Workshop Schedule

Day 1: Android Studio basics, basic app creation, interactivity

Day 2: Creating more Activities, layouts, views, Introduction to Firebase

Day 3: Firebase Integration, Android App Development Process, alternatives, expanding knowledge

Workshop resources

https://github.com/slimechips/AndroidIAP

All resources for this workshop can be found here

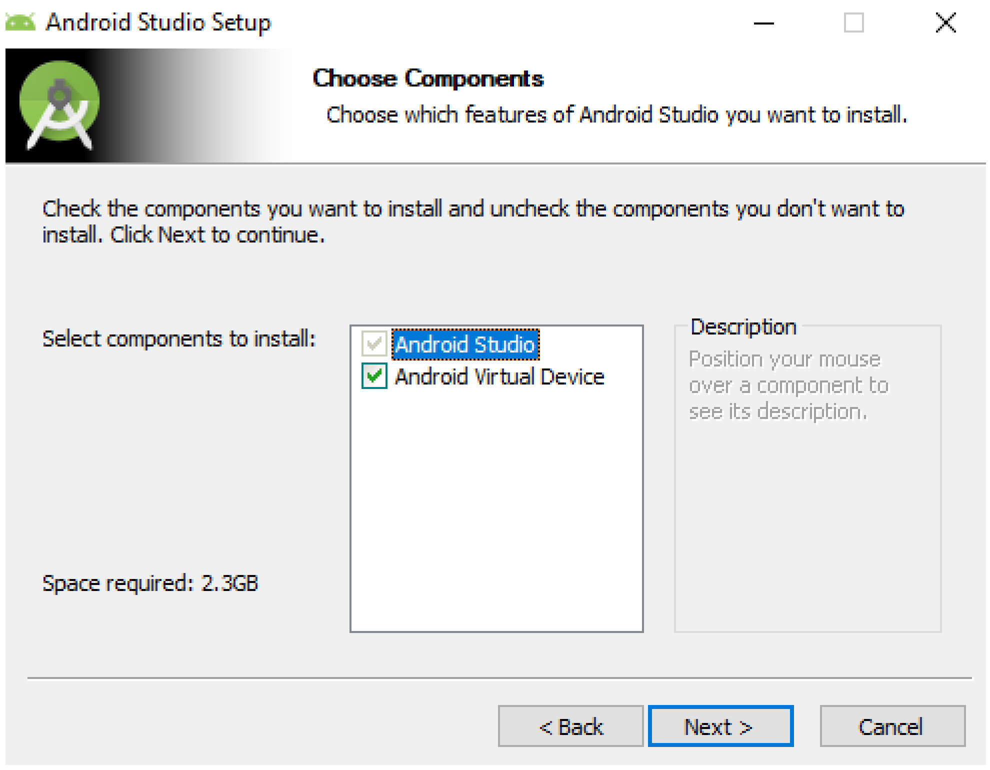
Notes:

The text in orange are meant for advanced users

1. Setting up Android Studio

Download Android Studio from https://developer.android.com/studio.

Follow the instructions to install android studio. It is available for Windows, Mac, Linux.



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

Position your mouse over a component to see its description.

Space required: 2.3GB

Be sure to select **Android Virtual Device** to install as a component.

2. What is Android App-making all about?

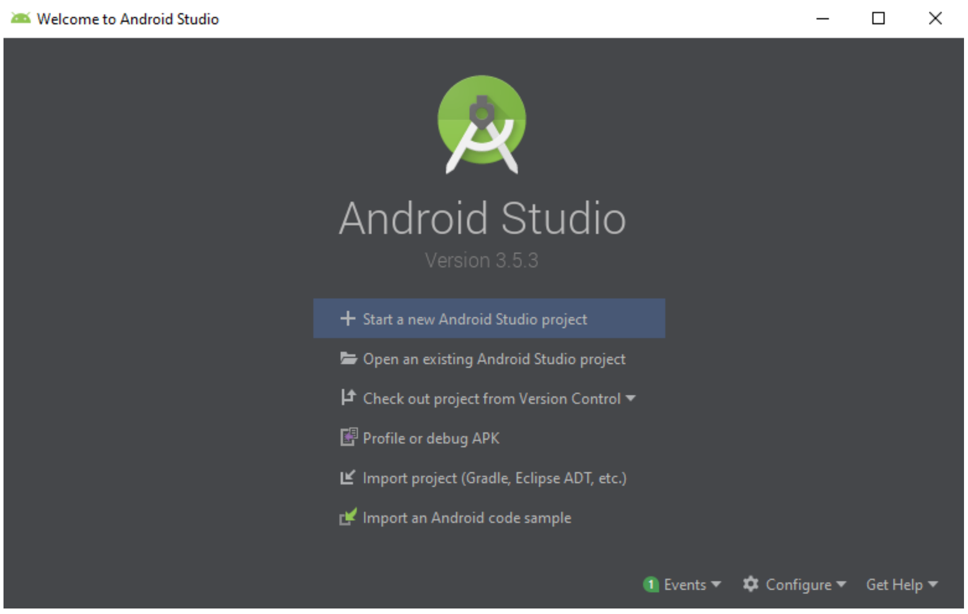
To create an Android Application, one needs to create an **APK file**, which is basically an installer for your Android App. Android Studio is a piece of software provided by Google, which can help you to create this APK file. It has the following functionalities

- A code-editing interface to use code (Java/Kotlin and XML) to create the graphical UI and functionalities of your application
- A GUI that allows beginner users to quickly create a simple UI without the need to dive into the code.

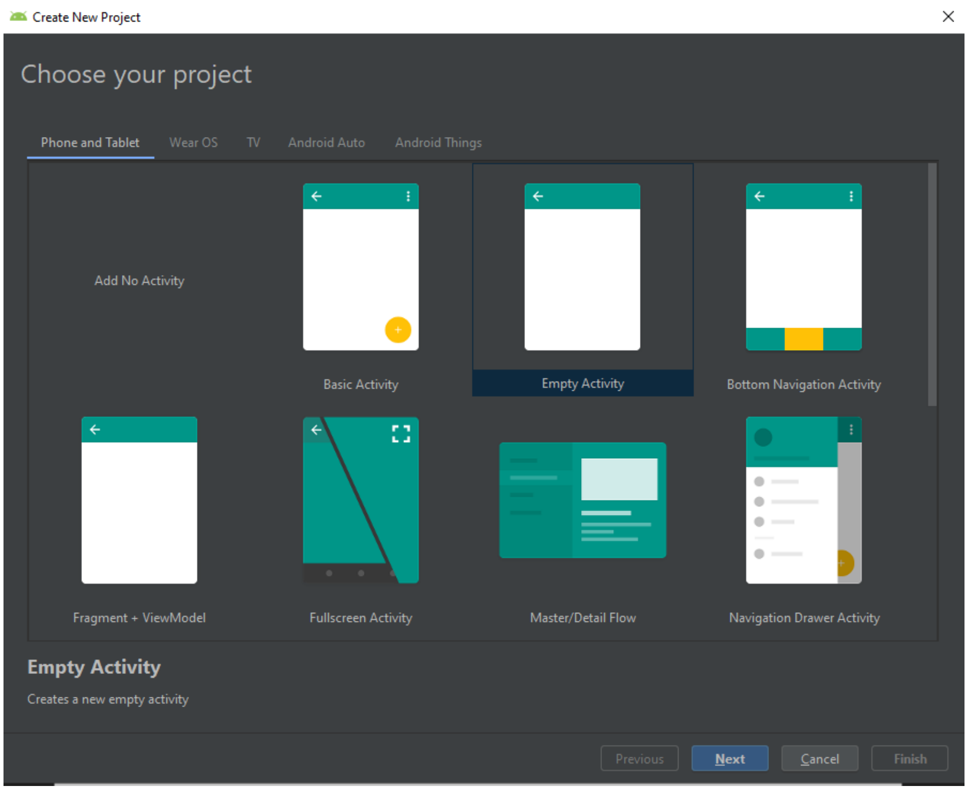
After you create an App in Android Studio, you can upload it into your Android Device or Android Emulator to run the application.

3. Creating an Empty Android Application

Open up your Android Studio. You should see something like this:

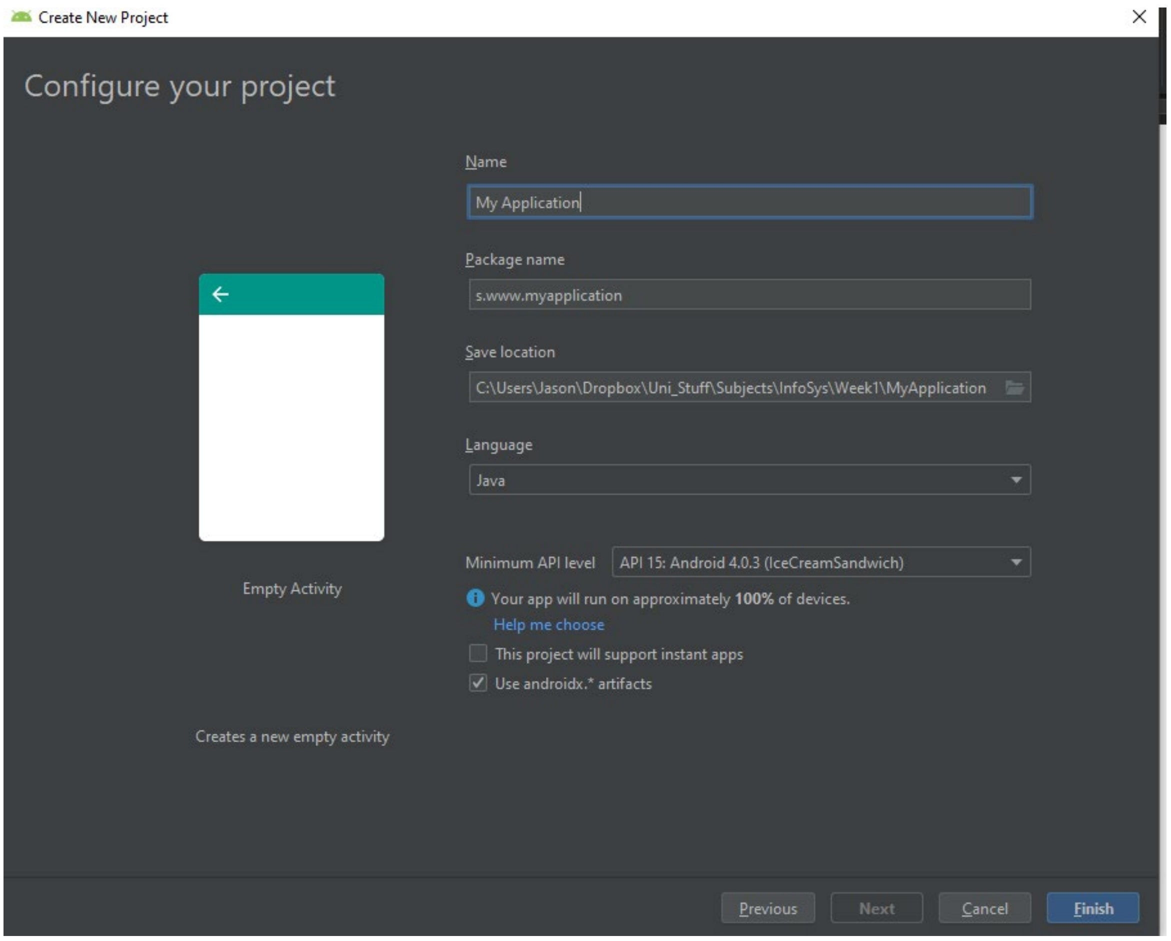

Welcome to Android Studio
Android Studio
Version 3.5.3
Start a new Android Studio project
Open an existing Android Studio project
Check out project from Version Control
Profile or debug APK
Import project (Gradle, Eclipse ADT, etc.)
Import an Android code sample
Events
Configure
Get Help

Click on **Start a new Android Studio Project**.



Select **Empty Activity**.

An **Activity** is a single logical component in your application. For our purposes, an Activity will usually be a single page in our application. We will look into greater detail regarding activities soon.



Create New Project

Configure your project

Name: My Application

Package name: s.www.myapplication

Save location: C:\Users\Jason\Dropbox\Uni_Stuff\Subjects\InfoSys\Week1\MyApplication

Language: Java

Minimum API level: API 15: Android 4.0.3 (IceCreamSandwich)

Your app will run on approximately **100%** of devices.

Help me choose

☐ This project will support instant apps

☑ Use androidx.* artifacts

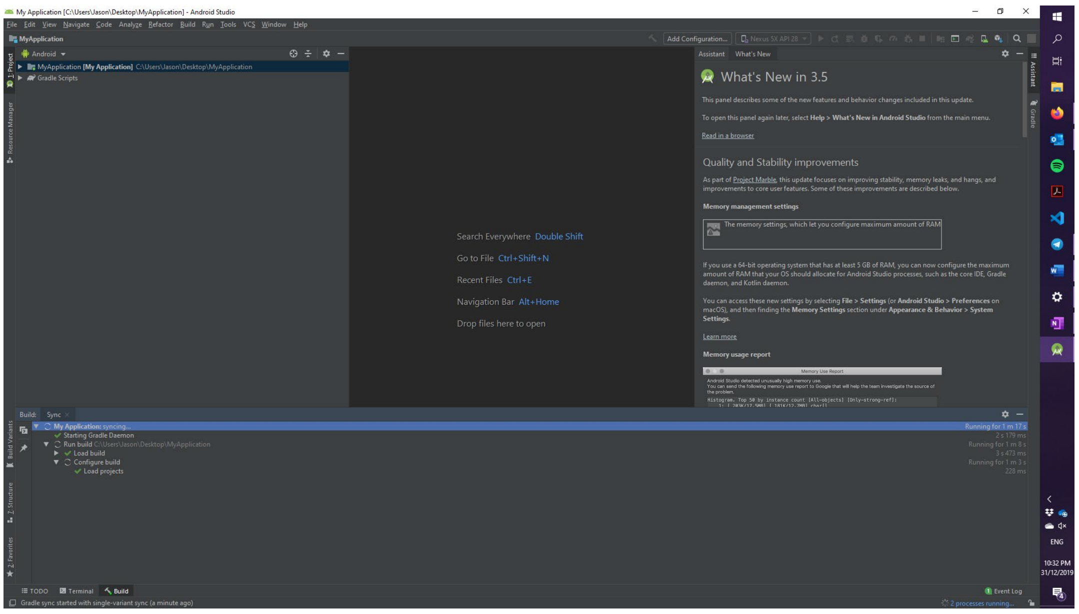
Empty Activity

Creates a new empty activity

Leave the settings as the default. You may change the save location of your project, be sure to pick a good and easy to access location! Also ensure the *Language* is **Java**.

Click **Finish**. If you are using Windows, simply click allow for all Windows Defender messages.

You should now see something like this:



My Application [C:\Users\Jason\Desktop\MyApplication] - Android Studio

MyApplication [My Application] C:\Users\Jason\Desktop\MyApplication

Gradle Scripts

Search Everywhere Double Shift

Go to File Ctrl+Shift+N

Recent Files Ctrl+E

Navigation Bar Alt+Home

Drop files here to open

What's New in 3.5

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

To open this panel again later, select **Help > What's New in Android Studio** from the main menu.

Read in a browser

Quality and Stability improvements

As part of Project Marble, this update focuses on improving stability, memory leaks, and hangs, and improvements to core user features. Some of these improvements are described below.

Memory management settings

The memory settings, which let you configure maximum amount of RAM

If you use a 64-bit operating system that has at least 5 GB of RAM, you can now configure the maximum amount of RAM that your OS should allocate for Android Studio processes, such as the core IDE, Gradle daemon, and Kotlin daemon.

You can access these new settings by selecting **File > Settings** (or **Android Studio > Preferences** on macOS), and then finding the **Memory Settings** section under **Appearance & Behavior > System Settings**.

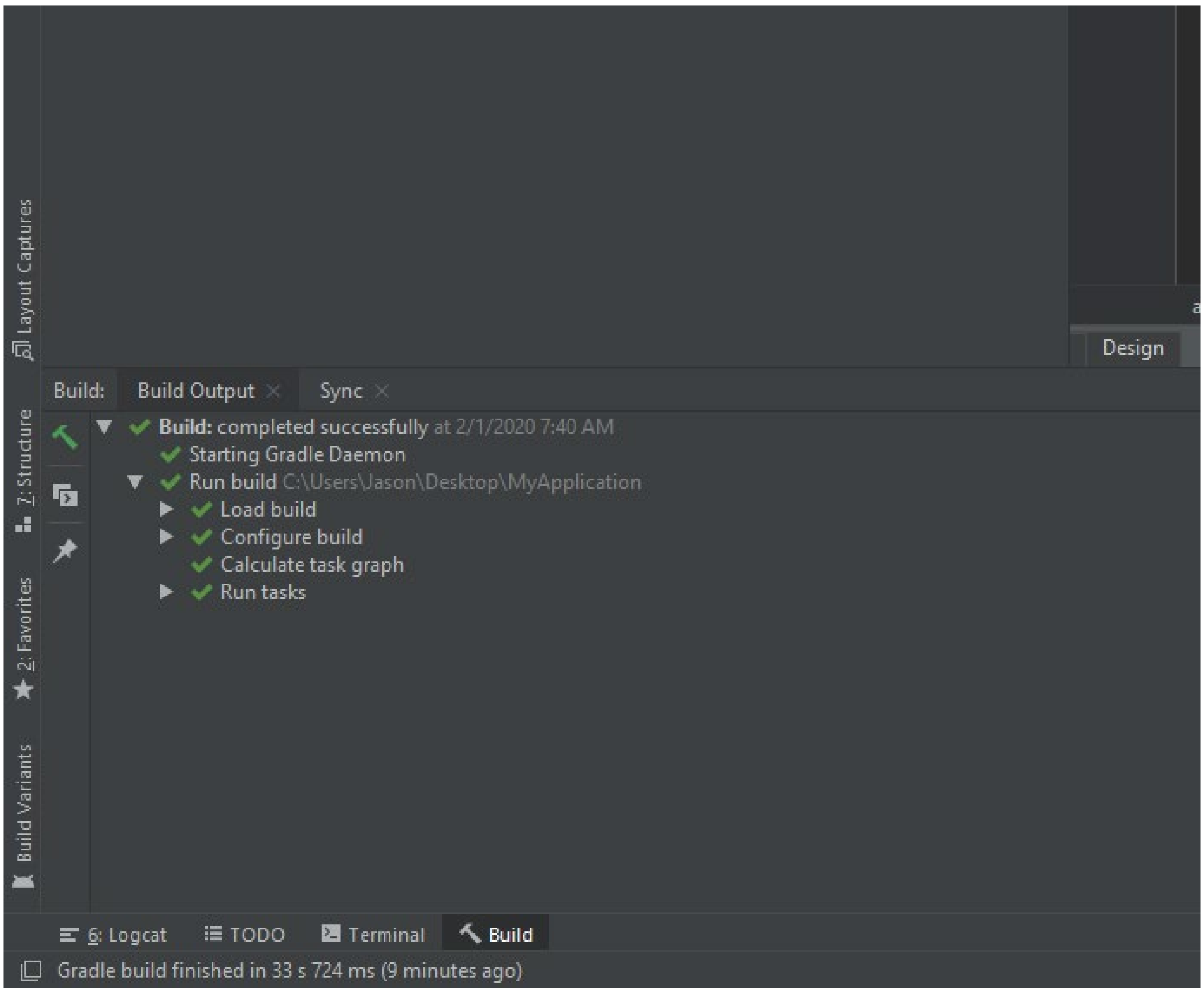
Learn more

Memory usage report

```
My Application: syncing...                          Running for 1 m 17 s
  Starting Gradle Daemon                                       2 s 179 ms
  Run build C:\Users\Jason\Desktop\MyApplication      Running for 1 m 8 s
    Load build                                                 3 s 473 ms
    Configure build                                   Running for 1 m 3 s
      Load projects                                                228 ms
```

Gradle sync started with single-variant sync (a minute ago)

Give Android Studio some time to finish the **Gradle sync.** You can look for the progress of Gradle Sync near the bottom of Android Studio.



```
▼ ✓ Build: completed successfully at 2/1/2020 7:40 AM
    ✓ Starting Gradle Daemon
  ▼ ✓ Run build C:\Users\Jason\Desktop\MyApplication
    ▶ ✓ Load build
    ▶ ✓ Configure build
      ✓ Calculate task graph
    ▶ ✓ Run tasks
```

Gradle build finished in 33 s 724 ms (9 minutes ago)

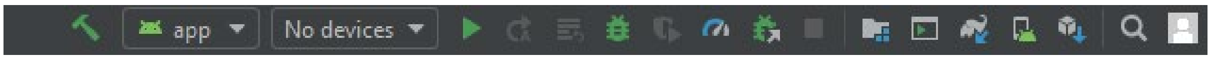
Gradle is a package manager that manages any dependencies/requirements that your Android Application might need and installs them from the internet. We will discuss more about Gradle later.

4. Building and running your first application

Right now, you actually already have an app (thought it does nothing now) that is ready to be deployed on your android device or emulator. Let us try to deploy it.

Building the App

Take a look at the **top-right** of your Android Studio. You should see the action panel that looks like this:



The main button we will be using is the green play button. The green play button

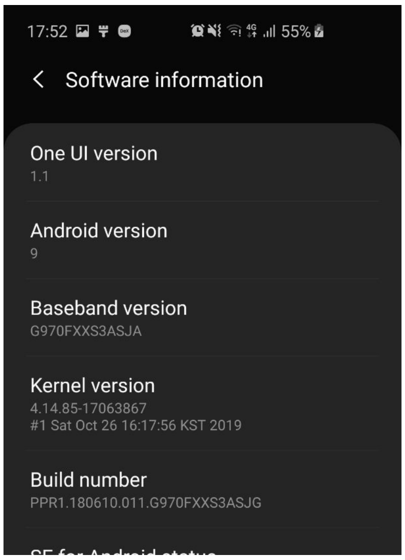
1. Compiles your application code
2. Builds the app
3. Deploys the app on your phone or emulator
4. Runs it

For now, we can see that there is a *"No devices"* dropdown box. This indicates there are currently no devices (or emulated device) for you to deploy your app on. There are two options to go from here. The first is to connect your own android phone to the computer using a USB connector. The second is to create an emulated android device and run it on your computer.

Option 1: Connecting Android Phone

Note: This tutorial was made using my Samsung Galaxy S10e, but the steps should be similar for most android devices

Go to your *Settings* app. Go to *About Phone* > *Software Information*, and tap the *Build Number* text 7 times.


17:52 55%

Software information

One UI version
1.1

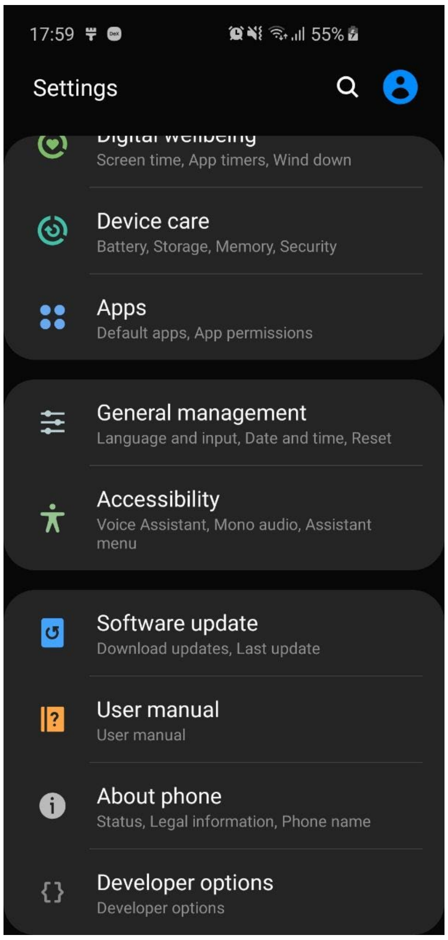
Android version
9

Baseband version
G970FXXS3ASJA

Kernel version
4.14.85-17063867
#1 Sat Oct 26 16:17:56 KST 2019

Build number
PPR1.180610.011.G970FXXS3ASJG

This should activate the *Developer options* menu in the settings app. Go back to the main settings menu and go ahead and click on ***Developer options***, and select ***USB debugging***.


17:59 55%

Settings

Digital wellbeing
Screen time, App timers, Wind down

Device care
Battery, Storage, Memory, Security

Apps
Default apps, App permissions

General management
Language and input, Date and time, Reset

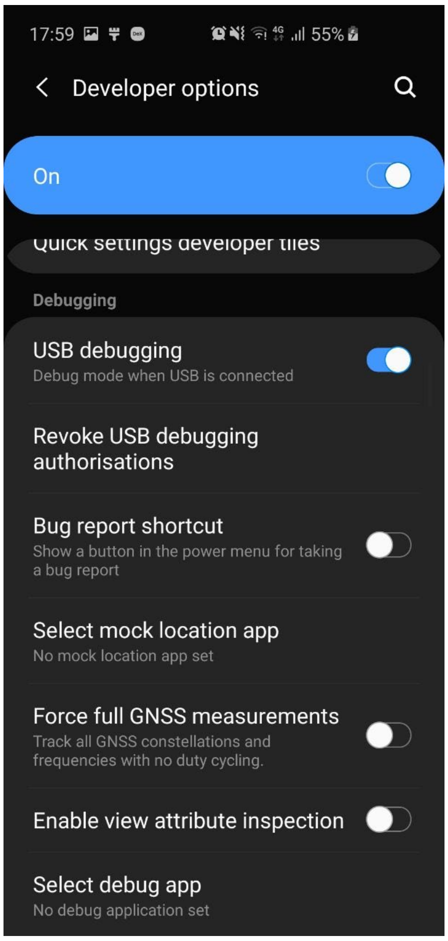
Accessibility
Voice Assistant, Mono audio, Assistant menu

Software update
Download updates, Last update

User manual
User manual

About phone
Status, Legal information, Phone name

Developer options
Developer options


17:59 55%

Developer options

On

Quick settings developer tiles

Debugging

USB debugging
Debug mode when USB is connected

Revoke USB debugging authorisations

Bug report shortcut
Show a button in the power menu for taking a bug report

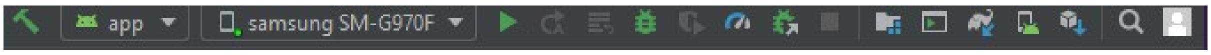
Select mock location app
No mock location app set

Force full GNSS measurements
Track all GNSS constellations and frequencies with no duty cycling.

Enable view attribute inspection

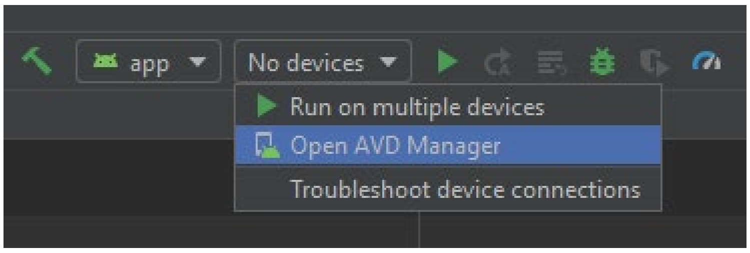
Select debug app
No debug application set

Then, connect your phone to the computer using the USB cable, and your phone should now show up as an available device.

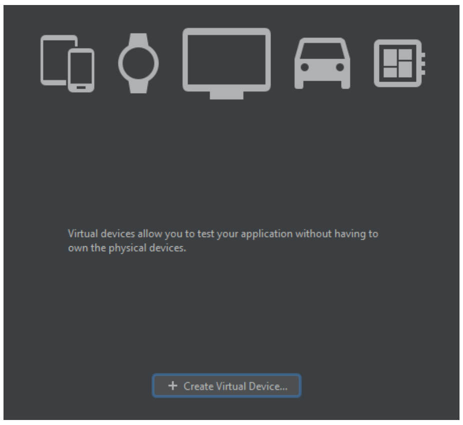


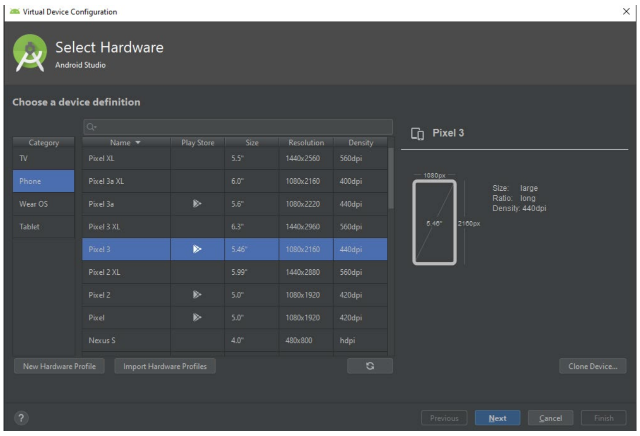
Option 2: Creating an emulated Android Device

Click on the dropdown box and select ***Open AVD Manager***.



Click on ***Create Virtual Device***.in the popup. Then choose Category: **Phone** and select a phone (recommended Pixel 3/3a.


Virtual devices allow you to test your application without having to own the physical devices.


Virtual Device Configuration

Select Hardware
Android Studio

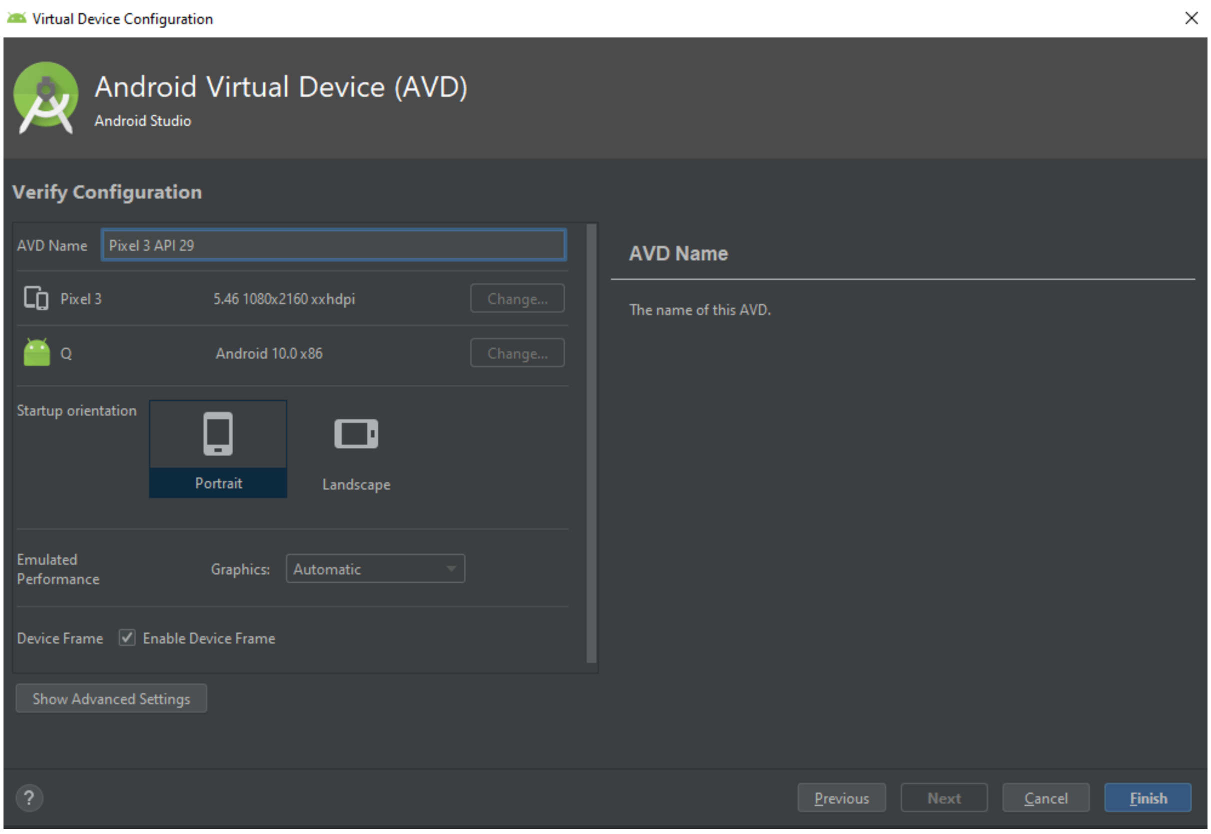
Choose a device definition

Category	Name	Play Store	Size	Resolution	Density
TV	Pixel XL		5.5"	1440x2560	560dpi
Phone	Pixel 3a XL		6.0"	1080x2160	400dpi
Wear OS	Pixel 3a	▶	5.6"	1080x2220	440dpi
Tablet	Pixel 3 XL		6.3"	1440x2960	560dpi
	Pixel 3	▶	5.46"	1080x2160	440dpi
	Pixel 2 XL		5.99"	1440x2880	560dpi
	Pixel 2	▶	5.0"	1080x1920	420dpi
	Pixel	▶	5.0"	1080x1920	420dpi
	Nexus S		4.0"	480x800	hdpi

Pixel 3

Size: large
Ratio: long
Density: 440dpi

Select a system image to download, preferably **Q.** Once that is done you can just click finish to create your Android Virtual Device.


Virtual Device Configuration

Android Virtual Device (AVD)
Android Studio

Verify Configuration

AVD Name: Pixel 3 API 29

Pixel 3 — 5.46 1080x2160 xxhdpi

Q — Android 10.0 x86

Startup orientation: [✓ Portrait] Landscape

Emulated Performance — Graphics: Automatic

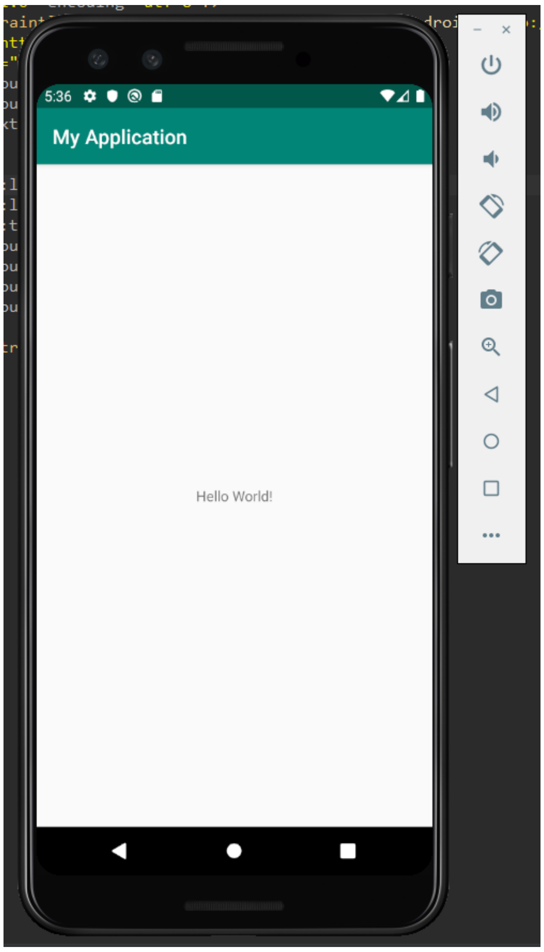
Device Frame: ✓ Enable Device Frame

AVD Name

The name of this AVD.

(After completing option 1/option2) Running the app

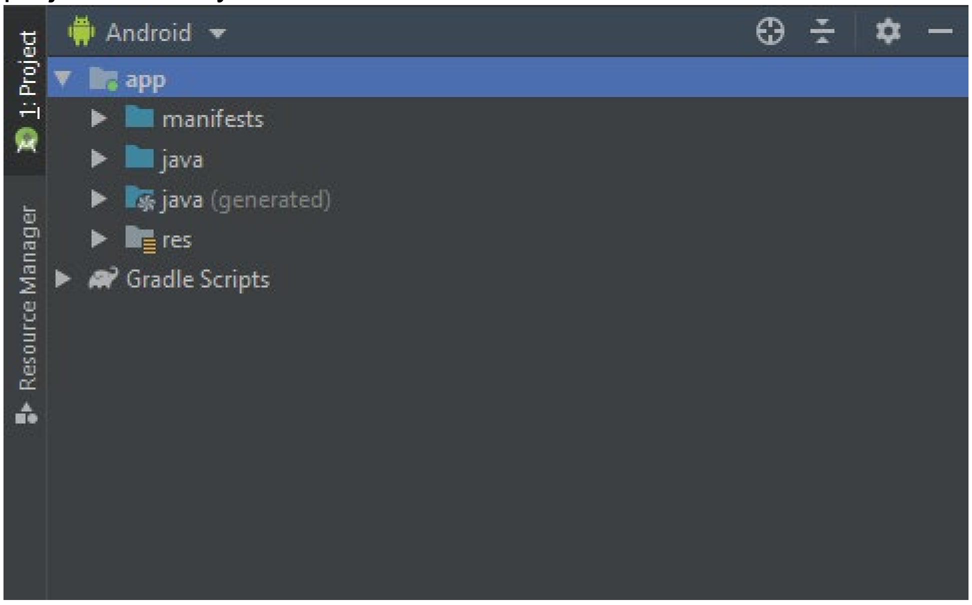
Just click on the green arrow button, and you will now see the app running on your Android device or emulator.



Great, we have created our first app (that doesn't do anything other than simply some text for now)! Let us get down to the basics and find out how it works!

5. Navigating your Android Project

On the left side of Android Studio, we can see a pane that describes your Android project directory.



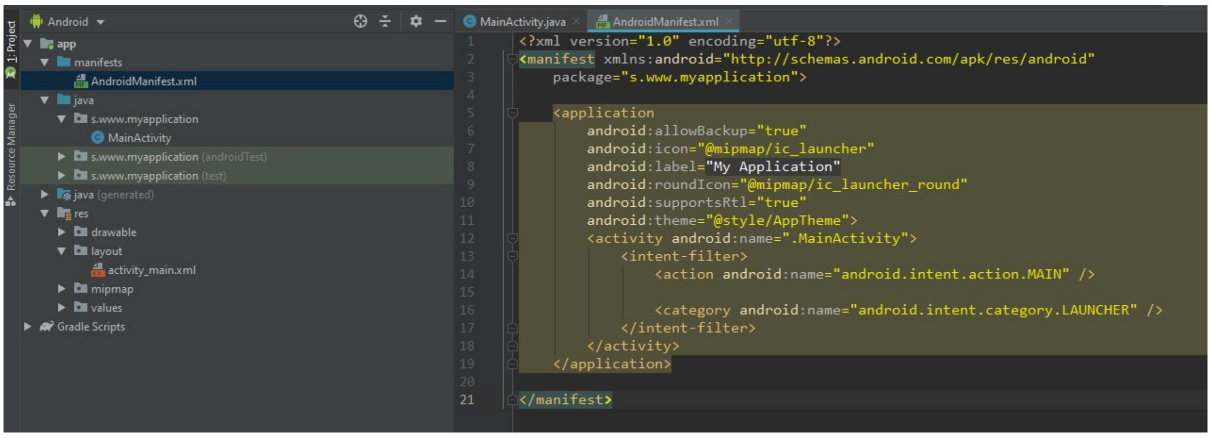
Do note that this directory view is not the actual folder directory, but a simplified directory that Android Studio is displaying for you.

If you wish the see the actual folder directory view instead (not recommended), you may click the Android button and choose *Project* instead.

The main folders we are interested in are the *manifests* folder, the *java* folder, and the *res* folder.

Manifests Folder

For now, expand the *manifests* folder, and you should see a file called *AndroidManifest.xml*. This file is the **Android Manifest**.



```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="s.www.myapplication">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="My Application"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

It is a file that

1. Describes the **structure** of the app
2. Determines the **entry point** and **main page** (or activity) of the app.
3. Describes important information like the **name of the app** and the **permissions required** for the app.

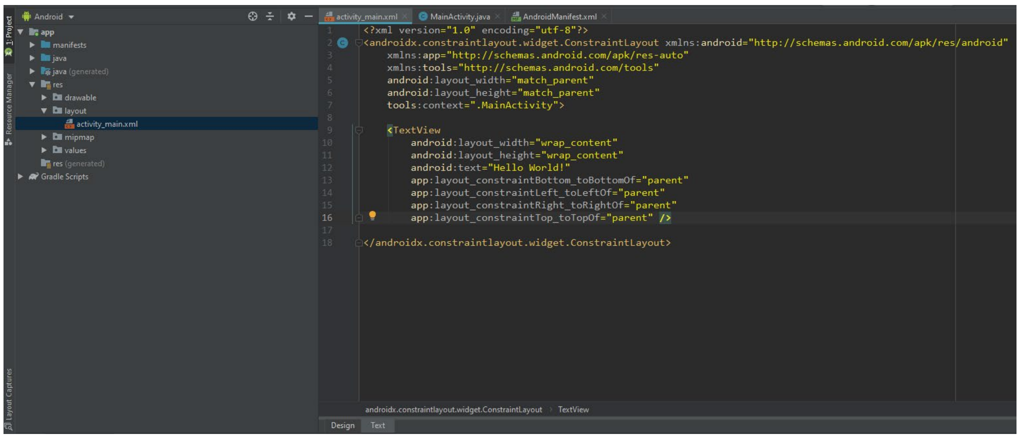
We will revisit the syntax of the Android Manifest later. For now, you just have to note that *MainActivity* is the activity being launched when the user starts the app. We will take a look at this activity soon.

Res Folder

The Res folder contains most of the files required for your android UI. Most of these files are written in XML format, a format similar to HTML. There are 3 important folders in the Res folder:

1. **Drawable:** Contains the images required by your app (e.g. your app logo). Some of these files may not be in XML format (e.g. png, jpeg)
2. **Layout:** Contains the layout file of your pages and activities. (e.g. physical locations of elements in your page)
3. **Values:** Contains global values that can be reused anywhere else in your app. (e.g. name of your app, colour scheme)

Let us take a look at the file that contained the Hello World page that we saw earlier. Expand the **layout** folder and you should see ***activity_main.xml***. Open it up and you will see something like this.



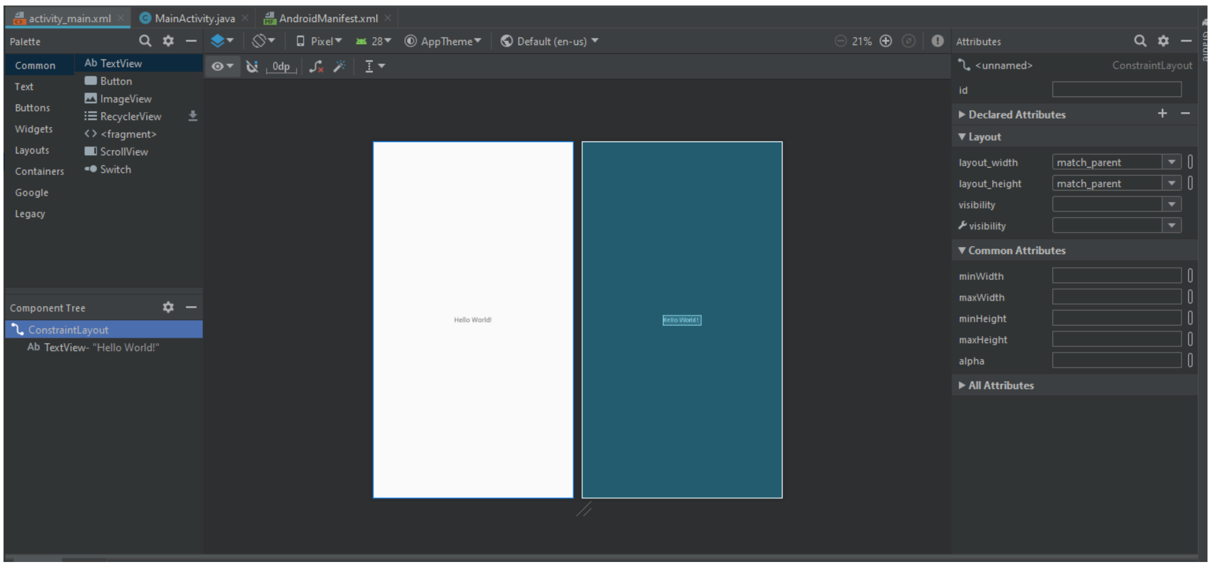
```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"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

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

Note that the line

```
android:text="Hello World!"
```

contains the text that was displayed on the app page. If you were to modify this text, it would change the text that would appear on the app page.

Note that at the bottom of the file, there two buttons side-by-side (Design and Text). If you click on the **Design** button, you will end up with a GUI showing the actual graphical layout of the XML page (which happens to be the first page of the app!).



id

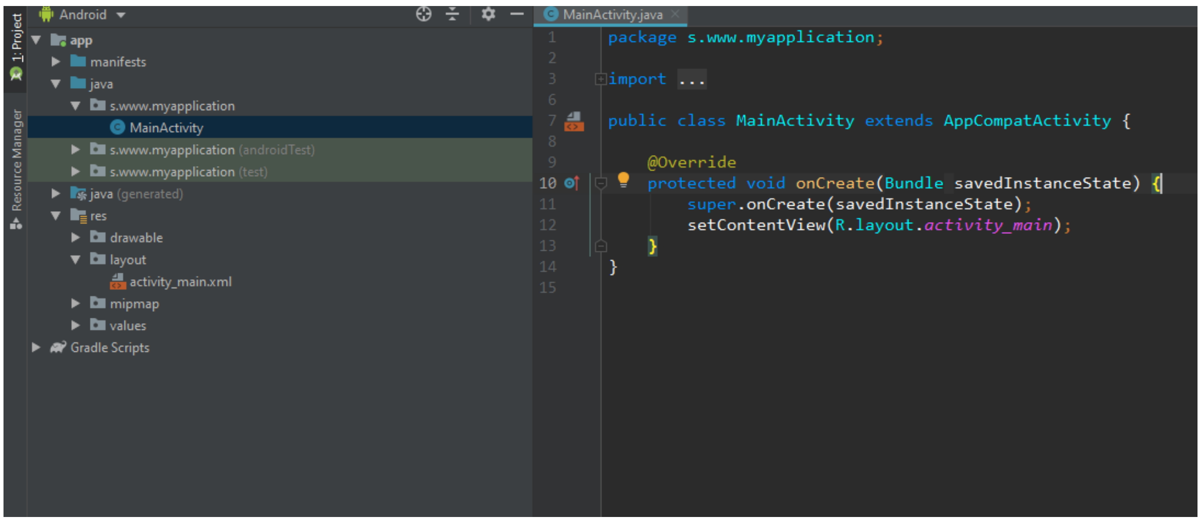
layout_width match_parent
layout_height match_parent
visibility
visibility

minWidth
maxWidth
minHeight
maxHeight
alpha

We will explore how this XML file works later.

Java Folder

For now, expand the *java* folder, where you should see another folder with a name ending with *www.myapplication*. Expand that folder and you should see a file named **MainActivity.java**. The Java folder contains all the necessary Java files you need to run your app. Double click *MainActivity.java* to open it. You should see something like this:



```java
package s.www.myapplication;

import ...

public class MainActivity extends AppCompatActivity {

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

This is the **Java code** responsible for all sorts of logical thinking and interactions between the **UI of the activity** and the user. Recall that *MainActivity* is the specified launching activity of our app, which means that the code in this file is executed when the app is launched.

We will examine this java code is more detail later. But for now, just know that the line

```
        setContentView(R.layout.activity_main);
```

simply sets the page to be displayed as ***activity_main.xml*** when MainActivity.java is run.

6. How does it all connect together?

Backtrack a bit to what we have understood in the previous section. Prior to the Application launching we take attributes of the app such as the **App name** from the **Android Manifest** (AndroidManfiest.xml). After the user clicks on the app, the Android OS now

1. Looks at the **Android Manifest** for which **Activity** (in the form a java file) should be launched. It executes the **MainActivity.java** in this case.
2. **MainActivity.java** now sets the page to be displayed as **activity_main.xml**.
3. **activity_main.xml** is the page to be displayed, and it contains elements such as the Hello World text.

7. Basic XML Syntax

XML is a markup language that is very similar to HTML. Note that XML is a declarative language that does not run any scripting language. It is not difficult to read or understand XML at all.

An XML file contains **tags**, and each **tag** encloses an **element**. A **tag** looks something like this:

```
<ElementName></ElementName>
```

An XML element can contain **attributes**, as well as **child elements**. **Attributes** help determine specific properties of an element. An element with some attributes look like this

```
<ElementName
    attribute1="somevalue"
    attribute2="anothervalue">
</ElementName>
```

Meanwhile, elements can contain **child elements** and more nested elements like this:

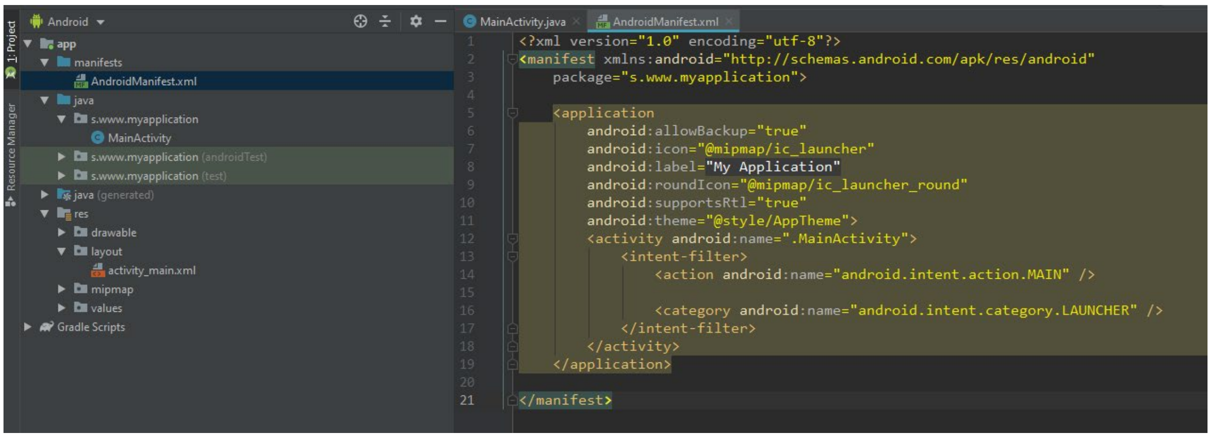
```
<ElementName>
    <ChildElementName>
        <GrandChildElementName>
        </GrandChildElementName>
    </ChildElementName>
</ElementName>
```

There is a short form for writing elements that **do not contain any child elements** (but may or may not contain attributes), they look like this:

```
<ElementName
    attribute1="somevalue"
    attribute2="anothervalue" />
```

Note that the closing `</ElementName>` tag was simply replaced with `/>`.

8. Examining the Android Manifest



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="s.www.myapplication">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="My Application"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

The positioning of the line of code

```
<category android:name="android.intent.category.LAUNCHER" />
```

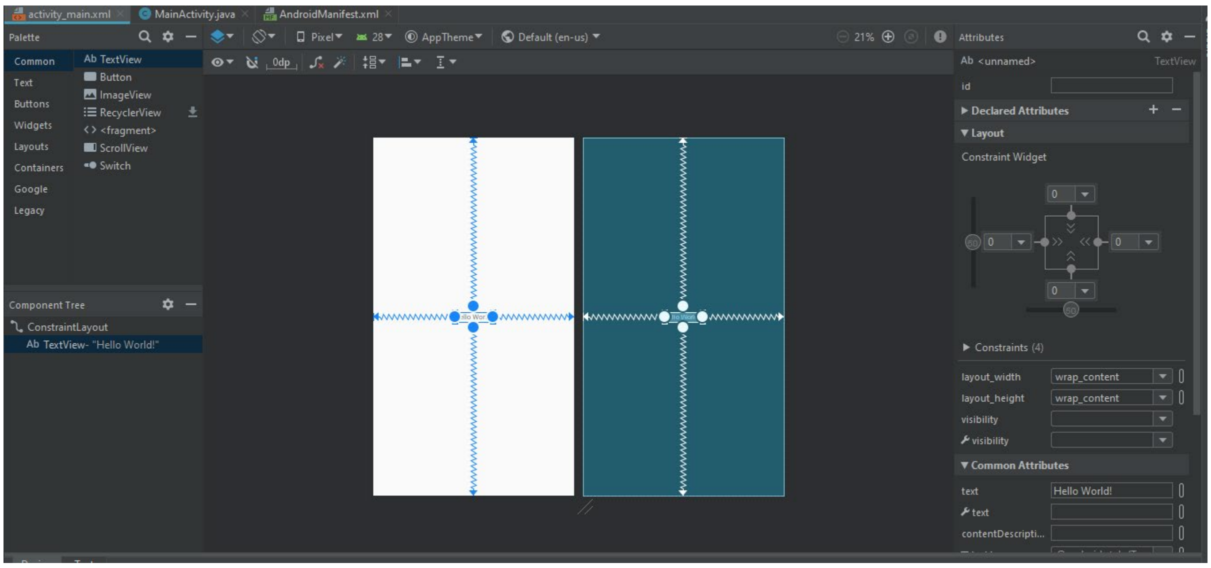
will tell us which activity will be launched when the app is launched. In this case, we see this line of code is enclosed within the XML tag

```
<activity android:name=".MainActivity">
       …
</activity>
```

This tells us the **MainActivity.java** has been chosen by the app as the activity to be launched when the app is started.

9. Examining the Layout XML File

Go back to **activity_main.xml.** To make our lives simpler, we will use the **Design** view of the XML file. Go ahead and left-click on the **Hello World** text. You will see something like this



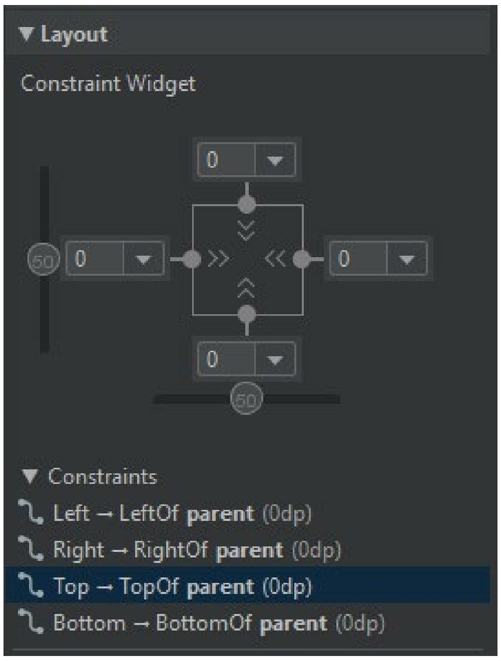
layout_width wrap_content
layout_height wrap_content
visibility
visibility
text Hello World!
text

Note that the left view is the actual view of the layout, while the right view shows a more barebones X-ray view of your layout.

On the bottom left side, you can see the **Component Tree** where you can see the elements contained within the layout file and their hierarchy. We see a single **ConstraintLayout** element containing a **TextView** element. We currently have the **TextView** element selected. A **ConstraintLayout** is simply a kind of layout that positions elements within it based on distances from edges or other objects within the layout. A **TextView** is simply an object that contains some text.

If we take a look at the right side, we can see the **Attributes** panel, which displays the attributes of the object currently selected. Looking near the bottom, we see that the **Text** attribute has been set to *Hello World!* Notice that if you try to change this attribute in the panel, the text on the screen will change as well!

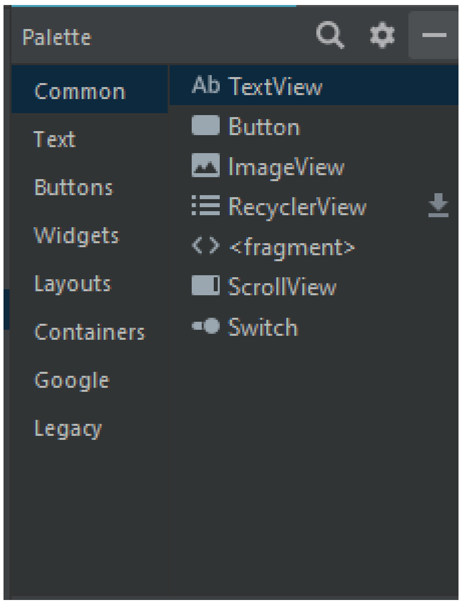
You can experiment and drag around the **TextView** as well. You will notice changes in the following part of the attributes panel.



Left → LeftOf **parent** (0dp)
Right → RightOf **parent** (0dp)
Top → TopOf **parent** (0dp)
Bottom → BottomOf **parent** (0dp)

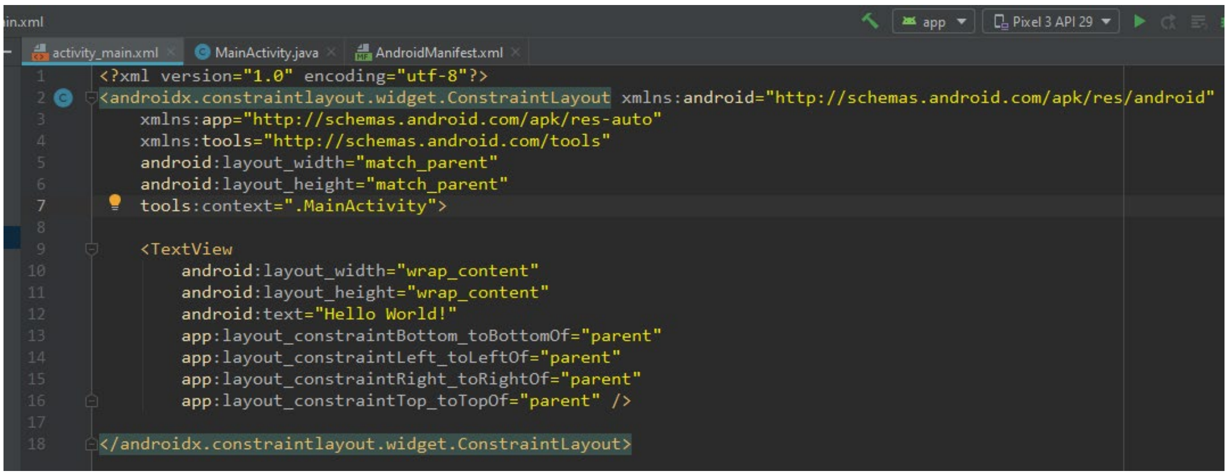
Note that this is a feature of the **ConstraintLayout.** You can drag around the **vertical slider** or the **horizontal slider** to make the TextView object higher, lower, more left or more right. You can edit the values in the **dropdown boxes** to add margins.

You may also drag in new objects into the page, by selecting and dragging an element from the top left **palette** into the layout. Try dragging in a **Button** object for example!



We will explore how we can more properly position the objects later on, but for now dragging and dropping works well enough.

You can switch back to the **Text** view of the XML file, by clicking on the **Text** button on the bottom.



```
<?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_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

We can see that there is a single XML element with the following tag

```
<androidx.constraintlayout.widget.ConstraintLayout
```

And this XML element contains a number of attributes and a single child element called a **TextView**, which also has its own attributes. These attributes are the same attributes as the ones you saw on the graphical editor. You are able to change attributes here as well (for example the attribute android:text), and you will be able to see those changes reflected when you switch back to the **Design** view.

10. Examining the Java File

The Java file contains multiple code segments, but we will only concern ourselves with the important parts.

The first line

```
package s.www.myapplication;
```

tells us which directory the Java file resides in

The second line(s)

```
import …;
```

contains statements required for using external dependencies/modules that are not found in the base Java library. This is often used to include Android dependencies.

```
public class MainActivity extends AppCompatActivity {
       …
}
```

This is simply the name of the file, and this file uses elements originally implemented in the base Android Activity (AppCompatActivity).

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

Any code contained between the curly braces **{ }** that follows this `protected void onCreate(Bundle savedInstanceState)`

is executed whenever the **Activity** starts.

I will not explain what

```
        super.onCreate(savedInstanceState);
```

does since it is beyond the scope of this workshop, but this line **must** be here.

```
        setContentView(R.layout.activity_main);
```

will set the current page to be displayed as **activity_main.xml**.

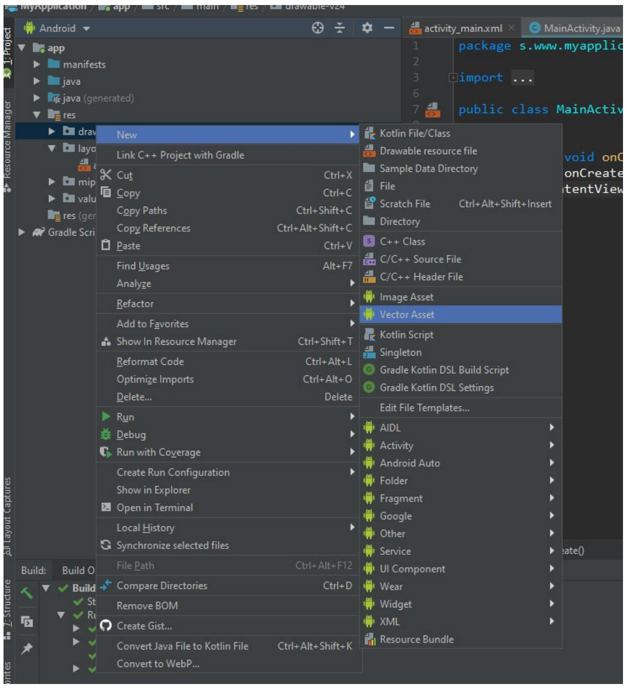
11. Adding an Image/Icon to the layout

Our page is looking a littttttle plain. Let's add an image or icon to make it more lively!

Adding an Icon

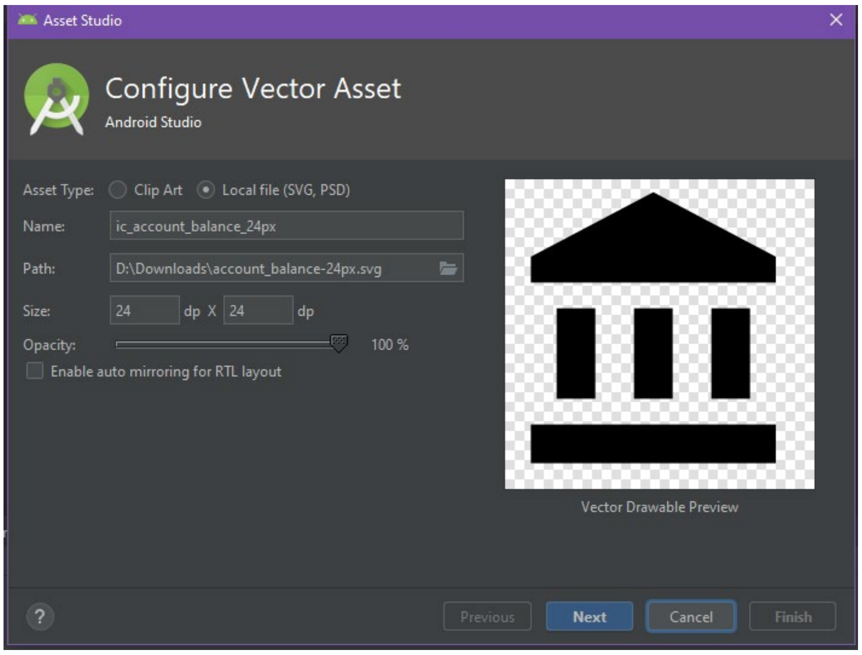
Choose any icon from https://material.io/resources/icons/?style=baseline

Download the SVG icon somewhere you can remember on your computer. Then, right click on your **drawable folder** and click **New -> Vector Asset**.



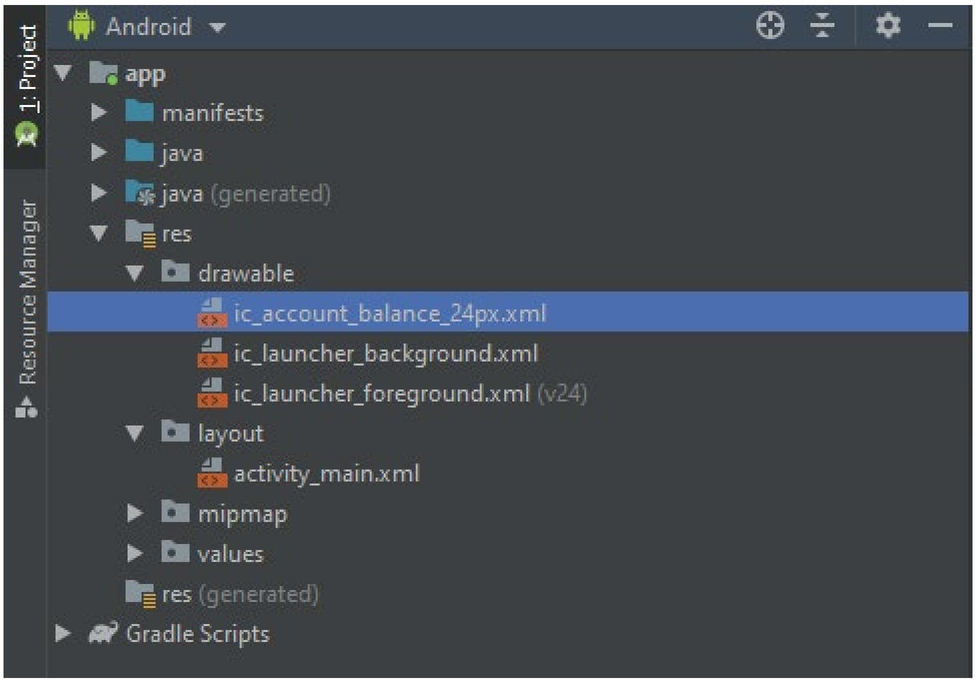
```
package s.www.myapplic
import ...
public class MainActiv
void onC
onCreate
tentView
```

You will see a window. Select **Asset Type** as **Local file**, then select the path to your SVG file. Click **Next** and **Finish.**



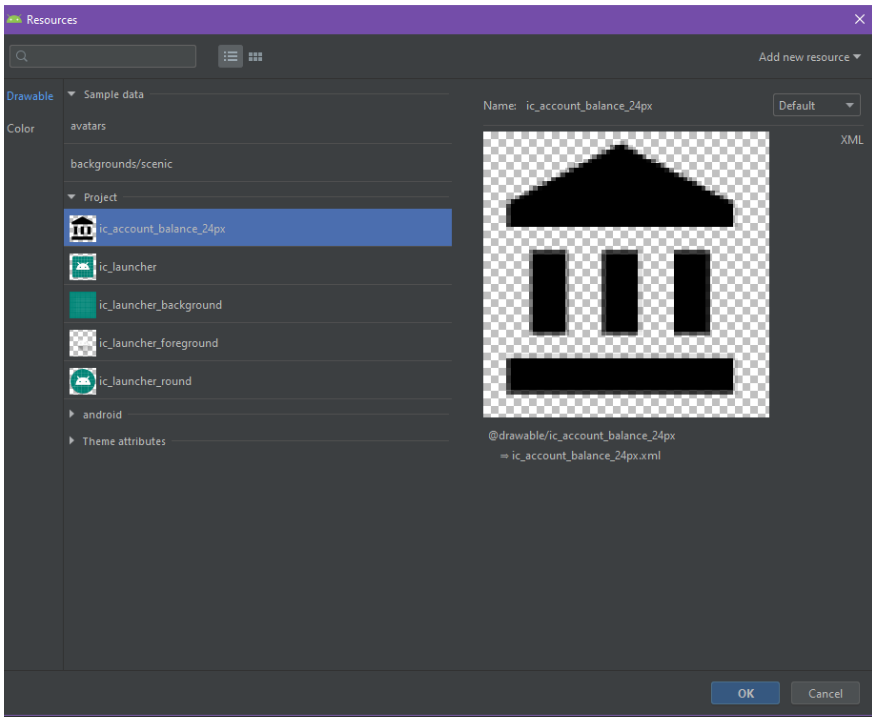
Asset Type: ( ) Clip Art (•) Local file (SVG, PSD)
Name: ic_account_balance_24px
Path: D:\Downloads\account_balance-24px.svg
Size: 24 dp X 24 dp
Opacity: 100 %
[ ] Enable auto mirroring for RTL layout

Check your drawable folder and you will now see your Vector Image now as a XML file.

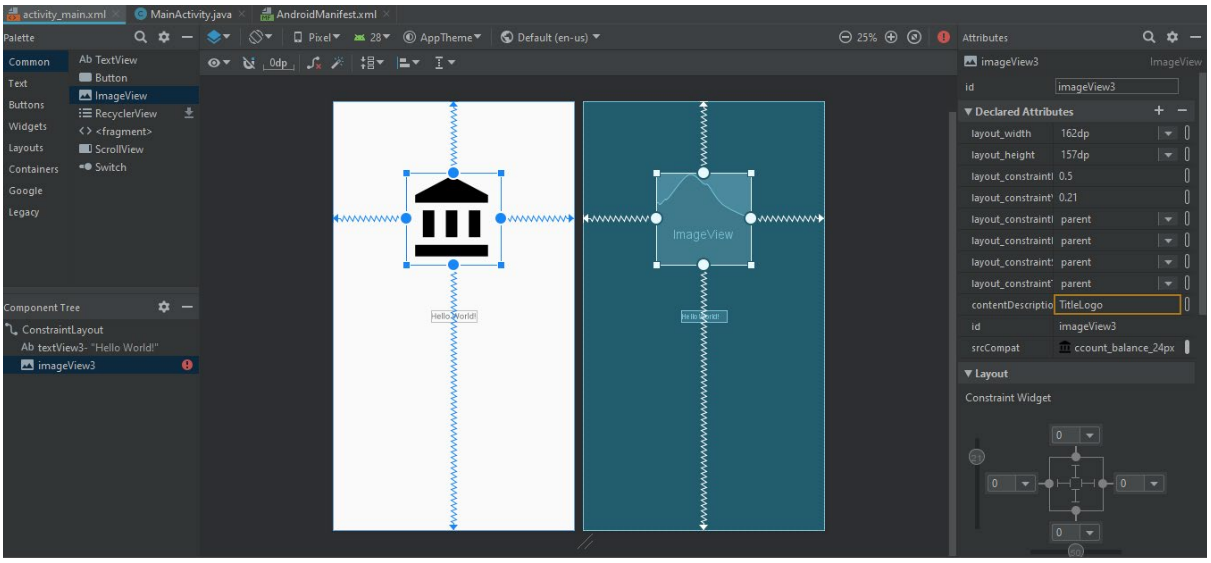


```
app
  manifests
  java
  java (generated)
  res
    drawable
      ic_account_balance_24px.xml
      ic_launcher_background.xml
      ic_launcher_foreground.xml (v24)
    layout
      activity_main.xml
    mipmap
    values
  res (generated)
Gradle Scripts
```

To use your vector asset, go back to **activity_main.xml**, drag in a new **ImageView** from the palette onto anywhere in the layout. Look for your vector asset under **Project** and click **OK**.



Voila! You now have a vector image in your main activity page! You can click and resize it, and drag the while dots to constraint the image to the ConstraintLayout!

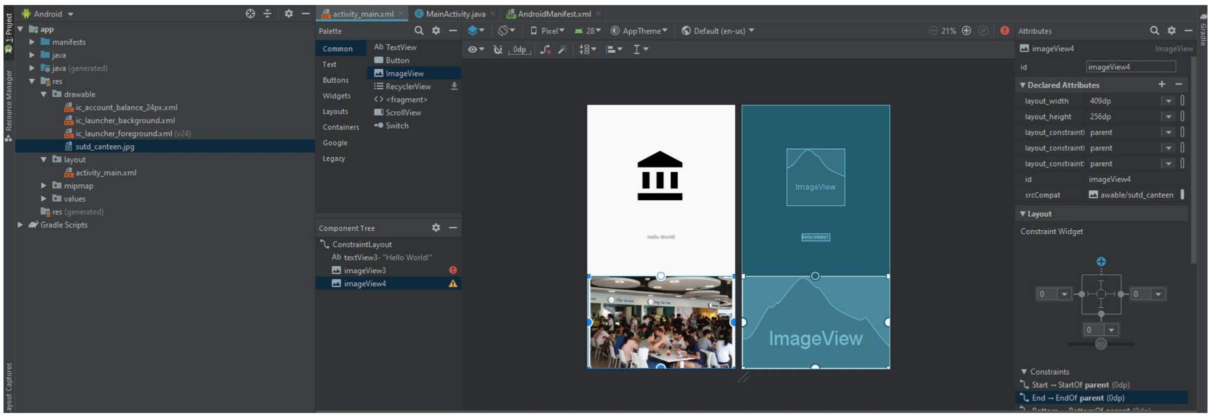


id	imageView3
layout_width	162dp
layout_height	157dp
layout_constraint	0.5
layout_constraint	0.21
layout_constraint	parent
layout_constraint	parent
layout_constraint	parent
layout_constraint	parent
contentDescriptio	TitleLogo
id	imageView3
srcCompat	ccount_balance_24px

Adding a Picture

You may choose to add something like a **PNG picture** instead as well.

To add a picture, you can simply copy the picture and paste it in the drawable folder. Then again switch back to **activity_main.xml** and drag an **ImageView** into the layout and select the PNG image. Remember to set your constraints or the image will display in the wrong place!


Attributes

imageView4 ImageView

id	imageView4

▼ Declared Attributes

layout_width	409dp
layout_height	256dp
layout_constraintI	parent
layout_constraintI	parent
layout_constraint:	parent
id	imageView4
srcCompat	awable/sutd_canteen

▼ Constraints

Start → StartOf **parent** (0dp)

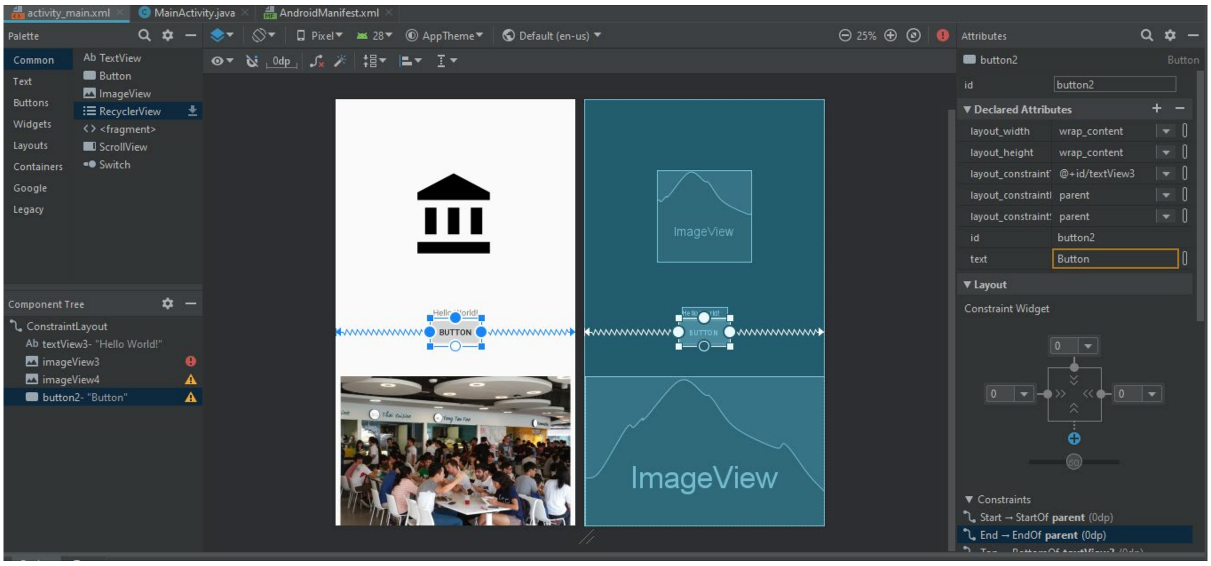
End → EndOf **parent** (0dp)

12. Interactivity, Debugging

It’s cool that we have a more interesting page now, but let’s try to include some interactivity in our app.

Adding a button

Let us add a button that displays some text when clicked. Drag a button from the **palette** into the layout, and constrain the top of the button to the bottom of the **TextView**. Constrain the left and right to the left side and right side of the **ConstraintLayout**.


Attributes

button2 Button

id	button2

▼ Declared Attributes

layout_width	wrap_content
layout_height	wrap_content
layout_constraint	@+id/textView3
layout_constraintI	parent
layout_constraint:	parent
id	button2
text	Button

Component Tree

ConstraintLayout

Ab textView3- "Hello World!"

imageView3

imageView4

button2- "Button"

▼ Constraints

Start → StartOf **parent** (0dp)

End → EndOf **parent** (0dp)

Take note of the id of the button. In this case, it is **button2**.

Getting reference to the button in the Java File

It is possible to get reference to the button from the Java file. We want to do this so we can add some interaction to the button.

First, we insert the following line underneath the setContentView():

```
        Button myButton = findViewById(R.id.button2);
```

This creates a new reference to a **Button** called **myButton**, and sets this reference to the button in the XML file with id **button2.**

Setting an onClickListener to the button

Now that we have a reference to the button in the Java file, we can add add some functionality to the button when it is clicked. Add in the following code underneath the previous code we added:

```
myButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

    }
});
```

This adds a listener to **myButton**, and when the button is clicked, anything within the parenthesis **{ }** of

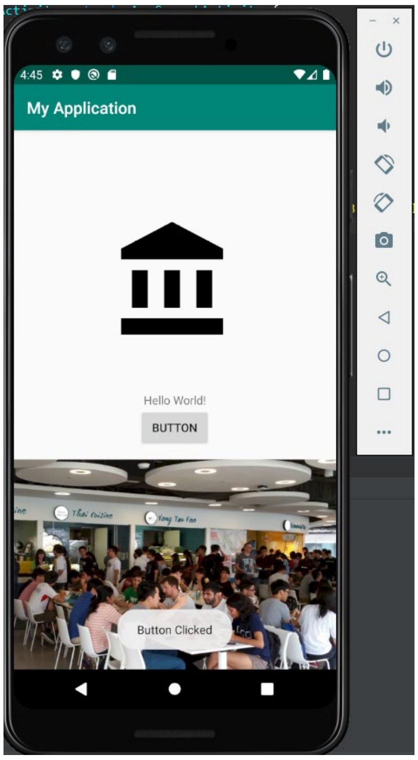
```
public void onClick(View v) {

}
```

will be executed. For now, there is nothing, so let us add some text (maybe *Button Clicked*) to be displayed when the button is clicked.

Insert the following line in between the parenthesis

```
Toast.makeText(getApplicationContext(), "Button Clicked",
Toast.LENGTH_LONG).show();
```

Run the android app and you should see something like this when the button is clicked.



Here is how the Java Code looks like after you are done:

```java
package s.www.myapplication;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

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

        Button myButton = findViewById(R.id.button2);

        myButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(getApplicationContext(), "Button
Clicked", Toast.LENGTH_LONG).show();
            }
        });
    }
}
```