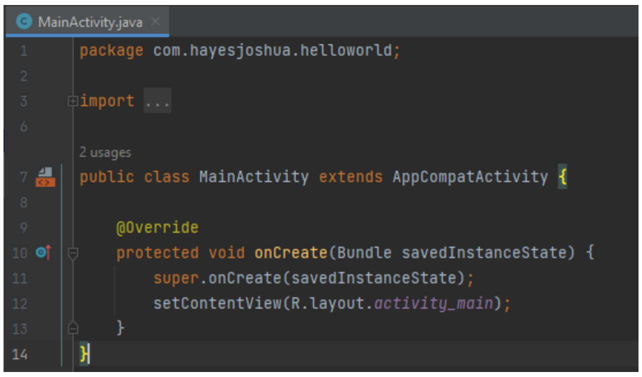


java
com.hayesjoshua.helloworld
com.hayesjoshua.helloworld (androidTest)
com.hayesjoshua.helloworld (test)

The first folder is where you will be holding all of your source code, whereas the other two are for running tests against your app. You can safely ignore those other two folders with the test tag.

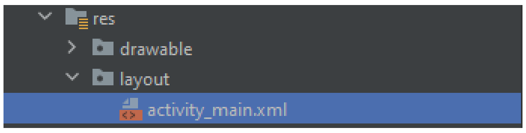
There are a few specific files I'd like to highlight here as well.

- Under **app** > **java** > **com.[name].[helloworld]** > **MainActivity** is the **MainActivity** file. As the name suggests, this is the main activity of your app. Each separate "screen" in your Android App is represented by an Activity, and the code inside represents the behaviour for that activity.

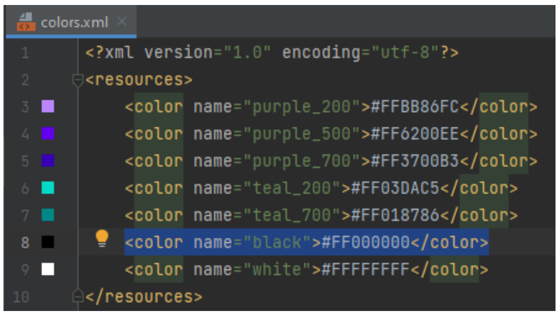


```
package com.hayesjoshua.helloworld;

import ...

2 usages
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

- Under **res** > **layout** > **activity_main.xml**, you'll find a layout file for your **Main Activity**. These XML files describe how each screen of your application should look.

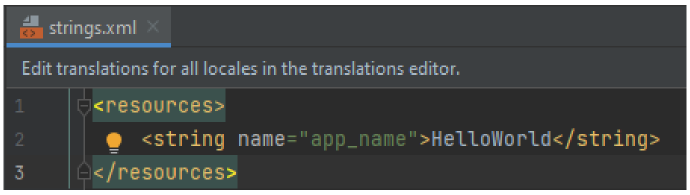


- Under **res** > **values** > **colors.xml**, you'll find another XML file that describes some pre-defined colours that will be used across your application. Consider this file similar to the way variables work when programming. You could create a variable called **black** that has the value "#FF000000", and use it throughout your application.



```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
</resources>
```

  There are some other files under the **values** folder for other variable names, such as **strings.xml** which just defines common text used in your app. This could be used for information such as your application's name, or for creating different translations of your app in different parts of the world.

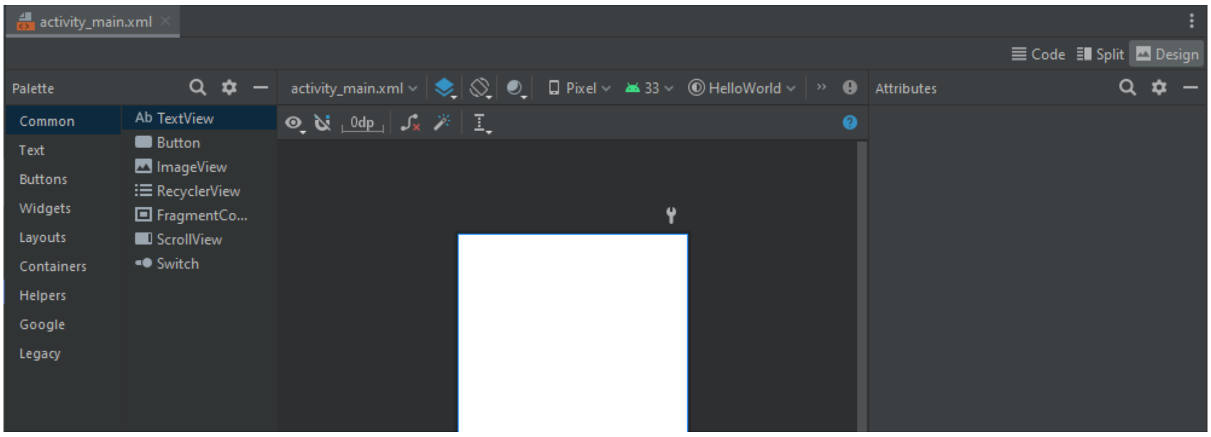


Edit translations for all locales in the translations editor.

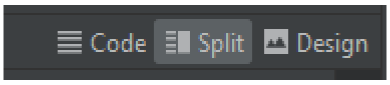
```xml
<resources>
    <string name="app_name">HelloWorld</string>
</resources>
```

Configuring our Layout

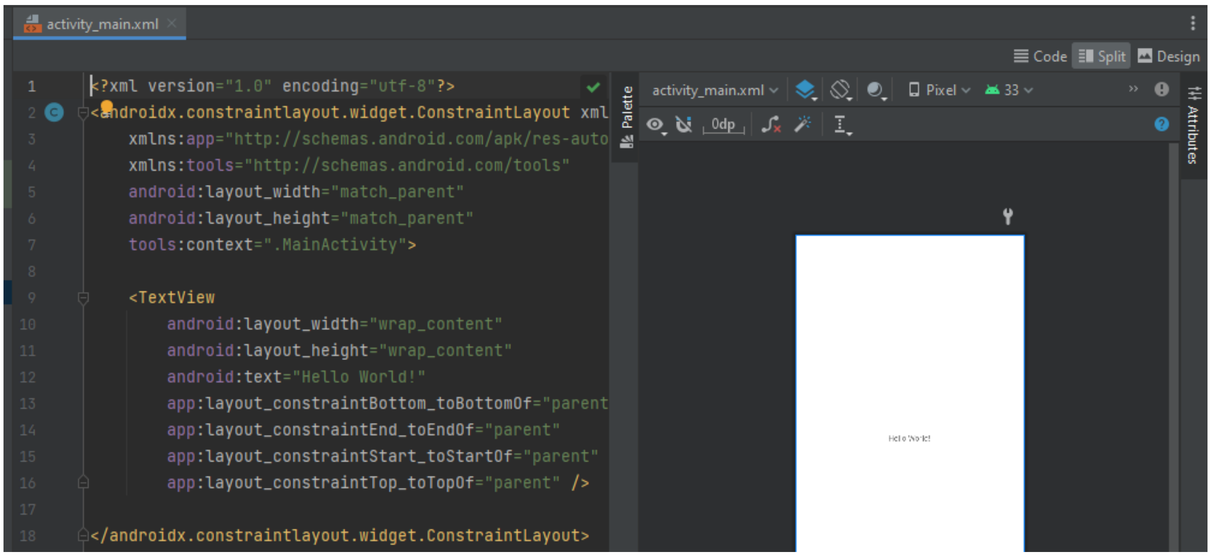
Let's start configuring our app. Open up your **layout_main.xml** file that I showed you earlier.



When you open it for the first time, Android Studio will automatically open it in **Design** view – which can help us quickly throw together views in our app. We're going to be modifying our XML directly in Code view, so switch to either **Code** or **Split** in the top right of the window. (I'd recommend Split so you can see your changes in real-time).



Your window should now look a bit like this:



```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xml
    xmlns:app="http://schemas.android.com/apk/res-auto
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Many files you'll be modifying while creating your apps on Android will be utilising XML, so it's important to understand the basics of how XML works. The easiest way to understand XML if you've never used it before, is to think of it like HTML. XML contains both opening and closing tags, and those tags contain different information. XML in itself doesn't do anything, but contain a bunch of information for another application to read. If you've never used XML before, I'd suggest reading up a bit more here before continuing.

The code of our **activity_main.xml** file should look like this:

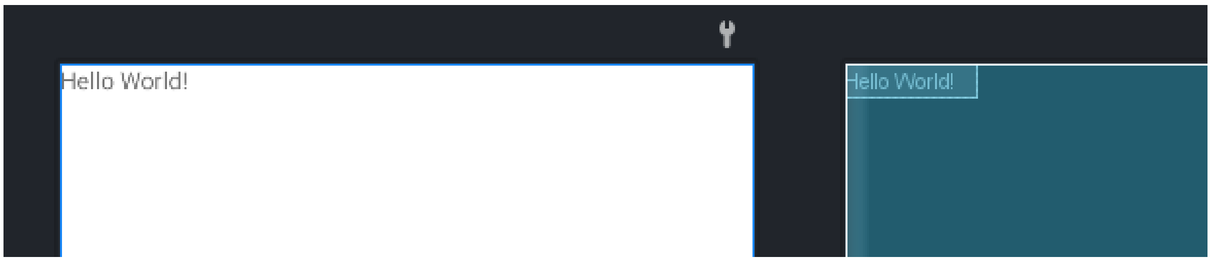
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Most of our layout defined here is currently wrapped inside of a **ConstraintLayout** XML tag. Go ahead and try change this to a **LinearLayout**.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</LinearLayout>
```

The LinearLayout displays all child tags in a stack one after the other in either a column or a row, similar to an unordered list in HTML without the bullet points.

If you've got the Split view open, you should immediately see the results. The text should now be displayed in the top left of our app, as we only have one item in our LinearLayout.



Let's add a few more attributes inside of our **LinearLayout** tag for orientation (as we want our items arranged in a column) and padding.

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    tools:context=".MainActivity">

    . . .

</LinearLayout>
```

The padding ensures that each item in our layout is spaced apart from each other in all directions. **dp** is a measurement of unit you'll see a lot in Android, that stands for **density-independent pixels**. Some screens have a much higher density of pixels than others, so if we use this unit of measurement, we can ensure that our app will be sized correctly across many different screen sizes.

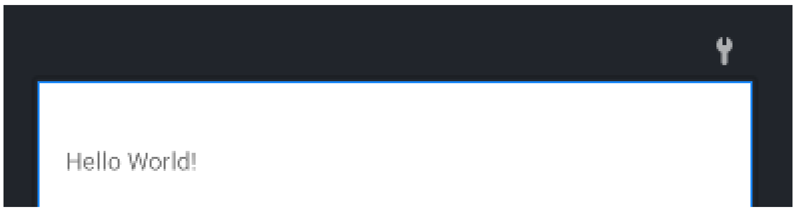
Let's add an attribute to our **TextView** tag called **marginTop** and set it to **20dp** to push our text item down slightly.

```
. . .
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:layout_marginTop="20dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
. . .
```

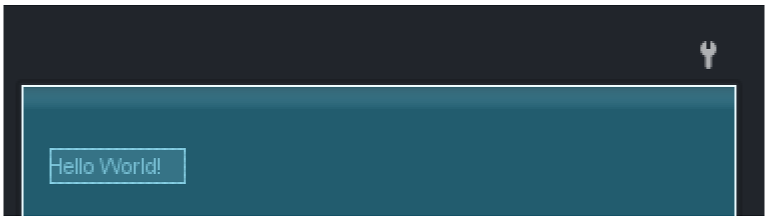
You should see our text live-update in our Design tab.

You've probably noticed by now that our design tab is split into two different parts. There's a white screen on the left, and an almost identical blue one on the right.

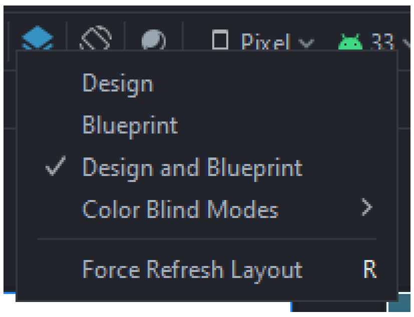
The white screen is the **Design** view itself. This will attempt to show you how the current activity will look when you compile and run the app on an actual Android device.



The blue screen is what we call the **Blueprint** view. This will try and remove the visual complexity so you can see how components are laid out in your app. This is useful for seeing how large components are compared to others, or to quickly find out if components are overlapping.



If you'd like to toggle between them, you can do so with the **blue icon** at the top of the **Design** tab.



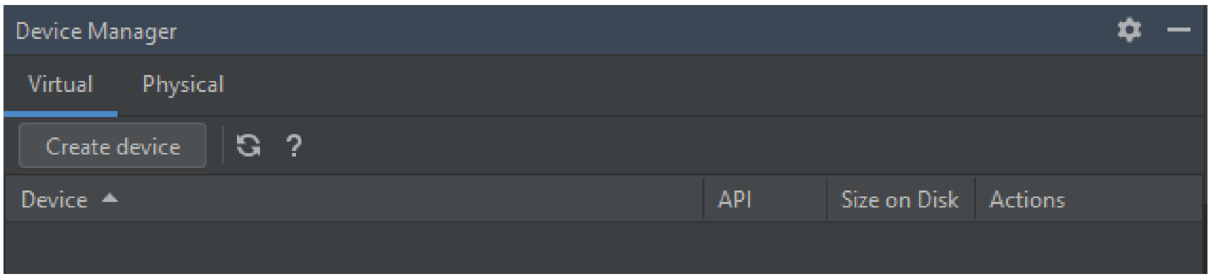
It's worth noting that the Design view itself isn't always completely representative of how our app will look on an actual device, but it is a good first start. Let's compile our app and see how it will look in Android itself!

Running our App

Android Studio provides us with two main ways we can test our application; either using a physical device, or a form of emulator. To begin with, we're going to be running you through setting up the emulator – but later down the line you're welcome to use your own Android phone for testing if you wish!

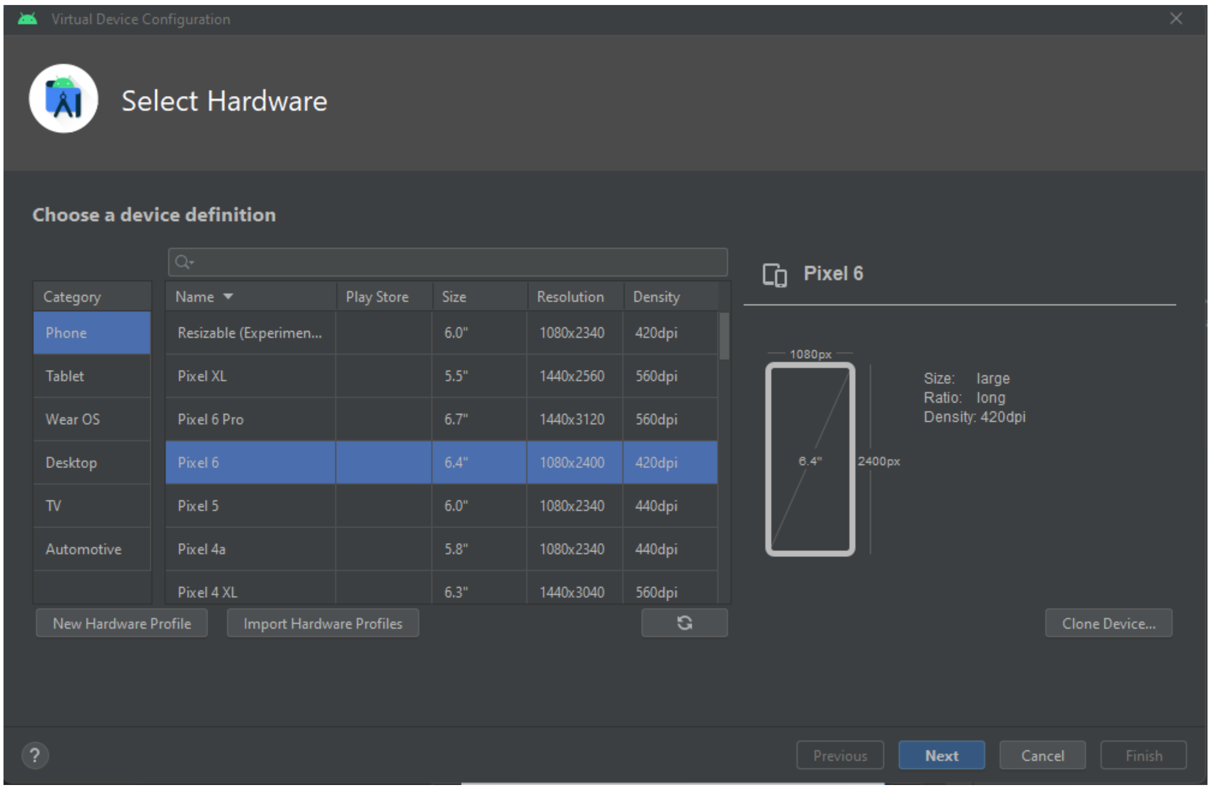
Android Studio has a built-in emulator called the **AVD**, or Android Virtual Device. This allows us to quickly test our app on a variety of different devices with different screen sizes and different Android versions – all without having to buy heaps of different phones! Let's go ahead and set this up now.

At the top of Android Studio, head to **Tools** > **Device Manager**.



Device	API	Size on Disk	Actions

Now click **Create Device**. This is where we can configure what kind of device we want to test our app on. It's up to you what device you'd like to create, but I'd recommend going for the **Pixel 6** dimensions. Once you've selected it, click **Next**.



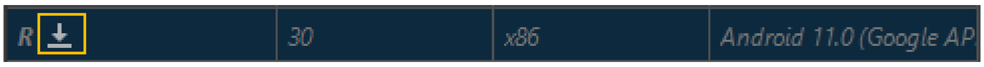
Category	Name	Play Store	Size	Resolution	Density
Phone	Resizable (Experimen...		6.0"	1080x2340	420dpi
Tablet	Pixel XL		5.5"	1440x2560	560dpi
Wear OS	Pixel 6 Pro		6.7"	1440x3120	560dpi
Desktop	Pixel 6		6.4"	1080x2400	420dpi
TV	Pixel 5		6.0"	1080x2340	440dpi
Automotive	Pixel 4a		5.8"	1080x2340	440dpi
	Pixel 4 XL		6.3"	1440x3040	560dpi

Pixel 6

1080px, 2400px, 6.4"

Size: large
Ratio: long
Density: 420dpi

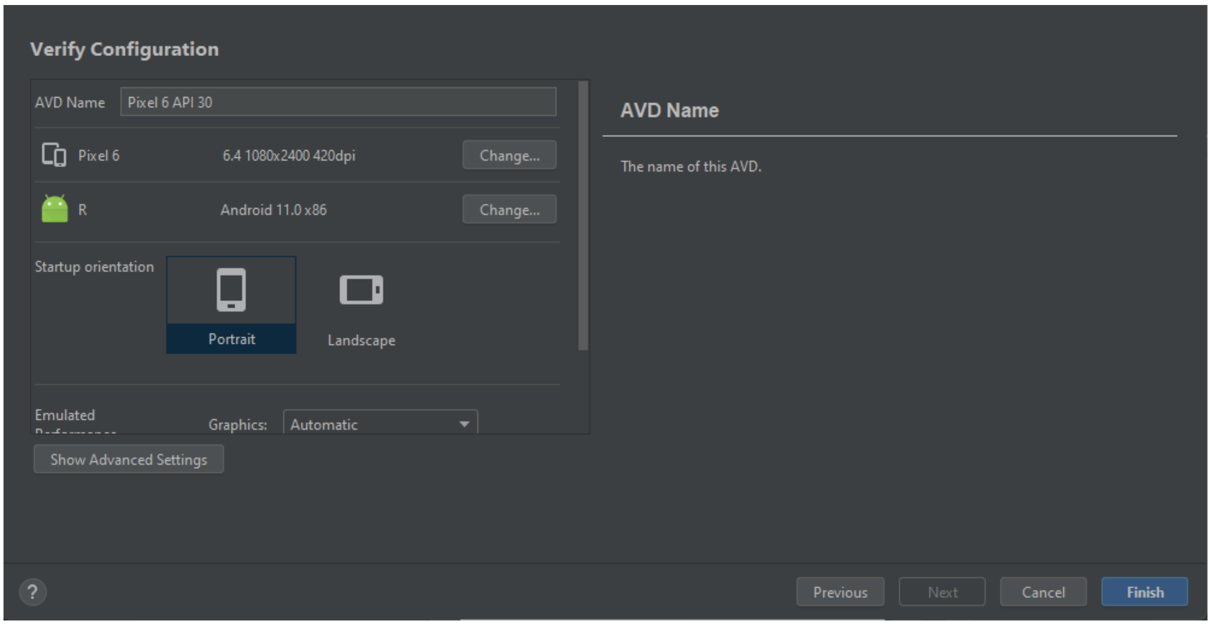
This is where we can select what Android Version we'd like to install on our device. For the purpose of this demo, I'll be installing Android R (11.0). You may need to download the image before you can continue by clicking the **download icon** beside the name.



R	30	x86	Android 11.0 (Google AP

It may take a while to download, so take a moment to relax and take in everything we've learned so far! Finally, hit **Next**.

We can now name our device if we wish. I'm going to keep the default name and hit **Finish**.


Verify Configuration

AVD Name: Pixel 6 API 30

Pixel 6 — 6.4 1080x2400 420dpi — Change...

R — Android 11.0 x86 — Change...

Startup orientation: Portrait / Landscape

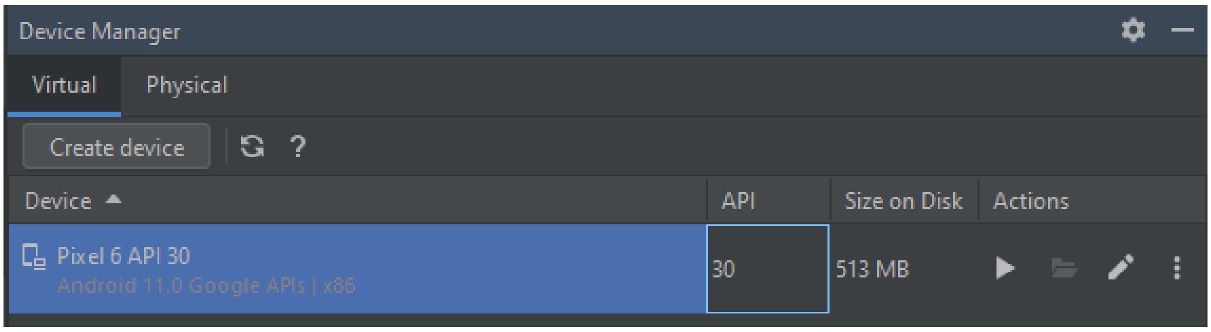
Emulated Performance — Graphics: Automatic

Show Advanced Settings

AVD Name

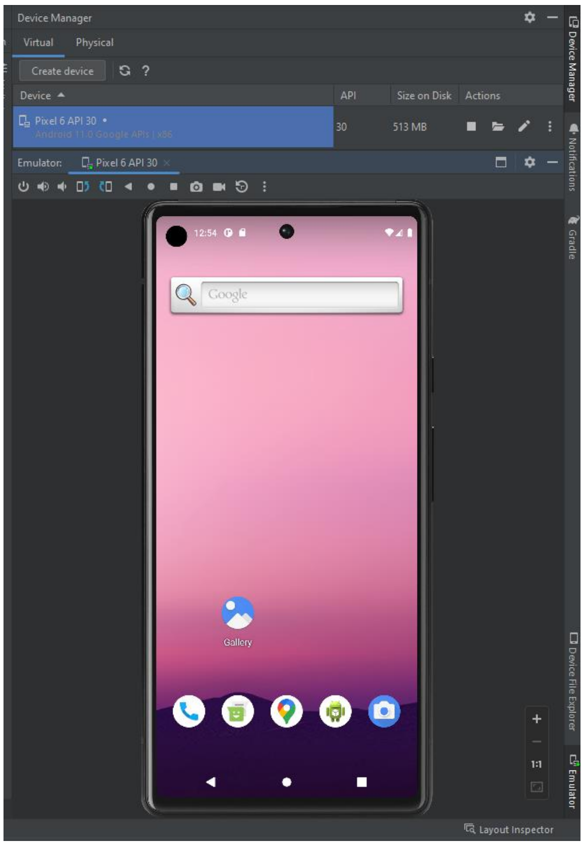
The name of this AVD.

Now under Device Manager, we should have our new Virtual Device created and ready to use. Click the **play** icon beside it to start up the device.



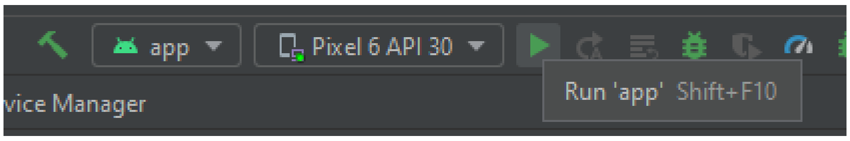
Device	API	Size on Disk	Actions
Pixel 6 API 30 Android 11.0 Google APIs | x86	30	513 MB	

It may take a while to boot for the first time, but once it's done it should look like this:

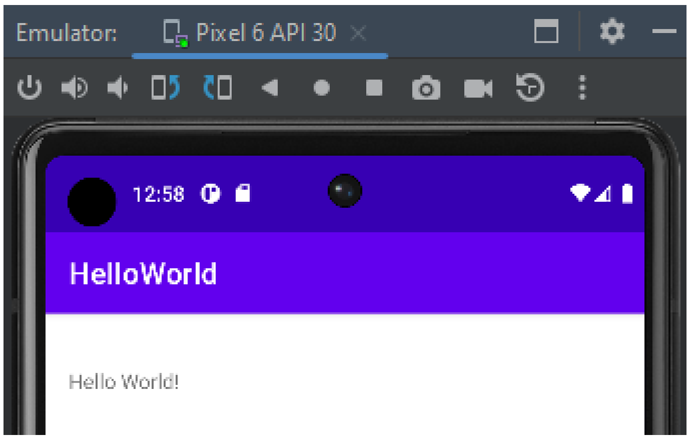


This is a very lightweight version of Android 11 which is why there are some widgets and apps that look a lot older, but it should still run like any physical Android device. Let's test our app.

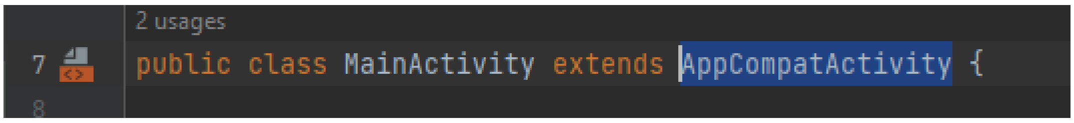
In the top right of Android Studio, ensure that the device dropdown has our device selected and then click the green **Play** button.



The app will take a moment to compile (you'll see the progress down the bottom of the IDE) and then launch!

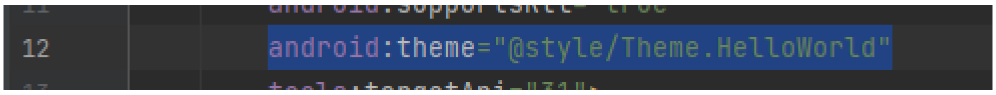


Notice that our app has a bar at the top that didn't display when we were coding our **activity_main.xml** file. This is because in our **MainActivity.java** file, we're extending the **AppCompatActivity** class.



```
2 usages
public class MainActivity extends AppCompatActivity {
```

This allows us to use some themes that exist in Android natively in order to display different bars at the top. The appearance of this bar is defined in our **AndroidManifest.xml** file under the **android:theme** attribute.



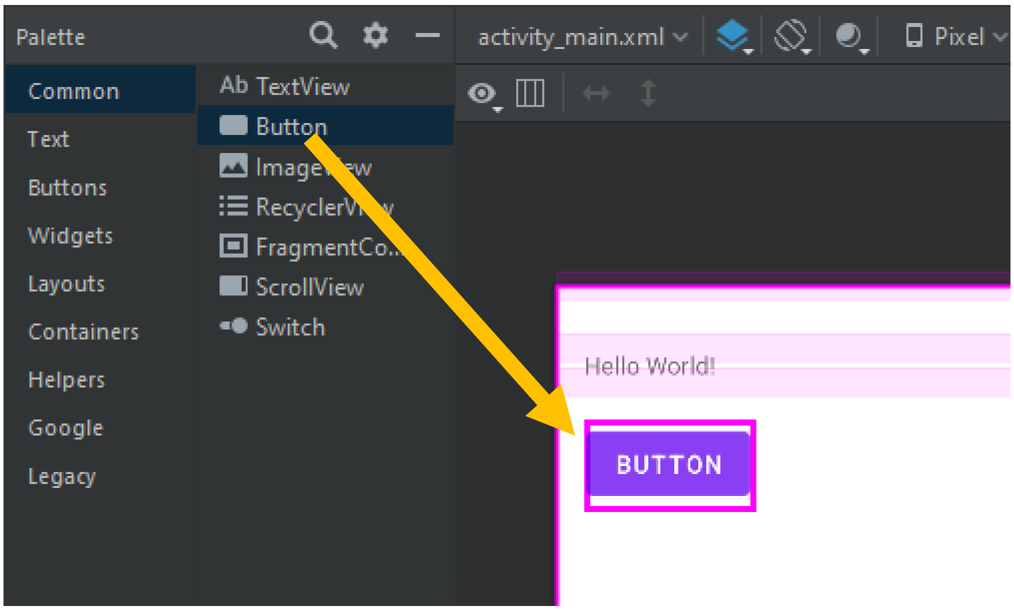
```
        android:theme="@style/Theme.HelloWorld"
```

Our app in its current state is a bit boring, so let's make it a bit more dynamic. We're going to add a button that will change our text to be something different.

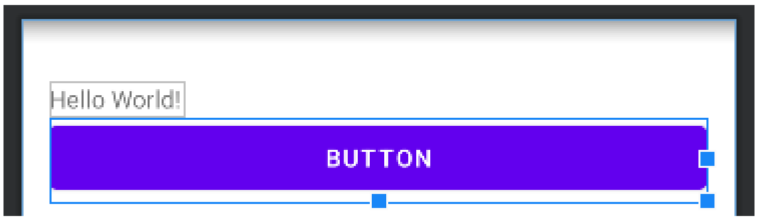
Back in **activity_main.xml**, let's give our **TextView** an ID attribute so we can find it later.

```
. . .
<TextView
    android:id="@+id/text_view"
    android:layout_width="wrap_content"
    . . .
```

Now let's add a button to our view. While we're going to mostly configure it in XML, I'd like to show you how you can add a component from the **Design** tag. Head to the Design tab, and from the left palette, drag a **Button** onto our view.



When you drop it, it should automatically place itself inside of our LinearLayout we made earlier.



Let’s head back to our Code view so we can program our button. There should be some code automatically generated for us.

```
    . . .
    <Button
        android:id="@+id/button3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button" />
    . . .
```

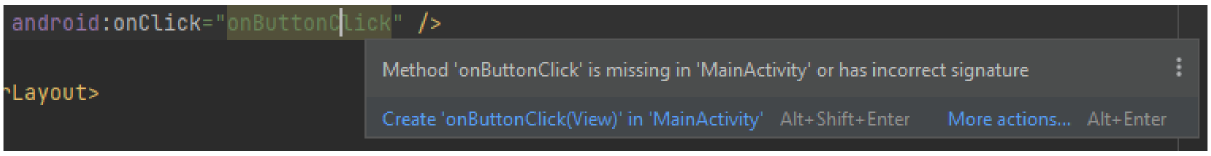
Add a new attribute called **marginTop** and set it to “20dp”, and modify the **text** to say “Click me!”.

```
        . . .
        android:layout_marginTop="20dp"
        android:text="Click me!"
        . . .
```

In order to get our button to do something, the first thing we have to do is add an **onClick** event for **onButtonClick**.

```
android:onClick="onButtonClick"
```

If we hover over our button, Android Studio will let us know we’re missing a bit of code in our **MainActivity.java** file.

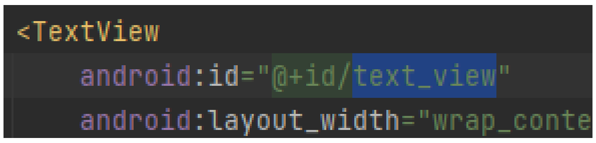


```
android:onClick="onButtonClick" />

Layout>
```

Method 'onButtonClick' is missing in 'MainActivity' or has incorrect signature

Create 'onButtonClick(View)' in 'MainActivity' Alt+Shift+Enter
More actions... Alt+Enter

Before we head over there, take note of the **TextView ID** that we created earlier. We’ll need to reference it later.



```
<TextView
    android:id="@+id/text_view"
    android:layout_width="wrap_conte
```

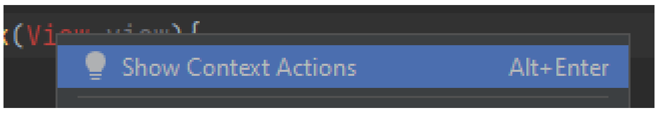
Now head to **MainActivity.java**. It should look like this:

```
package com.hayesjoshua.helloworld;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

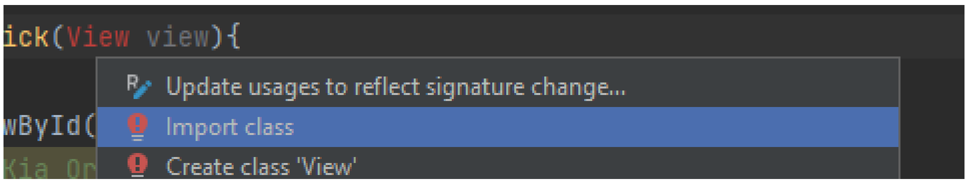
Let's create a new method that will act as our event handler for our button. Call it **onButtonClick** and pass it a **View** object.

```
public void onButtonClick(View view){

}
```

The **View** text will be red because we need to import a Java class. We can quickly do this by right-clicking the **View** text and selecting **Show Context Actions**.



Now click **Import class**.



That should have added this line to the import section.

```
import android.view.View;
```

Now back inside our new method, create a new **TextView** object.

```
TextView textView;
```

Using a method called **findViewById**, we can search for our TextView inside our **activity_main.xml** by using the ID we noted earlier, and then assign it to our textView variable we created.

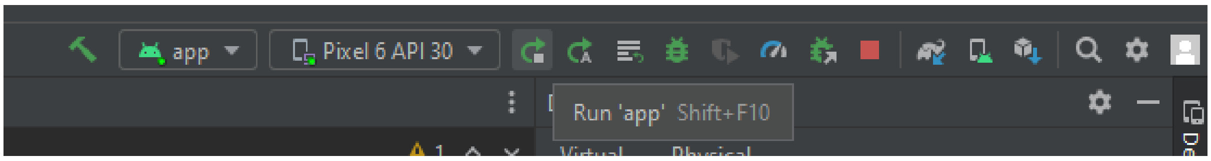
```
textView = findViewById(R.id.text_view);
```

Note the addition of the **R**. In Java, **R.java** is a special class that keeps references to all resource files so we can later access them from within our activity code. It is generated automatically when you make changes to resources, so you never have to worry about keeping it up to date.

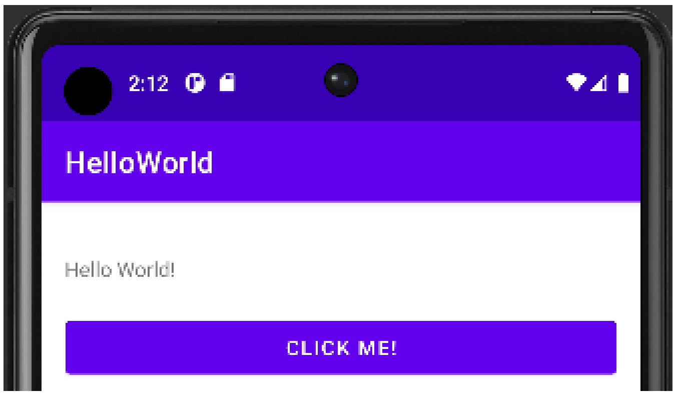
Finally, using our reference to our TextView, set our text to say something different.

```
textView.setText("Kia Ora");
```

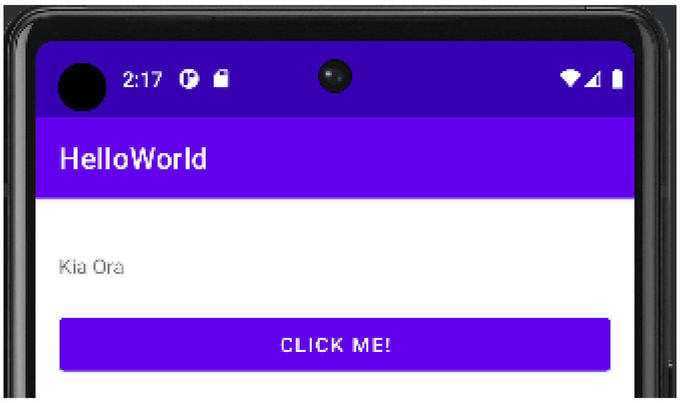
With that all completed, it's time to run our app. Click on this icon here to recompile and run our app.



The app should look a bit like this:



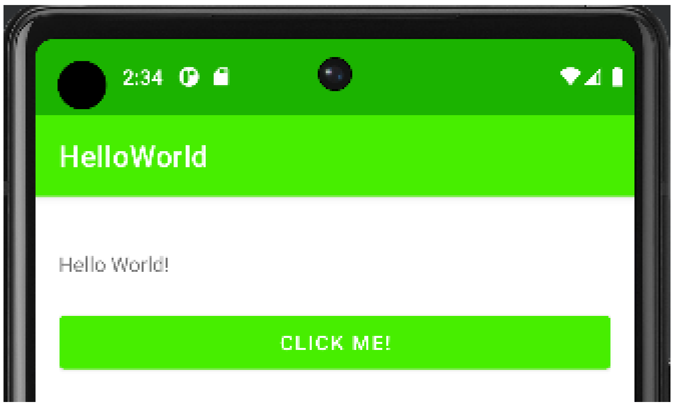
Now if we click on this button, it should change the text!



Final challenge

Before finishing with today's lab, I'd like you to have a play with the **colors.xml** file and the **themes.xml** file.

- Add some **new colour definitions** to **colors.xml**
- Add a new **Theme** to **themes.xml** by creating a new style tag with a custom name
- Use your custom colours inside of your new theme.
- Set your app's theme to your newly created theme within **AndroidManifest.xml**



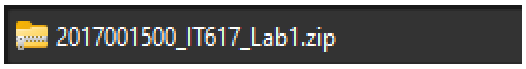
Congrats! You're finished your first Android App! Feel free to spend some time playing with different components within your **activity_main.xml**'s design view if you wish before heading to the submission section below.

Submission

When submitting your lab file, ensure that you have put the entire contents of your app's folder inside a **.zip** file. This will be in the location that you specified at the beginning of today's lab. It should be called **HelloWorld**.

Now rename your **.zip** file to

[YourStudentID]_IT617_Lab1.zip



We'll be using this naming convention for all of our labs moving forward.

Finally, upload this **.zip** to Blackboard. All labs are due on **Tuesday** at **11:59PM** the week after they are handed out for the duration of the semester.

If you have any issues getting a lab completed or have been away sick for a length of time, please ensure that you send me an email to explain the situation before the due date. If you do not email me, you may receive a 0.