OPEN SOURCE CODING (INTRODUCTION)
MODULE MANUAL 2024
(First Edition: 2023)

The Independent Institute of Education (Pty) Ltd is registered with the Department of Higher Education and Training as a private higher education institution under the Higher Education Act, 1997 (reg. no. 2007/HE07/002). Company registration number: 1987/004754/07.

Table of Contents

Introduction

Welcome to Open Source Coding (Introduction). In this module, we will be focussing on developing native apps for the Android Operating System (OS). We will make use of the Kotlin programming language.

In 2022, the Android OS has a market share of 70.97% of mobile devices (G., 2022). This means that the apps that we learn to develop here, will be able to run on most mobile devices out there right now.

In your previous programming modules, you have already learned object-oriented programming in Java or C#. Although the syntax of Kotlin is slightly different, you will find the concepts quite familiar.

Throughout this module, you will create several apps to master all the basic skills needed to build an Android app. It is important to get hands-on experience in any programming module, so it is essential that you complete all the activities provided on Learn.

We hope you will enjoy the module and take the opportunity to use the knowledge and experience gained in both future modules, and in your career.

Learning Unit 1: Introduction to Android Development	
Learning Objectives: • Identify the tools used in Android development. • Provide an overview of the history of the Android Operating System. • Explain the purpose of Git. • Contrast Git and GitHub. • Create a working copy of a repository hosted on GitHub.	**My notes**
Material used for this learning unit: • GitHub repository: LearningUnit1	
How to prepare for this learning unit: • Install Android Studio.	

1 Introduction

In this module, we will learn how to create apps for Android phones. We will start by creating a user interface that just uses hard coded data, and later in the module we will also look at how to read data from and write data to a database. In the last learning unit, we will do something different – creating a game from scratch using the APIs provided by Android.

This introductory learning unit provides an overview the Android Operating System (OS) and its features. We will also discuss Android software development and all the parts of the system that are involved in creating and building a native Android app.

And finally, this learning unit will introduce the sample source code from the GitHub repository for this module.

Note: All the sample source code in the GitHub repository, and in this Module Manual, is written in **Kotlin**.

2 Android Eco System

2.1 Android Operating System

The Android OS, created by Google, has been around for many years now – the first official public release happened in 2008. The next year Android 1.5 was released, and it was called Cupcake. That started the naming convention where releases were named after desserts, like Nougat, Oreo, and Pie. Android 10, released in 2019, was the first release to break the dessert-themed tradition. (Raphael, 2020) By August 2022, Android 13 Beta 4 was available for download. (Android Open Source Project, 2022)

Each new version of the operating system introduced new features. And these are not only features of the operating system apps that the user sees, but also features that app developers can make use of.

The source code for Android is open source. Most of the code is licensed under the Apache License, Version 2.0. There are exceptions though, like some kernel code that is licensed under GPLv2. (Android Open Source Project, 2020b) When manufacturers release devices running the Android OS, proprietary software is usually included too. (Chen, 2020)

The licensing sounds like it is getting complicated. But, for our purposes as app developers, working with Android means working with open-source software. So, for example we can view the code that displays images on the screen. And we can make use of tools that are completely free to develop our apps.

2.2 Building and Running an Android App

We will be developing our Android apps on a computer running a desktop operating system like Microsoft Windows. And yet we are creating software that is not meant to run on Windows at all. So, there are more pieces of software involved with developing for Android than there is for the desktop apps that you have developed before. Figure 1 shows all the important parts of the system.

The first important component is Android Studio. It is an Integrated Development Environment (IDE) that allows us to do everything we need to develop our apps. Android Studio supports both Java and Kotlin development.

Installed together with Android Studio is the Android Software Development Kit (SDK). The SDK is a bundle of the tools that are needed for Android development, such as the emulator. Although you could use the SDK tools directly, Android Studio actually integrates all of these things into a single user-friendly app. So, there is little need for anybody to use those tools directly anymore. (Sinicki, 2019)

The Gradle build system is used to manage the libraries that are used to build software, and to do the building and running of our apps.

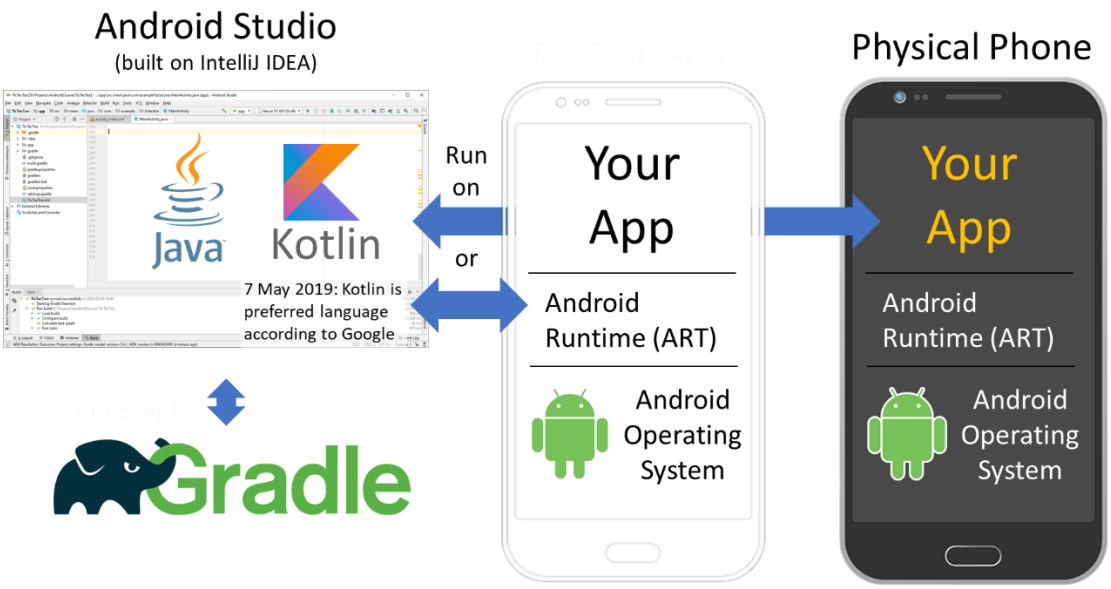


Figure 1. Android Eco System (created using images from (nevoski, n.d.), (Google, 2014), (JetBrains, 2020), (HowToDoInjava.com, n.d.), (Anon., 2018))

When you run an Android app during development, you need something running the Android OS to run it on. The first possibility is running the app directly on your phone, using USB debugging. Read more about how to do that in (Android Open Source Project, 2019). Of course, that means that you need an Android phone. And it also means that you are limited to running the version of the operating system that is currently installed on your phone.

Another way to run an app, is to make use of the emulator (also called Android Virtual Device). When using the emulator, an instance of the Android OS of your choice is run in a virtual device on your computer. The benefit of doing that is that you could create a virtual device that emulates hardware that you don't own, running any Android OS version. For phones and tablets, you can choose the screen resolution supported by the emulator. So, it allows for much greater flexibility when testing your software.

The emulator does have limitations though, for example not being able to emulate Bluetooth and device-attached headphones. (Android Open Source Project, 2020c) The emulator also doesn't support a step counter.

Whether you run the app on the emulator or on a phone, the Android Runtime (ART) is the software that runs the app on the target OS. It fulfils similar functions to the Java Virtual Machine (JVM) when you run a Java app on your computer. (Sinhal, 2017) Earlier versions of the Android OS had a different runtime called Dalvik. (Android Open Source Project, 2020d)

3 Git and GitHub

Git is a **distributed source control program** used to keep track of changes and updates to software during development. This allows software developers to revert to previous versions of software they have developed if any part of the code is lost, or if bugs are introduced into the code.

Git also allows developers to work together in teams and share code, while keeping track of the work done by each team member.

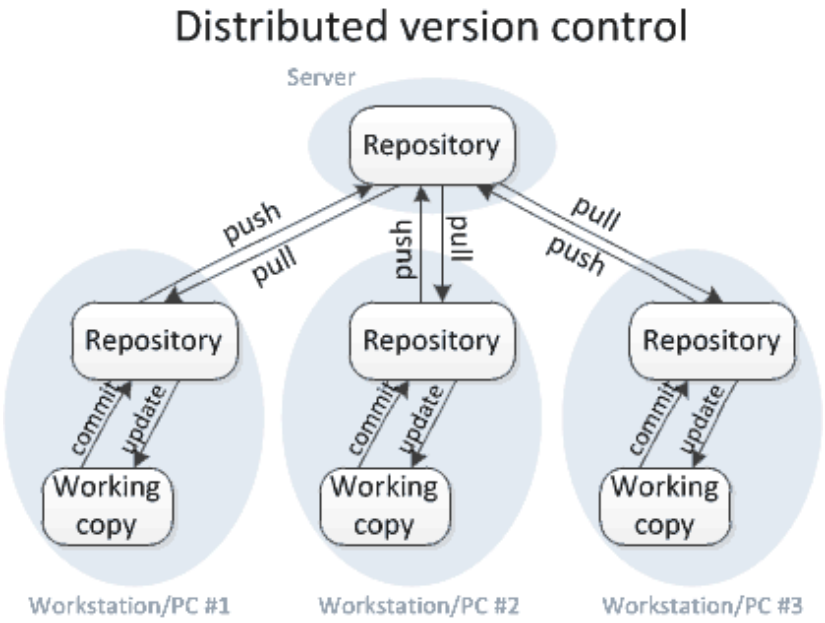


Figure 2. Distributed Version Control (Ernst, 2018)

Figure 2 shows how a distributed version control system such as Git works. Each developer has a clone of the repository on their computer, and changes are pushed to and pulled from a central repository that needs to be stored somewhere on a server.

GitHub (https://github.com/) is a service owned by Microsoft that provides **Git repositories** that are **hosted online**. The free tier of GitHub provides more than enough features for us to use.

GitHub is not the only option for hosted Git repositories. Other services include Microsoft Azure DevOps Services (https://azure.microsoft.com/en-us/services/devops/) and Bitbucket (https://bitbucket.org/product), both of which also have free tiers. But for our purposes in this module, we will be making use of GitHub.

3.1 Getting the Sample Code

The sample code for this module can be found in the following GitHub repository: https://github.com/iie-opsc/opsc7311kotlin [Accessed 17 November 2022].

As with most things in programming, there are multiple ways to clone the repository to your local computer.

3.1.1 Using GitHub Desktop

To clone the code using GitHub Desktop:

1. Download the free **GitHub Desktop** app from https://desktop.github.com/ [Accessed 17 November 2022].

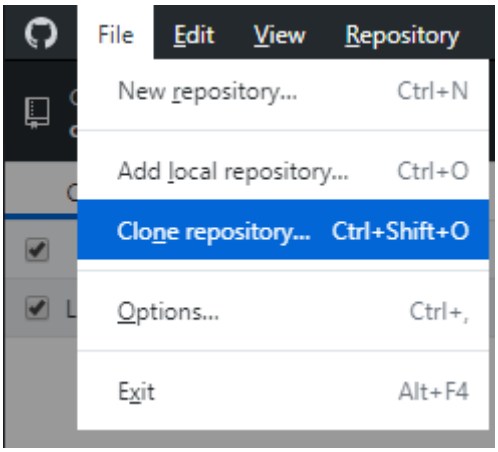


Figure 3. Clone Repository

2. In the **GitHub Desktop** app, click the **File** menu and then click **Clone Repository**.

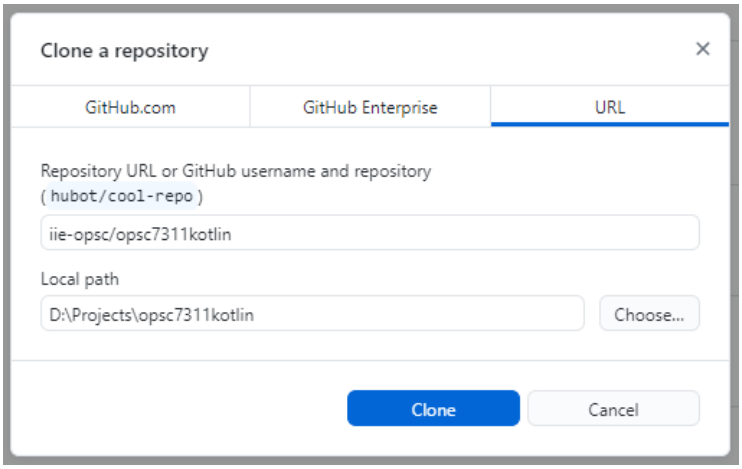


Figure 4. Repository Details

3. Click the URL tab.
4. Enter the following URL: iie-opsc/opsc7311kotlin
5. Choose a **Local path** (folder) to clone the repository to.

6. Click **Clone**.

Now you can open the various projects in Android Studio.

3.1.2 Using Command Line Tools

To clone the repository with the Git command line tools:

1. On the command line, go to the folder where you want to clone the source code.
2. Execute the following command:

```
git clone https://github.com/iie-opsc/opsc7311kotlin
```

3.1.3 Using Android Studio

To clone the repository using Android Studio:

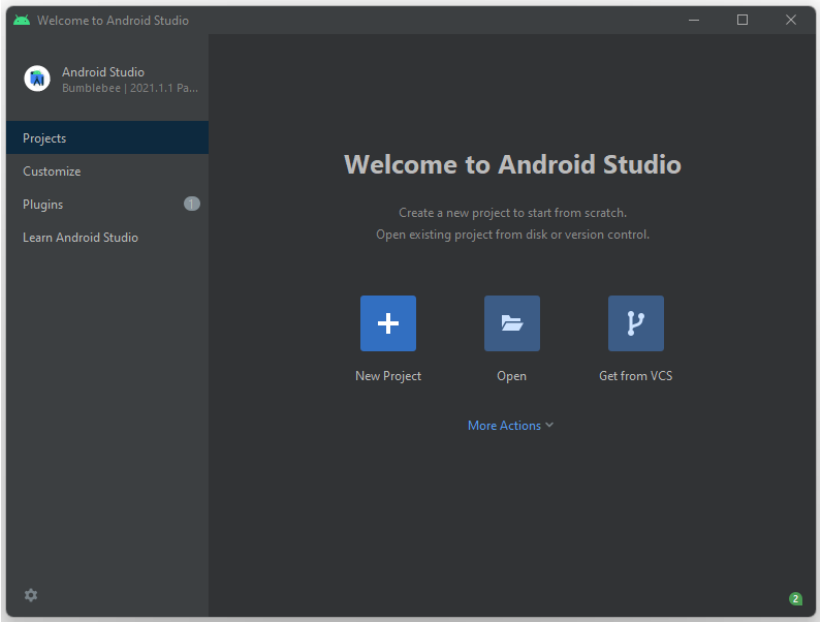


Figure 5. Android Studio Welcome Screen

1. On the **Welcome to Android Studio** screen, click **Get from VCS**.

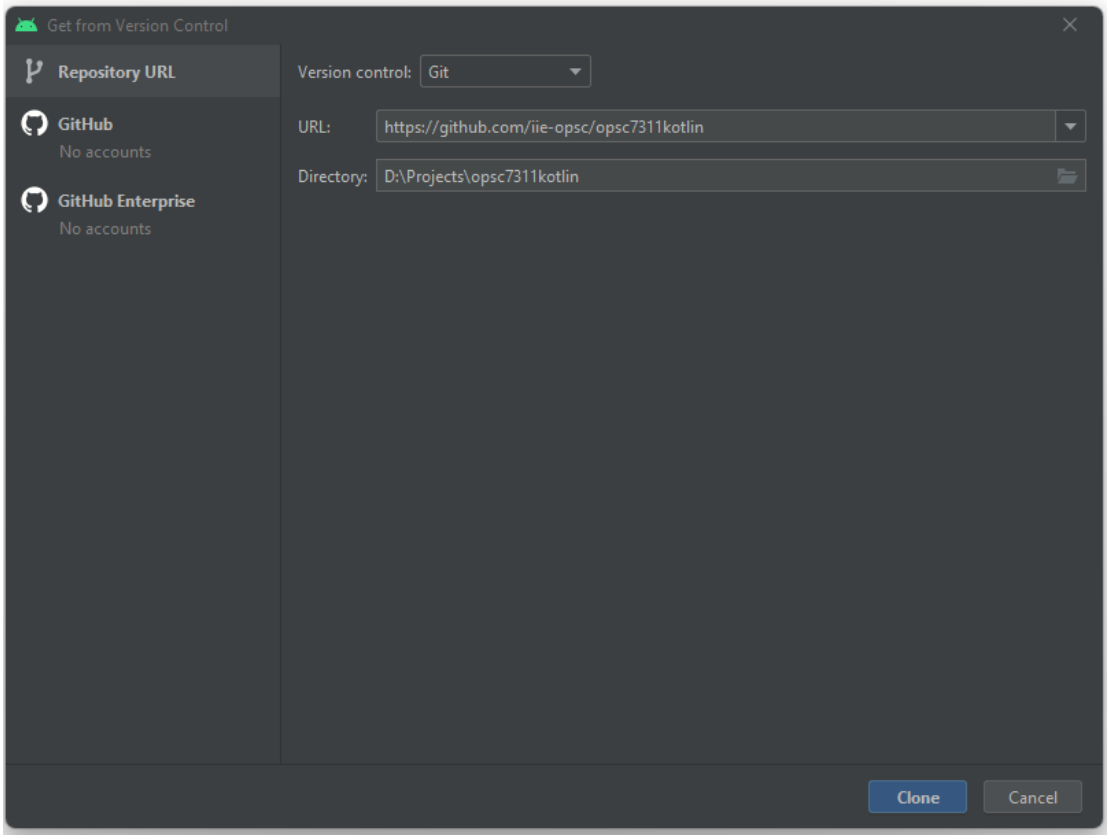


Figure 6. Enter the URL

2. Enter the URL for the repository: https://github.com/iie-opsc/opsc7311kotlin
3. Choose a **Directory** to store the code in.
4. Click **Clone**.

Now you can open any of the projects in the local working folder in Android Studio.

3.2 Our First Sample Project

Open the project in the `LearningUnit1\ WelcomeToOpenSource` folder in Android Studio.

Open the file `res\layout\activity_main.xml`. In the **Design** view, the main activity should appear as shown in Figure 7. If you see this, then you have successfully opened the first sample program for this module. And that means that you are ready to jump into creating your first app in Learning Unit 2!

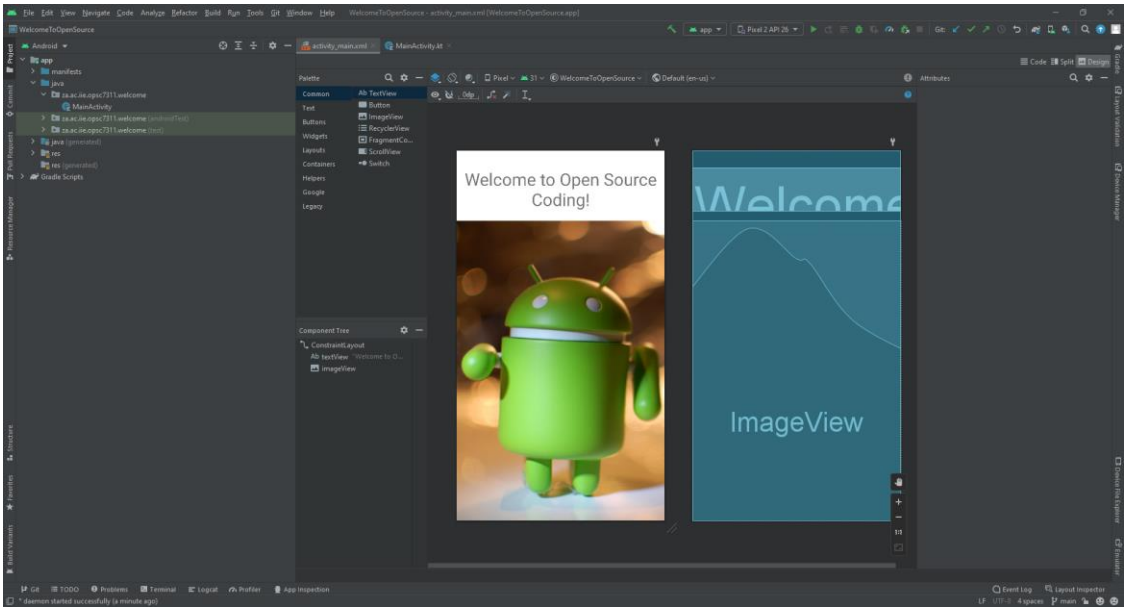


Figure 7. WelcomeToOpenSource Main Activity in Android Studio (Android picture from https://pxhere.com/en/photo/1085439)

4 Recommended Digital Engagement and Activities

The **Guru OPSC7311** playlist on **YouTube** has a lot of useful videos created for this module. The full playlist can be found here: https://www.youtube.com/playlist?list=PL480DYS-b_kdor_f0IFgS7iiEsOwxdx6w [Accessed 17 November 2022].

For this learning unit, watch the following videos from that playlist:

[YouTube] Connect Android Project to Github https://youtu.be/Kz1B0UwHQSU [Accessed 17 November 2022].

[YouTube] Commit and Push a change to GitHub https://youtu.be/q_j923SIZGI [Accessed 17 November 2022].

[YouTube] Clone Project from GitHub https://youtu.be/ag78QWpSVqw [Accessed 17 November 2022].

5 Activities

Do the activities that appear on Learn.

6 Revision Exercises

Create your own GitHub account and upload one of your own applications done in class to the repository.

7 Solutions to Revision Exercises

Share your GitHub repository link with one of your classmates. Ensure that they can clone your application and get it working.

Learning Unit 2: Creating a Basic Application	
Learning Objectives: • Create a new app. • Explain the purpose of the windows in Android Studio. • Explain the use of the layout editor. • Use images in an app. • Apply layouts to the user interface of an app. • Use the TextView and ImageView controls in an app. • Run a newly created app using the Android emulator. • Explain how to internationalize an app.	**My notes**
Material used for this learning unit: • GitHub repository: LearningUnit2	
How to prepare for this learning unit: • Make sure that you have the GitHub source code available and that your Android Studio is up to date.	

1 Introduction

In the first learning unit, we explored the Android eco system. This includes Android Studio, which is the Integrated Development Environment (IDE) that we are using in the module. We encountered Android Studio for the first time when we opened a sample project from the GitHub repository.

In this learning unit, we will be using Android Studio to build the user interface of an app.

2 The Android Studio User Interface

The main Android Studio user interface is visible to you when you have project open, that has been built. Figure 8 shows what the user interface looks like with the sample project from learning unit 1 opened. You will get to know all the elements in this screen well over time. But before we jump in, let's look at the main areas of the user interface.

These are the areas of the main window as shown in Figure 8 (Android Open Source Project, 2022b):

1. The **menu bar** lets you access all the actions that Android Studio can do.
2. The **toolbar** contains the most frequently used actions such as running the app. This is context sensitive, so for example in Figure 8 the Git actions appear on this toolbar since the project was created in a Git repository.

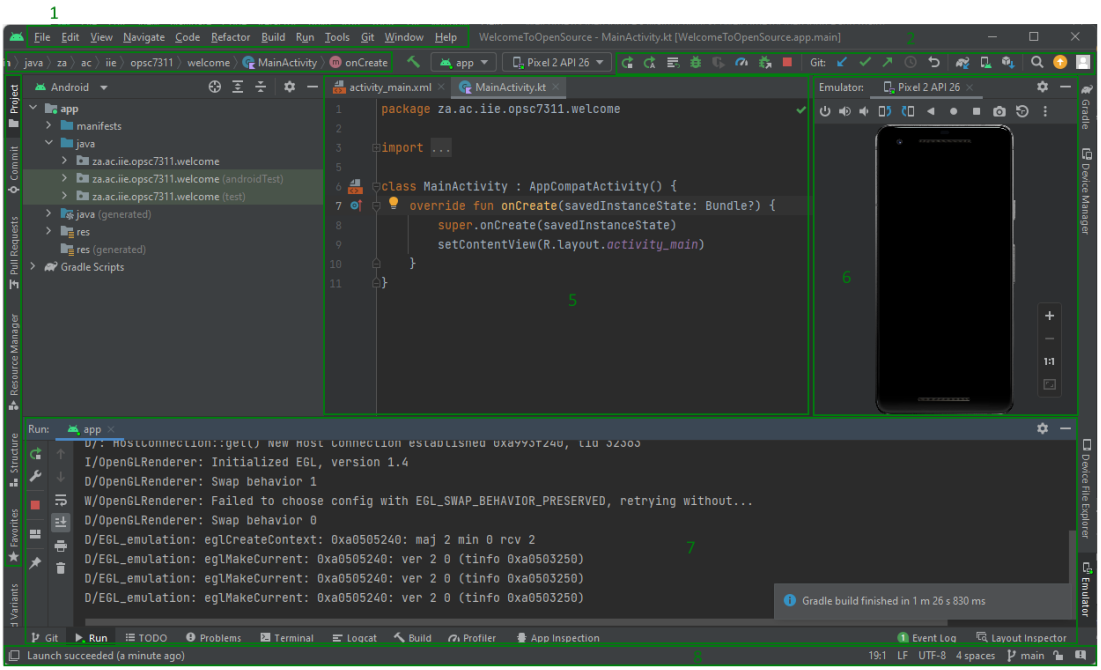


Figure 8. Main Window Areas

3. The **navigation bar** shows you a compact view of where the currently open is in the project.
4. The **tool window bar** allows you to expand or collapse tool windows.
5. The **editor window** allows you to edit code and layouts. Depending on which kind of file is currently open, this area will change.
6. The **emulator window** shows the app when it is running in the emulator.
7. The **output windows** display information about the compiling and running of an app.
8. The **status bar** displays status messages about the project and builds, including warnings and messages.

3 Creating an App

Now that we have seen the major areas of the Android Studio user interface, it is time to create our very first new project.

In the next three learning units, we are going to work on an app called *StarSucks.* It is an app for a coffee shop, and in each learning unit we will improve on the functionality. In this learning unit, we are going to create the basic user interface that displays the products sold by the coffee shop.

To create a new app using Android Studio:

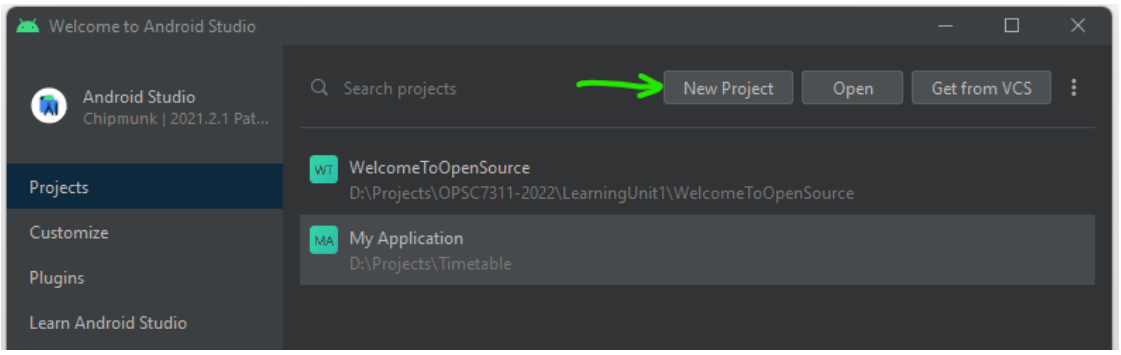


Figure 9. Creating a New Project from the Welcome Screen

1. If you are on the **Welcome to Android Studio** window (see Figure 9), click **New Project**.
2. If you are already in the main user interface of Android Studio, click the **File** menu, then **New** and then **New Project**.

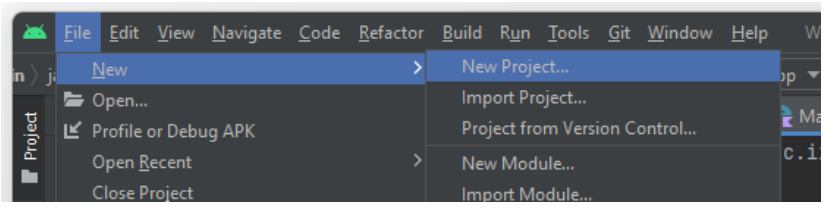


Figure 10. Creating a Project from the Main Menu

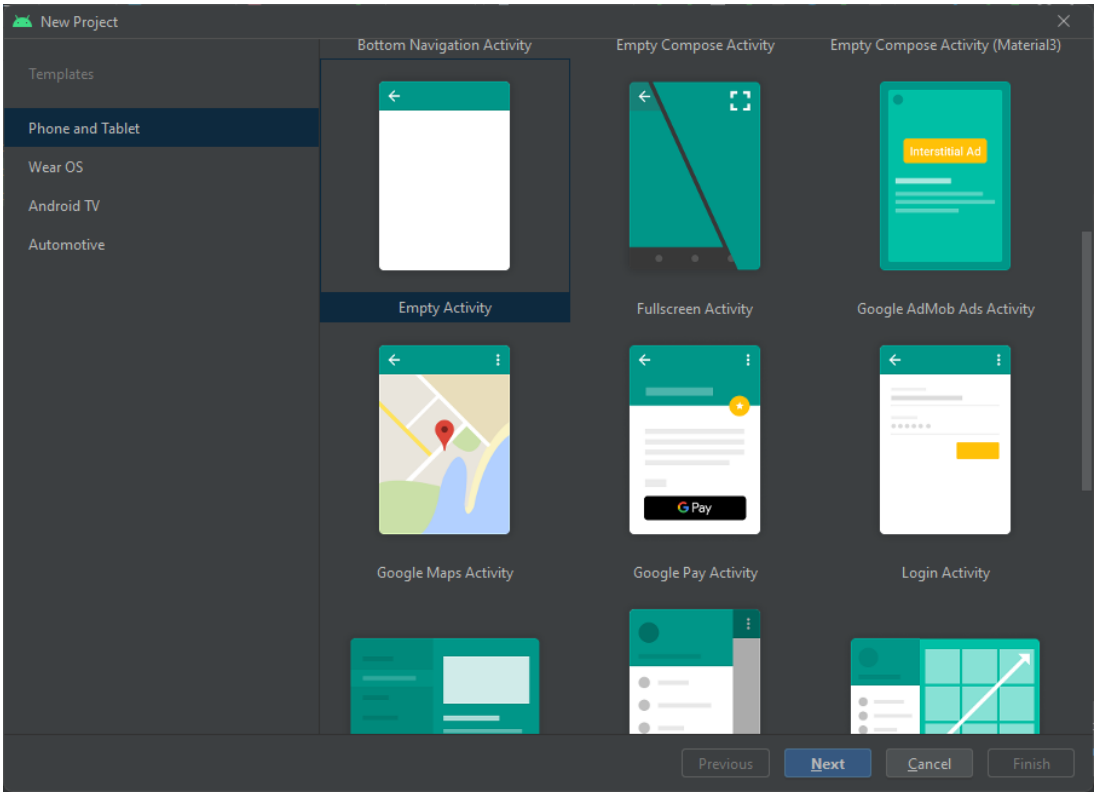


Figure 11. Select a Project Template

3. On the **Select a Project Template** page of the wizard, make sure that the **Phone and Tablet** category is selected.
4. Then select **Empty Activity** and click **Next**.

Selecting **Empty Activity** will start the project off with only a very basic, single screen. This is ideal for what we want to do here. The other activities create more advanced starting points for your app that you may want to explore later.

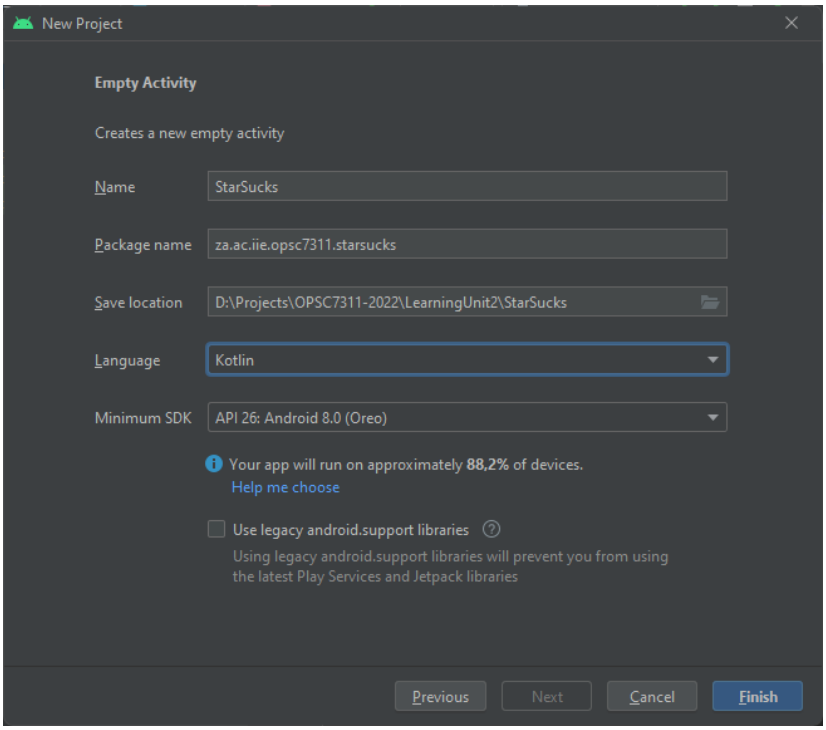


Figure 12. Configuring the Project

5. On the **New Project** wizard, we need to specify the following values:
   a. **Name** of the application. This is not easy to change, so choose well.
   b. **Package name**. The package that uniquely identifies your application. It needs to be a valid Kotlin package name.

Tip: If you ever want to **publish** your app on the Google Play Store, the **package** name that you choose when creating the app needs to be unique. The default `com.example` suggested by Android Studio will NOT be allowed by the Play Store. So, you must choose something that is going to be unique.

The Kotlin naming convention for packages is to use your organisation's website since that is already guaranteed to be unique. You could use for example:

```
com.vegaschool.st98765432.weatherapp
```

or

```
za.co.varsitycollege.st98765432.weatherapp
```

if st98765432 was your student number. Note how the top-level domain is first in the package name. It starts with com or za – the opposite of the website address.

c. **Save location**. The folder where the app will be created.
d. **Language**. For our purposes here we are using Kotlin. So, to follow along with this example make sure you select Kotlin.
e. **Finally**, you need to decide on your **Minimum API Level**. This affects which devices can “see” your app on the Play Store – only devices with Orea (8.0) and upwards will be able to “see” this app. You also need all the relevant API’s installed to use the min API. You should have Marshmallow, Nougat and Oreo installed. Android will set your Max API to the latest API that you have installed. In our case it is Oreo – these change quickly so you might even have several newer versions too.

6. Click **Finish**

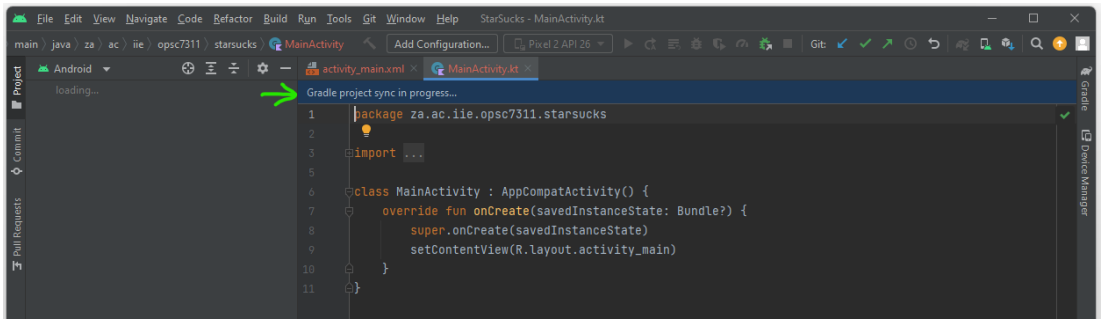


Figure 13. Gradle sync in progress

The project will now be displayed in the main Android Studio user interface. Click the **Build** tab (bottom of the user interface) to see what is happening in the background when the project is created. You will see a Gradle sync process that will be started. This may take a while when you first create the project, since it retrieves all the libraries that you need for this project from central artifact repositories online.

Once the initial build is complete, the files in the project will be displayed as shown in Figure 14.

Side Note: You may notice that the filenames in Figure 14 are displayed in red. This is because the files have not yet been committed to the repository.

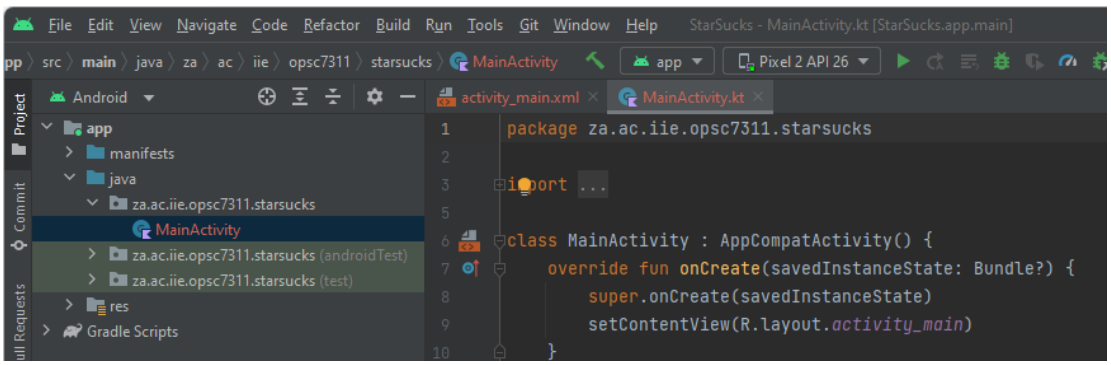


Figure 14. Project in the Main Android Studio Window

We selected the empty activity template when we created the app, to keep things simple. But there are several other activity templates that are available out of the box in Android Studio. Read more about all the templates in (Rout, 2022).

3.1 Running the App

We will look in more detail at what happens when you run an app later in this learning unit, but let's run the app on the emulator in the meantime.

Just as with any other programming work, it is important to run the app early and often, to make sure it works as expected. If you write a ton of code and then it doesn't compile, or doesn't work, it is much harder to debug than fixing a small piece of code.

Before you can run the app for the first time, you need to set up a Virtual Device. This is where you can choose what device to emulate.

To set up a new virtual device:

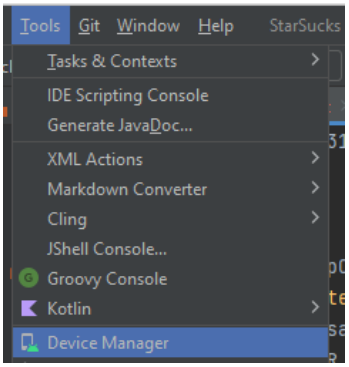


Figure 15. Accessing Device Manager

1. From the main menu, select **Tools** > **Device Manager**

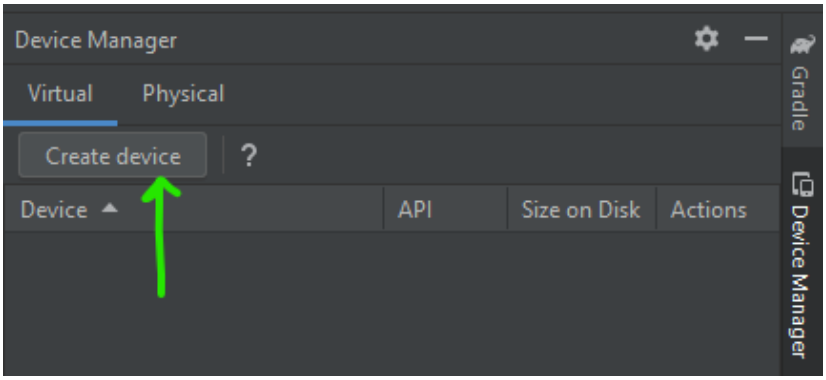


Figure 16. Device Manager

2. On the **Device Manager** window, on the **Virtual** tab, click **Create device**.

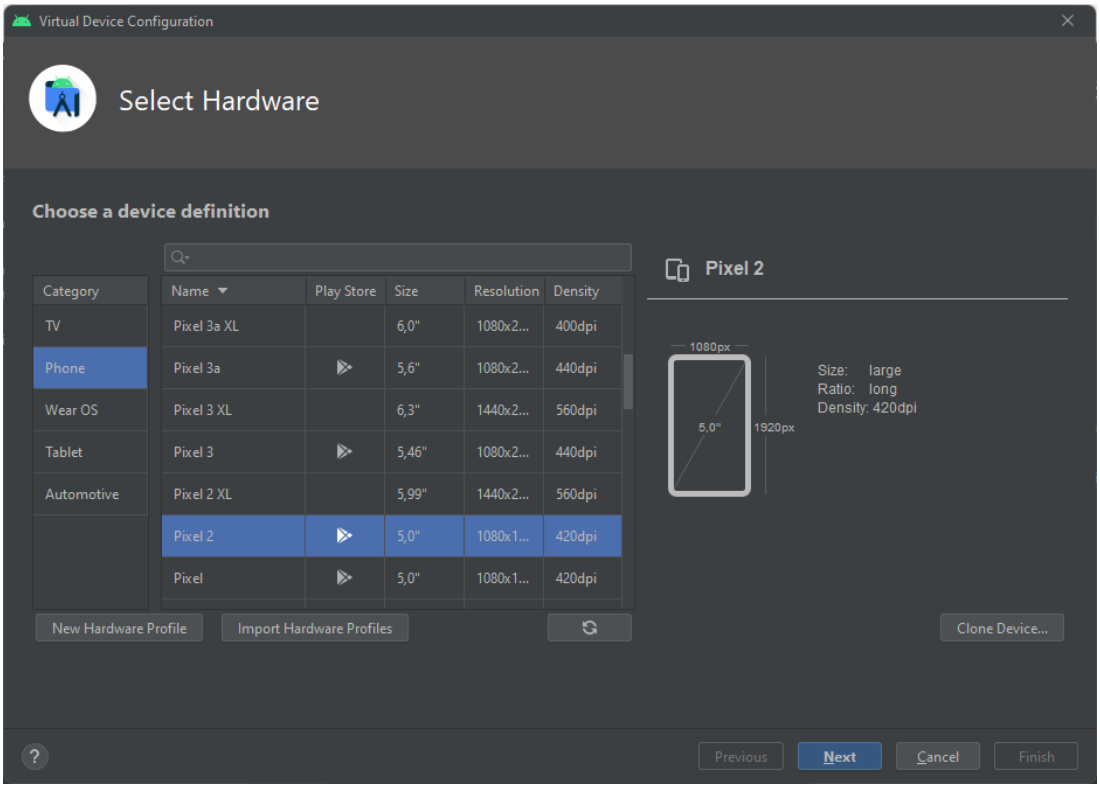


Figure 17. Selecting the Hardware

3. Select the **Phone** category since our app is meant to run on a phone.
4. Select the device that you want to emulate. Let's use the Pixel 2 device.

> **Tip:** The **Play Store** icon indicates whether the operating system will have the Play Store pre-installed. If it is installed, you can log in with your Google login and download apps on the emulator, just like you would for your phone.

5. Click **Next**.
6. Next, we select the operating system (see Figure 18) that we are going to use. Click the **Download** link next to any of the system images to download it. Once it is downloaded, it can be selected.
7. Select **Oreo** or newer.
8. Click **Next**.

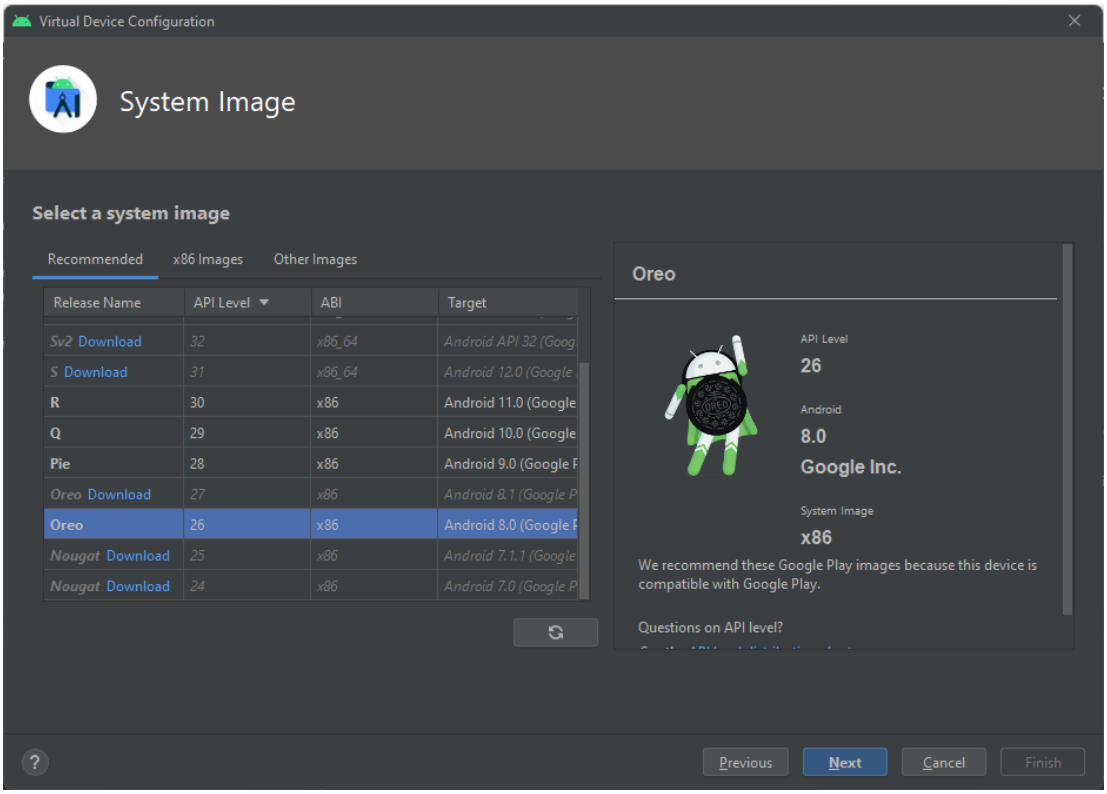


Figure 18. Selecting the Operating System

Tip: The system images are quite large. Make sure that you are connected to wi-fi and not a metered network before you start the download!

9. Lastly, we can name the virtual device (see Figure 20). This will be useful if you have multiple devices with different configurations. For now, the default will be fine.
10. Click **Finish**.

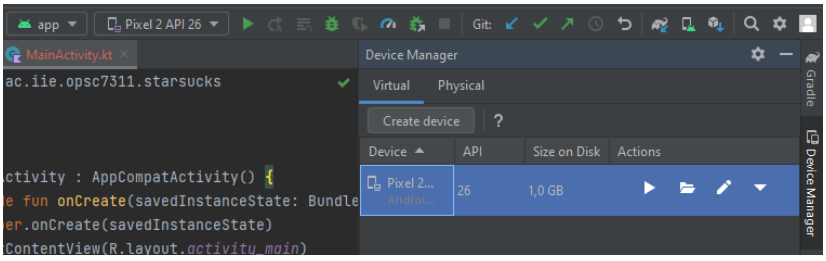


Figure 19. Virtual device appearing in Android Studio

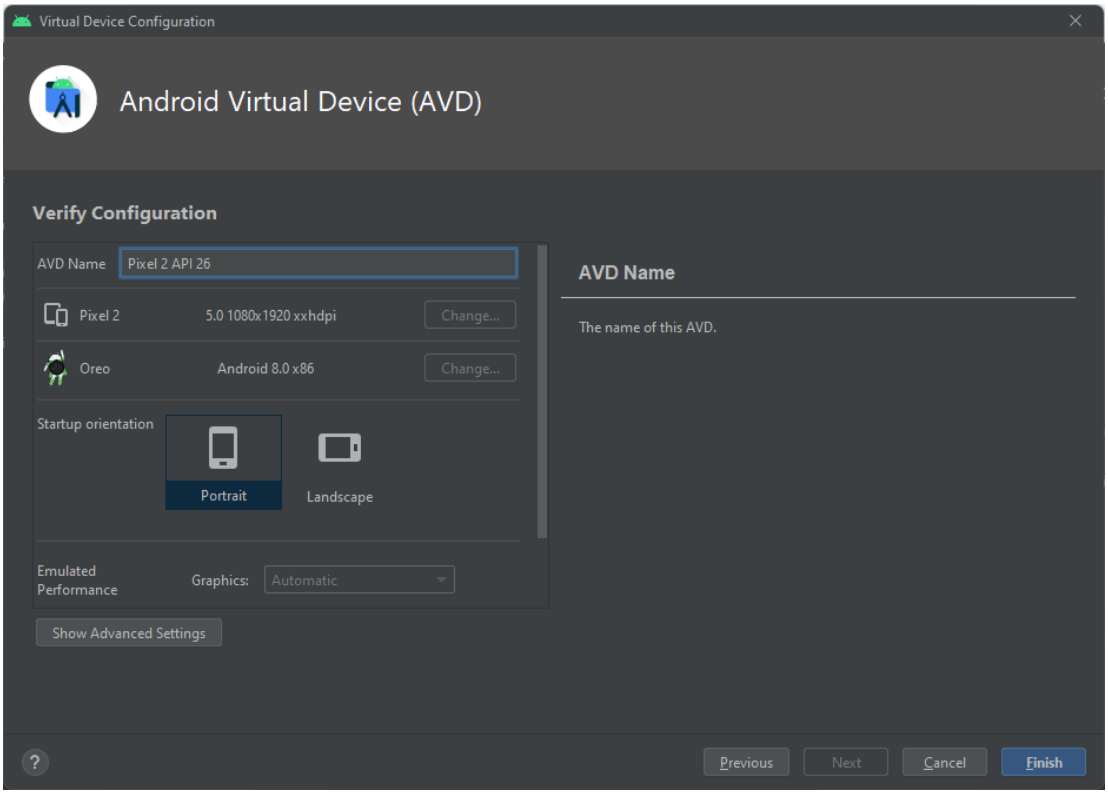


Figure 20. Name and Verify Configuration

Now that the virtual device has been set up, you can run the app for the first time.

To run the app, click the Run app button on the toolbar (▶) or click **Run** and then **Run App** on the main menu bar.

Tip: Minimise the **Device Manager** window to make more space for the **Emulator** window.

When the Virtual Device is started up for the very first time, it will take some time to start up. Eventually, you should see the app running in the emulator window in Android Studio as shown in Figure 21.

Tip: Starting the emulator up takes a while, even if it is not the very first run. But running the app on the emulator after that is quite fast. So, start the emulator early and leave it running. Then you can quickly run the app as you make changes.

Tip: If the emulator screen switching off annoys you, you can change the timeout in the Android operating system running in the emulator. It is under settings > **Display** > **Advanced** > **Sleep**. The maximum timeout is 30 minutes, which is much better for this use than the default 1 minute.

Figure 21. App Running on the Emulator

4 Building a User Interface

4.1 Android Studio Tools

Our app is currently still quite empty and not yet very exciting. So, let us start adding User Interface (UI) components to it.

There are two ways in which you can create a UI in Android Studio: you can code your UI in Extensible Markup Language (**XML**), or you can **design** it by using the Layout Editor. If you use the Layout Editor, it will update the XML file for you.

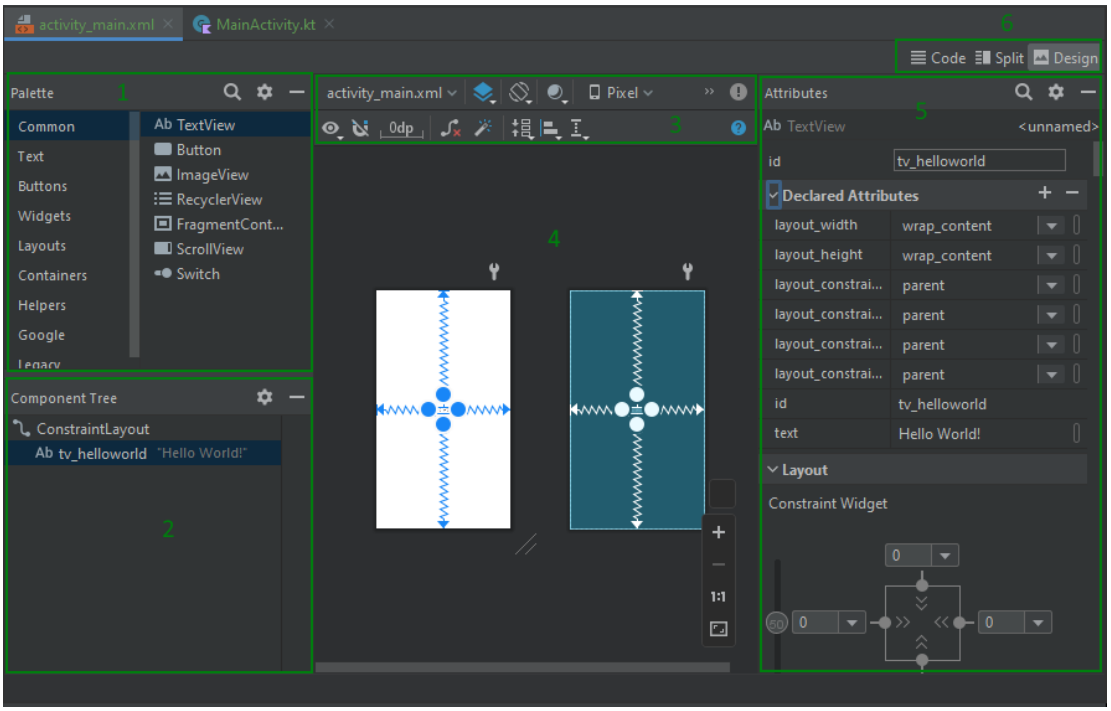


Figure 22. Layout Editor Components

The areas of the Layout Editor in Figure 22 are as follows (Android Open Source Project, 2020f):

1. The **Palette** contains the components that you can drag onto your layout.
2. The **Component Tree** shows all the components that are already on the layout.
3. The **toolbar** contains tools to change the layout appearance and attributes. For example, you could change the app's theme to see how it would look.
4. The **design editor** shows the layout in design view and blueprint view.
5. The **attributes** window shows the attributes of the currently selected component.
6. The **view mode** buttons allow you to switch between the **Code** (XML) view, **Design** view or a **Split** view showing both.

Let's switch now to the Code view to see what that looks like.

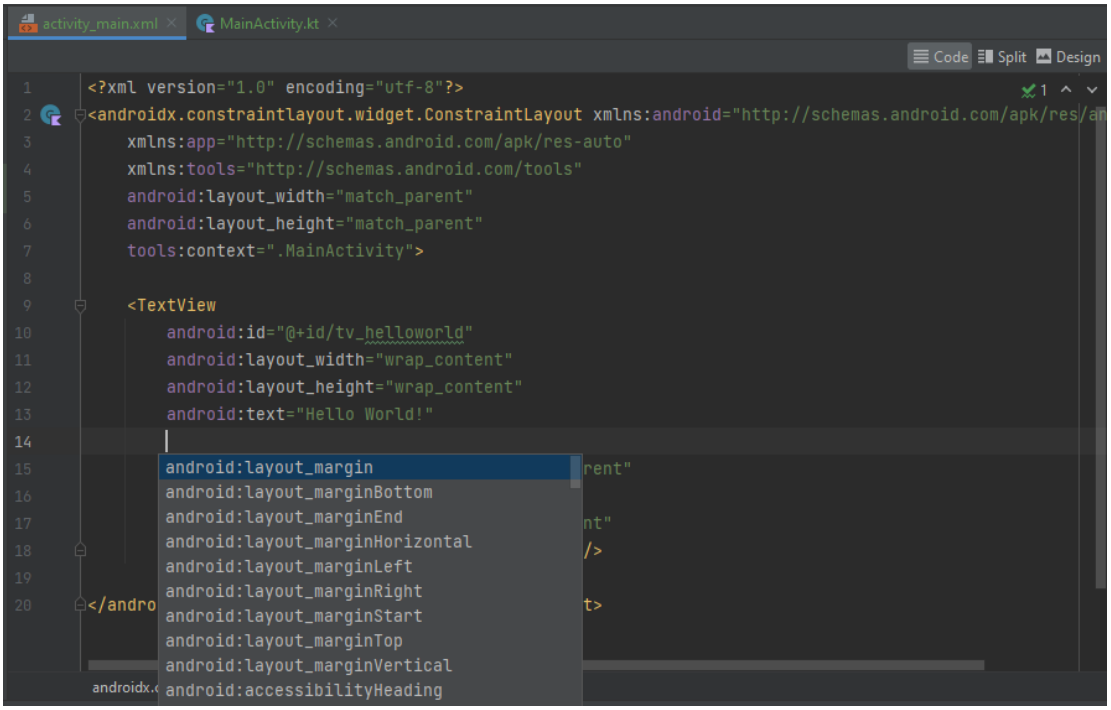


Figure 23. Code View

In Figure 23, we see the XML representation of our design. The XML editor does provide auto-complete functionality (by pressing `Ctrl+Space` in editor).

Some developers prefer using the Layout Editor to drag and drop and then customise UI controls in Android. Other developers prefer to code up the UI in XML.

4.2 Android Layouts

Android has different layouts which follow a hierarchical structure. You can use a Constraint, Linear, or Relative Layout to design your application UI. Each of these layouts have features that are useful in different UIs.

If we look at the structure below, we see the root element as a `ViewGroup`, we really need to understand what a `View` is before we can understand a `ViewGroup`. Android defines as a `View` as a UI element that draws something the user sees. For example, a button is view, an `ImageView` is a view, a scrollbar is a view.

A `ViewGroup` is a layout that contains multiple `Views`.

We create a hierarchical structure when we design our layouts, like what is shown in Figure 24. We will compare this diagram to the image of the XML layout below that.

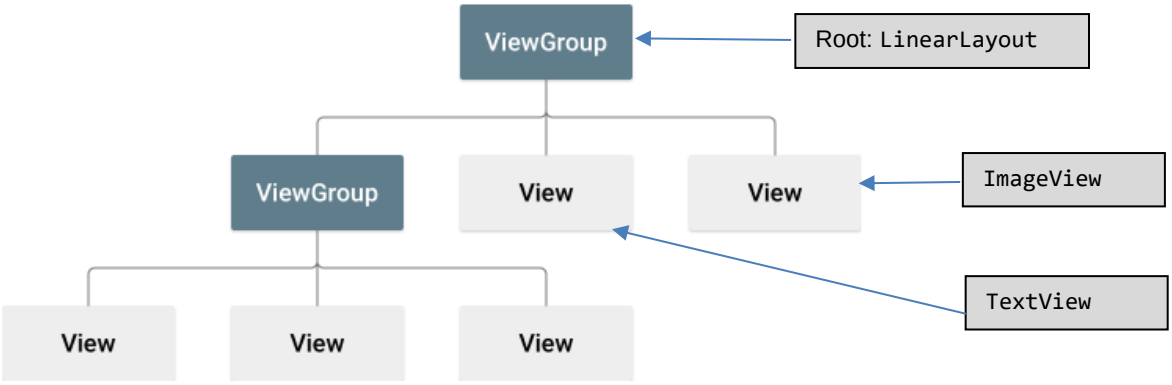


Figure 24. Hierarchy of Views (Android Open Source Project, 2020g)

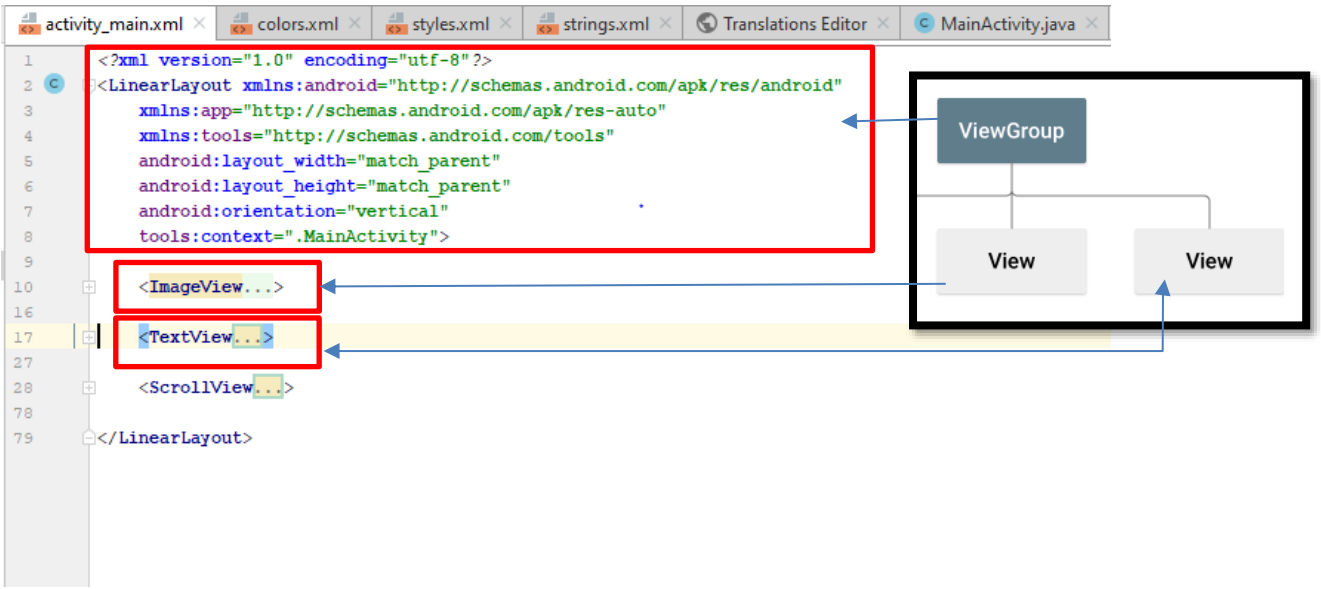


Figure 25. XML Layout

In Figure 25, we have our root view `LinearLayout` which contains two views: an `ImageView` and a `TextView`. Below that is a `ScrollView`. In Figure 26, we see that `ScrollView` expanded to show its contents: a `LinearLayout` with multiple `ImageViews`.

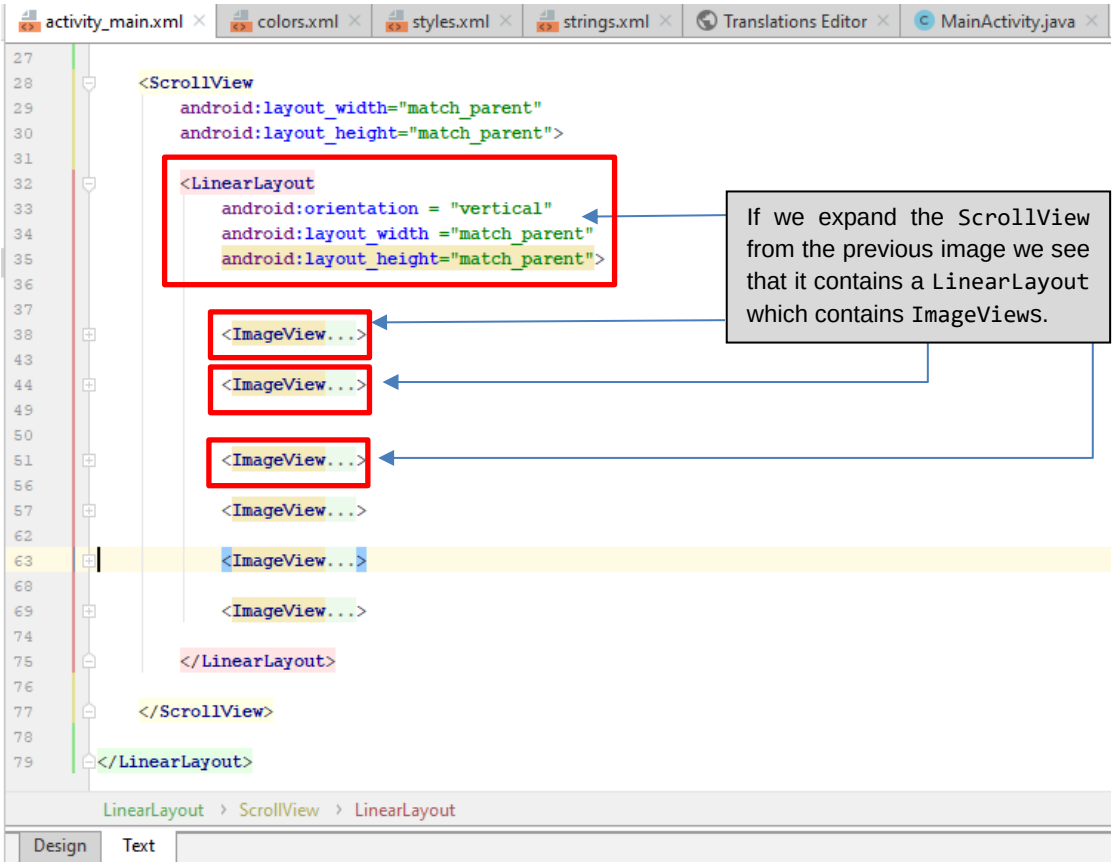


Figure 26. Expanded Scroll View

This `ViewGroup` that makes up our UI looks like the image below.

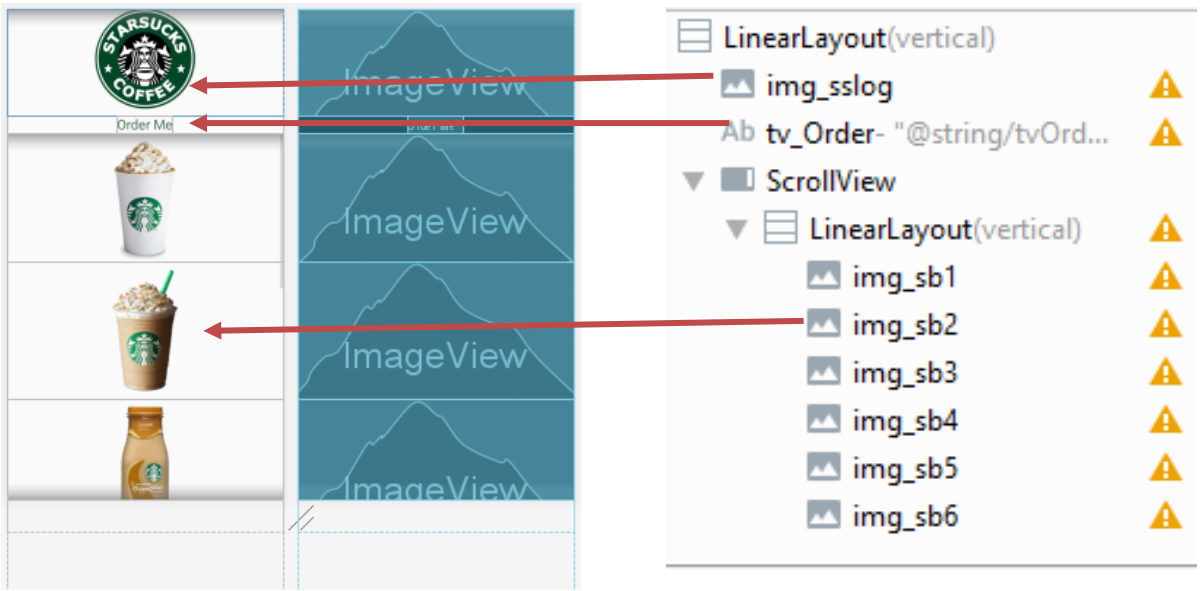


Figure 27. Hierarchy of Views

You cannot see the `LinearLayouts` (they are invisible) nor the `ScrollView` – but you can see the `ImageViews` and `TextViews` and how they fit into the hierarchical structure discussed above.

Tip: It is incredibly important to consider good UI and UX when you develop Android apps. Users are likely to abandon even the best developed app if it does not look good and flow well. First impressions matter!

Let us have a look at the different layouts that we have available in an Android App.

4.2.1 LinearLayout

`LinearLayout` is a root view that will stack all its children either horizontally next to each other or vertically on top of each other.

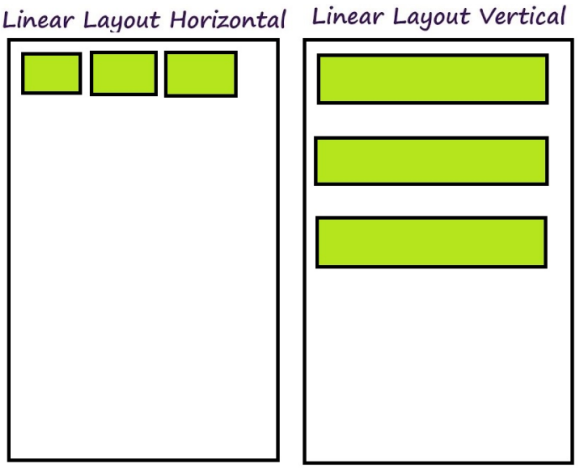


Figure 28. Horizontal and Vertical `LinearLayouts` (Ayub, 2017)

You can learn more about Linear Layouts (Ayub, 2017).

4.2.2 RelativeLayout

`RelativeLayout` aligns UI components relative to each other on the screen and does not need nested layouts.

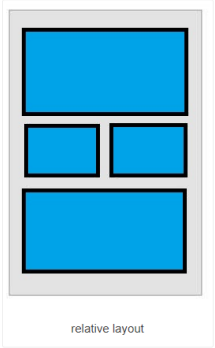


Figure 29. Relative Layout (Ayub, 2017b)

You can learn more about relative layouts in (Ayub, 2017b).

4.2.3 ConstraintLayout

`ConstraintLayout` is like `RelativeLayout` but a lot more flexible.

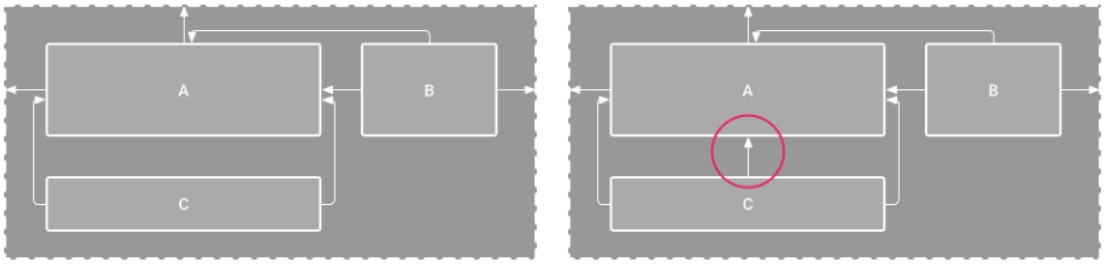


Figure 30. Constraint Layout (Android Open Source Project, 2020i)

You can learn more about `ConstraintLayout` in (Android Open Source Project, 2020i).

You can learn more about these layouts and how they function in (Lake, 2016) and (Android Open Source Project, 2020g).

4.3 About Resources

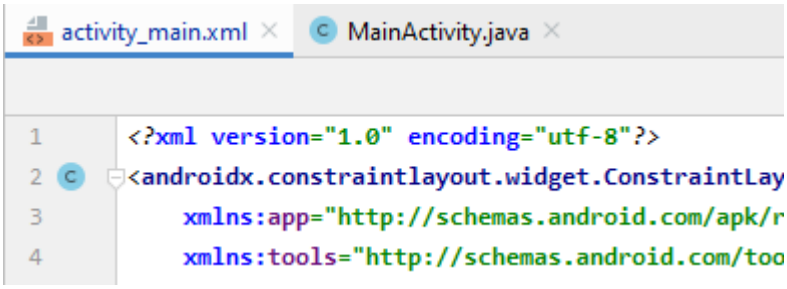


Figure 31. Two Main Activity Files

Each activity has two files that get created by Android Studio: a Kotlin source file, and a layout XML file. The Kotlin file contains the behaviour of the activity, and the XML file the layout. The XML file is a kind of resource file. So, to understand that we need to take a closer look at resources.

This split is like Windows Presentation Foundation (WPF) when the UI is declared in eXtensible Application Markup Language (XAML) and the behaviour in C#.

4.3.1 The `res` Folder

Definition

"**Resources** are the additional files and static content that your code uses, such as bitmaps, layout definitions, user interface strings, animation instructions, and more." (Android Open Source Project, 2020h)

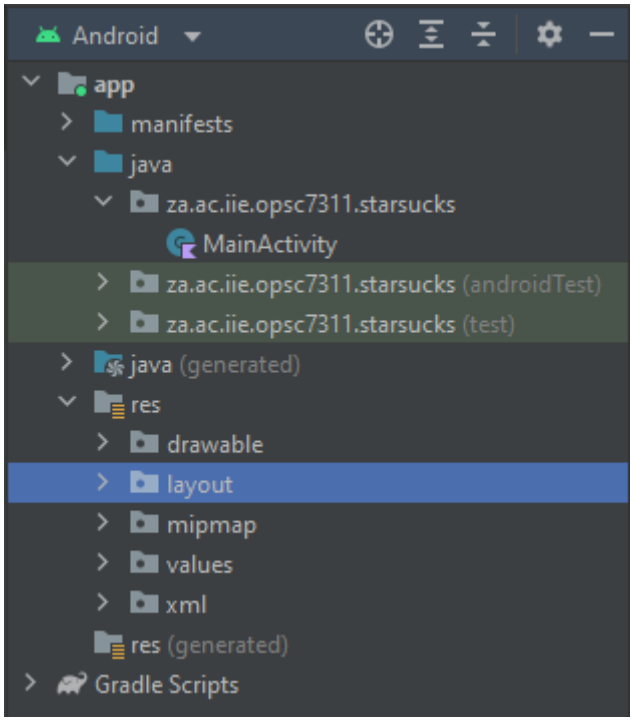


Figure 32. Resources Folder

Your Android resources are contained in the `res` folder, which correlates to the R class in Kotlin. You can think of the `R` class as the glue to combines your Kotlin code and any resource stored in the res folder. We will store our XML files as well as our images, colours, strings, and app icons in here. The `res` folder is broken down into the following folders:

- `drawable` will contain all our images, shapes, bitmaps, and vectors etc.
- `layout` contains the layouts for all our activities, fragments etc.
- `mipmap` will house our app icon and various place holders.
- `values` folder holds our String values, Colours, Themes etc.

We need to store all the images we want to use in our app in the `drawable` folder. We can store this using Windows Explorer, or we can just copy our images straight into the drawable folder.

The StarSucks images can be found in the GitHub repo, in the `LearningUnit2/assets` folder. While holding Ctrl, draw the files into the `drawable` folder in Android Studio.

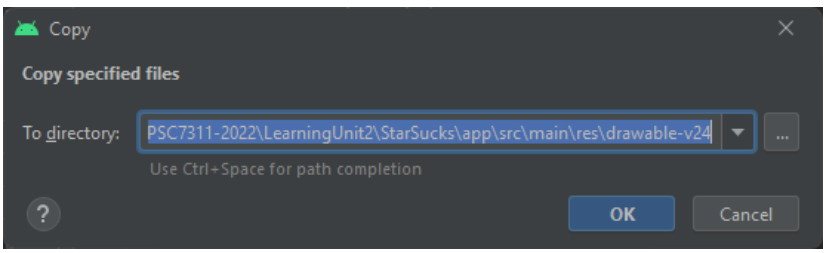


Figure 33. Copying Images

Make sure that the dialog title says **Copy** as shown in Figure 33. Otherwise, it will move the files, which might not be what you intended. The files should appear in your `res\drawable` folder once you have copied them over.

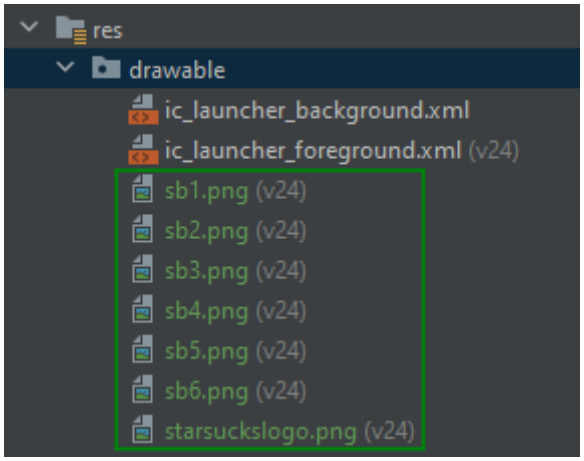


Figure 34. Images in the `drawable` folder

We are now ready to start building our first app that will display our images. There are just some rules that we need to discuss. Your app will crash if you use characters that are not allowed when naming your images.

Image naming convention: You can use small letters from a-z, numbers, and underscores.

No special characters are allowed because the image in the `drawable` folder is written into your `R` class when your Gradle builds. Characters that are used in Kotlin will then cause problems. For example, we use & in Kotlin to AND and || to OR etc.

4.3.2 The `R` Class

The `R` class is **auto generated** by the Android Asset Packaging Tool (AAPT) and contains all the resource ids to your resources in the `res` folder. The `R` class is automatically updated as you add UI elements in XML files, images in the `drawable` folder and strings in the `strings.xml` file. The `R` class is rebuilt every time you build or run your Android project.

Let's look at what the `R` class does.

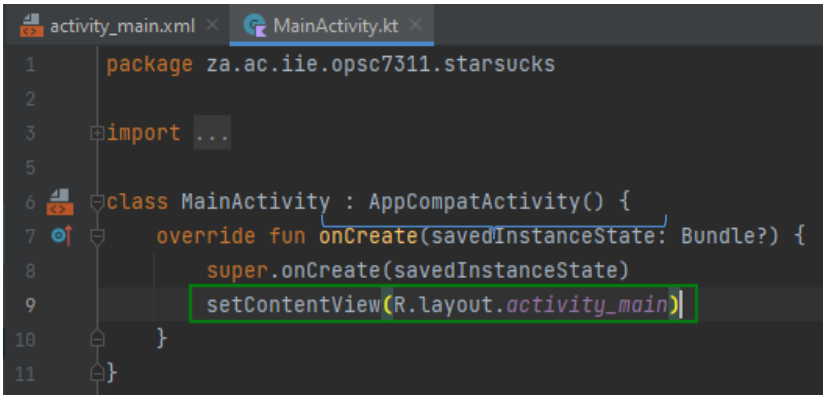


Figure 35. Using the `R` Class

In Figure 35, we see how the `R` class is used in the MainActivity.kt file. We are calling a method called `setContentView` here, which will of course set our view. This method is available because we are inheriting form the `AppCompatActivity` class. (We will look at Kotlin code in more detail in the next learning unit.)

We are also passing an argument into this method. We are telling Android where to find the view that we would like to display to the user when the app opens. We are passing in `R.layout.activity_main`. This means in the `R` class, there is nested class called `layout` and it contains a variable that holds the ID for the activity that we would like to load.

We don't need to know that numerical value for the ID, because the compiler takes care of it for us. What is useful to know, is that this is the line of code that links the XML file to the source code that specifies the related behaviour.

4.4 Creating the User Interface

All of this may seem quite overwhelming by now – so many layouts, so many components and so many moving parts just to display a UI. But you will see it is not that bad when we use it. So, let's jump into creating the UI.

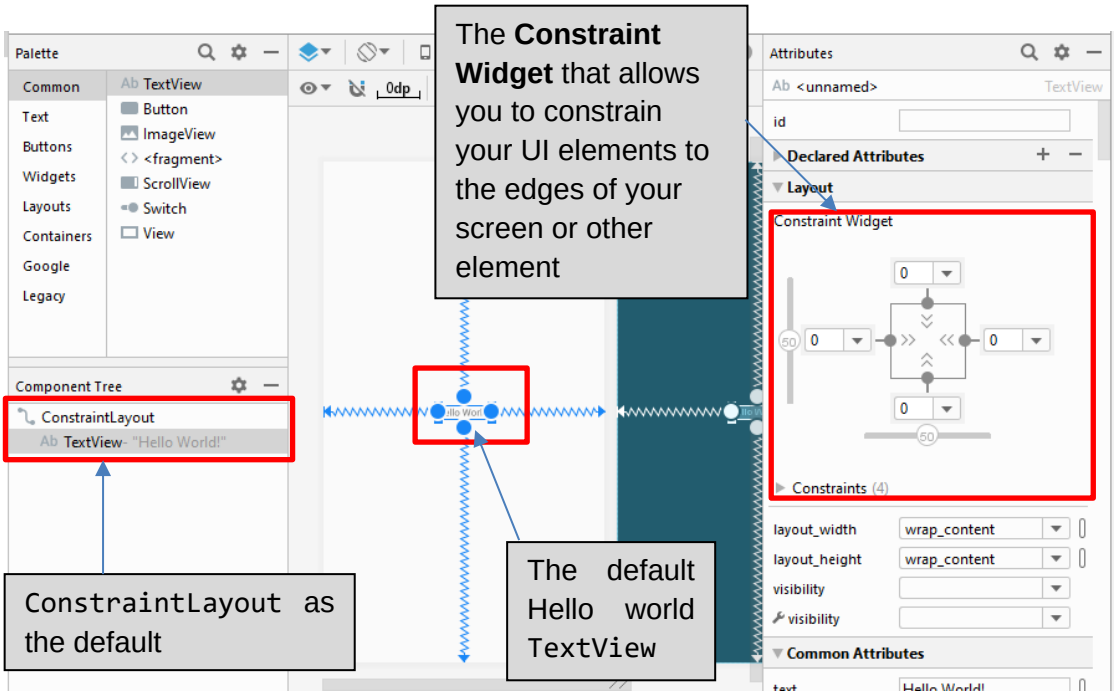


Figure 36. Empty Activity in the Layout Editor

4.4.1 Changing the Layout

Android Studio will have `ConstraintLayout` as its default layout when you create a new empty activity. We are going to design our first UI using `LinearLayout` because we want our elements vertically aligned. When you open your Layout Editor you will see the view shown in Figure 36.

We are going to change our layout to `LinearLayout` for now. There are two ways of doing this: using the Layout Editor or doing it in the XML file.

To change the layout using the Layout Editor:

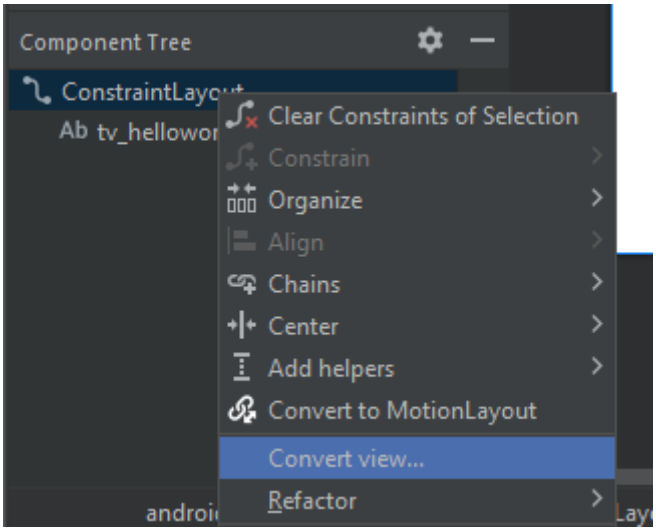


Figure 37. Converting the View

1. Right-click on the `ConstraintLayout` in the **Component Tree** and click **Convert view**.

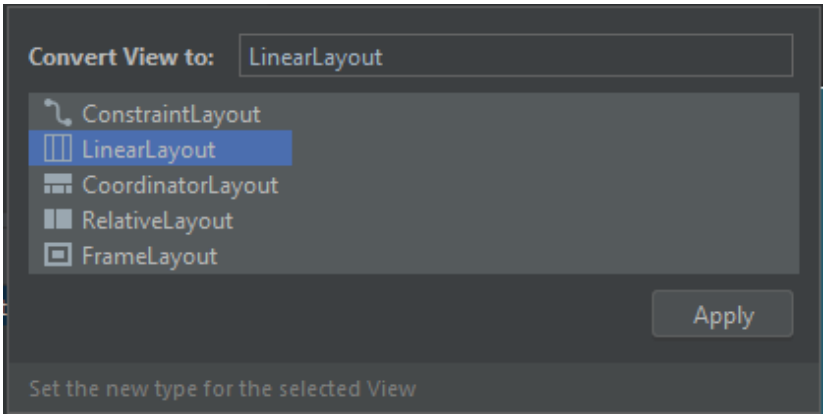


Figure 38. Selecting Linear Layout

2. Select **`LinearLayout`** and click **Apply**.

To change it by using the Code (XML) view:

1. Click **Code** (top-right corner of the Layout Editor).

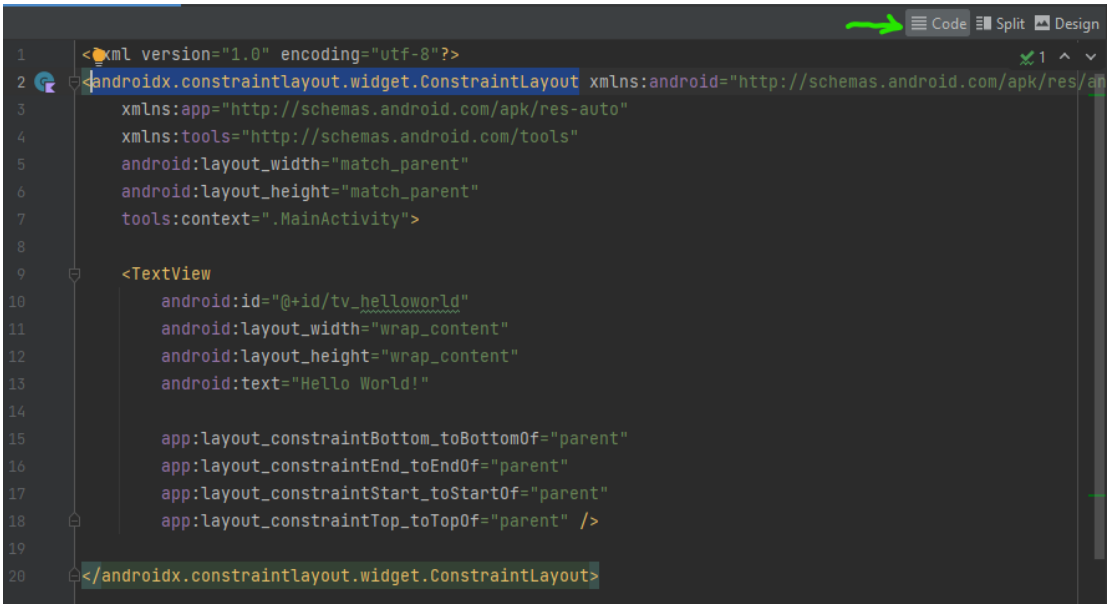


Figure 39. Select the Current Layout

2. Select the current layout's text as shown in Figure 39.

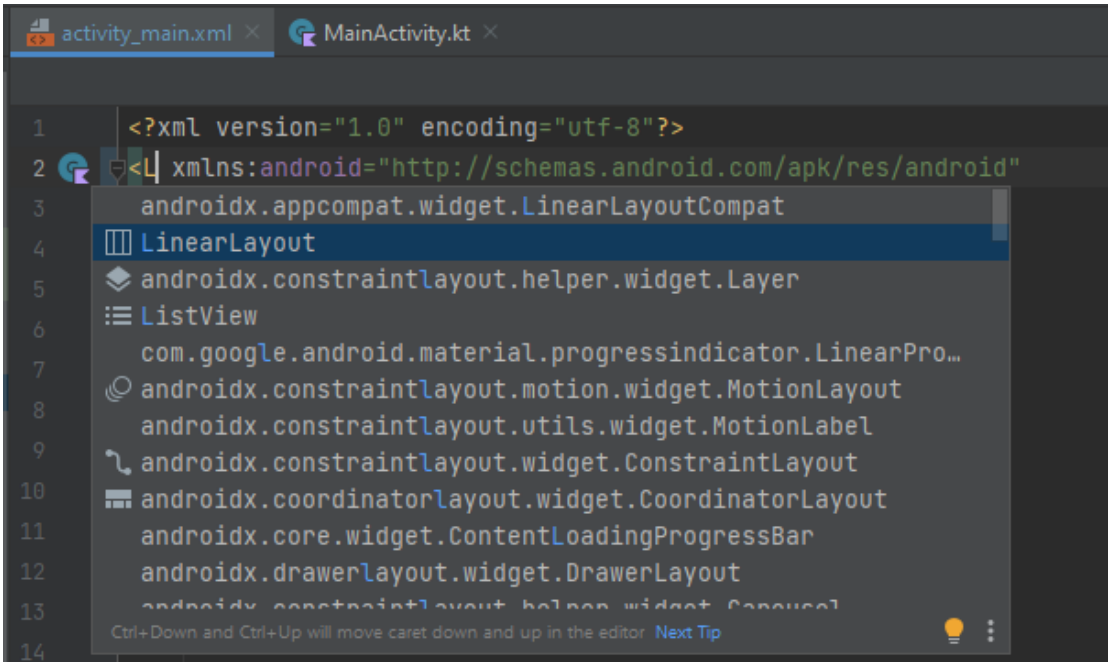


Figure 40. Autocomplete Pop-Up

3. Start typing a capital letter L. You should see a pop-up menu that lists `LinearLayout` near the top of the list.
4. Double click on `LinearLayout`.

Now the layout is changed to `LinearLayout`, and it is ready to use.


```
<?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:id="@+id/tv_helloworld"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
```

Figure 41. Changed to LinearLayout

Now we need to decide if we want to stack our UI elements (Views) vertically or horizontally. This is easily done in either the Layout Editor or XML.

You can simply select the correct orientation from the attributes for your layout in the Layout Editor. We are selecting **Vertical** for this example.

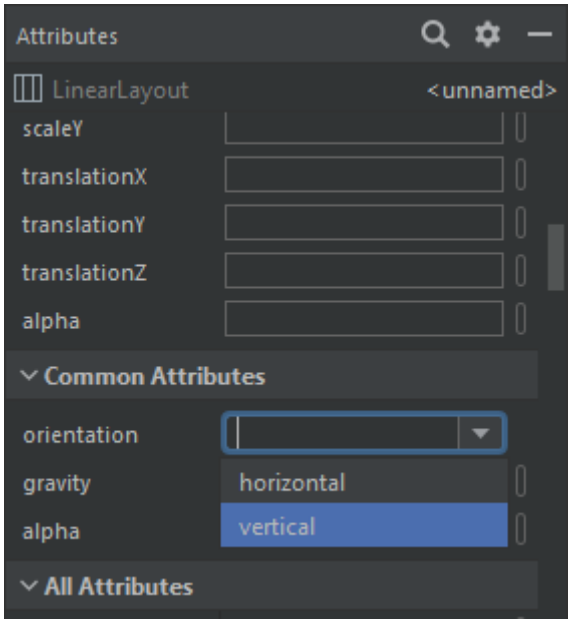


Figure 42. Selecting Vertical Orientation

You can also just add the following attribute to your XML by typing in the following attribute. (There will be *IntelliSense* – IDE pop-ups – to help you.)

```
android:orientation="vertical"
```

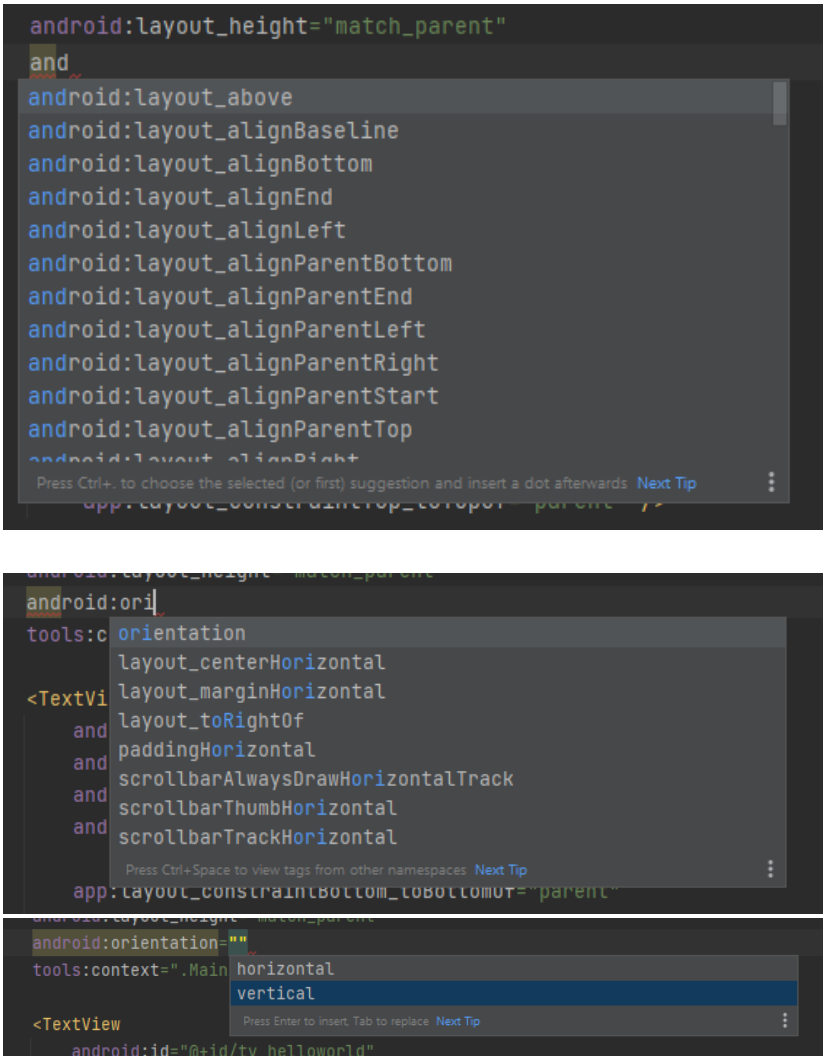


Figure 43. Adding the Orientation Attribute

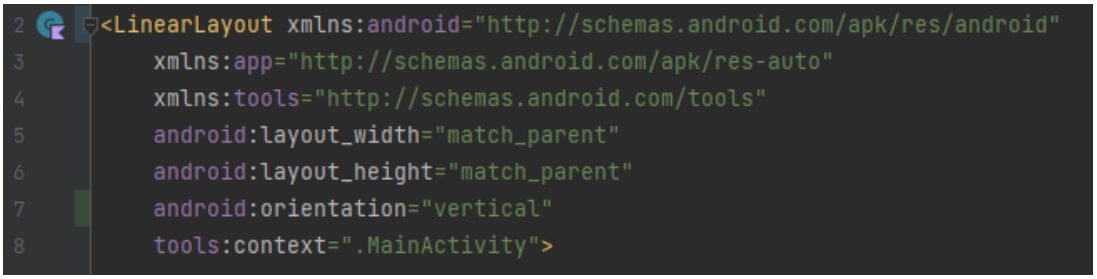


Figure 44. The Completed Orientation Attribute in the XML File

We will now have all our UI elements vertically stacked on top of each other in the order that we add them to XML file or Layout Editor.

4.4.2 Adding an ImageView

Now that we have our layout ready, we can start adding in our UI components (`Views`). For our first app we are only adding a `TextView` and an `ImageView`. We will start with the `ImageView`. Once again, this is possible through the Layout Editor or XML.

To add the ImageView using the Layout Editor:

1. Look for the **Palette** which contains the Views that can be added to the Layout.
2. Click the **Common** category to find the most used views. You should see `ImageView` listed there.

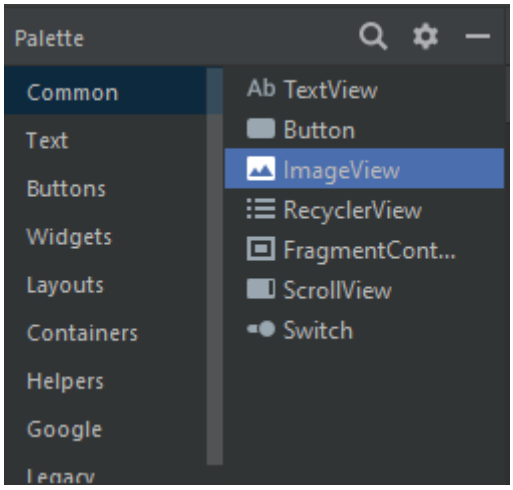


Figure 45. ImageView on the Palette

3. Drag and drop the `ImageView` onto your design preview.

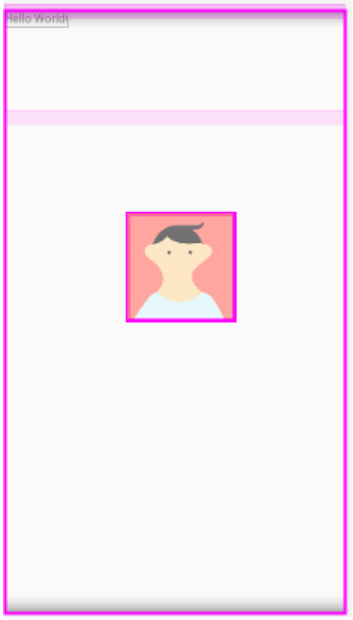


Figure 46. Dropping the ImageView

4. You will see the little guy shown in Figure 46 as you drag the `ImageView` onto the preview. Drop this image anywhere on the preview, we are going to move it where we like it while we design this UI.

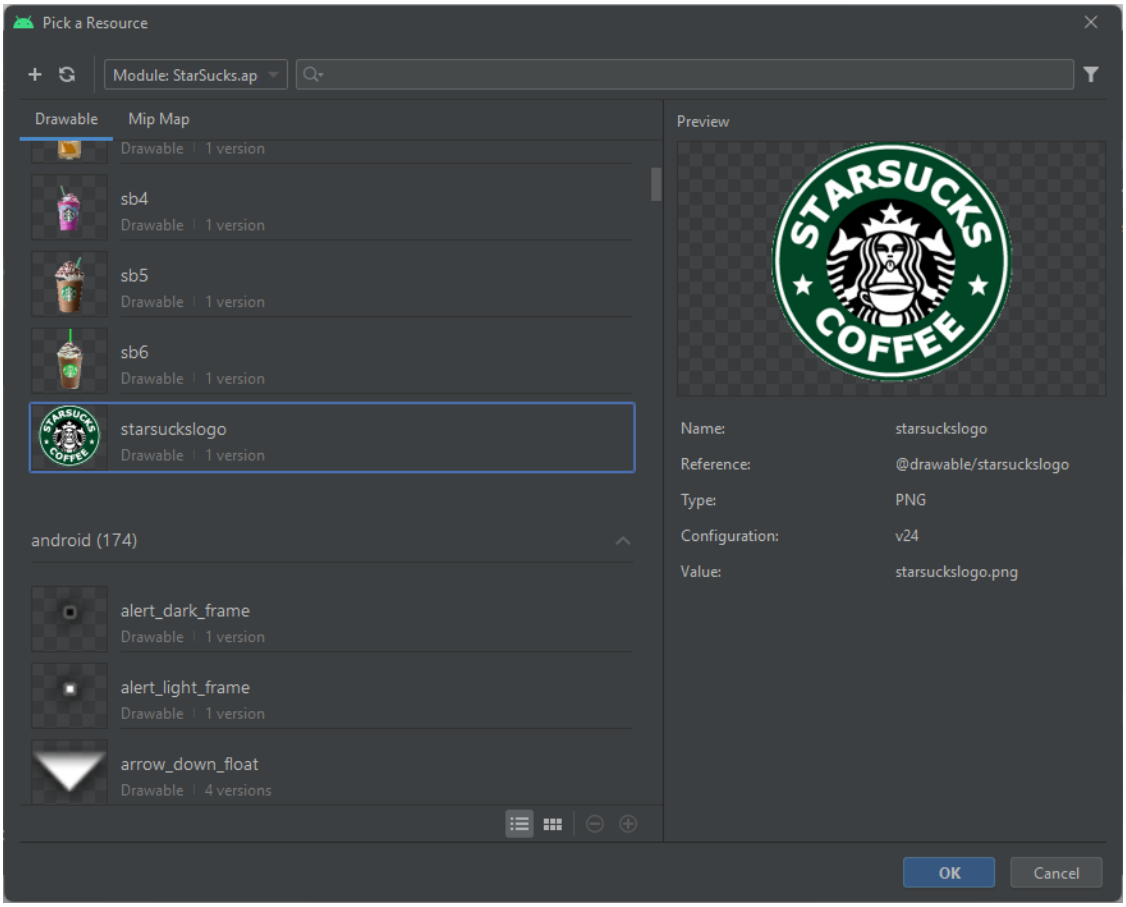


Figure 47. Picking an Image

5. Scroll down to the `starsuckslogo` under the `StarSucks.app.main` category.
6. Select the image and click **OK**.

> **Tip:** Do you see the checkered background for these images? That means that the images have a **transparent background**. It is important to design good UIs so please use images with transparent backgrounds!

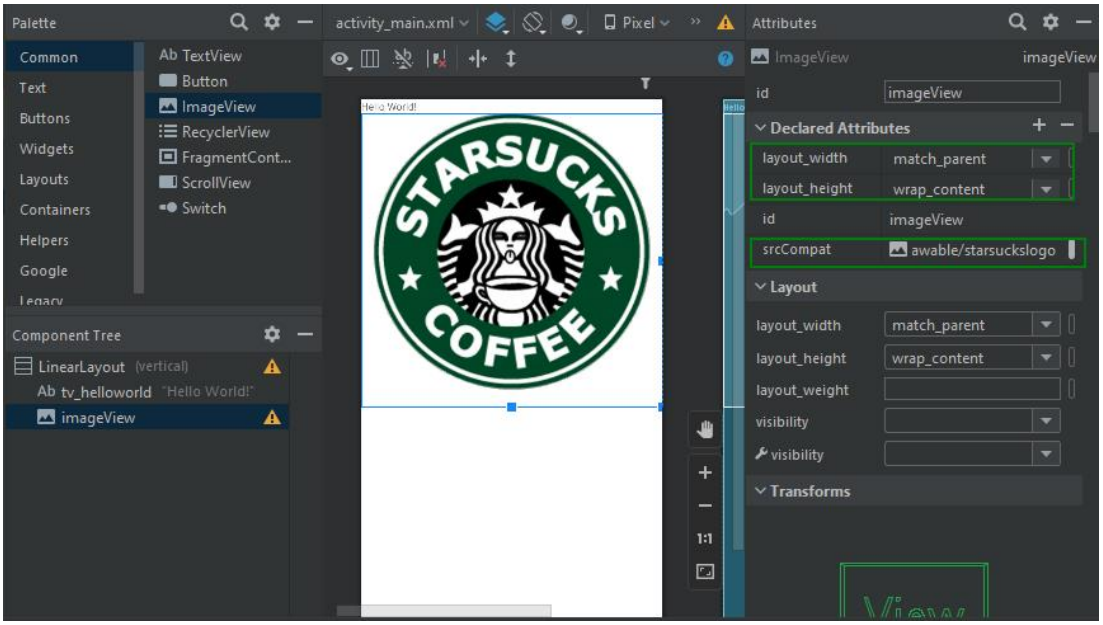


Figure 48. ImageView in the Preview

Now the `ImageView` will appear on your preview. There are a couple of things we should note here:

- `layout_width` and `layout_height`: The width and height attributes need to be set for every UI element (View) you add in Android. Your IDE will display an error if these are absent (only in XML – the Layout Editor will add defaults).
- `srcCompat`: Tells us where the image is stored.

This is image is currently too big. We can resize this image by dragging the blue borders until we get the size that we want.

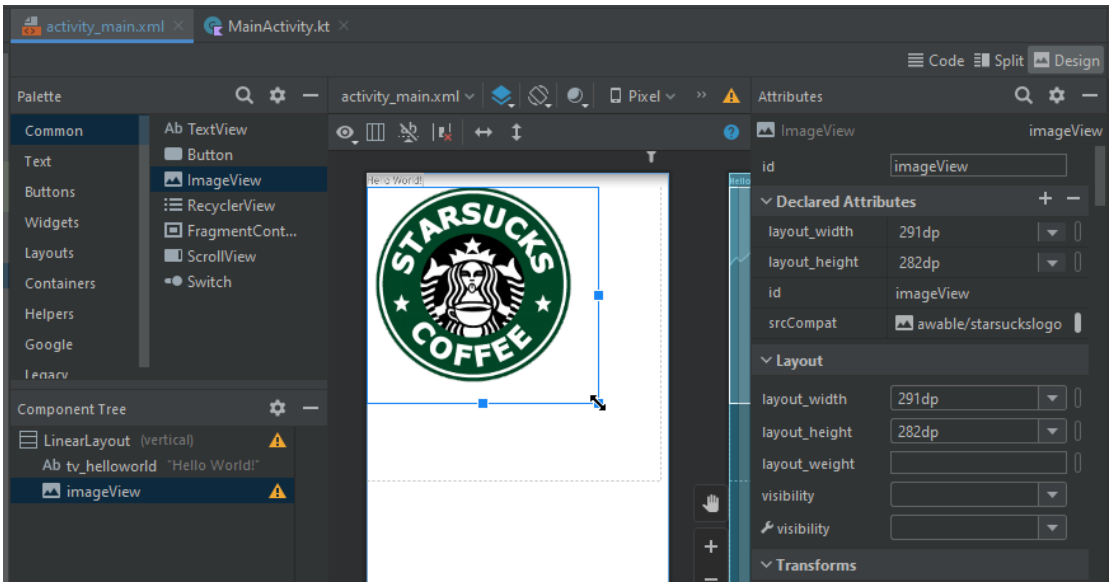


Figure 49. Resizing the ImageView

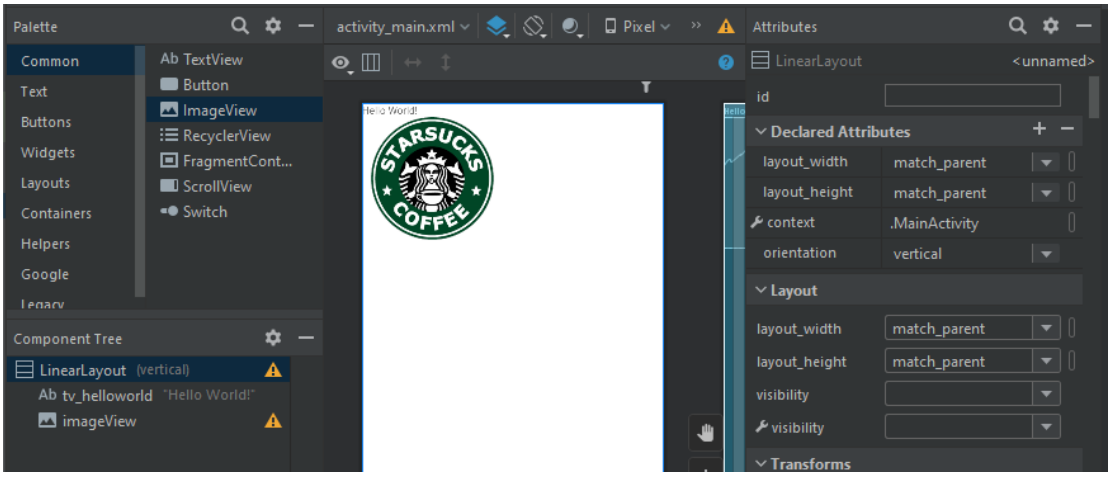


Figure 50. ImageView after the Resizing

That is much better, but now our image is stuck to the left of our screen, and we would like it right smack in the middle. We need to set some attributes for this.

Android has a lot of attributes for Views. You need to expand the drop-down arrow next to **All Attributes** and scroll quite a bit down to find the `layout_gravity` attribute.

Tip: We are looking for `layout_gravity` and not `gravity`.

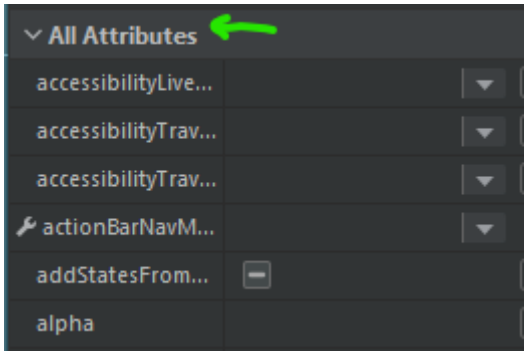


Figure 51. Expanding All Attributes

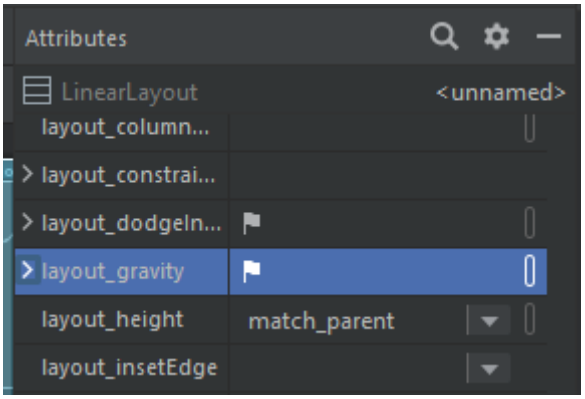


Figure 52. Setting the layout_gravity

Click the flag next to `layout_gravity`.

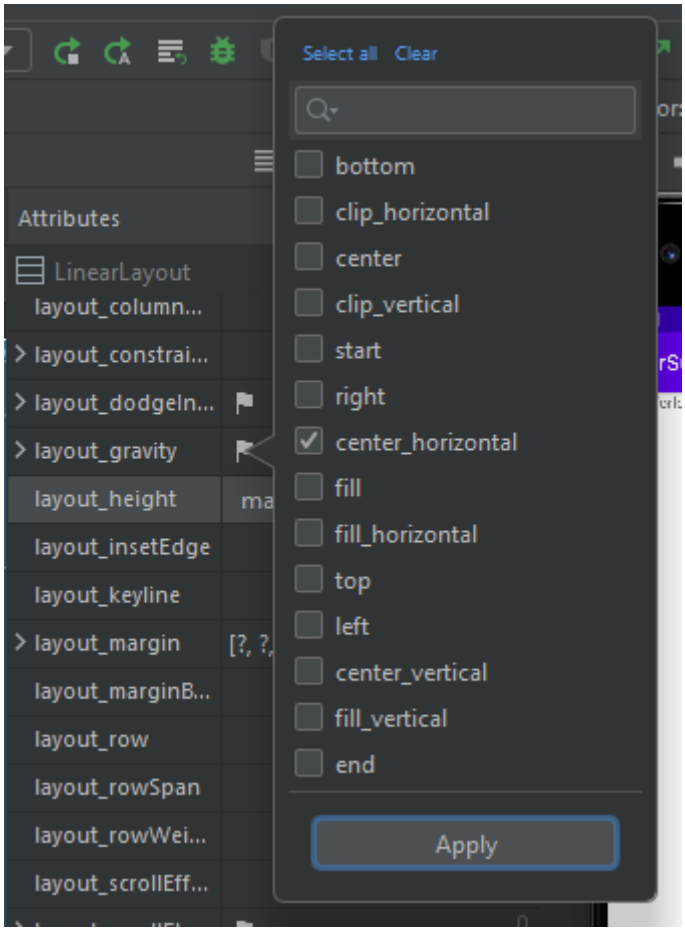


Figure 53. Setting center_horizontal

Check the `center_horizontal` check box and click **Apply**.

Now the image is where we wanted it to be (see Figure 54).

Figure 54. ImageView Position

Delete the `TextView` – we won't be using it.

To add the `ImageView` using the Code (XML) view:

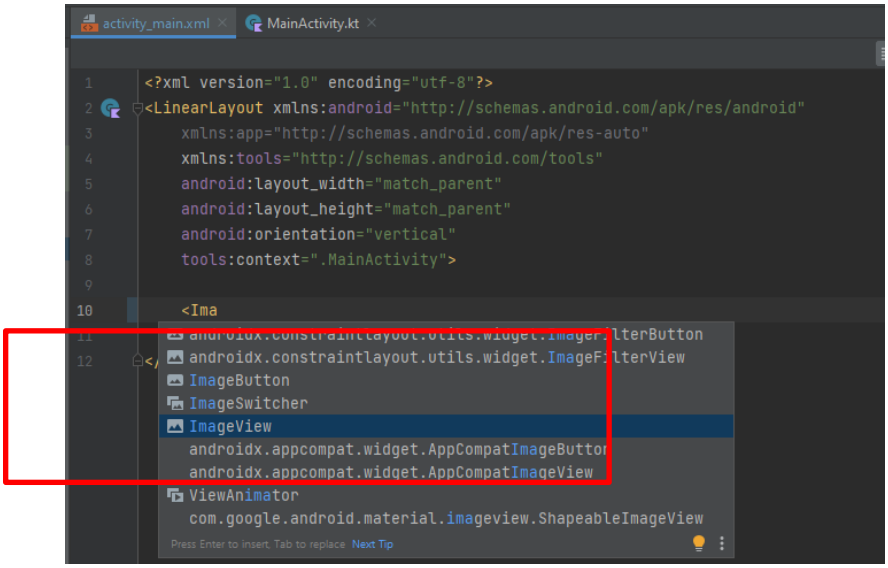


Figure 55. Inserting an ImageView

1. Open an XML tag by typing **<** and then start typing `ImageView`. IntelliSense should pop-up and you can simply press Enter on your keyboard to accept the suggestion.

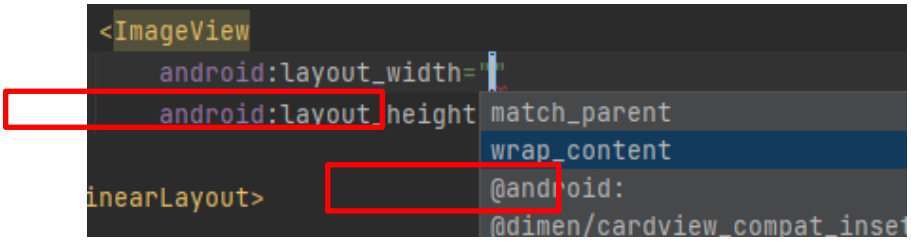


Figure 56. Adding `layout_width` and `layout_height`

2. Your `ImageView` will be inserted and Android Studio will prompt you to add the `layout_width` and `layout_height` as discussed above. You can click on `wrap_content` for now. `wrap_content` means the `ImageView` will wrap itself around the actual size of the image.

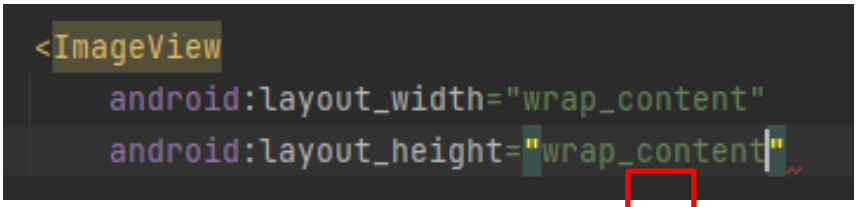


Figure 57. Syntax Error

3. You will notice that the IDE is displaying a small error by showing a red squiggly line behind the `wrap_content` entry for the layout height. That is because we need to close the XML tag.

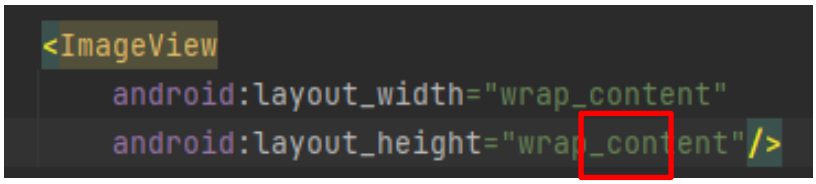


Figure 58. Closing the Tag

4. Just enter a forward slash (/) and Android will autocomplete the closing of the ImageView tag for you.
5. We next need to tell Android Studio where to find our image. We do this with the src attribute. The full attribute is:

```
android:src="@drawable/starsuckslogo"
```

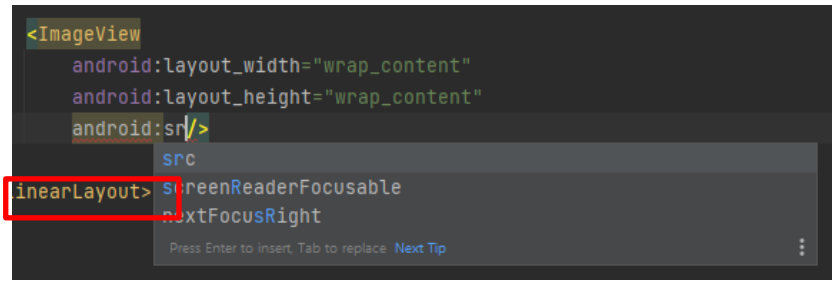


Figure 59. Adding the src Attribute

6. There should be IntelliSense popups that will assist you in adding in this entry quite easily. Add the `src` attribute.

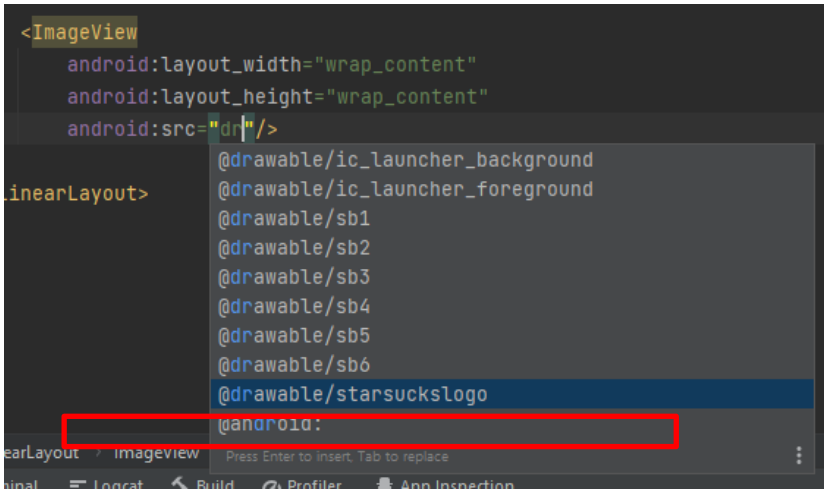


Figure 60. Choosing the Image

7. Choose the correct image from the correct resources folder (drawable).

8. Next, we need to centre our `ImageView`. The full XML attribute is:

```
android:layout_gravity="center_horizontal"
```

```
<ImageView
    android:id="@+id/starsuckslogo"
    android:layout_width="232dp"
    android:layout_height="152dp"
    android:src="@drawable/starsuckslogo"
    android:layout_gravity="center_horizontal"/>
```

Figure 61. Adding the Height and Width

9. We lastly need to change the height and width so that our image is the correct size compared to the screen. (See Figure 61.)

Let's have a look if the `ImageView` displays correctly. Run the app in the emulator. If you look to the bottom of your screen, you will see the **Layout Inspector** window in its minimised state. Click on it to display a window that shows what our layout looks like right now, with details as in the running app.

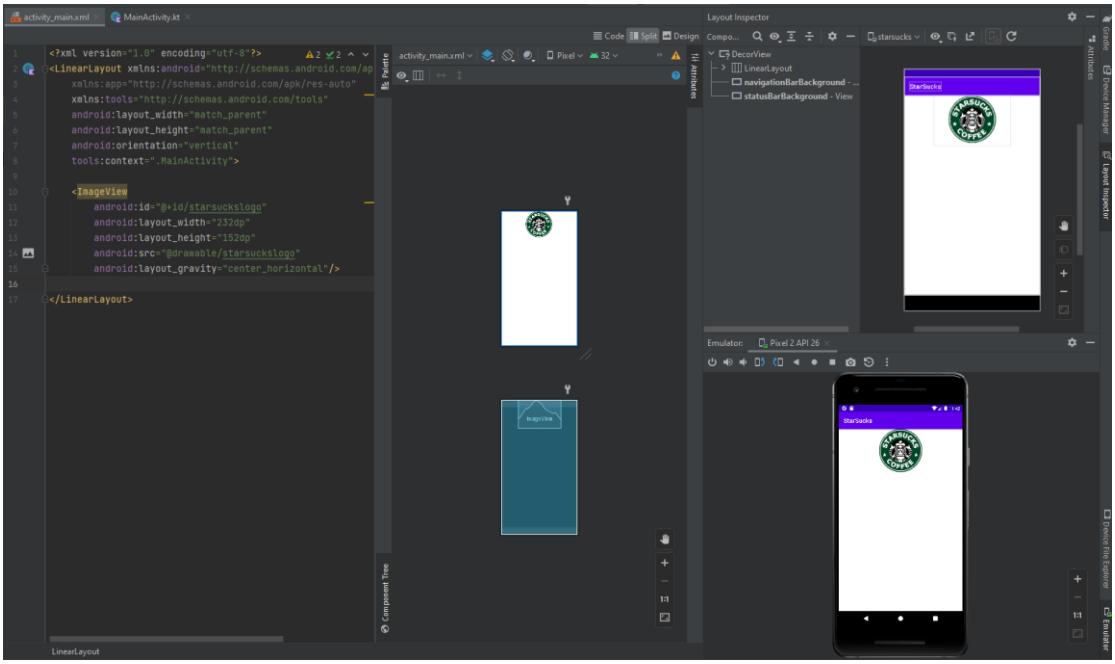


Figure 62. Layout Inspector Displayed

Everything is looking good so far. So, let's move on.

4.4.3 Adding a TextView

A `TextView` holds text. The text in your `TextView` can be set programmatically and using the Layout Editor or XML. Developers need to reuse as much as they can as well as change the look feel and information of applications quite quickly to remain competitive. There are shortcuts and easier ways to work String values and Colours in Android Studio. These however come with a cost later in time. It is worth our while to store our Strings and Colours in places where we can easily change them when necessary. We will cover adding a `TextView` as well as how to store our Strings and Colours in the `values` folder under the `res` folder. We will add our `TextView` and then we will add a String value in the correct colour.

You can follow the exact same process we followed to add a UI element (View) as we did for the `ImageView`. You can drag and drop it onto your preview in your Layout Editor or you can add the button using XML. The choice is yours. The truth is that you will probably jump between the Layout Editor and XML as you go along. I will focus on XML for the remainder of this module and jump to the Layout Editor where it is just easier to do. You can follow any pattern that works for you. Every attribute that is available in XML will also be available in the Layout Editor.

Let's start with the Layout Editor. You can find the `TextView` in the Palette, just as we did with the `ImageView`. You can now drag and drop this onto your preview.

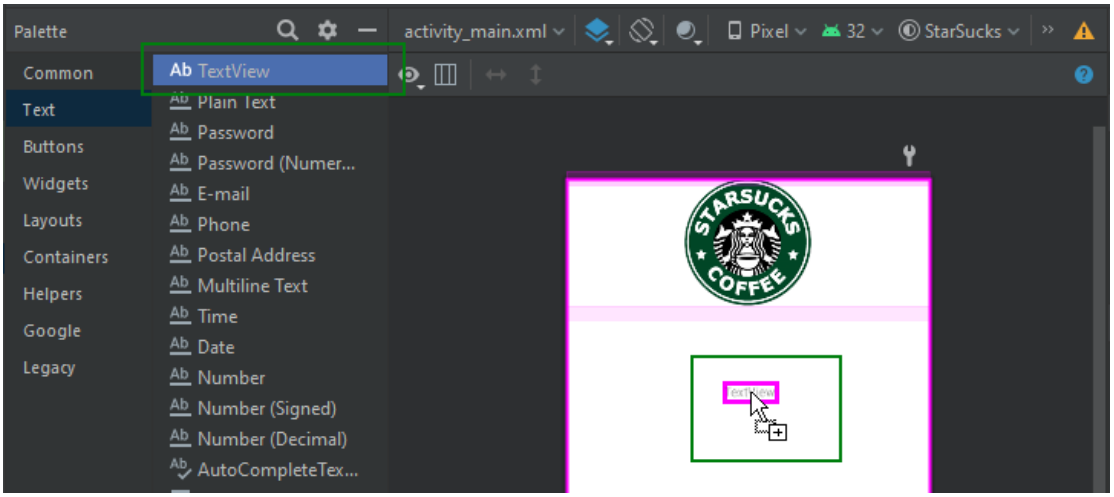


Figure 63. Drag and Drop TextView

Remember the Layout Editor will apply a default `layout_width` and `layout_height` attribute. In this case it chose `match_parent` for the width and `wrap_content` for the height. Which makes a rather ugly `TextView`.

Figure 64. Alignment of the TextView

We can off course just change this with the attributes, as shown in Figure 65.

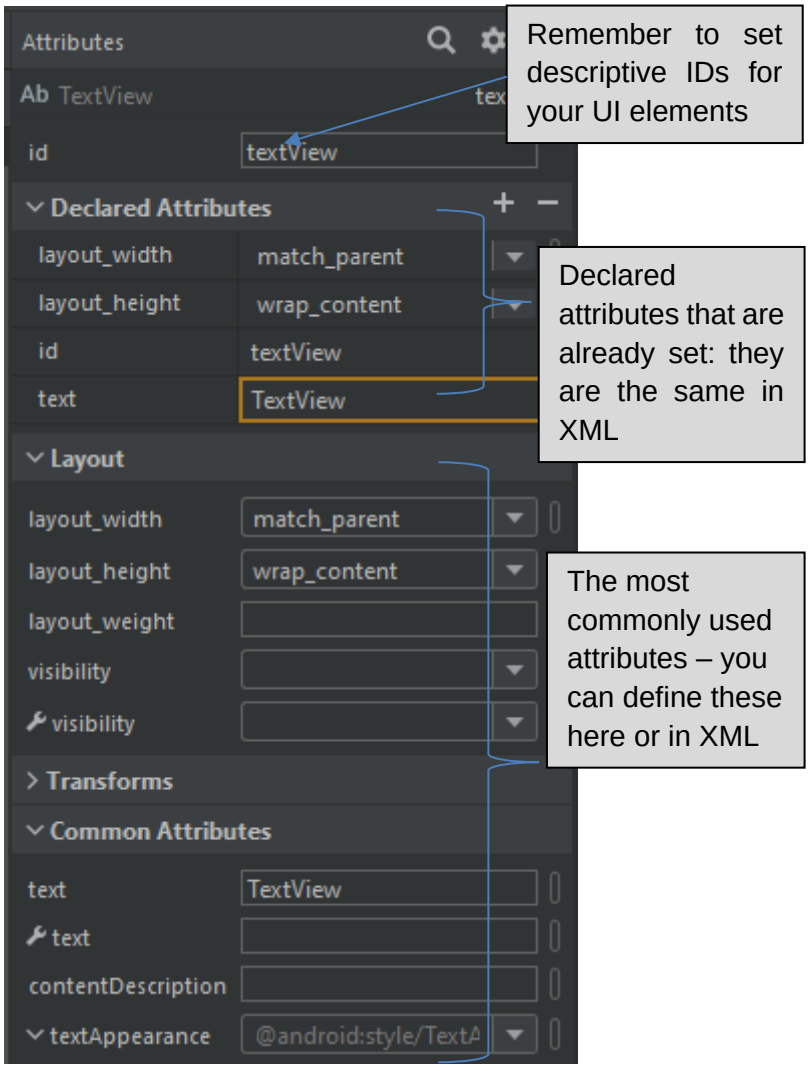


Figure 65. TextView Attributes

Which brings us to an important point – the id. Remember we spoke about the `R` class earlier and we discussed how the R class is the glue between our XML UI and our Kotlin code? That it stores a memory location usually as an int variable? We can only find our XML UI elements in our Kotlin code if we know the id. We need to set this value to a value that we will recognise – you will see why when we add some logic to this application.

The Layout Editor will set a default value as the ID, you are not that lucky in XML, you need to define your own. It is best practice to give your UI elements easy to understand descriptive names.

Here the exact same `TextView` in XML with the ID set correctly and the attributes we want to use defined.

```
17    <TextView
18        android:id="@+id/textView"
19        android:layout_width="wrap_content"
20        android:layout_height="wrap_content"
21        android:layout_gravity="center_horizontal"
22        android:textSize="30sp"
23        android:text="TextView" />
```

Figure 66. TextView in XML

If we run the app window, we see that the `TextView` is where we want it and the text is the correct size.

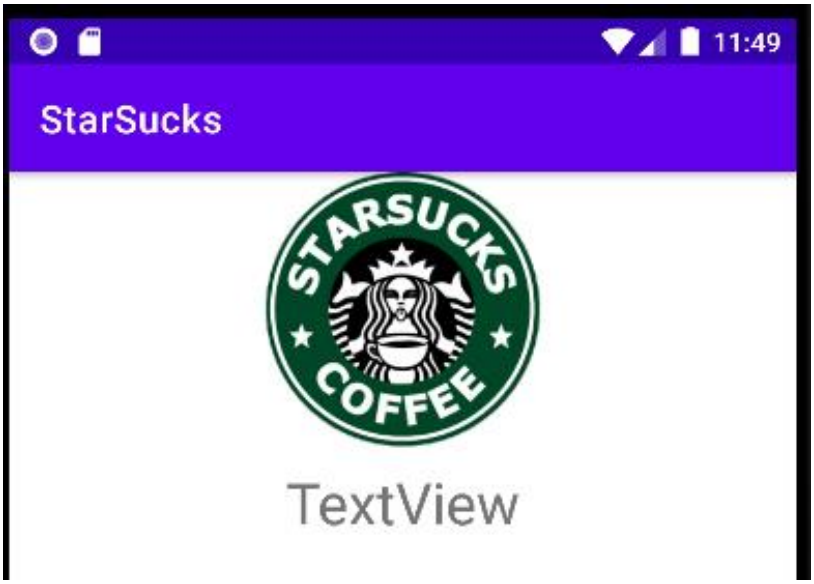


Figure 67. App with TextView

It is bad practice to hardcode the text that is displayed in the `TextView` in the XML file. What if you have used this value 10 000 times, and then the customer decides it needs to change? A better way is to define the value in strings.xml and reuse that everywhere. Then you can make the change in one place. The same is true for defining colours in color.xml too.

4.4.4 Strings.xml (Setting String values to easily maintain or modify them)

There is a folder called `values` under your `res` (resources) folder. If you expand it, you will see that it has three .xml files as shown in Figure 68.

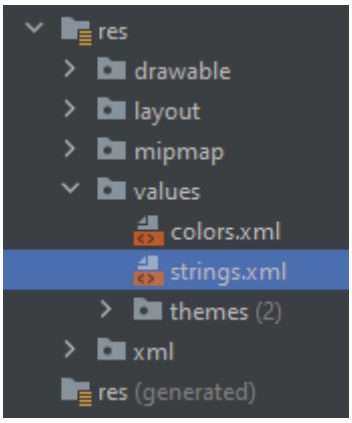


Figure 68. Files in the values folder

These are used to store the colours, string values and styles of your app in one central place. We can define our values here and then make a single change that could take effect in multiple places in our application.

For example, if we used the string “StarSucks originator of the mochaccino” on every single activity of our app. Let’s say we end up being sued by Luigi who is the originator of the mochaccino, and we need to chance it to “StarSucks originator of the chocciechino” if we ever want to sell coffee again. We could change this value instantly if we stored in strings.xml. Not so much if we hard coded it into every `TextView`…

Open the strings.xml file. The first string was created by Android. It is your apps name and used in your Manifest file (more on this later). Add the second entry to use as the string value for our `TextView.`

```
1  <resources>
2      <string name="app_name">StarSucks</string>
3      <string name="order_now_label">Order Now</string>
4  </resources>
```

Figure 69. Adding a New String Resource

We can now use this value in our XML. Instead of the literal value TextView, start typing an @ (see Figure 70). From the autocomplete pop-up, select our new string resource.

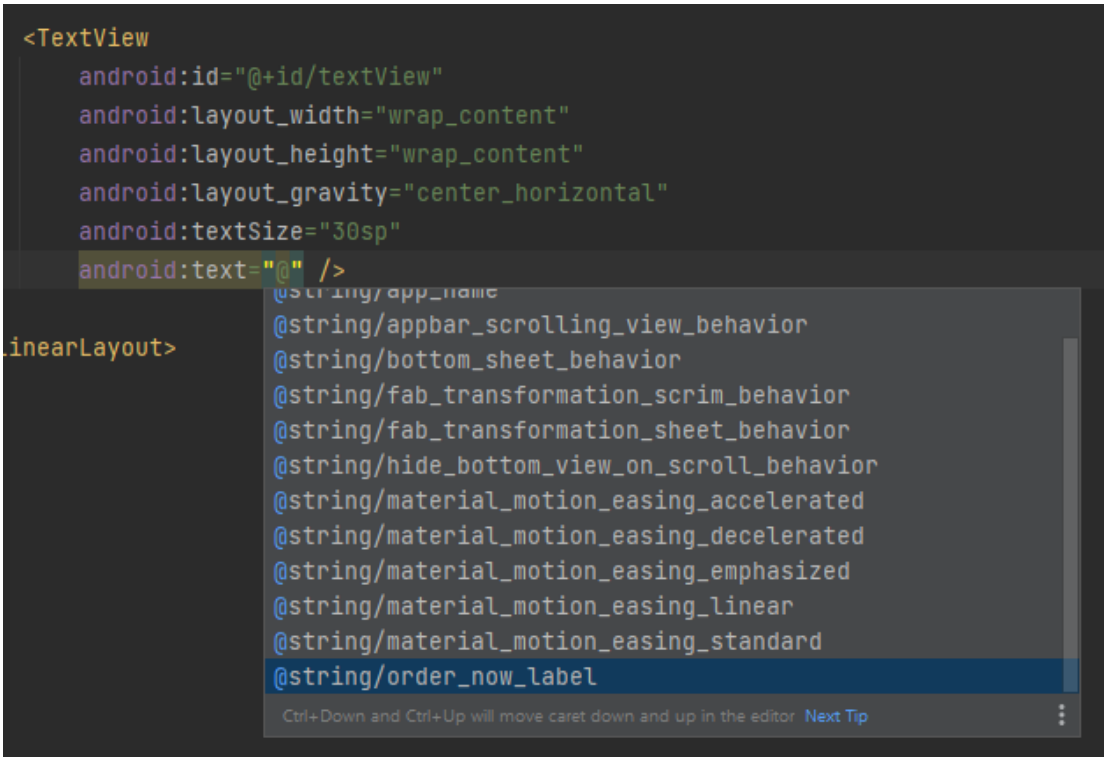


Figure 70. Using the String Resource

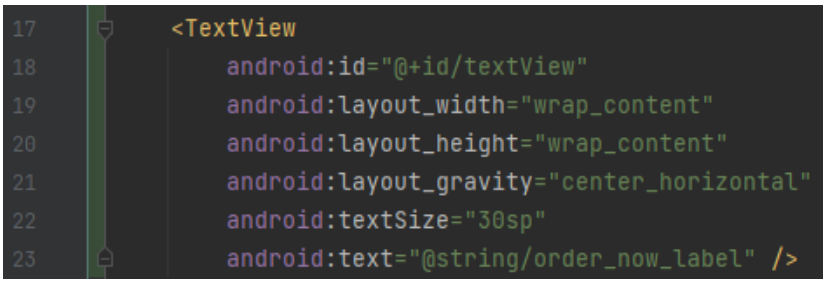


Figure 71. String Resource Now Used

Whenever we now need these same “Order Now” words in the app, we can make use again of this same string.

4.4.5 Colors.xml (Setting and easily maintaining of modifying app colours)

Customers change the look and feel of their brands and products to stay relevant. You as a developer will often have to update your application’s look and feel to keep up with logo and slogan changes etc. from your customers. Figure 72 shows how dramatically different some well-known brands looked before their rebranding.

Rebranding often involves entire new colour palettes too. It would be wise to have our colours stored so that they are easy to change.

Figure 72. Branding Changes (TFPP Writer, 2018)

4.4.5.1 Design and Colour Palettes

Applications need good user interfaces that are easy to use and provide a good user experience. It does not matter how brilliant the code behind is. If the interface is not intuitive, lovely to look at and pleasing to use – chances are that the application will fail. It is important to carefully consider the colours and placement of UI elements. Let's start with colour palettes. For those of us that are not trained designers, there are beautiful free colour palettes online that we can use.

Have a look at https://colorhunt.co/palettes. You can find predefined colour palettes on this site to use. [Accessed 17 November 2022].

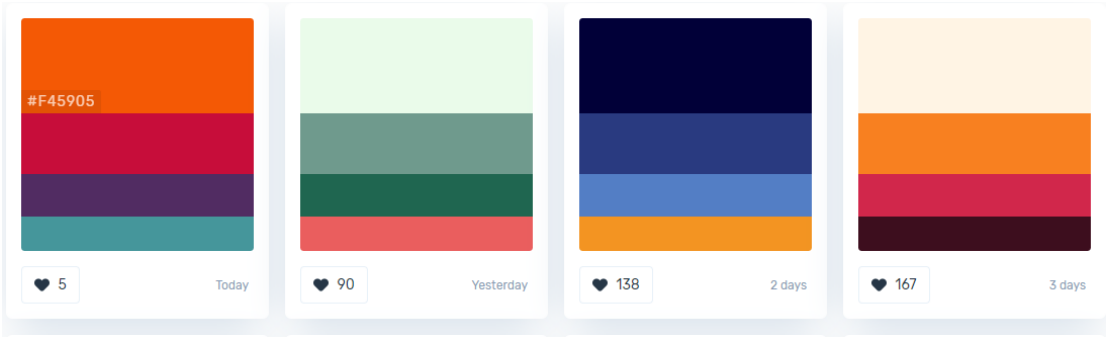


Figure 73. Palettes from https://colorhunt.co/palettes

Hovering over any of the colours will provide a hex value that you can use in colours resource file.

Figure 74. Hex Value for a Colour

4.4.5.2 Mobile Design Principles and Google Materials Design

Colours and the correct use of colours go a long way to assist with designing applications that are both pretty to look at and possibly easy to use. Let's start with Materials Design. Google provides the Materials Design website (available at https://material.io/design) that deals with everything from icons to colours to navigation. The good part about this is that Google has hired experts to provide developers with the know how so that we can create good applications.

For example – if we look at the following colour palette for the Google site, we see the colour palette as well as how to apply it to an application.

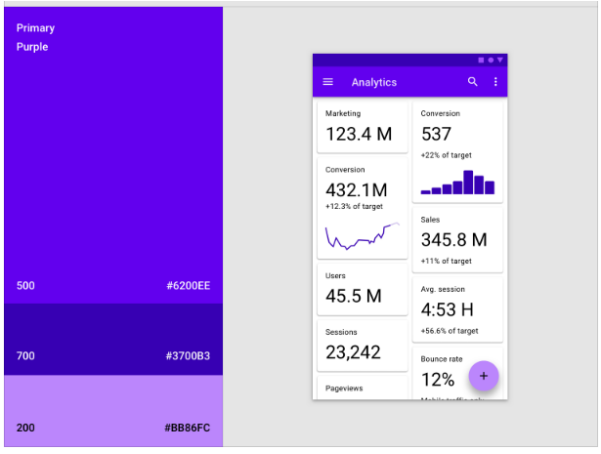


Figure 75. Applying Colours (Google, n.d.)

You will also find best practices as to how one should code up the navigation through your application. Please take the time to refer to this website.

Lastly, we need to carefully consider the placement of our UI elements so that our app is comfortable and easy to use. Look at the image below:

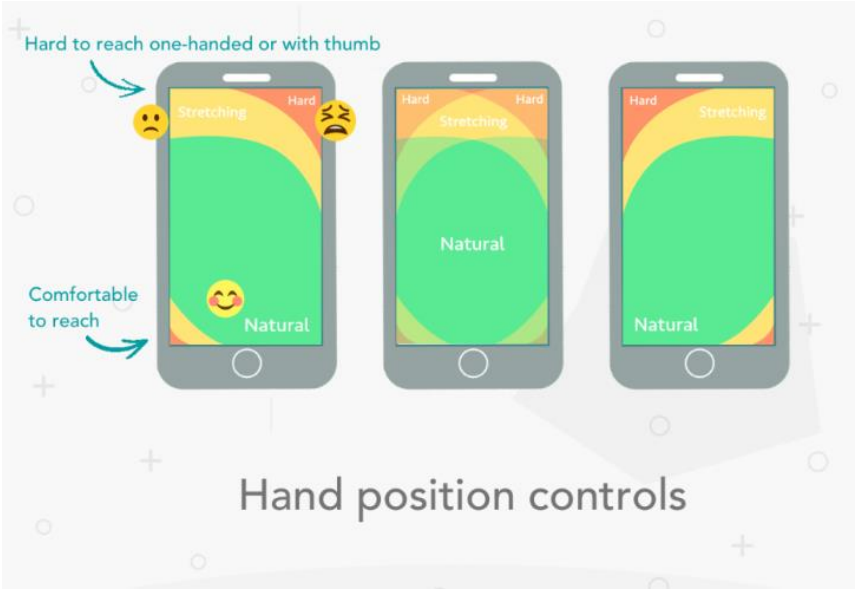


Figure 76. Hand Positions for Controls (Abu Experience, 2017)

UI and UX design have become an incredibly important part of software engineering and can no longer be ignored. The websites below discuss fantastic design principles that any app developer should try to adhere to:

Abu Experience, 2017. *10 Mobile UX Design Principles You Should Know.* [Online] Available at: http://uxbert.com/10-mobile-ux-design-principles/ [Accessed 17 November 2022].

Creative Bloq, 2012. *The 10 principles of mobile interface design.* [Online] Available at: https://www.creativebloq.com/ mobile/10-principles-mobile-interface-design-4122910 [Accessed 17 November 2022].

4.4.5.3 Adding Colours to the color.xml File

Now that we have looked at good use of colours and design principles, we can finally add some colours to our colors.xml file and use them.

We can “borrow” the Starbucks colours which you can find on the following website: https://usbrandcolors.com/starbucks-colors/ [Accessed 17 November 2022].

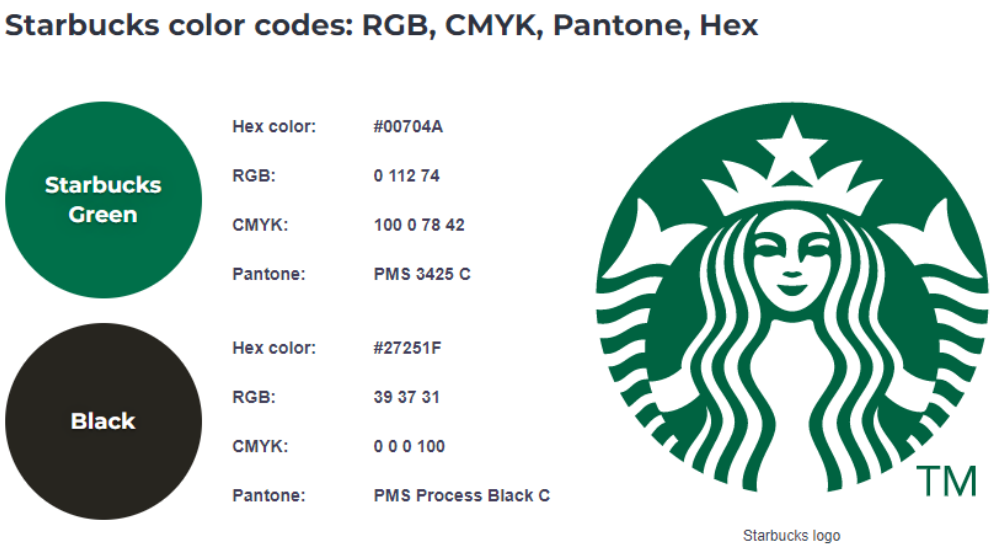


Figure 77. Starbucks Colours (U.S. Brand Colours, n.d.)

For easy access, the hex colour is **#00704A**

Open your colors.xml file and add the hex value to this file. You will notice that there are some default android colours available already.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3      <color name="purple_200">#FFBB86FC</color>
4      <color name="purple_500">#FF6200EE</color>
5      <color name="purple_700">#FF3700B3</color>
6      <color name="teal_200">#FF03DAC5</color>
7      <color name="teal_700">#FF018786</color>
8      <color name="black">#FF000000</color>
9      <color name="white">#FFFFFFFF</color>
10 </resources>
```

Figure 78. Colours Already in colors.xml

Add the starsucksGreen colour as illustrated

```
10     <color name="starsucksGreen">#00704A</color>
```

Figure 79. Adding the New Colour

You will note that the colour appears in the margin. You can edit the colour here by clicking on the block and using the slider as shown in Figure 80.

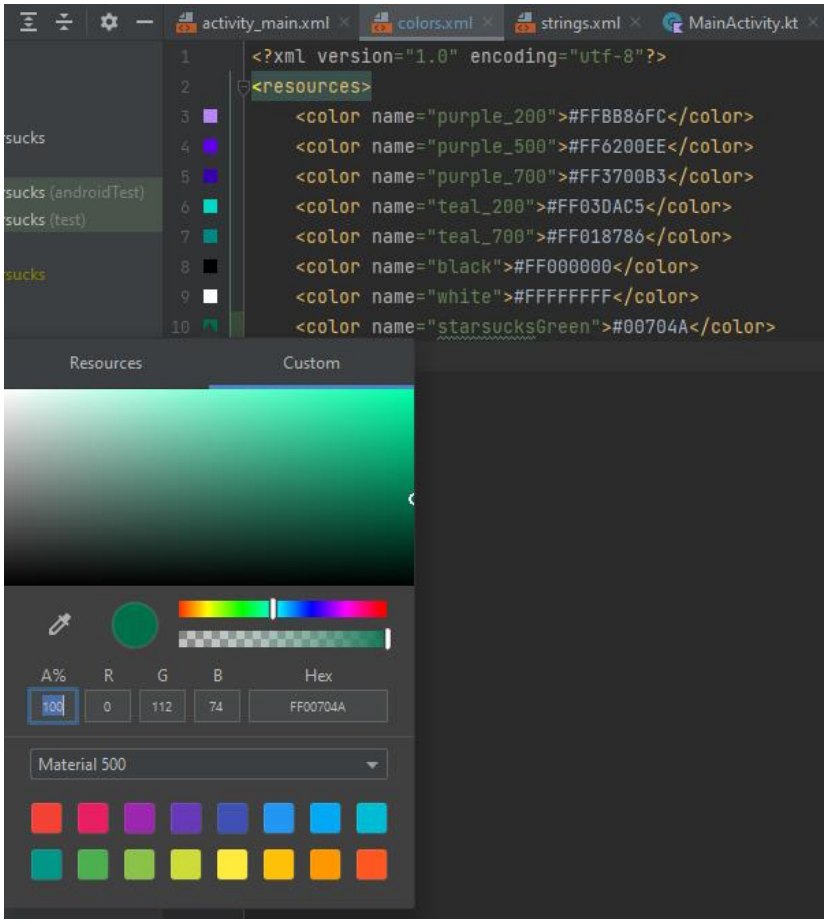


Figure 80. Editing the Colours

This makes it quite easy to add your own colours – just copy over a colour, change the name, and edit it using the colour mixer.

You can now apply the colour to the text in the edit text as shown in Figure 81.

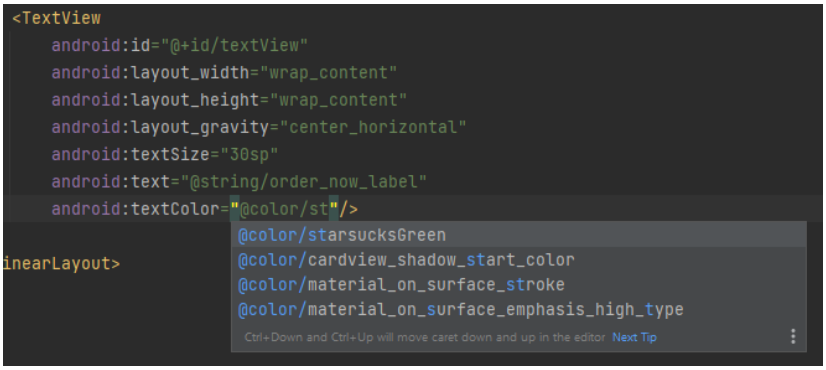


Figure 81. Adding the textColor Attribute

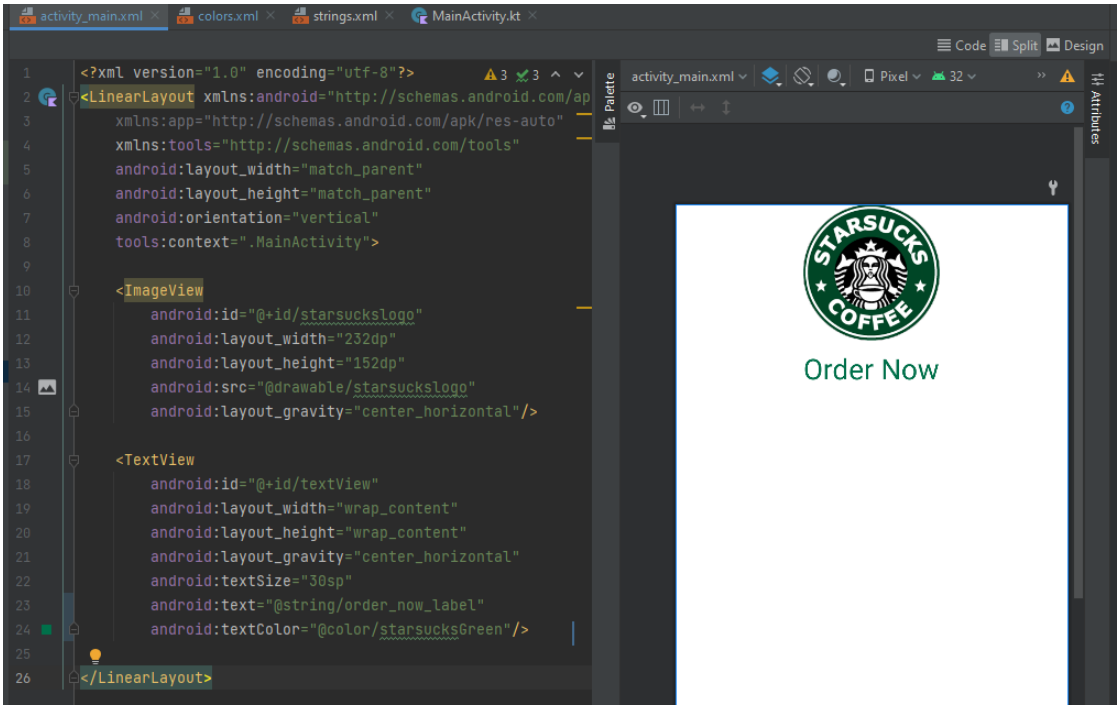


Figure 82. Text Colour Now Applied to the TextView

Our text is now Starbucks green but still fairly simple. We will change that next by adding a font.

4.4.6 Adding Fonts to your Project

We will be using the Layout Editor to add our fonts. It is easier and quicker to do it here and not in XML. Click on the `TextView` and search for the `fontFamily` attribute.

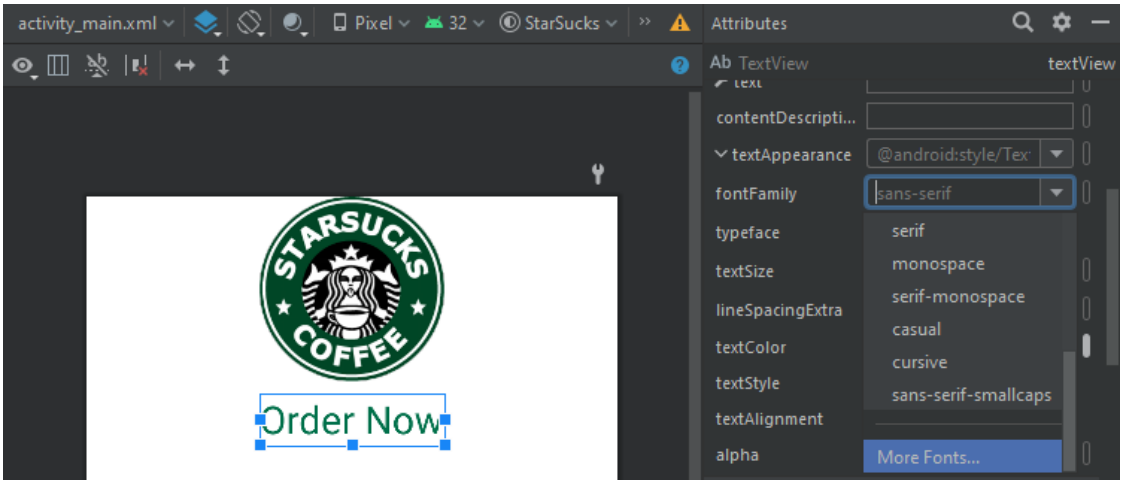


Figure 83. Choosing a Font

Click **More Fonts** to get access to the Google fonts. (Scroll down to see that option.)

These fonts are open source and easy to use.

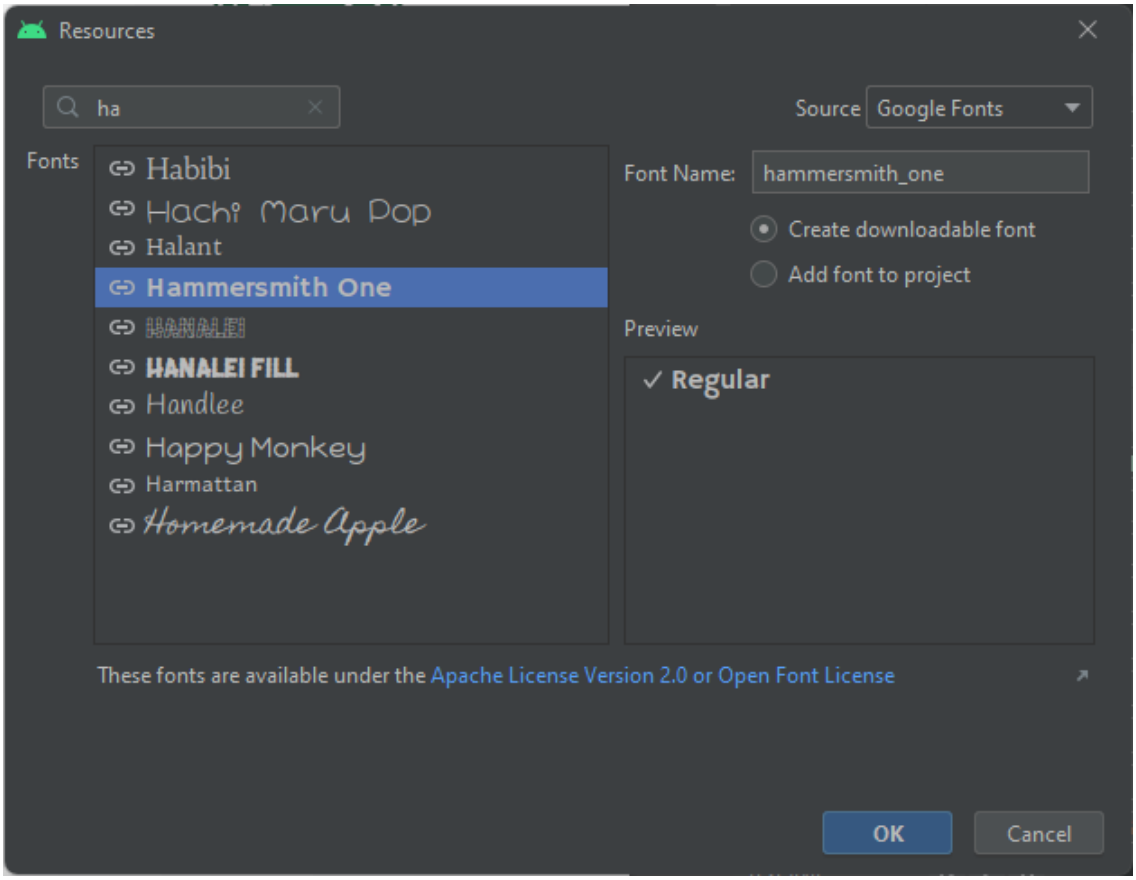


Figure 84. Choosing a Google Fonts Font

Select any font you like from the list. You will notice that there are two options when you select a font, these are:

- Create downloadable font
- Add font to project

The downloadable font might not work if there is no internet connection as the font needs to be downloaded. If you add the font to the project it will always be available, but this creates large *apk* (Android install) files.

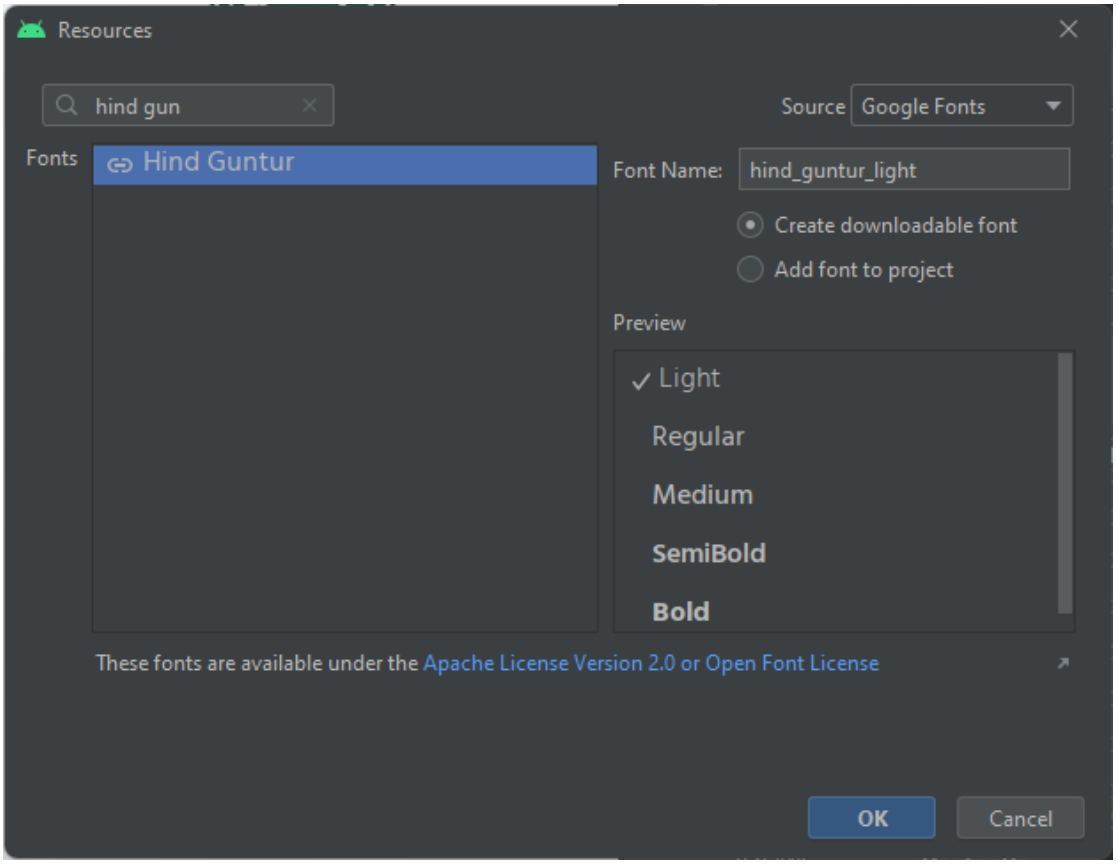


Figure 85. Selecting a Downloadable Font

Android Studio will automatically create the font files in the res folder.

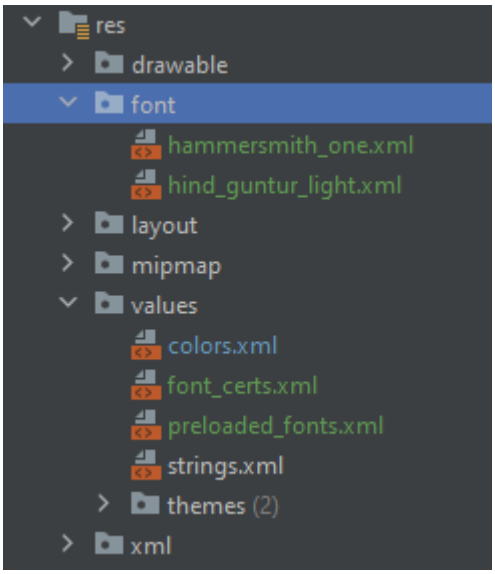


Figure 86. Fonts Added

You can learn more about fonts here:
https://developer.android.com/guide/topics/ui/look-and-feel/fonts-in-xml [Accessed 17 November 2022].

The XML for our TextView is shown in Figure 87.

```
17    <TextView
18        android:id="@+id/textView"
19        android:layout_width="wrap_content"
20        android:layout_height="wrap_content"
21        android:layout_gravity="center_horizontal"
22        android:fontFamily="@font/hind_guntur_light"
23        android:text="@string/order_now_label"
24        android:textColor="@color/starsucksGreen"
25        android:textSize="30sp" />
```

Figure 87. TextView XML with Font

And the app is shown in Figure 88.

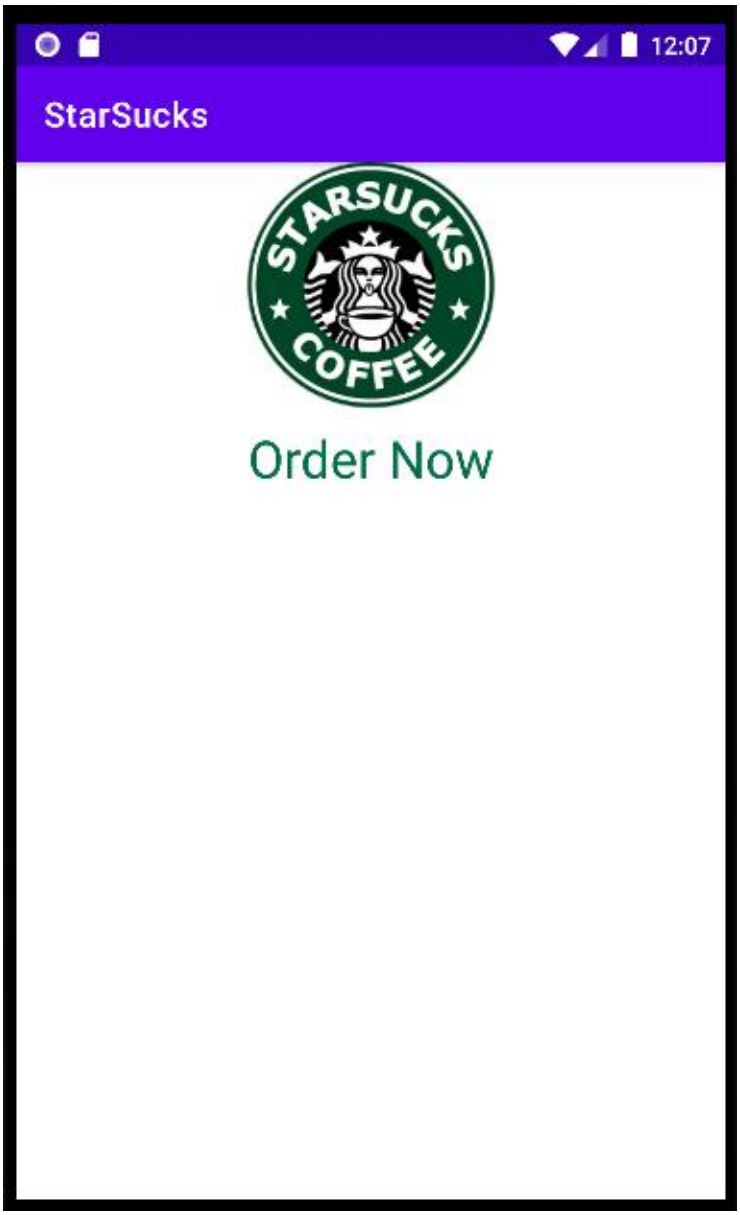


Figure 88. App running with font

5 Running an App

Now we can run the app in the emulator again to see what it looks like. And so far, it looks good, even though it doesn't do much yet.

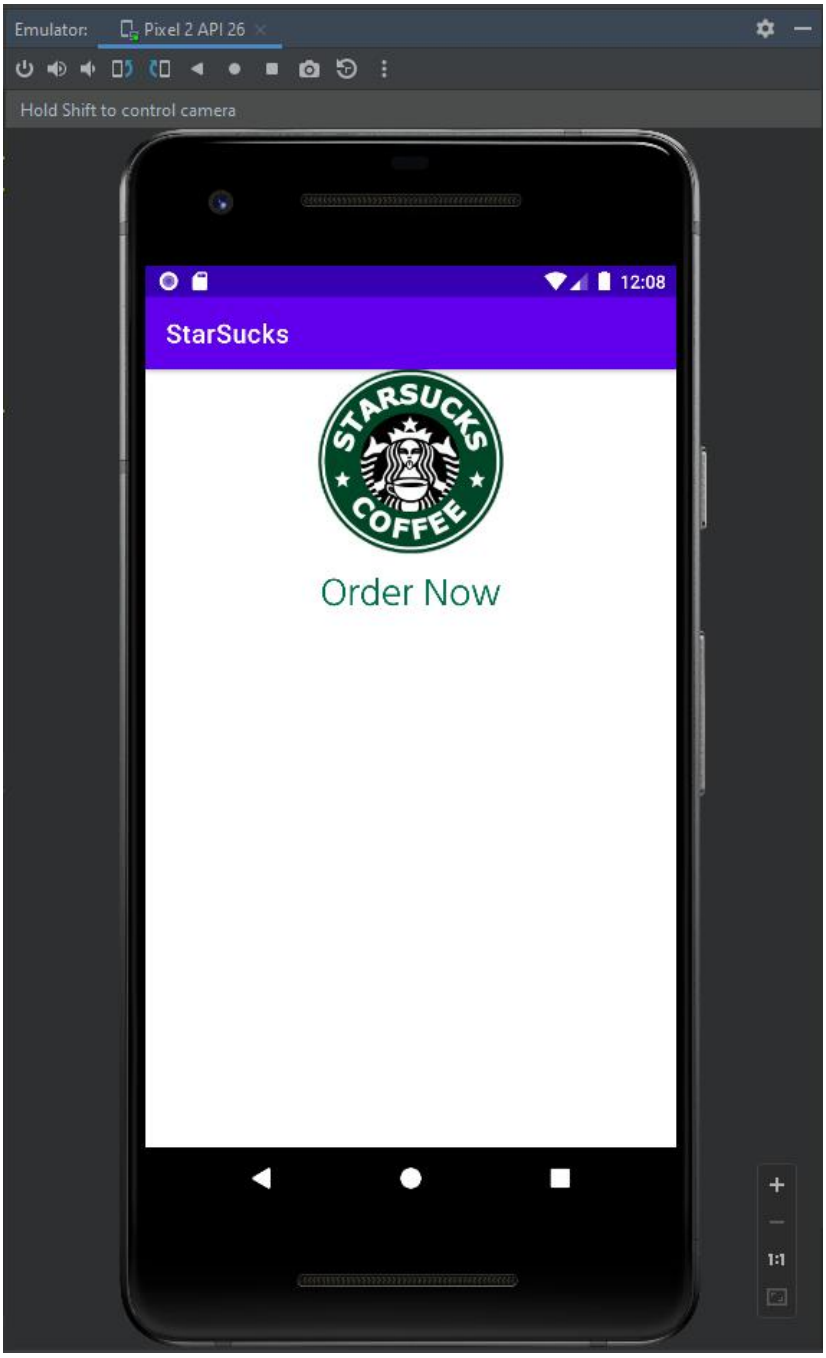


Figure 89. Our App Running in the Emulator

Let's look at what happens behind the scenes when the app is built and executed.

Gradle is the build system that creates your Android Package Kit (APK). The APK is what is installed on your emulator or phone when you run your app. Gradle takes all your XML files in the `res` folder as well as all your Java files in the `src` folder as well

as any dependencies that you have added to your project (you will see later in Firebase) and runs a script to build your APK.

A script is a coding file that automates processes – in this case creating the APK. It is completely possible to create your own build script. You could use a language such as Groovy or bash scripting to do so. For now, we will rely on Gradle.

The Gradle configuration is made up of several files which can be divided into two categories:

- Top level `build.gradle`
- Module level `build.gradle`.

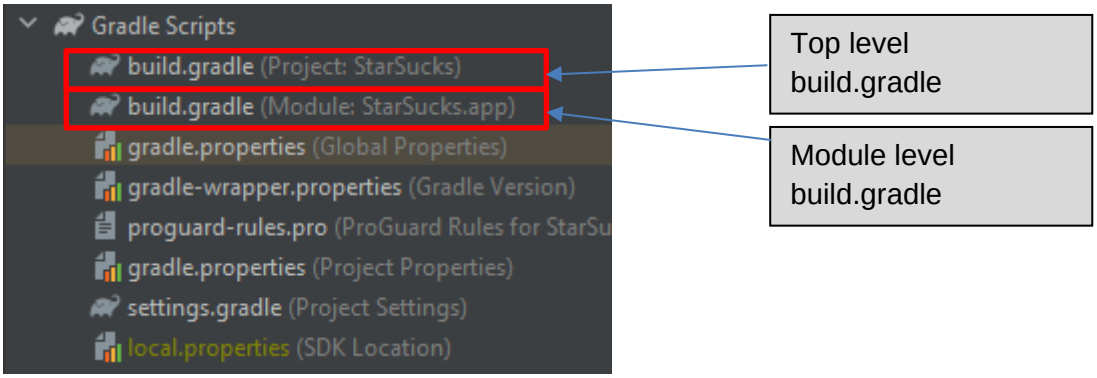


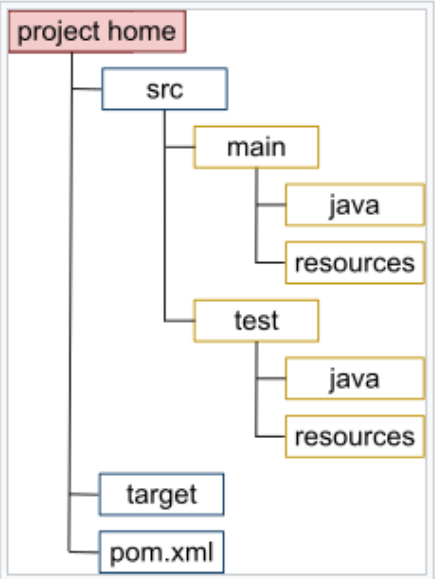
Figure 90. Gradle Files

The top level Gradle script contains the settings for your entire project and the module layer Gradle script contains the settings for just this application.

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.
plugins {
    id 'com.android.application' version '7.2.2' apply false
    id 'com.android.library' version '7.2.2' apply false
    id 'org.jetbrains.kotlin.android' version '1.6.10' apply false
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Figure 91. Top-level Gradle Script

You will notice that Android uses *Maven* for libraries. *Maven* is repository that automates a lot of the build process for us. Including creating the folder structure below. You should be familiar with it by now.



A directory structure for a Java project auto-generated by Maven

Directory name	Purpose
project home	Contains the pom.xml and all subdirectories.
src/main/java	Contains the deliverable Java sourcecode for the project.
src/main/resources	Contains the deliverable resources for the project, such as property files.
src/test/java	Contains the testing Java sourcecode (JUnit or TestNG test cases, for example) for the project.
src/test/resources	Contains resources necessary for testing.

Figure 92. Maven Folder Structure

The module Gradle file contains information specific to our application from the compiled SDK version, the min SDK version, the version code of our app to the external libraries (dependencies) that we import.

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

android {
    compileSdk 32

    defaultConfig {
        applicationId "za.ac.iie.opsc7311.starsucks"
        minSdk 26
        targetSdk 32
        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:1.5.1'
    implementation 'com.google.android.material:material:1.6.1'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
}
```

Figure 93. Module Level Gradle Script

Gradle can break spectacularly – don't worry too much if this happens. You should easily find a solution if you Google the error that Gradle throws.

5.1 Running on your phone

Besides running the app in the emulator, you can also run the app on your phone if you have an Android device. You can connect using USB or even wi-fi (Android 11 or later). Read more about the setup process in (Android Open Source Project, 2022c)

When the app is just being run on the emulator, or on your phone, you can successfully use an APK file. But when we get to publishing the app (more about that in OPSC7312), apps are built as a bundle instead. It is a different format that allows the Play Store to do smart things like deliver game assets. Read more in (Android Open Source Project, 2022d).

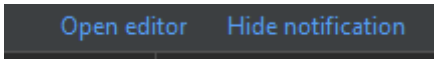
6 Internationalizing an App

We next want to make translations available to our users. Android displays applications in information based on the language settings of the device it runs on. We would like to translate the values in our app for different language settings. You start by setting the string values that you are willing to translate to `translatable="true"` for the values that you want to translate and then `translatable="false"` for the values you wish to not translate.

See https://developer.android.com/guide/topics/resources/localization?hl=en [Accessed 17 November 2022].

```
<resources>
    <string name="app_name" translatable="false">StarSucks</string>
    <string name="order_now_label" translatable="true">Order Now</string>
</resources>
```

Figure 94. Marking Strings as Translatable



Click **Open editor** (top-right corner of the editor window). This can also be accessed by right-clicking on the strings.xml file and choosing **Open Translations Editor**.

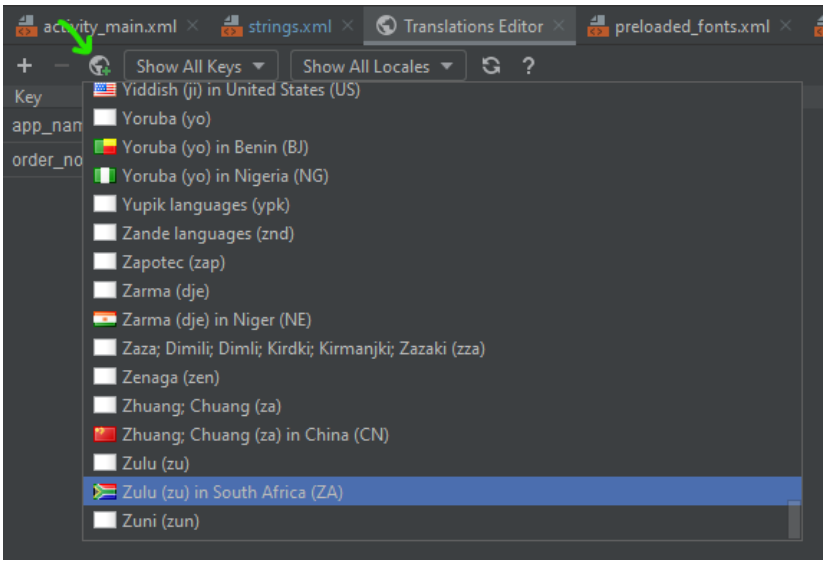


Figure 95. Adding a Locale

Click Add Locale ( ) and then select the language you would like to translate your values to.

Enter the translation in the column for the new locale.

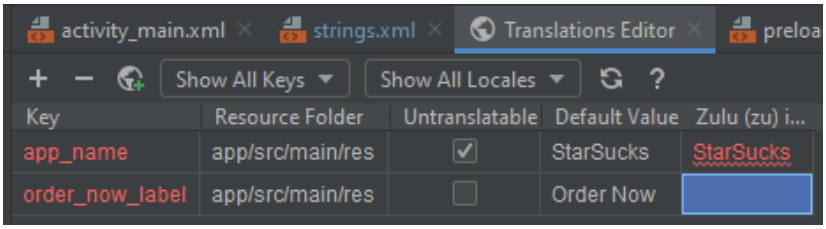


Figure 96. Enter the Translation

And that is all we need to do. The label already uses the string resource, and Android will take care of loading the resources for the right language when the app runs.

Tip: You can change the language of the Android operating system running on the emulator to see the translation in action.

7 Recommended Additional Reading

Abu Experience, 2017. *10 Mobile UX Design Principles You Should Know.* [Online] Available at: http://uxbert.com/10-mobile-ux-design-principles/ [Accessed 17 November 2022].

Android Open Source Project, 2020g. *Layouts.* [Online] Available at: https://developer.android.com/guide/topics/ui/declaring-layout [Accessed 17 November 2022].

Creative Bloq, 2012. *The 10 principles of mobile interface design.* [Online] Available at: https://www.creativebloq.com/ mobile/10-principles-mobile-interface-design-4122910 [Accessed 17 November 2022].

Lake, I., 2016. *Layouts, Attributes, and you.* [Online] Available at: https://medium.com/androiddevelopers/layouts-attributes-and-you-9e5a4b4fe32c [Accessed 17 November 2022].

8 Recommended Digital Engagement and Activities

The **Guru OPSC7311** playlist on **YouTube** has a lot of useful videos created for this module. The full playlist can be found here: https://www.youtube.com/playlist?list=PL480DYS-b_kdor_f0IFgS7iiEsOwxdx6w [Accessed 17 November 2022].

For this learning unit, watch the following videos from that playlist:

[YouTube] Create an Android Studio Project
https://youtu.be/WwIpOwgR6aI [Accessed 17 November 2022].

[YouTube] Android Studio Features
https://youtu.be/YtjRZhKFYvY [Accessed 17 November 2022].

[YouTube] Additional Android Studio Features
https://youtu.be/Ibqkzv9Vp7g [Accessed 17 November 2022].

[YouTube] Introduction to UI layouts and adding an image to the drawable folder
https://youtu.be/7xy2_qVK5b0 [Accessed 17 November 2022].

[YouTube] Understanding Code and Design view as well as ViewGroups
https://youtu.be/FKADfBdNLns [Accessed 17 November 2022].

[YouTube] Add Image to Constraint and Linear Layout
https://youtu.be/Qy5ZZyiyjRI [Accessed 17 November 2022].

[YouTube] Add a Textview and work with Strings Colors and Fonts
https://youtu.be/zJU0UmOeTck [Accessed 17 November 2022].

[YouTube] Add Emulator and Run App
https://youtu.be/UcGjgeE-l2k [Accessed 17 November 2022].

[YouTube] Running app and working with colours and thumb reach
https://youtu.be/rspEeRZ6wqA [Accessed 17 November 2022].

9 Activities

Do the activities that appear on Learn.

10 Revision Exercises

Create your own application that displays a list of funny cat pictures (or any kind of *meme* that you enjoy). Make sure you include a caption (using a `TextView`) for each image.

11 Solutions to Revision Exercises

Compare your solution to the example *StarSucks* application on GitHub. Also check if all your images and `TextViews` are displaying properly.

Learning Unit 3: Introduction to Kotlin	

Learning Objectives:
• Differentiate between Kotlin and Java or C#.
• Use Kotlin to write a basic program with variables and calculations.
• Explain the following object-oriented programming concepts:
  ○ Inheritance;
  ○ encapsulation;
  ○ polymorphism.
• Explain object-oriented programming in Kotlin.
	My notes

Material used for this learning unit: • GitHub repository: LearningUnit3	
How to prepare for this learning unit: • Make sure that you have the GitHub source code available and that your Android Studio is up to date.	

1 Introduction

In the first two learning units, we have only really looked at using Android Studio to build the user interface of our app. This is of course a very important part of an app, since the user interface affects very directly the user experience of the app. But a user interface without code behind it that handles the logic can only go so far.

In this module, we are using the Kotlin programming language. Unless you have done Android development before, you are not likely to have used this language before. The purpose of this learning unit is to introduce you to the basic features of the language. Whether you consider your main programming language to be Java or C#, you will see lots of familiar concepts in Kotlin.

2 Kotlin Basics

2.1 What is Kotlin?

Definition

"Kotlin is a general purpose, free, open source, statically typed 'pragmatic' programming language initially designed for the JVM (Java Virtual Machine) and Android, and combines object-oriented and functional programming features." (Heller, 2022)

Looking at this definition, we see that Kotlin was designed with Android development in mind. That is good news for use, since it means that it is particularly well suited to what we want to achieve in this module.

Another point is that the language is statically typed. This means that the types of variables are known at compile time, and the compiler will be able to catch a lot of errors before you even run the program for the first time. (Bhatnagar, 2018)

The last point that is important is that it has features from object-oriented programming as well as functional programming. You have done a lot of object-oriented programming already, so the concepts will be familiar even if the syntax is slightly different.

Kotlin has great documentation that describes all the features of the language. If you want to dive straight into the docs, start here: https://kotlinlang.org/docs/basic-syntax.html [Accessed 17 November 2022].

There is also quite a useful tool for learning Kotlin - the Kotlin Playground. It is an online environment where you can write and run basic Kotlin programs without having install anything. We will do most of our coding for this module in Android Studio, of course. But for the purpose of writing simple Hello World kind of programs, the Kotlin Playground is useful. You can access it here: https://play.kotlinlang.org/ [Accessed 17 November 2022].

2.2 Basic Syntax

All programming languages have their quirks that developers don't appreciate. You have probably had your fair share of compiler errors complaining about semicolons (;) before in Java and/or C#, for example. One of the obvious ways to spot Kotlin code is the absence of semicolons. If you space the code in an unambiguous way, you can leave out almost all semicolons!

Figure 97. Hello World in Kotlin (kotlinlang, n.d.)

The code will compile with semi-colons too, but that isn't considered good style in Kotlin.

Just like Java, Kotlin has the concept of a package that contains related classes (called a namespace in C#). And just like Java it uses the import keyword to make use of a package (like using in C#). (Kotlin Foundation, 2022)

The starting point for a Kotlin program is a function called main. An example of that is shown in Figure 97. This is the first place where we encounter a functional aspect of the language. You can have a function that exists outside of a class.

2.3 Declaring Variables

Two keywords are used in Kotlin to declare variables: val (can assign a value only once), and var (a normal variable). (Kotlin Foundation, 2022)

You will recall that we said that Kotlin is statically typed. So, the types of the variables are known at compile time. However, that doesn't mean that you need to specify the type – the compiler can infer the type if you assign a value.

```
fun main() {
    val favouriteNumber = 14
    println("Hello, OPSC7311!")
    println("My favourite number is $favouriteNumber")
}

Hello, OPSC7311!
My favourite number is 14
```

Figure 98. Declaring and printing a variable

In Figure 98, we have a very small Kotlin program. It declares a variable called favouriteNumber and sets the value to 14. Because the value is assigned, the compiler infers that the variable must be an integer.

```
fun main() {
    val favouriteNumber: Int = 14
    println("Hello, OPSC7311!")
    println("My favourite number is $favouriteNumber")
}
```

Figure 99. Adding an explicit type

If we wanted to, we could also specify the type of the variable explicitly when it is declared, like the example in Figure 99.

```
fun main() {
    val favouriteNumber: Int = 14
    favouriteNumber++
    println("Hello, OPSC7311!")
    println("My favourite number is $favouriteNumber")
}

Val cannot be reassigned
```

Figure 100. Trying to change the value of a val

Recall that if a variable is declared with the val keyword, the value can only be assigned once. If we try to change the value later, we get an error at runtime indicate that we cannot do this. So, let us change the declaration to var instead.


```
fun main() {
    var favouriteNumber: Int = 14
    favouriteNumber++
    println("Hello, OPSC7311!")
    println("My favourite number is $favouriteNumber")
}
```

```
Hello, OPSC7311!
My favourite number is 15
```

Figure 101. Incrementing a var

Yep, that works as expected. All the normal arithmetic operators like +, -, *, / work just like they would in Java and C#.

2.4 Conditional Statements

A normal if statement in Kotlin works pretty much like you would expect from Java and C#. Let us look at an example.

```
fun main() {
    var favouriteNumber: Int = 14
    favouriteNumber++
    println("Hello, OPSC7311!")
    if (favouriteNumber > 10) {
        println("My favourite number has at least two digits")
    }
    else {
        println("My favourite number has just one digit!")
    }
    println("My favourite number is $favouriteNumber")
}
```

```
Hello, OPSC7311!
My favourite number has at least two digits
My favourite number is 15
```

Figure 102. If statement

No surprises there. We see the curly brackets around the if and else blocks just like in Java and C#.
There is no ternary operator (?:) in Kotlin, because you can use an if instead. (Kotlin Foundation, 2022b) The syntax look like this:

```
fun main() {
    var favouriteNumber: Int = 14
    var display = if (favouriteNumber >= 10)
                    "Two digits" else "One digit"
    println(display)
}
```

```
Two digits
```

Figure 103. Using if as an expression

The when statement in Kotlin works a lot like switch in Java and C#. (Kotlin Foundation, 2022b) It is ideal when you have a variable that can have different values, and you need to do something different for each value.

```
fun main() {
    var favouriteNumber: Int = 14
    when (favouriteNumber) {
        0, 1 -> println("Boring... :O")
        9 -> print("Mine is 14 too! :)")
        else -> print("Who cares... :|")
    }
}
```

```
Who cares... :|
```

Figure 104. When statement

The interesting thing here is the syntax. The first line matches both 0 and 1. This illustrates the conciseness of the language.

2.5 Loops

For-loops in Kotlin can only be used for looping over something that provides an iterator (like a collection or a range). These behave a lot like a foreach loop in C# or a for-loop in Java that loops over a collection. Let us look at two simple examples.

```
fun main() {
    var total: Int = 0
    for (i in 1..5) {
        print("$i ")
        total += i
    }
    println("Total: $total")
}
```

```
1 2 3 4 5 Total: 15
```

Figure 105. For-loop adding up five first five integers

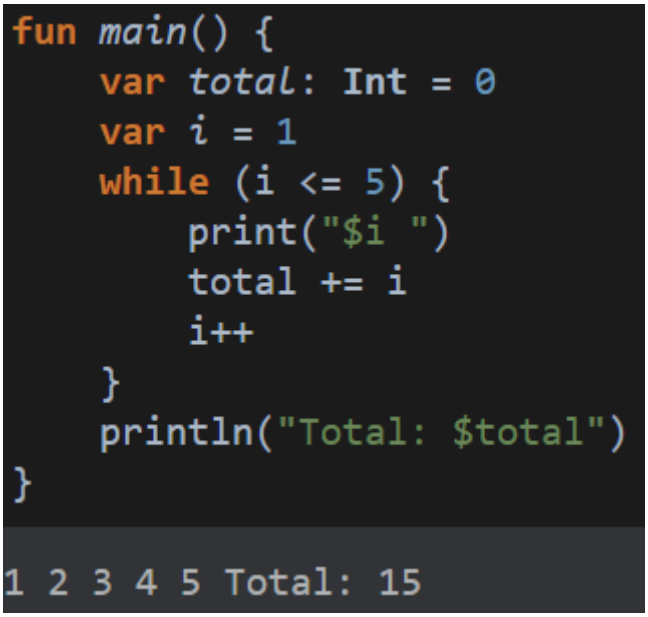


Figure 106. While-loop adding up five first five integers

The difference between the loops is that the for-loop example uses a range expression to create something the loop can iterate over. And the while-loop example has a more traditional control variable that we control ourselves.

Kotlin supports the break and continue keywords just like Java and C#.

For more details about the other language features of Kotlin not described here, do read through the official documentation. It is a concise description of the language, building on existing programming knowledge of the reader. Perfect for you with your Java and/or C# skills.

Table 1. Quick comparison of Java, Kotlin and C#

Java	Kotlin	C#
Semicolons mandatory	Semicolons optional	Semicolons mandatory
package	package	namespace
import	import	using
void method() {}	fun function() {}	void Method() {}
System.out.println("Java");	println("Kotlin")	Console.WriteLine("C#");
int j = 0;	var j = 0	var j = 0;
final c = 9;	val c = 9 // or val c: Int = 9	const int c = 9;
int a = b > 0 ? b : 0;	var a = if (b>0) b else 0	int a = b > 0 ? b : 0;
switch	when	switch
for can loop over anything	for loops over iterable	for can loop over anything

Now is a good time for you to write a few small programs in Kotlin, just so you can get used to the differences in syntax.

3 Object-Oriented Programming in Kotlin

Before we jump into object-oriented programming (OOP) in Kotlin, let us revise some essential concepts.

3.1 Object-oriented Concepts

3.1.1 Encapsulation

Encapsulation is about hiding “the internal state of one object from the others”. (Shaukat, 2016) The idea is to separate the parts of the program from one another, which helps to make our programs easier to debug and maintain.

3.1.2 Inheritance

“Inheritance is an “is-a” relation, which inherits the attributes and behaviors from its parent class.” (Shaukat, 2016) For example, we could have a vehicle class that specifies behaviour which can be inherited by the car and truck classes.

3.1.3 Polymorphism

“Polymorphism is the ability of one object to be treated and used like another object.” (Shaukat, 2016) In our vehicle example, all vehicles could have a turn method. And we can call the method in the same way for any vehicle. But what the vehicle does in its turn method would be quite different between a motorcycle and a big rig truck, for example.

3.2 Object-oriented Programming in Kotlin

3.2.1 Classes and Inheritance

Let us look again at the code from learning unit 2. We very briefly just mentioned that we have Kotlin code that specified behaviour. Let us look at the generated MainActivity class in Figure 107.

```
1   package za.ac.iie.opsc7311.starsucks
2
3   import ...
5
6   class MainActivity : AppCompatActivity() {
7       override fun onCreate(savedInstanceState: Bundle?) {
8           super.onCreate(savedInstanceState)
9           setContentView(R.layout.activity_main)
10      }
11  }
```

Figure 107. Generated MainActivity class

A class is declared using the class keyword, just like you would expect in Java or C#. Inheritance is specified using : just like in C#. But there is a difference though. You will see that there are round brackets () after the parent class name, in this case AppCompatActivity. That means that the default constructor of the parent class is called then this MainActivity class is instantiated.

Android Studio makes it easy to navigate between different classes. If you hold Ctrl and click on the name of the AppCompatActivity class in the above code, Android Studio will take you to the code for that class. Recall that the Android code is open source, so we have access to all the code that our program is calling.

```
80  public class AppCompatActivity extends FragmentActivity implements AppCompatCallback,
81          TaskStackBuilder.SupportParentable, ActionBarDrawerToggle.DelegateProvider {
82
83      private static final String DELEGATE_TAG = "androidx:appcompat";
84
85      private AppCompatDelegate mDelegate;
86      private Resources mResources;
87
        Default constructor for AppCompatActivity. All Activities must have a default constructor for API 27 and
        lower devices or when using the default android.app.AppComponentFactory.
93      public AppCompatActivity() {
94          super();
95          initDelegate();
```

Figure 108. AppCompatActivity code in Android Studio

3.2.2 Declaring a Method

In our example in Figure 107, we have a method that is declared. It happens to also be an override of a method in AppCompatActivity. But here we do see the syntax for declaring a method. The fun keyword is used to declare a function. It can be either a top-level function (outside a class), or a method (inside a class).

The syntax for parameters follows the format name: type, which is unlike Java and C#. But you will recognise the format from when we declared variables with explicit types.

3.2.3 Properties

Properties in Kotlin work a lot like properties in C#. A property is a member variable of a class that allows access to a field by means of a getter and setter. Auto-implemented properties in C# automatically create a backing variable without you having to specify it. (Wagner, et al., 2022) Properties in Kotlin work like that too.

```
class MainActivity : AppCompatActivity() {

    var greeting: String = "Hello"
        set(value) {
            if (value.isNotEmpty()){
                field = value
            }
        }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        greeting = "Sawubona"
    }
}
```

Figure 109. Property with a setter

For the property greeting, we defined a custom setter, that checks if the value is empty, and only sets the value of the backing field if it is in fact not empty. We don't want to end up with an empty greeting message in our app, after all.

Remember, for more details about the Kotlin language, read the official documentation.

Now we are ready to use Kotlin code in our apps in the next learning unit.

4 Recommended Additional Reading

Kotlin Foundation, 2022. *Basic syntax.* [Online] Available at: https://kotlinlang.org/docs/basic-syntax.html [Accessed 17 November 2022].

Maurya, A. 2022. *Object-Oriented Programming in Kotlin [OOP in Kotlin — 2022]* [Online] Available at: https://blog. devgenius.io/object-oriented-programming-in-kotlin-oop-in-kotlin-2022-90dea0f5776c [Accessed 17 November 2022].

5 Recommended Digital Engagement and Activities

Learn Kotlin by Example is a collection of example programs that you can explore to get more familiar with the language. Start here: https://play.kotlinlang.org/byExample/overview [Accessed 17 November 2022].

Work through the *Introduction to Kotlin* training material on the android.com website at: https://developer.android.com/ courses/pathways/android-basics-kotlin-one [Accessed 17 November 2022].

6 Activities

Do the activities that appear on Learn.

7 Revision Exercises

Write a small Kotlin program that prints out the following text pattern:

8 Solutions to Revision Exercises

Read section 2.5 about loops above.

Learning Unit 4: More Advanced Techniques	
Learning Objectives: • Apply layouts in an app. • Use the EditText, NumberFormat and SeekBar in an app. • Use a navigation drawer in an app. • Apply colours to an app. • Create a launcher icon for an app. • Apply event handling in an app. • Explain the activity life cycle in an Android App. • Create an activity. • Use overridden methods. • Explain the purpose of an intent. • Apply an intent in an application.	**My notes**
Material used for this learning unit: • GitHub repository: LearningUnit4	
How to prepare for this learning unit: • Make sure that you have the GitHub source code available and that your Android Studio is up to date.	

1 Introduction

Now that we can design a UI and run an application, and we know some Kotlin too, the time has come to add logic to our app. We are going to slowly build up to an application that has two activities and that can pass data between the two activities. We are also going to delve deeper into the linear layout and create a nested hierarchy. We will use toasts and intents on our second activity.

And finally, we are going to add a navigation drawer to the app to improve the user experience.

2 Layouts and Controls

2.1 Removing the Action Bar

Let's start by updating our UI. We are currently only displaying an `ImageView` and a `TextView`. We want our application to resemble the UIs shown in Figure 110. We want to display a scrollable list of all the products that our coffee shop sells.

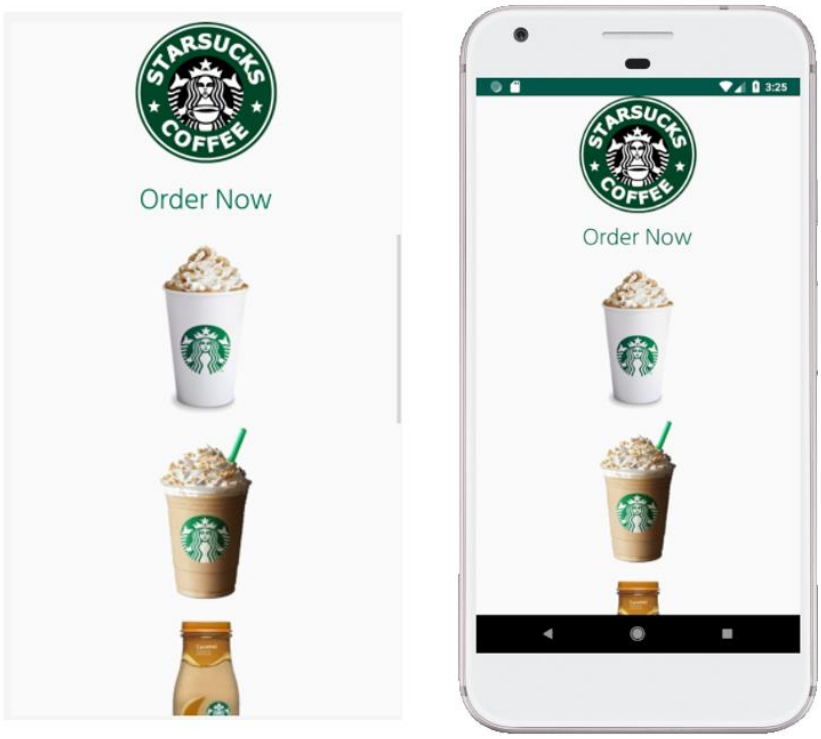


Figure 110. Updated User Interface – Our Goal

The first thing we need to do, is to remove the action bar. The action bar is the ugly looking bar on the top of your app that shows your application name.

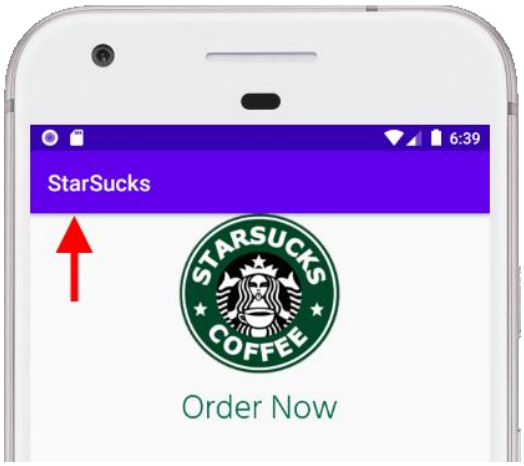


Figure 111. Action bar in the StarSucks App

You could have actions appearing there in your app. But we are not going to do that. So, the action bar needs to go.

The easiest way to remove the action bar is to access the themes.xml file from the `values\themes` folder and to edit the `style` entry. Change the parent attribute to:

```
Theme.MaterialComponents.DayNight.NoActionBar
```

```
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Theme.StarSucks" parent="Theme.MaterialComponents.DayNight.NoActionBar">
        <!-- Primary brand color. -->
        <item name="colorPrimary">@color/purple_500</item>
        <item name="colorPrimaryVariant">@color/purple_700</item>
        <item name="colorOnPrimary">@color/white</item>
        <!-- Secondary brand color. -->
        <item name="colorSecondary">@color/teal_200</item>
        <item name="colorSecondaryVariant">@color/teal_700</item>
        <item name="colorOnSecondary">@color/black</item>
        <!-- Status bar color. -->
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>
        <!-- Customize your theme here. -->
    </style>
</resources>
```

Figure 112. Changing the style parent

Make a similar change to the themes.xml (night) file too, setting the parent of the style to:

```
Theme.MaterialComponents.DayNight.NoActionBar
```

Now when we run the app again, the action bar no longer appears.

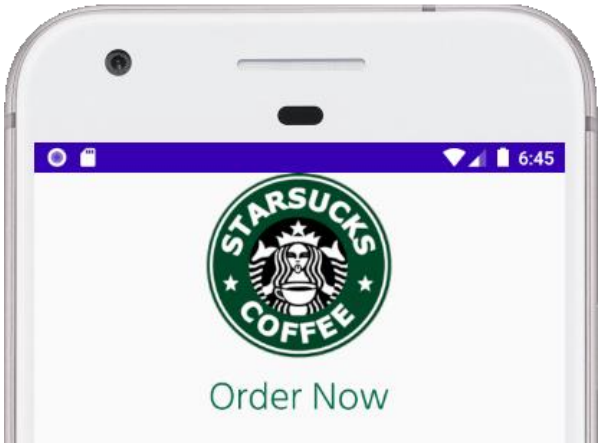


Figure 113. App without the Action Bar

2.2 Adding More UI Components

We previously spoke about the fact that you can nest different views inside each other. To recreate what we see in Figure 110, we need to first add a `ScrollView`. That is the component that will allow the list to scroll if it is longer than can be displayed on the screen at the same time.

You can find the `ScrollView` component under the **Common** category on the **Palette**.

Inside the `ScrollView`, we want to display a couple of `ImageViews`, vertically below one another. That is probably feeling familiar by now – we can use a `LinearLayout` (with **vertical** orientation) to do that.

Next, we add six `ImageViews` – one for each of the image assets sb1, sb2, sb3, sb4, sb5 and sb6.

> **Tip:** If you are using the Layout Editor, drag the `ImageView` onto the `LinearLayout` in the **Component Tree**. If you drag it onto the visual representation, it will not add it.

Go ahead and create the UI (either using the Layout Editor or the XML view – it is your choice). The completed UI should look like the one shown in Figure 114. Look specifically at the hierarchy of components shown in the **Component Tree**. We have a `LinearLayout` that contains a `ScrollView` that contains a `LinearLayout` that contains the six `ImageViews`.

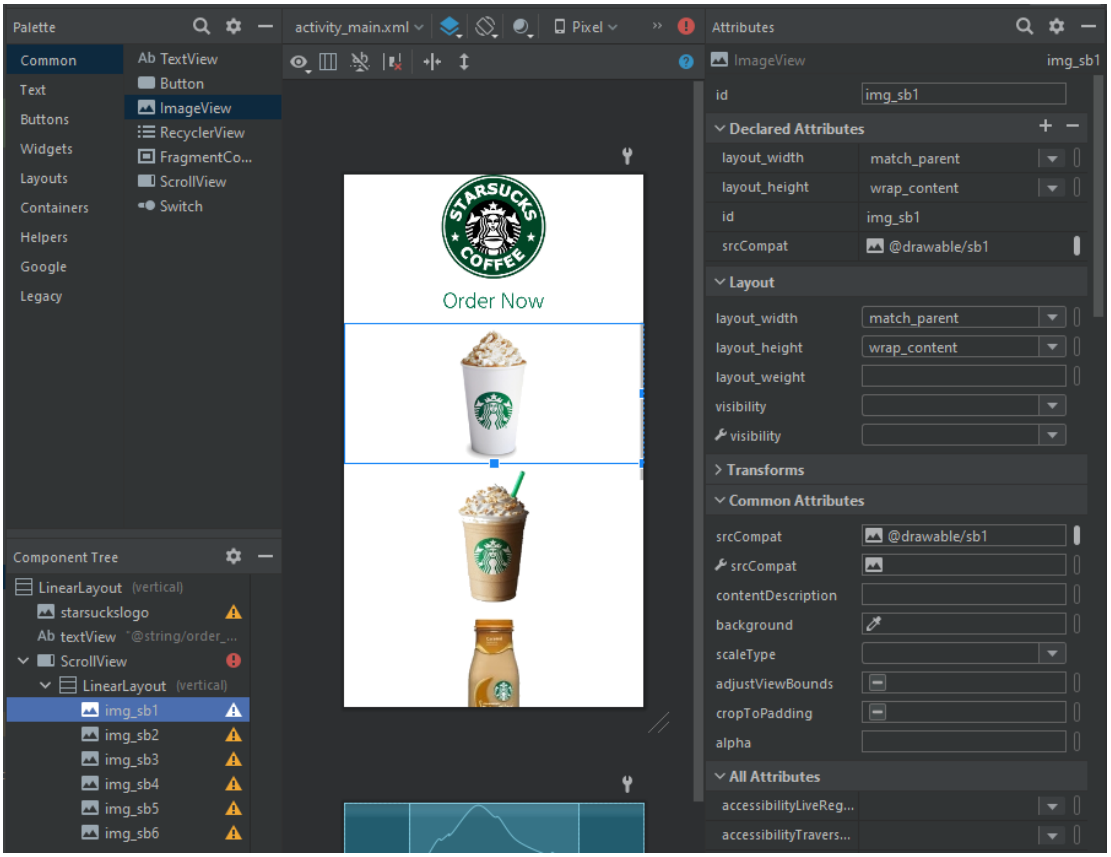


Figure 114. Completed UI in the Layout Editor

Note that the order that the components appear in the **Component Tree** will be the order that they will appear in the `LinearLayout` at runtime too.

Another important thing to note here is the **naming convention** that is used for the various components. There is no standard naming convention that is specified by the Android Open Source Project for components in the XML file. But it is important to **stick** to a naming convention throughout the app. And if you name your components well, it will be easy to spot mistakes when you access these from the code a little later.

For this module manual, we will make use of the convention of adding a prefix followed by an underscore followed by an easy-to-understand name. The prefixes that we use are as follows (inspired by (Jethro, 2018)):

- `ImageView`: img
- `TextView`: tv
- `Button`: btn
- `EditText`: et
- `FloatingActionButton`: fab

Here is the complete layout in XML. Pay careful attention to how the elements are nested.

```
<?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"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <ImageView
        android:id="@+id/starsuckslogo"
        android:layout_width="232dp"
        android:layout_height="152dp"
        android:src="@drawable/starsuckslogo"
        android:layout_gravity="center_horizontal"/>

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:fontFamily="@font/hind_guntur_light"
        android:text="@string/order_now_label"
        android:textColor="@color/starsucksGreen"
        android:textSize="30sp" />

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical" >

            <ImageView
                android:id="@+id/img_sb1"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                app:srcCompat="@drawable/sb1" />

            <ImageView
                android:id="@+id/img_sb2"
                android:layout_width="match_parent"
```

```
                android:layout_height="wrap_content"
                app:srcCompat="@drawable/sb2" />

            <ImageView
                android:id="@+id/img_sb3"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                app:srcCompat="@drawable/sb3" />

            <ImageView
                android:id="@+id/img_sb4"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                app:srcCompat="@drawable/sb4" />

            <ImageView
                android:id="@+id/img_sb5"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                app:srcCompat="@drawable/sb5" />

            <ImageView
                android:id="@+id/img_sb6"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                app:srcCompat="@drawable/sb6" />
        </LinearLayout>
    </ScrollView>
</LinearLayout>
```

3 Event Handling

We now have plenty UI elements (Views) and it is time to add some logic and event handling to our application. We have previously discussed how the `R` class and the resource (`res`) folders work. We will now put this information to work.

Android developers use either **activities** or fragments to design their applications UI and code up the logic. We will work with fragments in Open Source Coding (Intermediate) (OPSC7312). We will focus on activities for now. You can think of an activity as a single screen in your application. Your application might have multiples screens, similarly to how the one we are about the develop will have. Each screen (activity) is made up of two parts. The Java file and the XML file.

The `MainActivity` is created by default when you create your application and the corresponding two files that make up the activity is immediately available for use.

These two files are:

- **MainActivity.kt** which is a Kotlin class that allows us to code up the logic of our application.
- **activity_main.xml:** Which you by now know is the XML file where we create the UI for the activity.

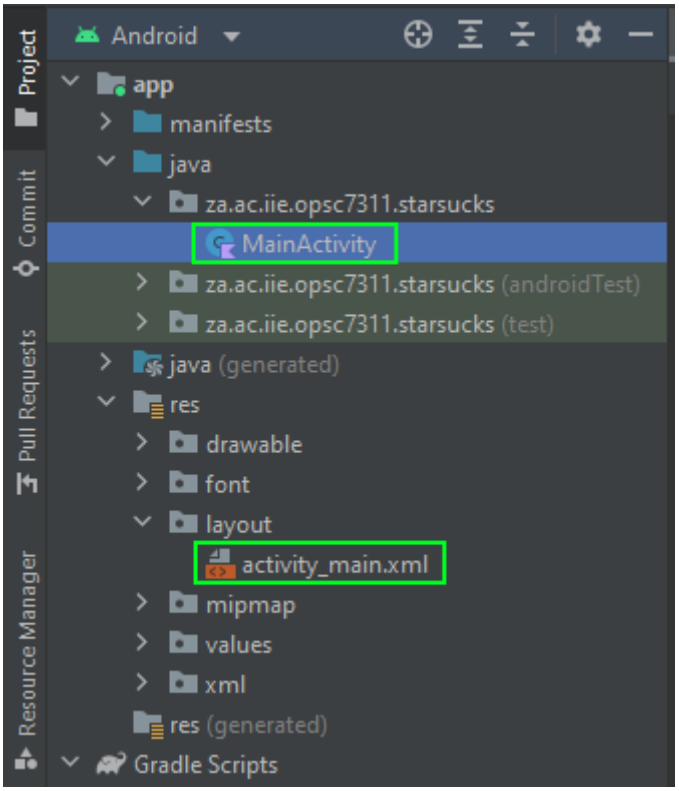


Figure 115. Two Files that Make Up the `MainActivity`

3.1 Declaring and Initialising UI Elements

We want to add logic to our program now, so we are going to be working in the Kotlin code. When working in the Kotlin code, we don't automatically have access to the UI components. But there are two different ways of getting access to these components.

3.1.1 Old-fashioned manual way

To get the variables in the old-fashioned way, we need to declare fields for them and then initialise them too.
Open the `MainActivity.kt` file and add the following properties to it:

```
7  class MainActivity : AppCompatActivity() {
8
9      lateinit var img_Sb1: ImageView
10     lateinit var img_Sb2: ImageView
11     lateinit var img_Sb3: ImageView
12     lateinit var img_Sb4: ImageView
13     lateinit var img_Sb5: ImageView
14     lateinit var img_Sb6: ImageView
```

Figure 116. Properties in the `MainActivity`

We don't have a value for these properties when we declare them, so we add the `lateinit` keyword to indicate that we will initialise these later.

You will notice that we use the UI elements (`Views`) much like data types such as String, int, float and double. You should also notice that the “data types” (class names) start with a capital letter. If you think back to the naming conventions that you have learned, you will remember that class names start with capital letters in Kotlin. The `ImageView` “data type” is named in Pascal case (starts with a capital letter) because it is a class. If you have the sources installed, and you press Ctrl on your keyboard and click on the `ImageView` “data type”, the `ImageView` class will open as shown in Figure 117.

```
99      @RemoteView
100     public class ImageView extends View {
101         private static final String LOG_TAG = "ImageView";
102
103         // settable by the client
104         @UnsupportedAppUsage
105         private Uri mUri;
106         @UnsupportedAppUsage
107         private int mResource = 0;
108         private Matrix mMatrix;
109         private ScaleType mScaleType;
110         private boolean mHaveFrame = false;
111         @UnsupportedAppUsage
112         private boolean mAdjustViewBounds = false;
113         @UnsupportedAppUsage(maxTargetSdk = Build.VERSION_CODES.P)
114         private int mMaxWidth = Integer.MAX_VALUE;
115         @UnsupportedAppUsage(maxTargetSdk = Build.VERSION_CODES.P)
116         private int mMaxHeight = Integer.MAX_VALUE;
```

Figure 117. ImageView Class from the Android Code

If you look at this code, you will notice that it is written in Java, not Kotlin.
This class contains almost 2000 of lines of code which defines constructors, and all the methods and properties that we can programmatically use for an `ImageView`.

One of the methods of `ImageView` that we will be using often in `setImageResource()` – the method shown in Figure 118.

Luckily, the Android UI classes abstracts away the complexities of drawing user interfaces in the Android Operating System. It is sometimes useful to be able to read the code when something is not working, but we don’t need to understand all the details to be able to make good use of an `ImageView`.


```
@android.view.RemotableViewMethod(asyncImpl="setImageResourceAsync")
public void setImageResource( @DrawableRes @DrawableRes int resId) {
    // The resource configuration may have changed, so we should always
    // try to load the resource even if the resId hasn't changed.
    final int oldWidth = mDrawableWidth;
    final int oldHeight = mDrawableHeight;

    updateDrawable( d: null);
    mResource = resId;
    mUri = null;

    resolveUri();

    if (oldWidth != mDrawableWidth || oldHeight != mDrawableHeight) {
        requestLayout();
    }
    invalidate();
}
```

Figure 118. setImageResource() Method

When you declare the `ImageView` fields shown in Figure 116, you may run into the issue shown in Figure 119.

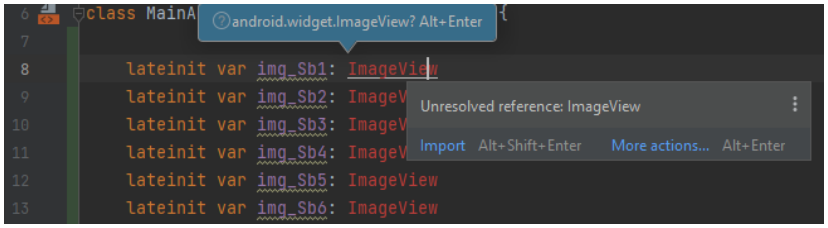


Figure 119. Unresolved reference error

This error happens when the import is missing and is most likely to happen if you copy and paste code rather than typing it out. As the blue tooltip suggests, press `Alt + Enter`. This will add the import as shown in Figure 120.

```
package za.ac.iie.opsc7311.starsucks

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.ImageView

class MainActivity : AppCompatActivity() {

    // declare fields for ImageViews
    lateinit var img_Sb1: ImageView
```

Figure 120. Import Added

Right now, all the variables are still uninitialized. So, we now need to initialise our variables. We do this by pointing our Kotlin class to the correct XML ID which is stored in our `R` class. We do this in the `onCreate()` method, which is one of the activity life cycle methods that we will discuss in more detail in section 4. We will use the `findViewById()` method to link our XML and Kotlin.

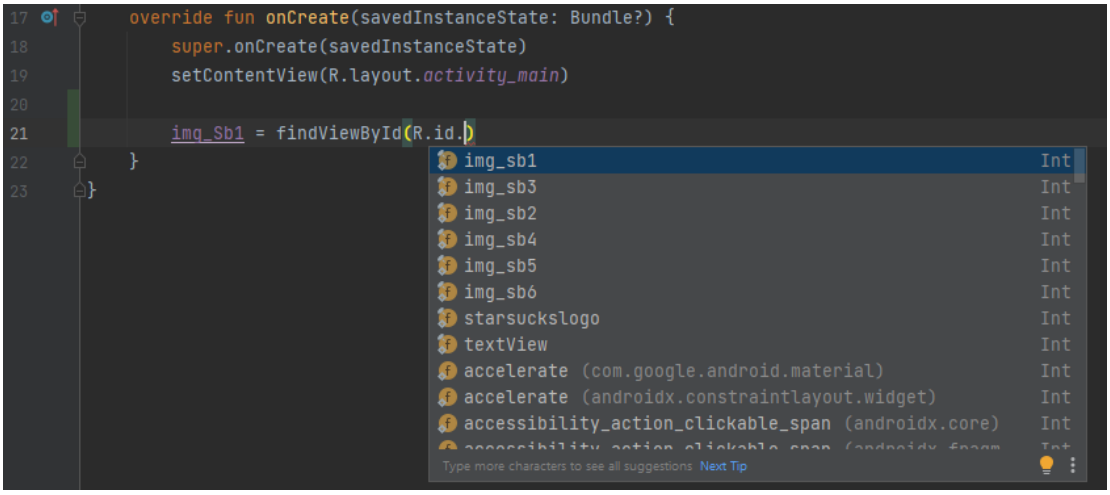


Figure 121. Finding the ImageView by its ID

The `findViewById()` method (which our `MainActivity` inherits from `AppCompatActivity`) expects an int value as a parameter.

We access that int value by accessing our `R` class. Inside the R class we will find the nested (inner) class called id, and inside that we find the IDs for all our components in the app.

Note that component IDs need to be unique in the whole app, not just on the specific layout where it is used.

All the initialisation code in the `onCreate()` method is shown in Figure 122.

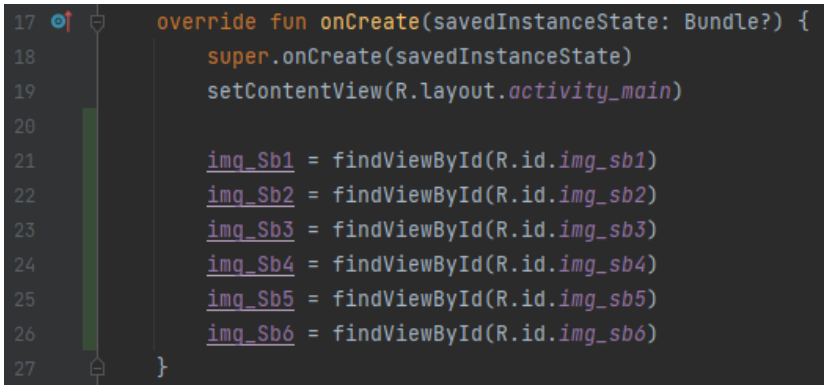


Figure 122. All the Fields Initialised

Note: It is very important that the **initialisations** must be **after** the `setContentView()` method is called. Before `setContentView()` completes, `findViewById()` will just return null for all the elements.

3.1.2 Using View Binding

I am sure you will agree that this is quite tedious. We have already created the `ImageViews` in the XML – why do we need to create a property for each one? Well,

the good news is there is a more modern and easier way, using View Binding. (Leiva, 2020)

The first step is to change something in a gradle.build file. Open the gradle.build file for the Module: StarSucks.app. In this file, there are a whole bunch of properties used by Gradle during the build process. Add the below highlighted row to the file, save it, and rebuild the project.

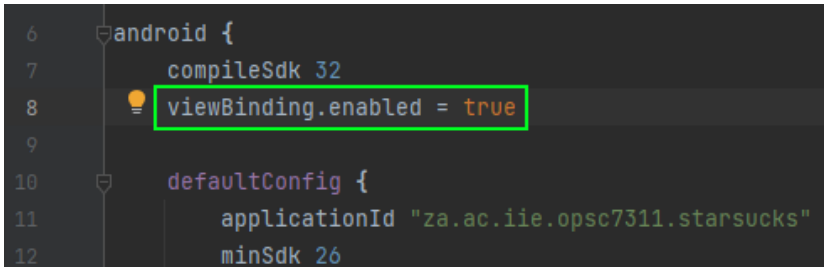


Figure 123. Enabling View Binding

Now we can change the code to use View Binding. You can delete the properties and delete all the rows to find the views. Instead, replace the `setContentView` line with the two lines shown Figure 124.

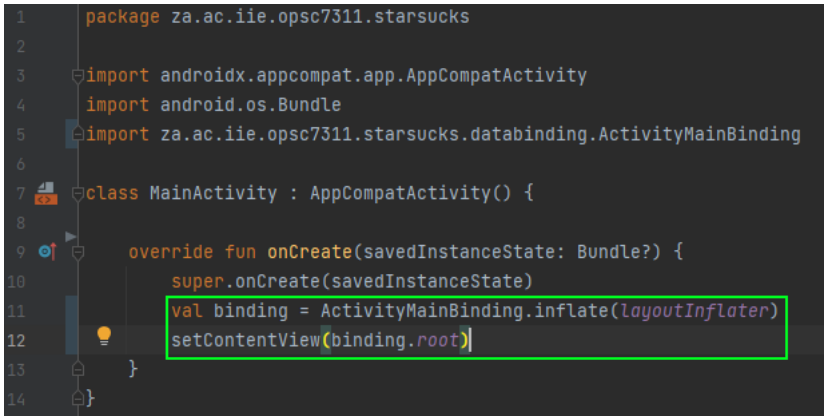


Figure 124. Using View Binding

Much less code to write. And we don't need to add code to get components if we add more of those to the view later.

3.2 Adding Event Handlers

Event handlers allow us to provide the logic that we would like our application to perform when our users interact with the UI elements of our application. There are many event handlers such as `onTouch`, `onSelectionChanged`, `onClick` etc. available in Java for Android. You will learn most of them as you start working of different UI elements. We are going to start with a simple `OnClickListener`.

An `OnClickListener` reacts when the user clicks or taps on one of our UI elements. We are going to set an `OnClickListener` on each of our `ImageViews` and pop up a `Toast` message when a user taps on the `ImageView`. We will optimise this code later. We are going to practice for now.

It is easy to add an `OnClickListener`. In the `onCreate` method, we can now use the binding to access our controls. Simply type `binding.` followed by the name we gave the `ImageView` and start typing `.setOnClickListener`.

Auto complete will pop up with the list of possible methods that can be called (see Figure 125). With the `setOnClickListener` method selected on the autocomplete pop-up, press `Enter`.

Figure 125. Start Typing setOnClickListener Shows Autocomplete Pop-Up

You should have the following code now:

```
binding.imgSb1.setOnClickListener()
```

Add {} so we have a space to put the code that gets executed when the image is tapped.

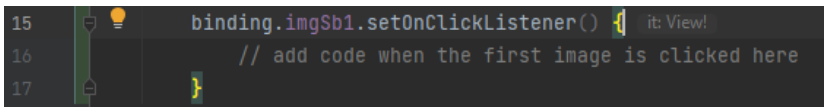


Figure 126. Adding the {}

So, what exactly is going on in the generated code? If we Ctrl+click on the setOnClickListener, we see that the method is expecting a class implementing the interface `View.OnClickListener`. And if we click through to that, we see that it has a single method – `onClick()`.

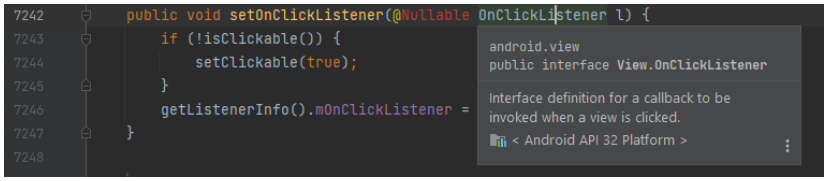


Figure 127. setOnClickListener

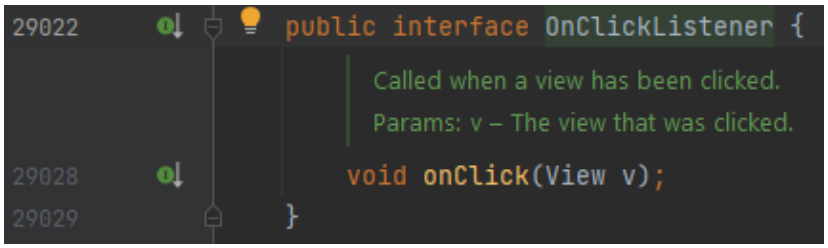


Figure 128. Declaration of View.OnClickListener

This syntax makes use of a Lambda in Kotlin. (al3c, 2022) so, we can avoid having to use anonymous inner classes.

3.3 Adding Toast Messages

`Toast` messages are handy little messages that pop up to give your user information and status updates about your app. They appear for a short period of time and then disappear. We are going to use `Toast` messages to display the name of the selected product when we click on it. Add the following code inside the `OnClick()` method.

Start typing `Toast` and select the second option from the IntelliSense pop-up.

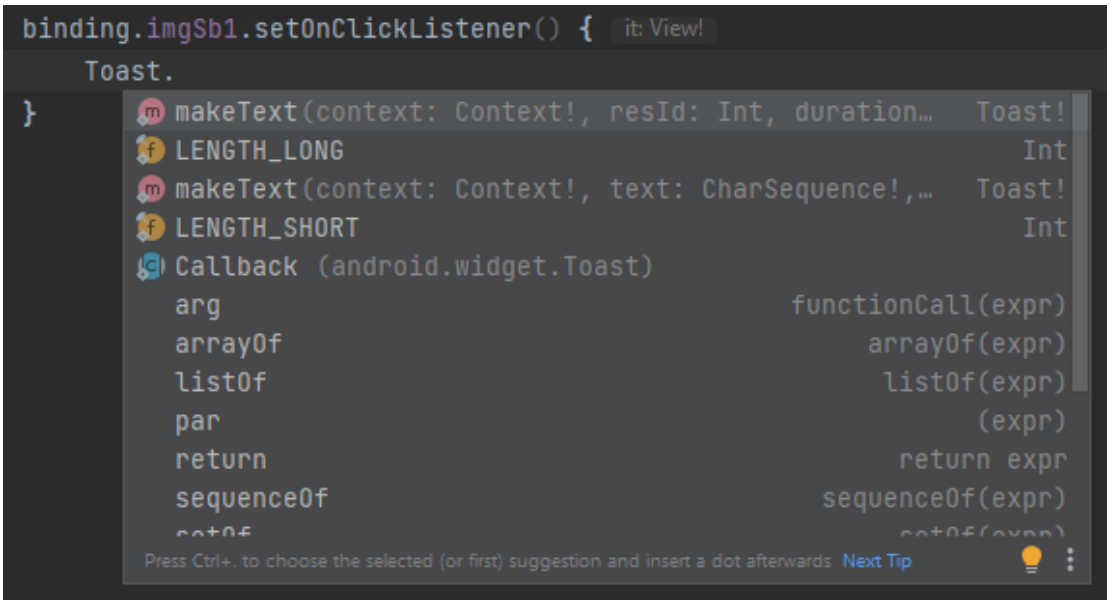


Figure 129. Starting with the Toast

A Toast message needs the following arguments:

- A `context` (telling Android which activity the toast should appear in)

- The `text` that you would like to display
- The `duration` of the Toast (you have two options: `LENGTH_SHORT` and `LENGTH_LONG`)

```
15        binding.imgSb1.setOnClickListener() {  it: View!
16            Toast.makeText( context: this@MainActivity,  text: "MMM Soy Latte",
17                            Toast.LENGTH_SHORT).show()
18        }
```

Figure 130. Displaying the Toast

The `Toast` message will appear on the `MainActivity`, for a short duration. Also note the `show()` method. Without this our `Toast` will never appear!

Note: Kotlin, like C#, does have the concept of **named arguments**. But the context: label here is just something that is displayed by the IDE for convenience, not something that you should type! Named arguments in Kotlin use an = not a colon.

Now when we run the app and we tap the first `ImageView` we will see our message.

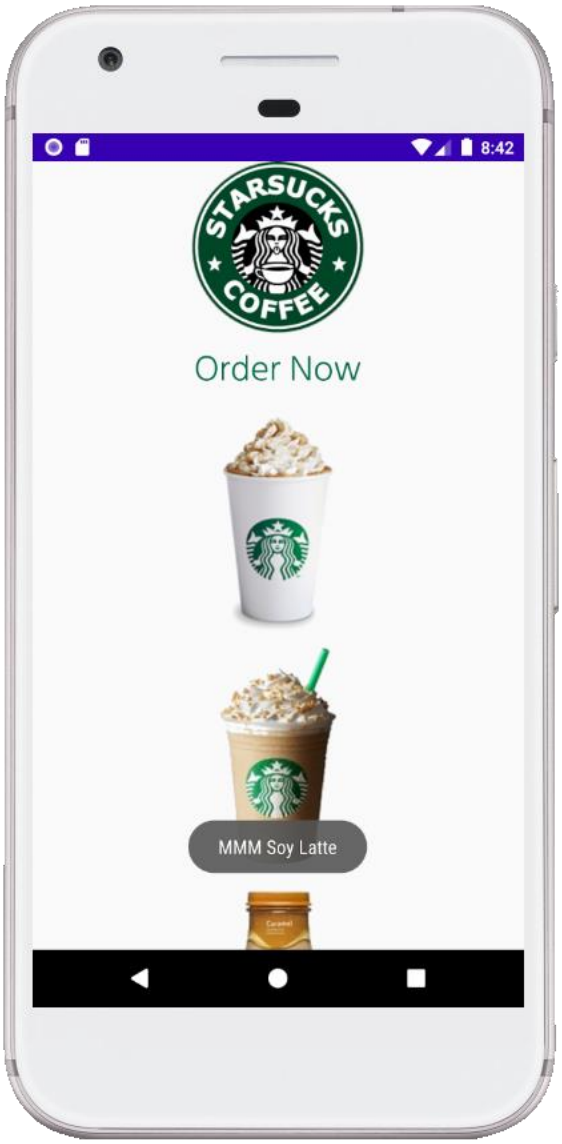


Figure 131. Toast Message in the Running App

Go ahead and add the Toast message for all the other images too. The messages should be as follows:

- sb2: MMM Chocco Frapp
- sb3: MMM Bottled Americano
- sb4: MMM Rainbow Frapp
- sb5: MMM Caramel Frapp
- sb6: MMM Black Forest Frapp

Compare your code to the code in the sample repository if you struggle to make it work.

You might be thinking right now that having so many copies of these anonymous inner classes can't possibly be the best way to do this. And you would be right. We will improve this implementation in section 5.1.

4 Activity Life Cycle

Each Android activity has a lifecycle. This consists of methods we use to provide logic for when users click off the activity, click back on the activity, start the activity, or close the activity. The life cycle is shown in Figure 132.

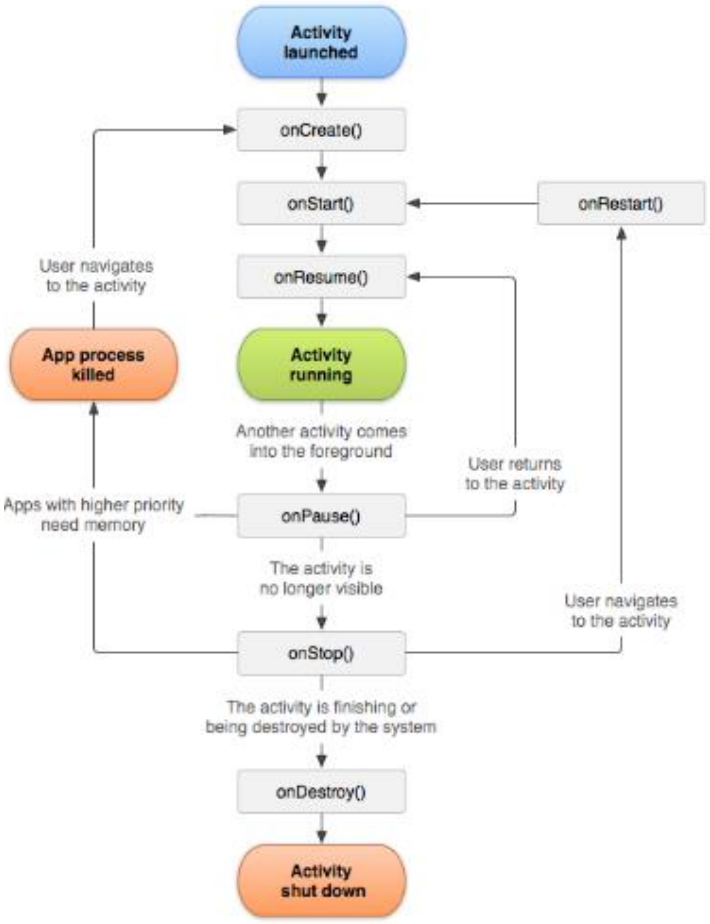


Figure 132. Simplified Activity Life Cycle (Android Open Source Project, 2020j)

Any activity has four essential states and are essentially stacked in the memory. When you click off one app and onto another the activity you left is placed “behind” the activity you are currently on. These states can be explained as:

- **Active/ Running:** This occurs when the activity is on the top of the stack and currently being used.
- **Lost Focus:** This activity is still visible to user but not currently being used.
- **Stopped:** This activity is completely taken over by another activity and no longer visible to the user. The window is hidden until the user recalls it.
- **Destroyed:** The activity is dropped from the memory (killed)

As previously mentioned, the activity life cycles are methods that we use to provide logic for user actions when using, pausing, and stopping the application. These methods are listed in Figure 133.


```
public class Activity extends ApplicationContext {
    protected void onCreate(Bundle savedInstanceState);

    protected void onStart();

    protected void onRestart();

    protected void onResume();

    protected void onPause();

    protected void onStop();

    protected void onDestroy();
}
```

Figure 133. Life Cycle Methods (Android Open Source Project, 2020j)

These methods are described below. From (Android Open Source Project, 2020j).

Method	Function
onCreate()	Called when the activity is first created. This is where you should do all of your normal static set up: create views, bind data to lists, etc. This method also provides you with a Bundle containing the activity's previously frozen state, if there was one. Always followed by onStart()
onRestart()	Called after your activity has been stopped, prior to it being started again. Always followed by onStart()
onStart()	Called when the activity is becoming visible to the user. Followed by onResume() if the activity comes to the foreground, or onStop() if it becomes hidden.
onResume()	Called when the activity will start interacting with the user. At this point your activity is at the top of its activity stack, with user input going to it. Always followed by onPause().
onPause()	Called when the activity loses foreground state, is no longer focusable or before transition to stopped/hidden or destroyed state. The activity is still visible to user, so it's recommended to keep it visually active and continue updating the UI. Implementations of this method must be very quick because the next activity will not be resumed until this method returns. Followed by either onResume() if the activity returns back to the front, or onStop() if it becomes invisible to the user.

Method	Function
onStop()	Called when the activity is no longer visible to the user. This may happen either because a new activity is being started on top, an existing one is being brought in front of this one, or this one is being destroyed. This is typically used to stop animations and refreshing the UI, etc. Followed by either onRestart() if this activity is coming back to interact with the user, or onDestroy() if this activity is going away.
onDestroy()	The final call you receive before your activity is destroyed. This can happen either because the activity is finishing (someone called Activity#finish on it), or because the system is temporarily destroying this instance of the activity to save space. You can distinguish between these two scenarios with the Activity#isFinishing method.

4.1 Adding a New Activity

Now that we have added some logic to our app, it is time to add an extra activity and some more sophisticated functionality. We will start by adding a second activity and creating a UI for this activity.

Adding a new activity is very easy to do. To create a new empty activity:

1. Right-click on your package in the project folder where you want to create it.
2. Click **New**, then **Activity** and then **Empty Activity** (see Figure 134). The **New Android Activity** dialog will be displayed.
3. On the **New Android Activity** dialog (see Figure 135), give the new activity a **name** – call it `OrderDetailsActivity`.
4. Make sure that the **Generate a Layout File** checkbox is selected.
5. Double check that the package name is the one where you want to create it. Click **Finish**.

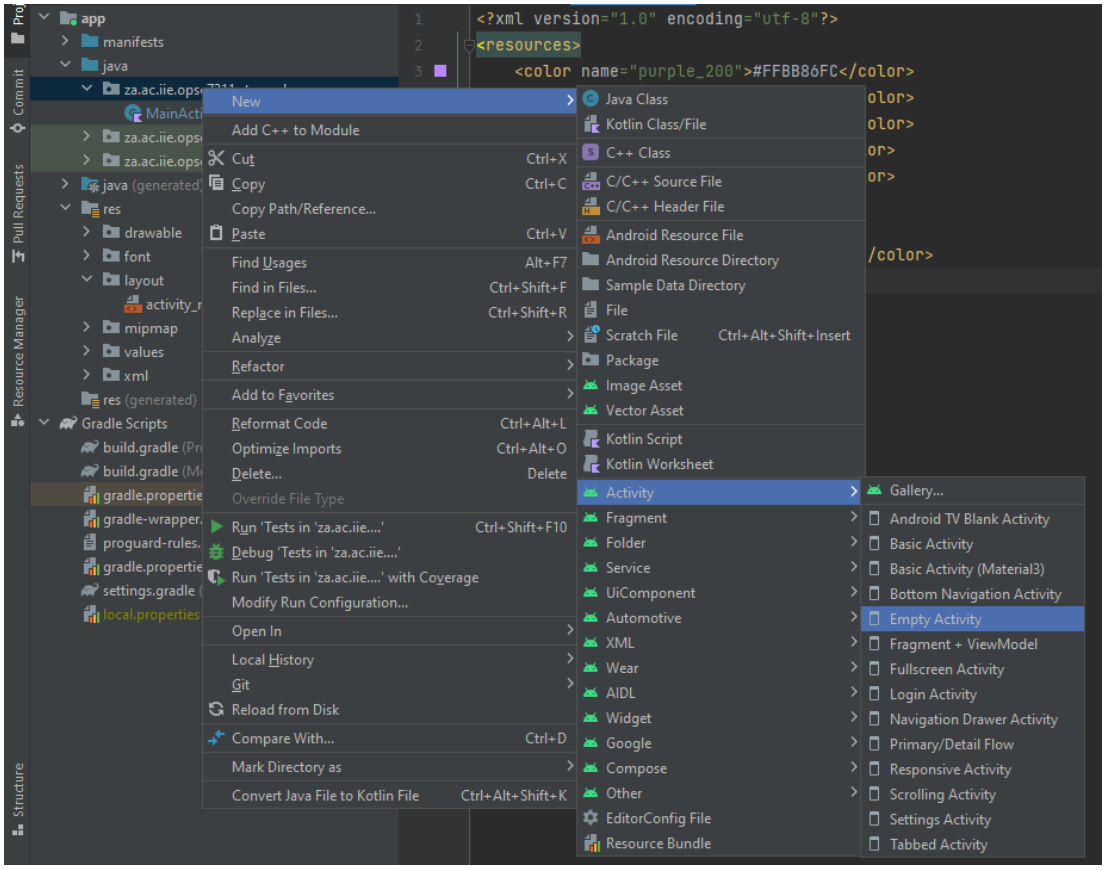


Figure 134. Creating a New Activity

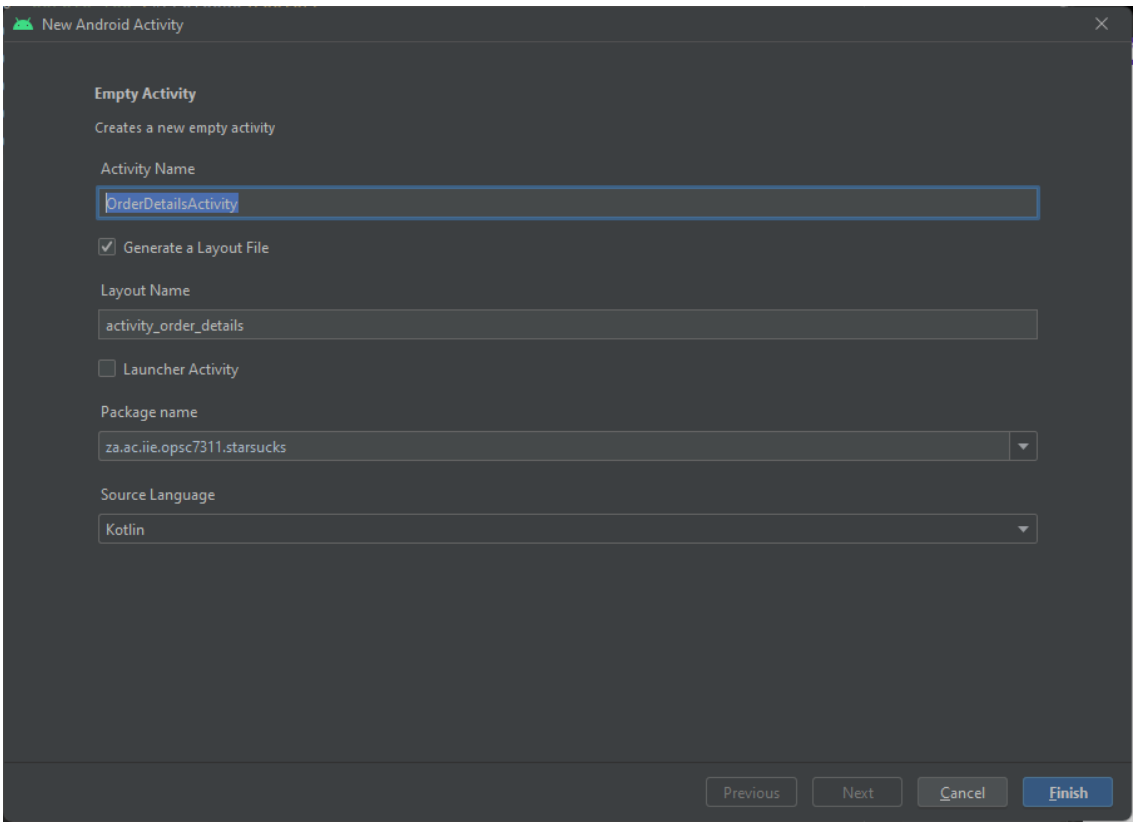


Figure 135. Configuring the New Activity

The new Kotlin file and XML file will be added to the project, as shown in Figure 136.

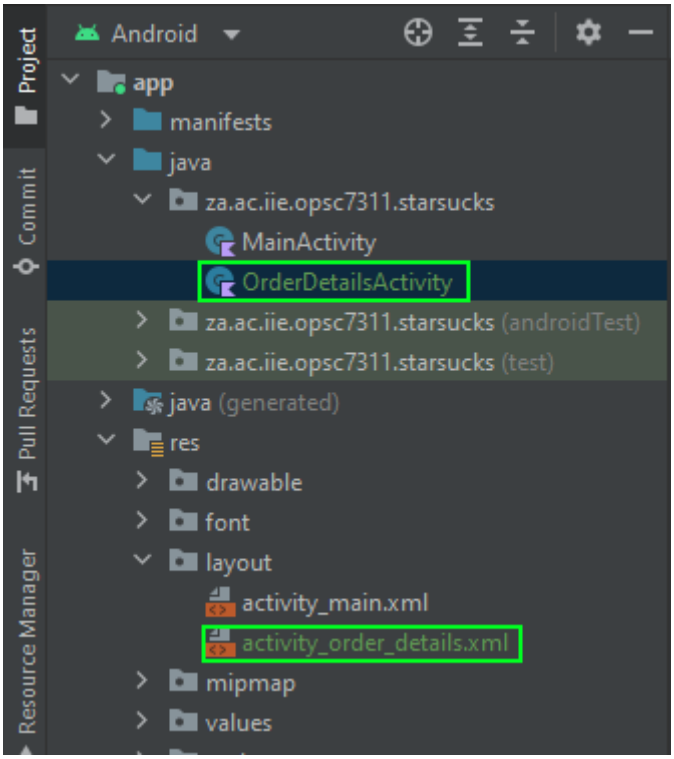


Figure 136. New Files added to the App Project

4.1.1 Adding the UI for the New Activity

We are going to create the UI as shown in Figure 137.

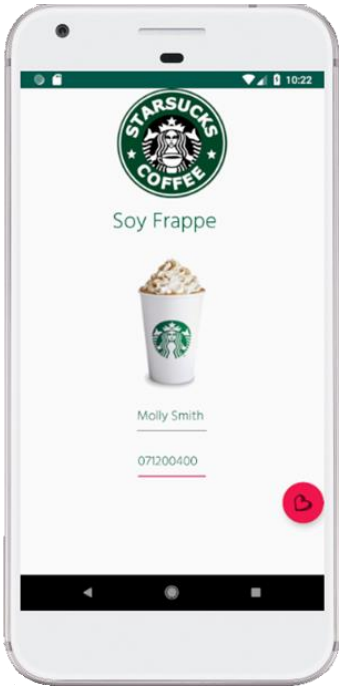


Figure 137. Order Details UI

This UI will display the Image of the product we selected, the name of the product and allow us to enter our name and nell phone number for delivery.

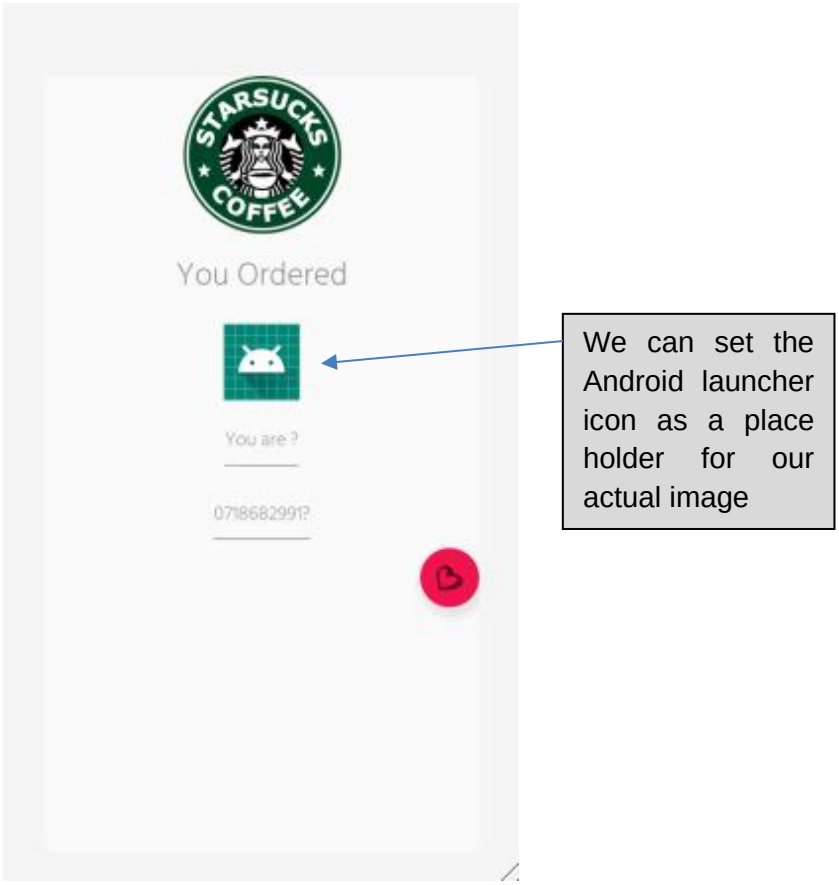


Figure 138. The New User Interface in the Designer

The new UI elements that you will encounter is an `EditText` and a `FloatingActionButton` (FAB). `EditTexts` allows users to enter information into the application and can have predefined formats such as Text, Number, E-mail, Phone number etc.

You will design this UI in the same as we did for the first activity. We will have a nested layout that contains our UI elements, as shown in Figure 139.

For now, we are just using @mipmap/ic_launcher for the image. It is just a place holder – we will set it programmatically after the user choses their beverage.

Notice the hints that are set for the two `EditTexts`. The hint will only be displayed when nothing has been entered yet.

The `FloatingActionButton` makes use of the smallheart.png image, the you will find in the `LearningUnit4\assets` folder in the repository.

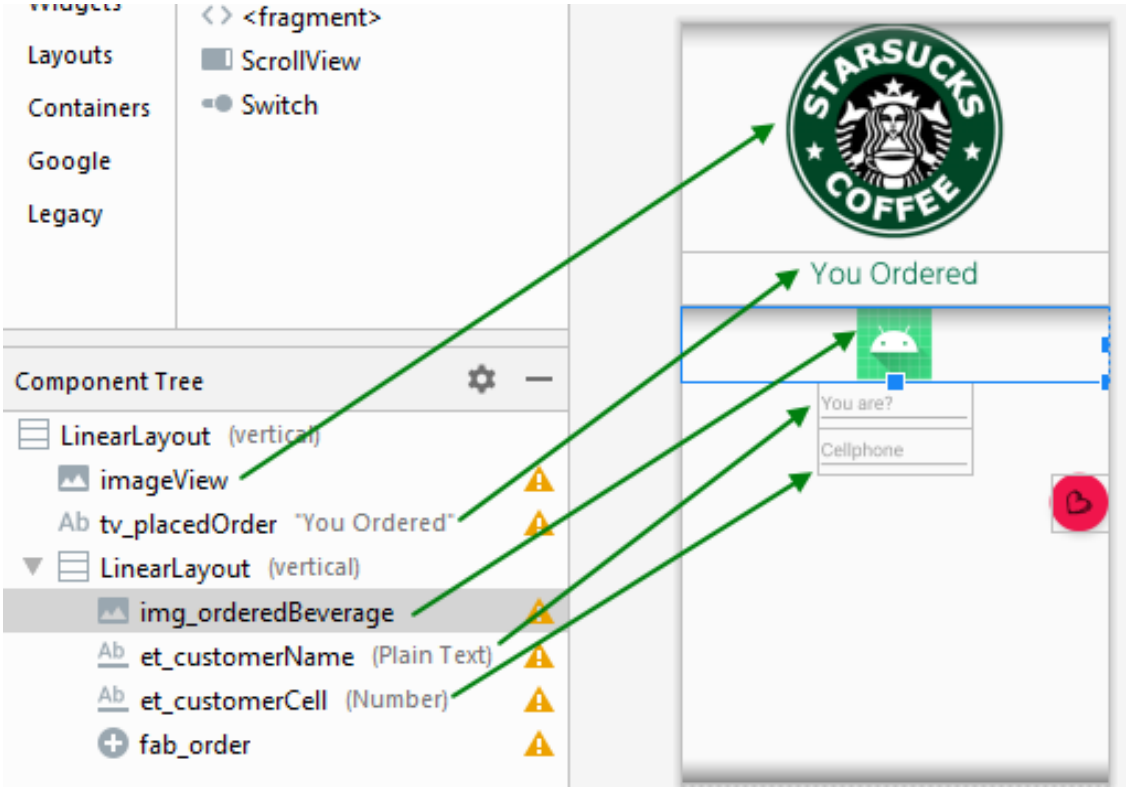


Figure 139. New User Interface in the Layout Editor

Add the `FloatingActionButton` by using the Layout Editor. If it prompts you to add a dependency, accept it. This will add the library to your project that you need for the `FloatingActionButton` to work. Wait for the Gradle sync to complete, then the button will appear.

The colour for the button is starsucksRed, which is #F0154C.

The Layout looks like this in XML:

```
<?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"
    android:orientation="vertical"
    tools:context=".OrderDetailsActivity">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="match_parent"
        android:layout_height="221dp"
        app:srcCompat="@drawable/starsuckslogo" />

    <TextView
        android:id="@+id/tv_placedOrder"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
```

```
        android:fontFamily="@font/hind_guntur_light"
        android:text="You Ordered"
        android:textAlignment="center"
        android:textColor="@color/starsucksGreen"
        android:textSize="30sp" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <ImageView
            android:id="@+id/img_orderedBeverage"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            app:srcCompat="@mipmap/ic_launcher" />

        <EditText
            android:id="@+id/et_customerName"
            android:layout_width="148dp"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:ems="10"
            android:hint="You are?"
            android:inputType="textPersonName" />

        <EditText
            android:id="@+id/et_customerCell"
            android:layout_width="148dp"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:ems="10"
            android:hint="Cellphone"
            android:inputType="number" />
      <com.google.android.material.floatingactionbutton.FloatingActionButton
            android:id="@+id/fab_order"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="right"
            android:clickable="true"
            app:backgroundTint="@color/starsucksRed"
            app:srcCompat="@drawable/smallheart" />

    </LinearLayout>
</LinearLayout>
```

For guidelines about using floating action buttons in your user interfaces, read https://material.io/components/buttons-floating-action-button [Accessed 17 November 2022].

5 Using Intents

Now that we have our UI ready, we can add the necessary logic to our application. We want to click on one of our products in the `MainActivity` and use an `Intent` to open the new activity and to pass the name of the product over to the `OrderDetailsActivity`. We are also going to start by optimising our code on the

`MainActivity`. We currently have 6 event handlers on this activity. This introduces quite a bit of redundant code that can be done more efficiently.

5.1 Optimising the Event Handlers

The previous app functioned just fine - it popped up a toast message every time we clicked on one of our products. But there is code and then there is efficient code. Your compiler basically compiled the same method with slight variations six times. That is not very efficient, and not very easy to maintain.

We previously discussed the `OnClickListener` and the fact that it is an interface that lives in the `View` class. Android enables use a lambda created from that interface into the `setOnClickListener()`. That is one way of using an `onClickListener`. It works perfectly – most of the time.

But we have multiple `ImageViews` and we only want one `OnClickListener` to fire when we click the any of the `ImageViews`. We would also like to avoid redundant and repeated code.

We can do this implementing the `OnClickListener` interface on the `MainActivity`, and then passing the `MainActivity` to the `setOnClickListener()` method. (Engel, 2017) (See Figure 140.)

When the `MainActivity` implements `View.OnClickListener`, you will need to override the `onClick()` method from that interface.

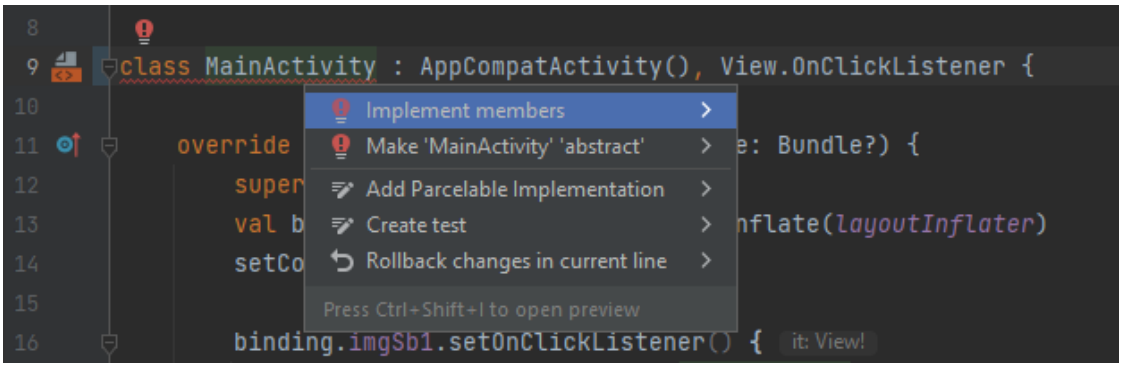


Figure 140. Implementing View.OnClickListener

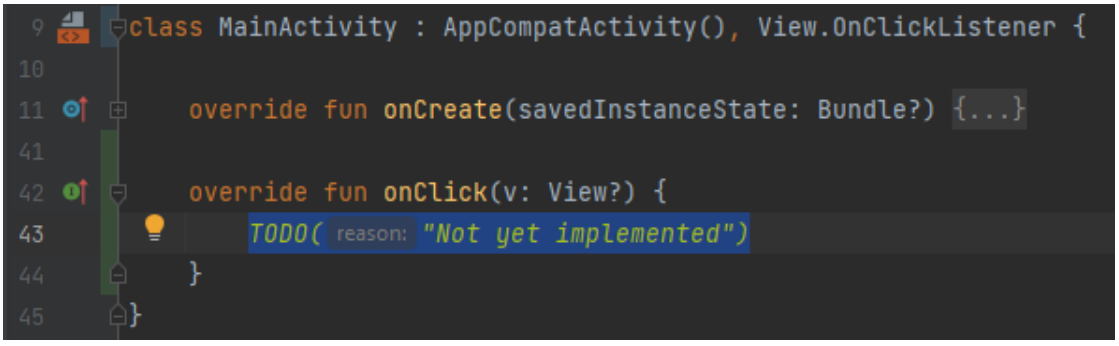


Figure 141. Generated onClick method

We can then set the `onClickListener` to each `ImageView` with the `setOnClickListener` method as shown below. We pass through `this`, which is the reference to the current class (i.e., the `MainActivity` instance).

```
class MainActivity : AppCompatActivity(), View.OnClickListener {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        binding.imgSb1.setOnClickListener(this)
        binding.imgSb2.setOnClickListener(this)
        binding.imgSb3.setOnClickListener(this)
        binding.imgSb4.setOnClickListener(this)
        binding.imgSb5.setOnClickListener(this)
        binding.imgSb6.setOnClickListener(this)
    }
```

Figure 142. Using this as the Listener

Now our `onClick()` method needs to be implemented to do the same work that the individual `onClick()` methods did before. For that, we need to be able to determine which `View` was clicked. Luckily, the method takes a parameter which is the `View`, so we can use the id of that view to determine what to do. This is the perfect opportunity to use the `when` statement.

The code is shown in Figure 143.

```
    override fun onClick(v: View?) {
        when(v?.id) {
            R.id.img_sb1 -> Toast.makeText( context: this@MainActivity,
                text: "MMM Soy Latte", Toast.LENGTH_SHORT).show()
            R.id.img_sb2 -> Toast.makeText( context: this@MainActivity,
                text: "MMM Chocco Frapp", Toast.LENGTH_SHORT).show()
            R.id.img_sb3 -> Toast.makeText( context: this@MainActivity,
                text: "MMM Bottled Americano", Toast.LENGTH_SHORT).show()
            R.id.img_sb4 -> Toast.makeText( context: this@MainActivity,
                text: "MMM Rainbow Frapp", Toast.LENGTH_SHORT).show()
            R.id.img_sb5 -> Toast.makeText( context: this@MainActivity,
                text: "MMM Caramel Frapp", Toast.LENGTH_SHORT).show()
            R.id.img_sb6 -> Toast.makeText( context: this@MainActivity,
                text: "MMM Black Forest Frapp", Toast.LENGTH_SHORT).show()
        }
    }
```

Figure 143. Implemented onClick Method

This might not feel like much of a win right now, but when the event handlers get more complex later, you will appreciate this improvement!

5.2 Creating the Order Class

We are going to optimise our code to include an `Order` class which we will use to store our `productName` for now. We also use this class later when we push our data to a Firebase database. You can create the order Class shown in Figure 144.

```
1  package za.ac.iie.opsc7311.starsucks
2
3  class Order {
4      lateinit var productName: String
5      lateinit var customerName: String
6      lateinit var customerCell: String
7      lateinit var orderDate: String
8  }
```

Figure 144. Order Class

The four properties declared here are public, even though that is not explicitly stated here. But we don't need to make these private to maintain encapsulation. The getters and setters are automatically created, just like in C#. So, there is no need for get and set methods. (Agrawal, n.d.)

We do still want a few different constructors that we can use, depending on what data we have available when we create the instance. The syntax for constructors is probably the strangest thing if you are used to Java or C#.

Each Kotlin class has a primary constructor and (potentially) secondary constructors. The primary constructor forms part of the header of the class. (Programiz, 2022)


```
package za.ac.iie.opsc7311.starsucks

// primary constructor - no parameters
class Order() {
    lateinit var productName: String
    lateinit var customerName: String
    lateinit var customerCell: String
    lateinit var orderDate: String

    // secondary constructor
    constructor(pName: String) : this() {
        productName = pName
    }

    // secondary constructor
    constructor(pName: String, cName: String, cCell: String,
                oDate: String) : this(pName) {
        customerName = cName
        customerCell = cCell
        orderDate = oDate
    }
}
```

Figure 145. Order Class with Constructors

5.3 Understanding Intents

We are writing this application to allow users to order beverages from Starsucks. This means that we would like to gather data on one activity and then send this data to another activity where we would like to do something with it. In our case we want to pass the product that we selected on our first activity to the second activity to gather all the order details that we need. We need to know:

- What the customer ordered
- Who the customer is (so that we can spell it wrong 😊)
- When the customer wants it delivered (functionality that will be added later)
- The customer's cell number
- The price of the product (functionality that will be added later)

Our app currently gets the product name from our first activity (`MainActivity`). Here is what we would like to happen:

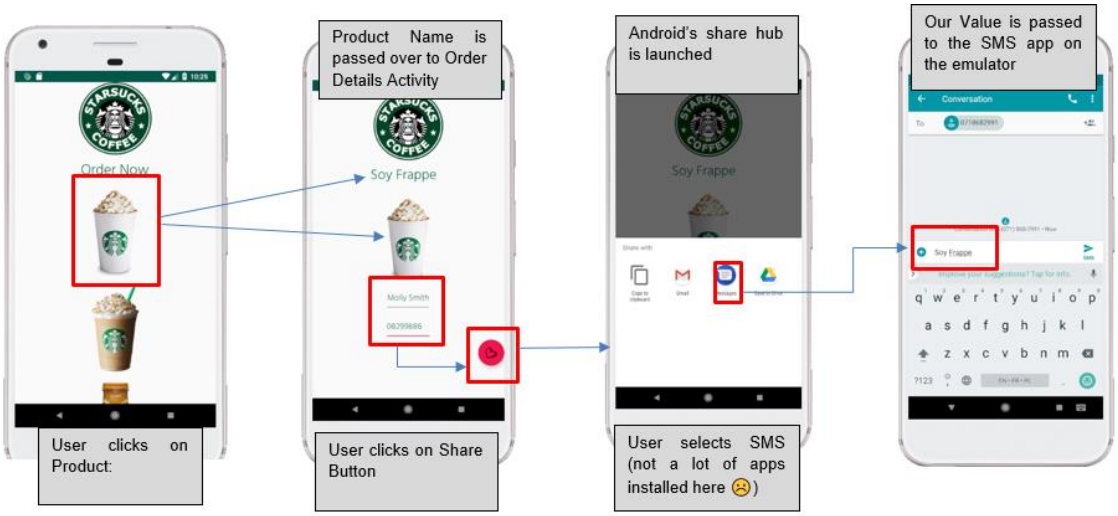


Figure 146. Flow of Activities

The process above is completed with `Intents`. Our first `Intent` will simply open a new activity and pass a single value over to the new activity.

Our second `Intent` will open the share hub and use a bundle to pass multiple values to which ever app choose to use to share our order.

We can use Intents and extras to share information quickly and easily between activities and applications.

You can read more about this here: https://developer.android.com/training/sharing/send [Accessed 17 November 2022].

Intents are not only used to share information, they are also used to open new activities, send broadcasts and to start services. You can read up more about this here: https://developer.android.com/guide/components/intents-filters [Accessed 17 November 2022].

There are two types of Intents namely explicit and implicit Intents. We will be using both. They can be described as:

Explicit Intents: Specify which application or activity they will open. We will be using an explicit Intent to open our Order Details activity and to pass our product to this Activity using an Explicit Intent. Explicit Intents are often used within your own application since you will know the class name of the Activity that you would like to open.

Implicit Intents: These Intents do not specify a specific application or Activity. They declare a general action instead. For example, we will create an Implicit Intent that will allow us to open the Android share sheet and share the product name the customer has selected as well as other information via any of the installed applications that we choose.
https://vogella.com/tutorials/AndroidIntent/article.html[Accessed 17 November 2022].

5.4 Adding the IntentHelper class

We will be using Intents a lot, and across all our activities. We have the option of writing the same line of code every time we need an intent, which if you think about it is a bit of a waste of time and it introduces unnecessary redundancy in our code. Good practice is to create a helper class. GeeksforGeeks.com provides the simplest description of a helper class:

Definition

"**Helper classes** contains functions that help in assisting the program. This Class intends to give quick implementation of basic functions such that programmers do not have to implement again and again." (sahilkhoslaa, 2018)

We are going to create a helper class that will help our application with Intents.

5.4.1 Creating an Explicit Intent

The first thing we need to accomplish is opening the Order Details activity using an explicit intent. You can go ahead and add a new Kotlin Class and call it `IntentHelper`.

```
1   package za.ac.iie.opsc7311.starsucks
2
3   import android.content.Context
4   import android.content.Intent
5
6   fun openIntent(context: Context, order: String,
7                  activityToOpen: Class<*>) {
8       // declare intent with context and class to pass the value to
9       val intent = Intent(context, activityToOpen)
10      // pass through the string value with key "order"
11      intent.putExtra( name: "order", order)
12      // start the activity
13      context.startActivity(intent)
14  }
```

Figure 147. openIntent() Helper Function

We will add our first function called `openIntent()`. This method accepts the `Context`, a `String` value and the `Class` to open (the activity class) as parameters.

Notice that the function isn't inside a class. Kotlin supports having functions that are visible in the package where they are declared. (Doan, 2020) And that is what we are doing here.

```
7116    public Intent(Context packageContext, Class<?> cls) {
7117        mComponent = new ComponentName(packageContext, cls);
7118    }
```

Figure 148. Intent constructor

Looking at the code of the intent class shown in Figure 147, we see that the second parameter is the class that we want to open. Kotlin makes use of generics here. But we don't know which class we will be passing to our function yet. So, the `Class<*>` indicates that it can be any class. (Foundation, 2022c)

5.4.2 What exactly is a Context and why do we need it?

You will see `Context` everywhere in Android development. There are lots of explanations on the internet as to what a `Context` really is. These can be incredibly confusing. The Android documentation defines a `Context` as:

> Context: "Interface to global information about an application environment. This is an abstract class whose implementation is provided by the Android system. It allows access to application-specific resources and classes, as well as up-calls for application-level operations such as launching activities, broadcasting and receiving intents, etc." (Android Open Source Project, 2022e)

A `Context` is often called a God object. A God object is an object that knows too much and does too much. That is because `Contexts` are so integral to Android Development. You can read more about `Context` here: https://www.freecodecamp.org/ news/mastering-android-context-7055c8478a22/ [Accessed 17 November 2022].

A very simple explanation is that a `Context` provides context to the newly added components of an Android application. It allows us to tell an activity where it is originating form or to tell a `Toast` message in which activity it should pop up. Don't let this break your head for now.

5.4.3 Creating an Implicit Intent (method)

We next need to create an Implicit Intent to share our order. We will use polymorphism to create two methods that will provide the functionality to use implicit Intents to our application. This is because we can send a single value with an Intent and an Extra (`putExtra`) or we can send multiple values with a Bundle. We will start by sending just a single value to any application the user selects form the Android share sheet. We will later optimise this code to send a bundle of data to the android app that we call with the Implicit Intent.

An Implicit `Intent` is created much the same as an Explicit `Intent`.


```
fun shareIntent(context: Context, order: String) {
    var sendIntent = Intent()
    // set the action to indicate what to do - send in this case
    sendIntent.setAction(Intent.ACTION_SEND)
    sendIntent.putExtra(Intent.EXTRA_TEXT, order)
    // we are sending plain text
    sendIntent.setType("text/plain")
    // show the share sheet
    var shareIntent = Intent.createChooser(sendIntent,  title: null)
    context.startActivity(shareIntent)
}
```

Figure 149. Creating the Implicit Intent

The MIME type defines the format of our Data – plain Text in our case. You can read more about MIME types here: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types

Finally we will create the overloaded `shareIntent()` method. This method accepts all the same parameters as the original `shareIntent()` method, but adds the `customerName` and `customerCell` as further data that we would like to share.

```
fun shareIntent(context: Context, order: Order) {
    var sendIntent = Intent()
    sendIntent.setAction(Intent.ACTION_SEND)

    // create a bundle and add multiple values to it
    var shareOrderDetails = Bundle()
    shareOrderDetails.putString("productName", order.productName)
    shareOrderDetails.putString("customerName", order.customerName)
    shareOrderDetails.putString("customerCell", order.customerCell)

    // share the whole bundle
    sendIntent.putExtra(Intent.EXTRA_TEXT, shareOrderDetails)
    sendIntent.setType("text/plain")

    var shareIntent = Intent.createChooser(sendIntent,  title: null)
    context.startActivity(shareIntent)
}
```

Figure 150. Overloaded shareIntent Method

5.5 Using the Intents

Now we can make use of the `IntentHelper` and `Order` classes. Declare a property of type `Order` in the `MainActivity` class.

```
class MainActivity : AppCompatActivity(), View.OnClickListener {

    var order = Order()
```

Figure 151. Declaring an order

We can now update the `onClick()` method to make use of the order and the `IntentHelper`.

```
override fun onClick(v: View?) {
    when(v?.id) {
        R.id.img_sb1 -> order.productName = "Soy Latte"
        R.id.img_sb2 -> order.productName = "Chocco Frapp"
        R.id.img_sb3 -> order.productName = "Bottled Americano"
        R.id.img_sb4 -> order.productName = "Rainbow Frapp"
        R.id.img_sb5 -> order.productName = "Caramel Frapp"
        R.id.img_sb6 -> order.productName = "Black Forest Frapp"
    }
    Toast.makeText( context: this@MainActivity,
        text: "MMM " + order.productName, Toast.LENGTH_SHORT).show()
    openIntent(applicationContext, order.productName,
        OrderDetailsActivity::class.java)
}
```

Figure 152. Using the Order and IntentHelper

If you run the app now, our new activity will be opened when you tap on one of the products. But it won't do anything with the data yet.

5.6 Adding Logic to OrderDetailsActivity

We are now ready to code up our next activity. We will make use of View Binding again to get access to all the user interface elements on this activity.


```
package za.ac.iie.opsc7311.starsucks

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import za.ac.iie.opsc7311.starsucks.databinding.ActivityOrderDetailsBinding

class OrderDetailsActivity : AppCompatActivity() {
    var order = Order()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val binding = ActivityOrderDetailsBinding.inflate(layoutInflater)
        setContentView(binding.root)

        // get the name of the ordered product from the intent
        order.productName = intent.getStringExtra( name: "order").toString()

        // set the product name on the text view
        binding.tvPlacedOrder.text = order.productName
    }
}
```

Figure 153. Using View Binding to access the controls

Have a close look at the last two lines of code in Figure 153. We can get the value that we passed through from the intent, by calling the `getStringExtra()` method.

The last thing we need to do, at the end of the `onCreate` method, is to change the image based on what the customer picked.

```
    when(order.productName) {
        "Soy Latte" -> binding.imgOrderedBeverage.setImageResource(R.drawable.sb1)
        "Chocco Frapp" -> binding.imgOrderedBeverage.setImageResource(R.drawable.sb2)
        "Bottled Americano" -> binding.imgOrderedBeverage.setImageResource(R.drawable.sb3)
        "Rainbow Frapp" -> binding.imgOrderedBeverage.setImageResource(R.drawable.sb4)
        "Caramel Frapp" -> binding.imgOrderedBeverage.setImageResource(R.drawable.sb5)
        "Black Forest Frapp" -> binding.imgOrderedBeverage.setImageResource(R.drawable.sb6)
    }
```

Figure 154. Changing the Image

If we run the app now, and click on one of the products, the order details will be displayed correctly including the picture for the product.

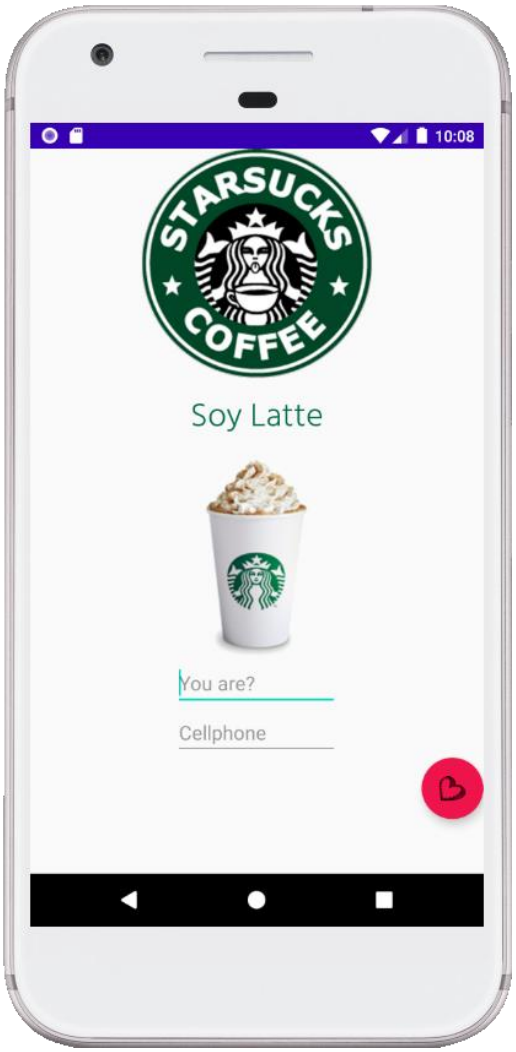


Figure 155. Order Details in the App

5.7 Sharing the Order

The last part of the user interface that we haven't implemented yet, is the floating action button's action. We will use it to share our order using external apps installed on the phone. To do that, we will make use of the implicit intent methods in the `IntentHelper`.

We can add an `OnClickListener` to it just like any button.

```
binding.fabOrder.setOnClickListener() {  it: View!
    shareIntent(applicationContext, order.productName)
}
```

Figure 156. Adding the OnClickListener

Now when we tap the `FloatingActionButton`, the Android share sheet is show, where the user can choose which app to use for sharing.

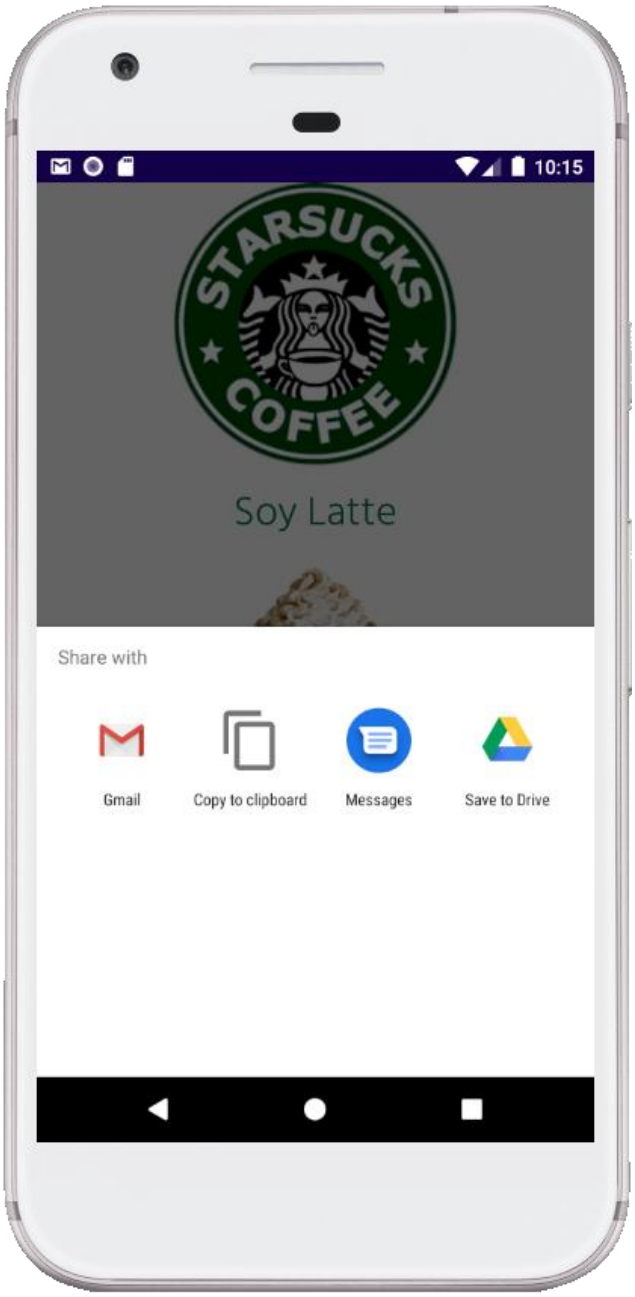


Figure 157. Android Share Sheet

5.8 Navigation Drawer

A navigation drawer is a menu that pops in from the side of the app, that can be used to access various areas of our app. Right now, we only have the `MainActivity`, and from that the `OrderDetailsActivity` is opened. But we want to add another activity where the user can take a photo of their drink. And when we get there, we are going to need some way to navigate between these different activities.

When a new app is created, there is an option for creating a **Navigation Drawer Activity** as the starting point for the app. But the code generated for that is fairly complex, so let's see how to implement this from scratch.

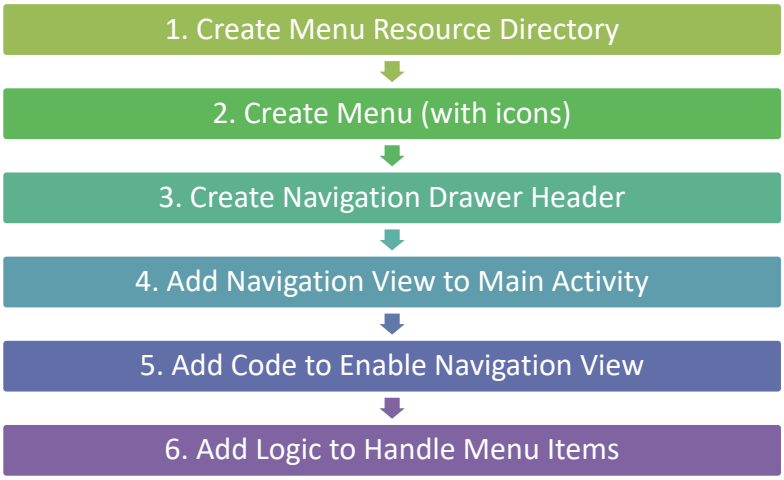


Figure 158. Steps for Adding a Navigation Drawer

The six steps above are described in the following subsections.

5.8.1 Create Menu Resource Directory

To create the menu resource directory:

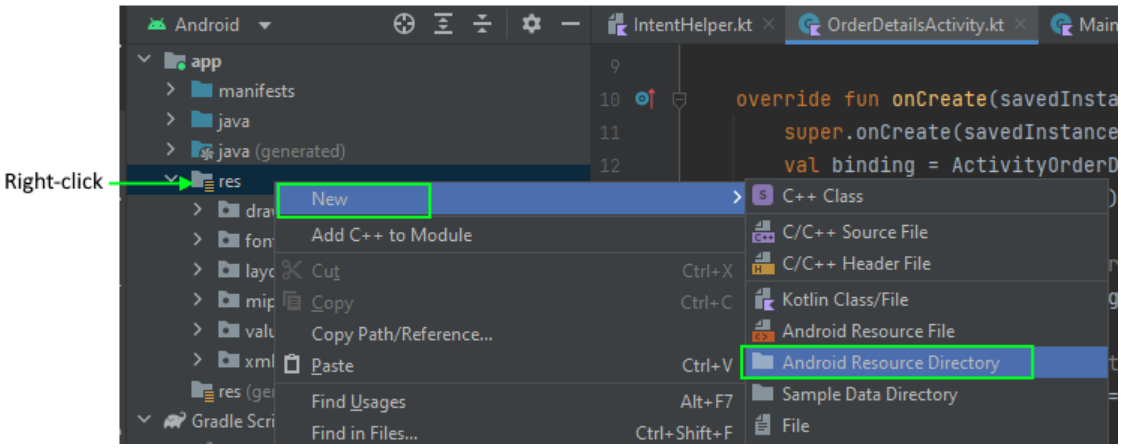


Figure 159. Adding the Resource Directory

1. Right-click on the res folder in your app project and click **New** followed by **Android Resource Directory**.

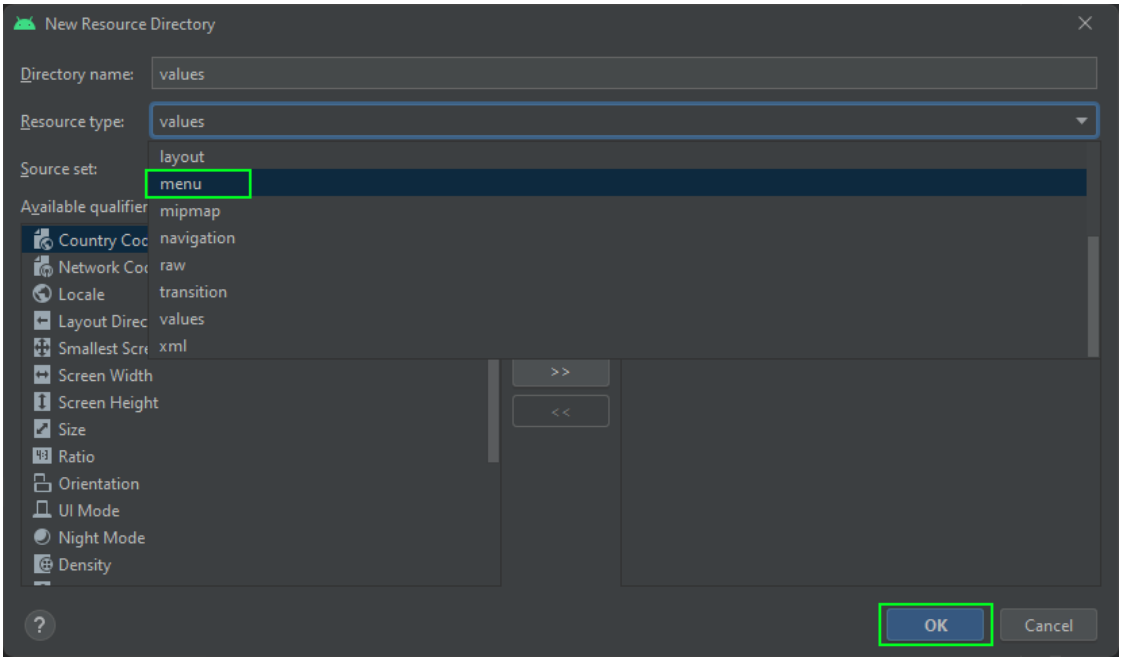


Figure 160. Selecting the Resource Type

2. On the **New Resource Directory** dialog, select the **Resource Type** *menu*.
3. Click **OK**.

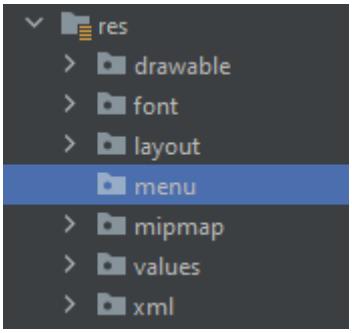


Figure 161. Menu Folder Created

This will create the `menu` folder under `res`, as shown in Figure 161.

5.8.2 Create Menu (With Icons)

Next, we need to create the menu. It is the XML file that tells the navigation drawer which items need to be displayed on the menu.

To create the menu:

1. Right-click on the menu folder that we just created. Click **New** and then **Menu Resource File** (see Figure 162).

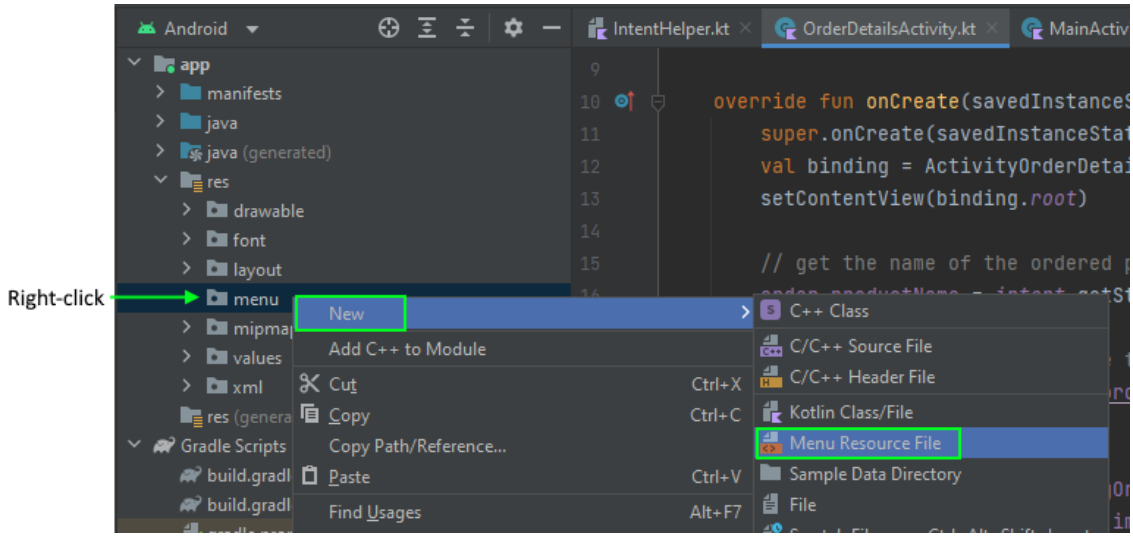


Figure 162. Creating the Menu Resource File

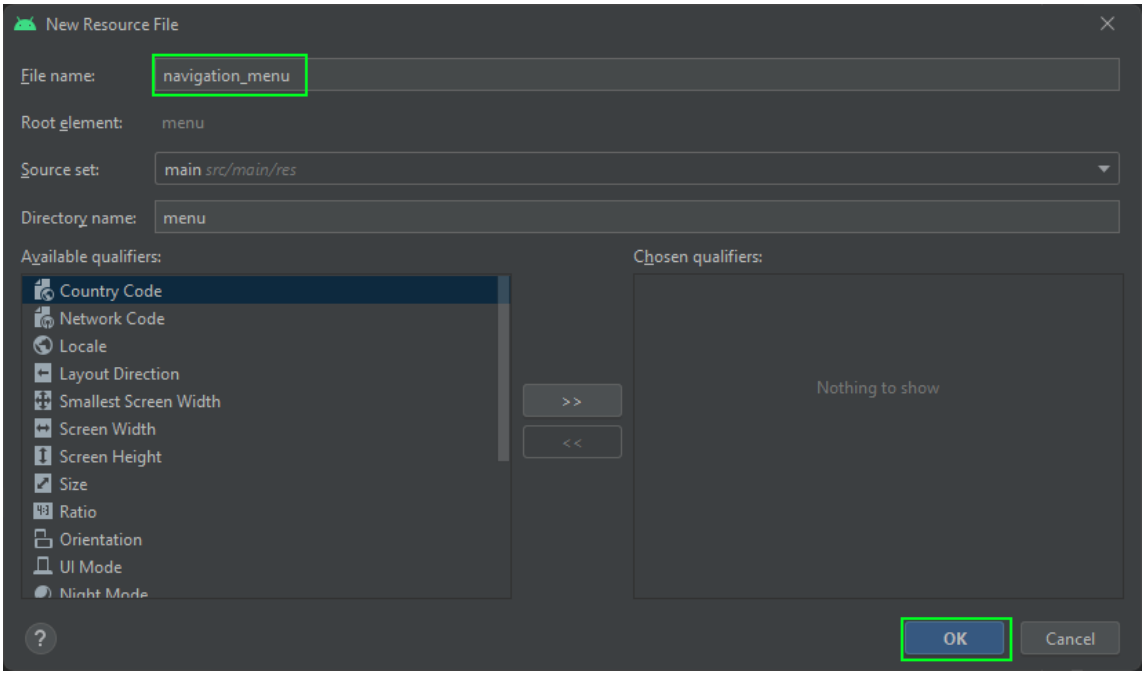


Figure 163. Entering the File Name

2. On the **New Resource File** dialog, enter the **File name** *navigation_menu*.
3. Click **OK**.

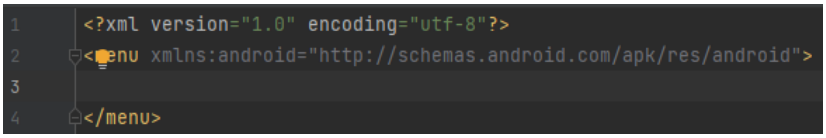


Figure 164. Empty Menu File

You will see the design view for the menu, and the XML file with only the top-level menu tag created can be viewed using the Code view. Next, we will create the icons. And then we will return here to add the menu items.

We are going to need two icons: one for the main page where we are placing our order, and one for the photo activity that we will add later.

To create an icon:

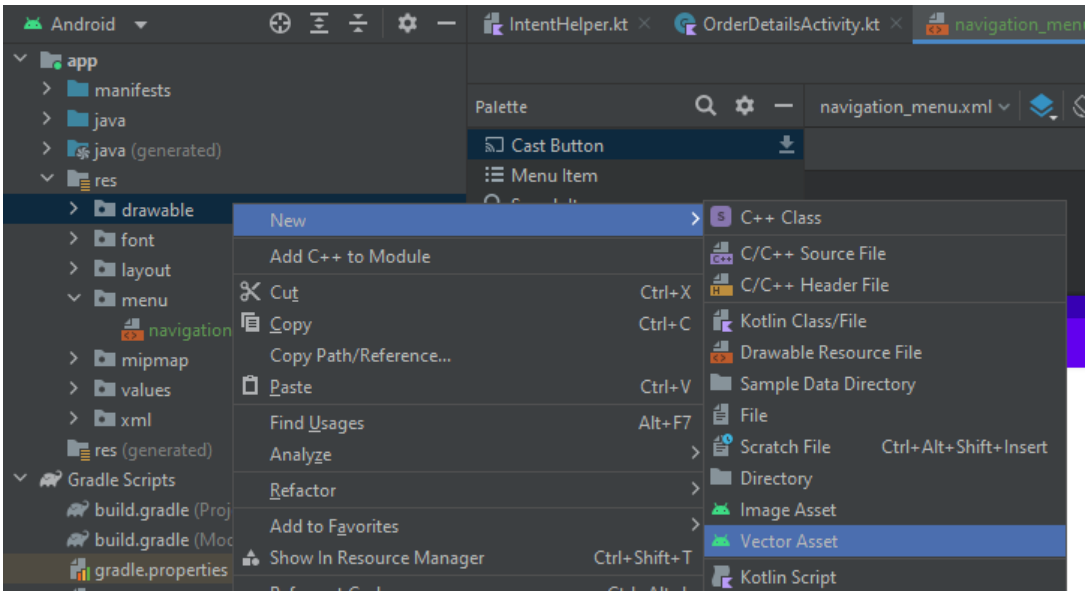


Figure 165. Creating a vector asset

1. Right-click on the `drawable` folder, click **New** and then **Vector Asset**.

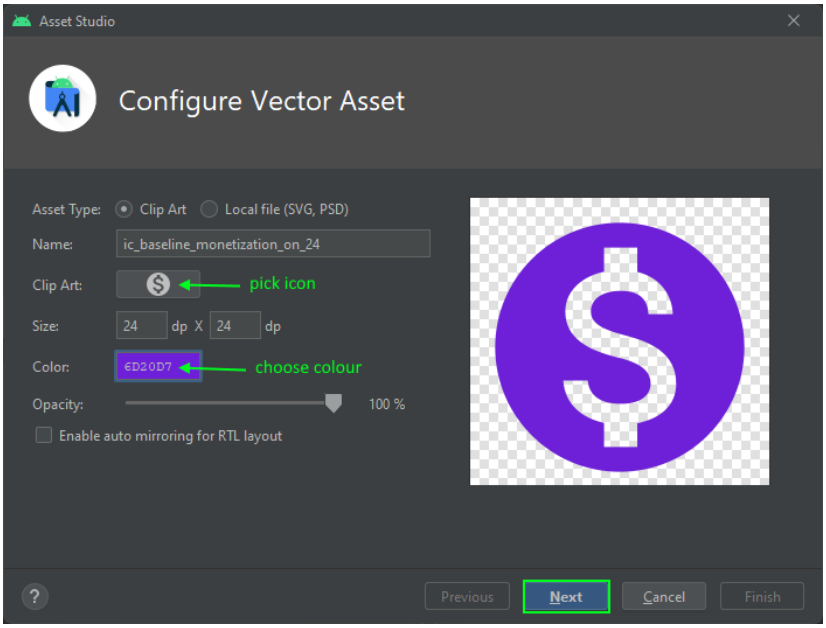


Figure 166. Editing the Vector Asset

2. Enter the **name** ic_menu_main for the icon name.
3. Pic an icon by clicking on the image next to **Clip Art**.
4. Choose a **colour** for the icon.
5. Click **Next**.

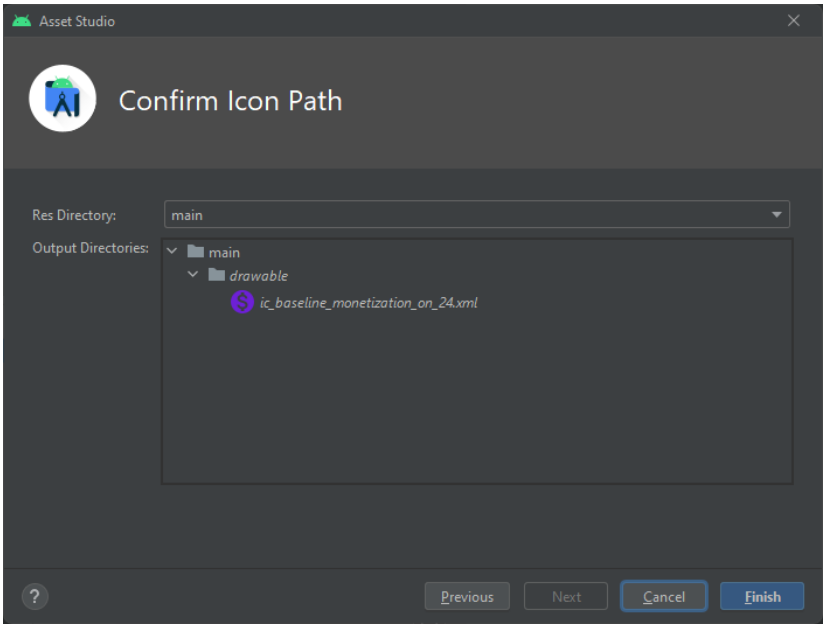


Figure 167. Confirm the Location

6. Check the directory where the resource will get created – it should be in drawable. Then click `Finish`.

Create another vector asset for the photo activity and call it `ic_menu_photo`.

Now we can get back to creating the menu structure. This can be done either in the XML or the Design view. Here we will describe how to do this using the Design view. So, open the navigation_menu.xml file, and switch to the Design view (top-right corner).

To create the menu structure:

1. Drag a Group from the Pallet into the Component Tree (see Figure 168).
2. Set the `checkableBehaviour` attribute of the group to `single`. This is to ensure that only one page at a time will be highlighted in the menu.

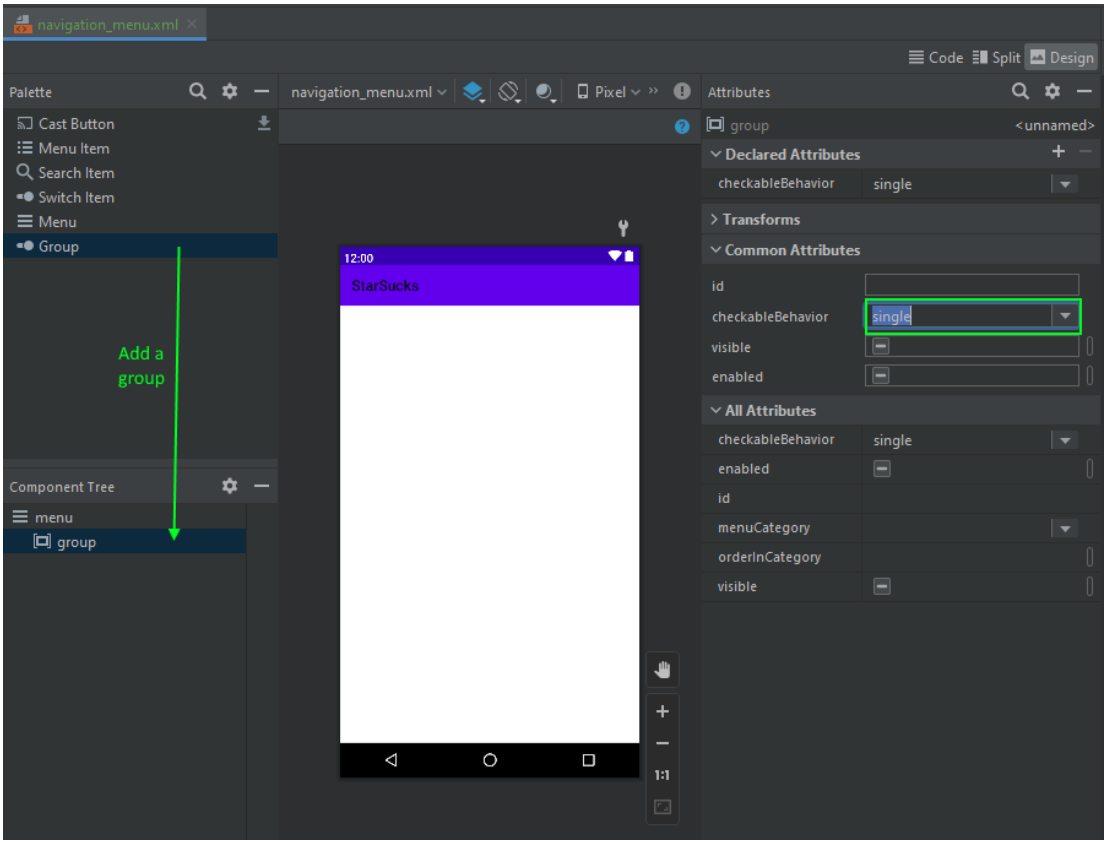


Figure 168. Adding the Group

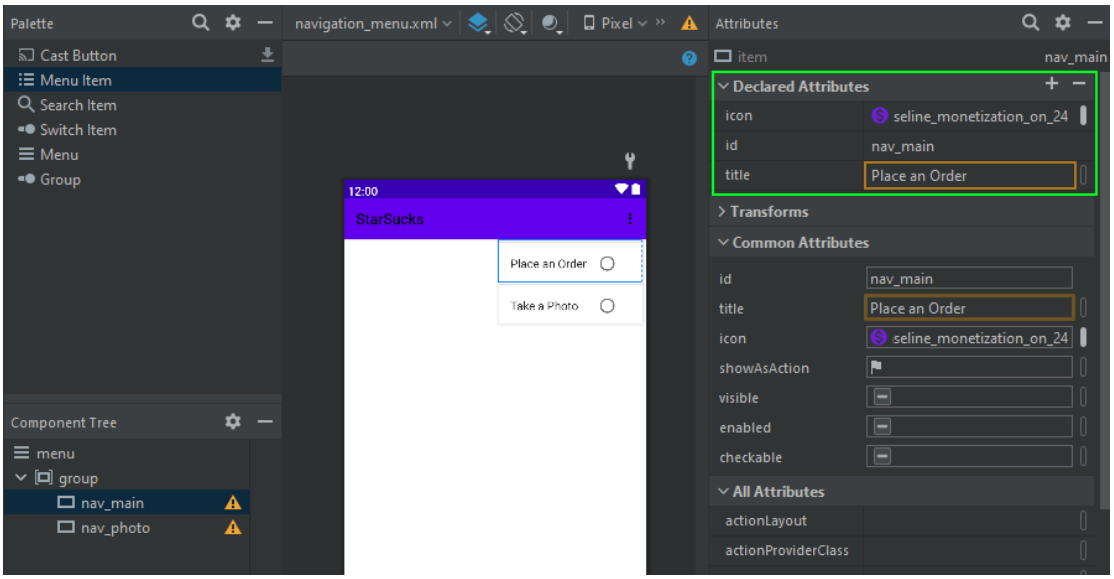


Figure 169. Adding Menu Items

3. Add two **menu items** to the group – one for the main activity and one for the pending photo activity.
4. Set the `id`, `title` and `icon` for each of the items.

5.8.3 Create the Navigation Drawer Header

At the top of a navigation drawer, there is an area where you display something graphical to make the menu look good. We are going to display the StarSucks logo there.

We are going to create a layout resource file, that works just like the layouts that we have used for activities so far.

To create the navigation drawer header:

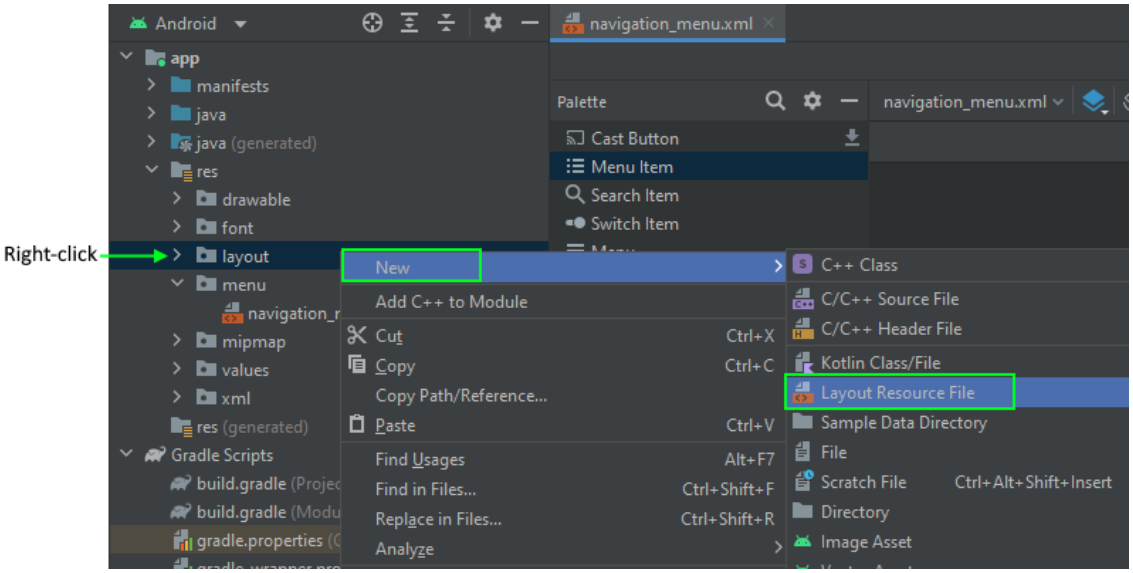


Figure 170. Adding a Layout Resource File

1. Right-click on the `layout` folder, click **New** and then click **Layout Resource File**.

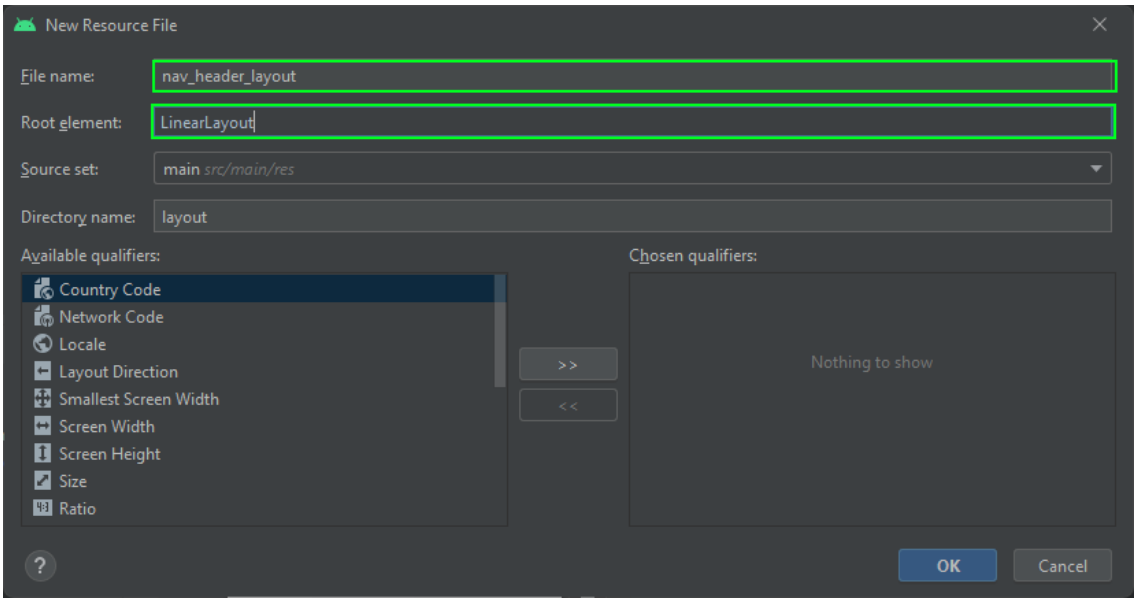


Figure 171. Setting the Resource File Properties

2. Enter the **File name** `nav_header_layout`.
3. Change the **Root element** to `LinearLayout`.
4. Click **OK**. A layout file will be created where you can add all the components like usual.
5. Add an `ImageView` that displays the Starsucks logo.

5.8.4 Add Navigation View to Main Activity Layout

Now we have all the components created, and we can use these now to add the Navigation View to the `MainActivity`.

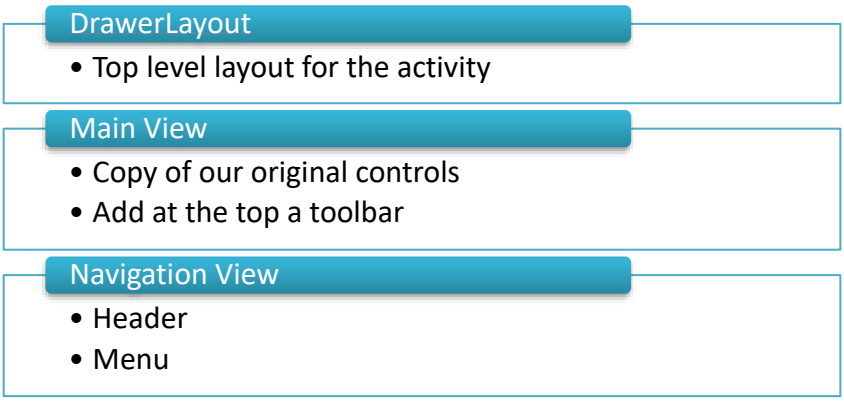


Figure 172. Components We Need on the MainActivity

Figure 172 shows all the components that we need eventually on the `MainActivity`.

We are going to implement the changes using the Layout Editor. The safest way to do that, is to create a new layout and copy the existing controls into that. At the end you can then delete the original layout.

Create a new layout now, following the same steps that we did for the navigation drawer header, and call it `activity_main_with_nav_drawer`.

To build the navigation view:

1. **Convert** the top level component from `ConstraintLayout` to `androidx.drawerlayout.widget.DrawerLayout`.

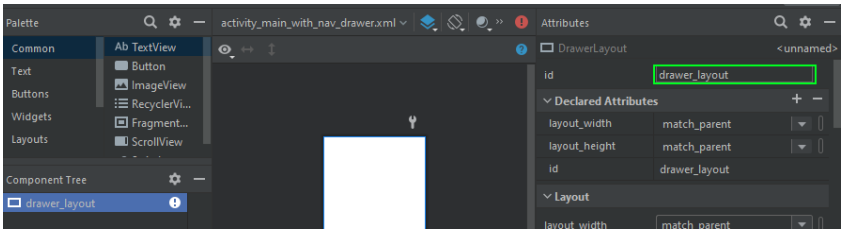


Figure 173. Setting the ID

2. Set the id for the top-level component to `drawer_layout`.

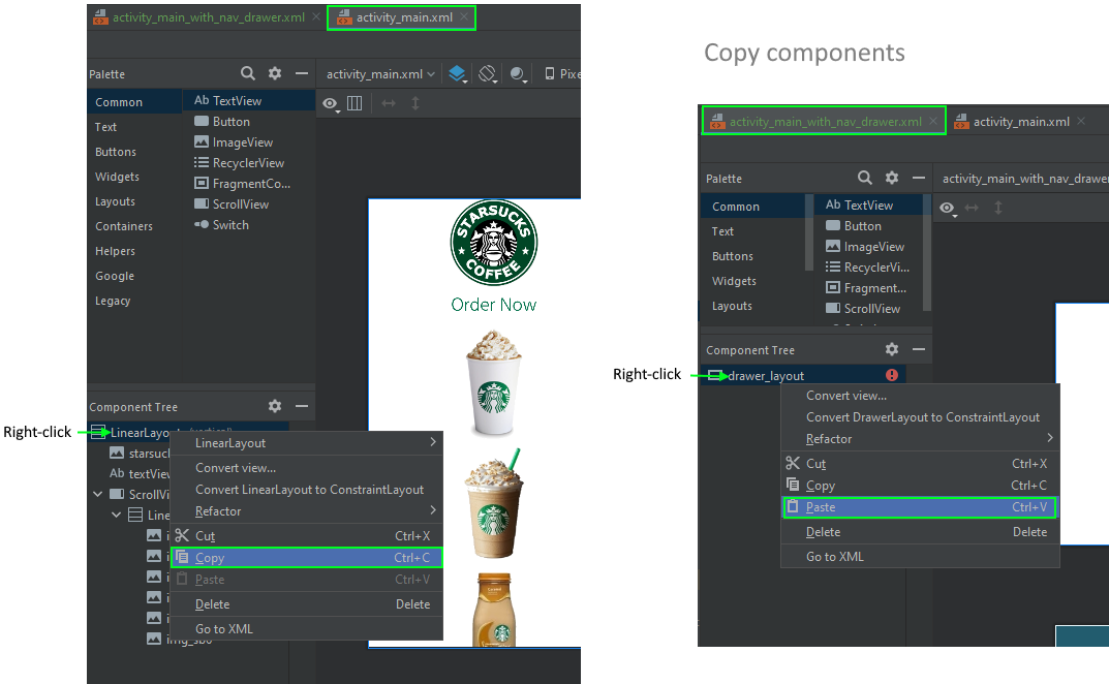


Figure 174. Copy and Paste Previous Layout's Components

3. Copy and paste the components, starting from the top-level element, into the new layout. The component tree should look like Figure 175.

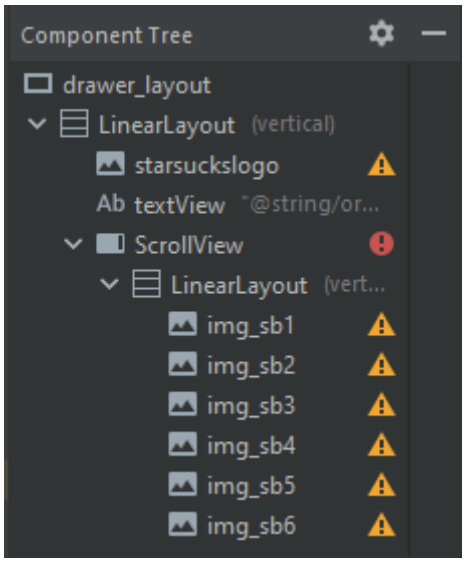


Figure 175. After the Copy

4. Drag and drop a `NavigationView` above the `LinearLayout` (see Figure 176). Hint: use the search function on the palette.
5. Set the id for the `NavigationView` to `nav_view`.
6. Set the `fitsSystemWindows` attribute to `true`.
7. Choose the `headerLayout` that we created earlier.
8. Choose the `menu` that we created earlier.

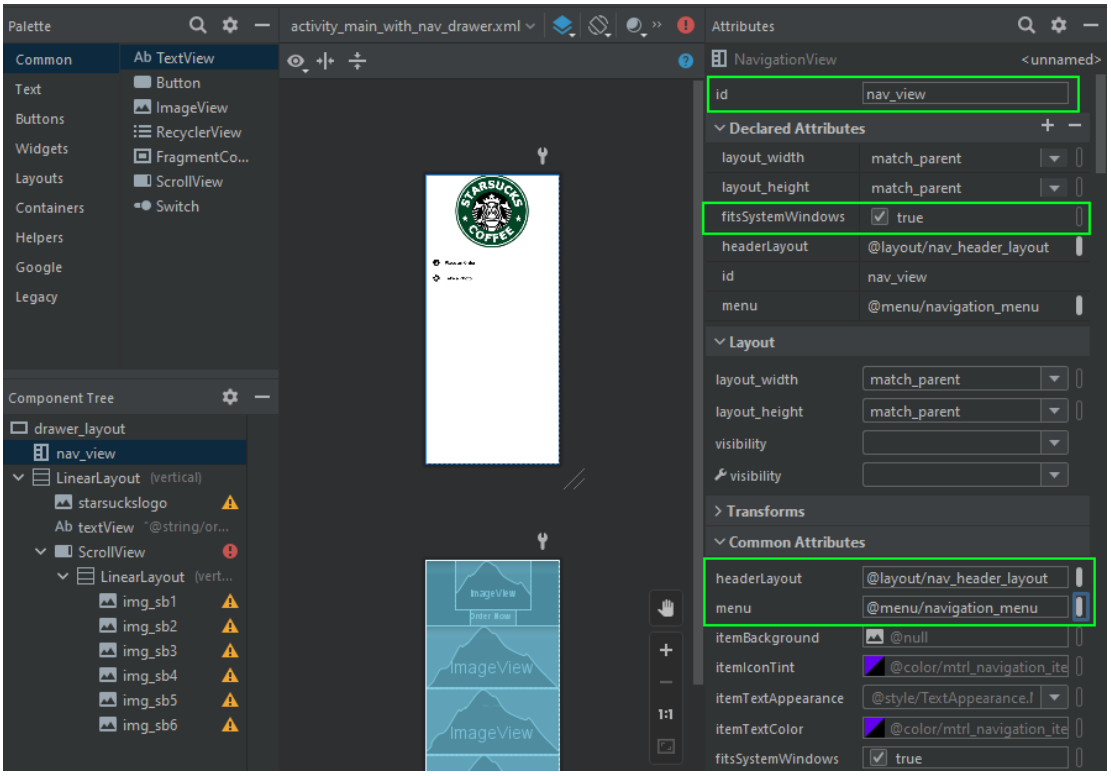


Figure 176. Adding the Navigation View

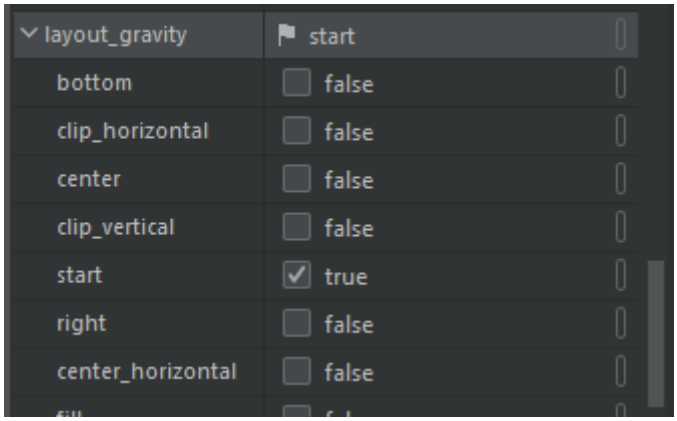


Figure 177. Layout Gravity

Tip: If you don't see the content of the LinearLayout after this, check that the layout_gravity isn't set for the LinearLayout. It should not be.

9. Set the `layout_gravity` attribute as shown in Figure 177.
10. Add a `Toolbar` above the logo and change its ID to `nav_toolbar`.

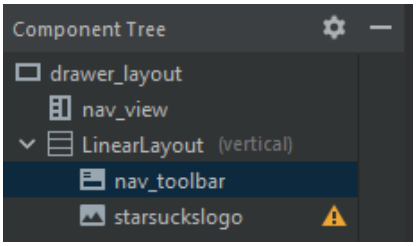


Figure 178. Adding the Toolbar

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    val binding = ActivityMainWithNavDrawerBinding.inflate(layoutInflater)
    setContentView(binding.root)
```

Figure 179. Updating the Main Activity

11. Open the `MainActivity` Kotlin class and change the code that creates the binding as shown in Figure 179.

If you run the app now, you will see the toolbar but not yet the menu or the navigation drawer. We still need to add a little code for that. If you see your layout and it works, you can now safely delete the original layout file.

5.8.5 Add Code to Enable the Navigation Drawer

To add the code that will enable the Navigation Drawer to work:

```
<resources>
    <string name="app_name" translatable="false">StarSucks</string>
    <string name="order_now_label" translatable="true">Order Now</string>
    <string name="navigation_drawer_open">Open navigation drawer</string>
    <string name="navigation_drawer_close">Close navigation drawer</string>
</resources>
```

Figure 180. New String Resources

1. In `strings.xml`, add the two new strings shown in Figure 180.
2. To the MainActivity Kotlin class, add the code highlighted in green in Figure 182.

If you run the app now, you will see that the navigation drawer can appear and be hidden again.

Figure 181. Navigation Drawer in the App


```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    val binding = ActivityMainWithNavDrawerBinding.inflate(layoutInflater)
    setContentView(binding.root)

    binding.imgSb1.setOnClickListener(this)
    binding.imgSb2.setOnClickListener(this)
    binding.imgSb3.setOnClickListener(this)
    binding.imgSb4.setOnClickListener(this)
    binding.imgSb5.setOnClickListener(this)
    binding.imgSb6.setOnClickListener(this)

    setSupportActionBar(binding.navToolbar)
    supportActionBar?.setDisplayHomeAsUpEnabled(true)
    supportActionBar?.setDisplayShowHomeEnabled(true)
    var toggleOnOff = ActionBarDrawerToggle( activity: this,
        binding.drawerLayout, binding.navToolbar,
        R.string.navigation_drawer_open,
        R.string.navigation_drawer_close)
    binding.drawerLayout.addDrawerListener(toggleOnOff)
    toggleOnOff.syncState()
}
```

Figure 182. Code to Enable the Navigation Drawer

The menu now appears. But if the back button is pressed, the menu is not closed as you would expect. Instead, the app is exited. Let's see how we can fix that.

In the `MainActivity` Kotlin class, we need to override the `onBackPressed` method. But now we will need to access the binding there too. So, let's create a property for that.

```
class MainActivity : AppCompatActivity(), View.OnClickListener {

    var order = Order()
    private lateinit var binding: ActivityMainWithNavDrawerBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainWithNavDrawerBinding.inflate(layoutInflater)
        setContentView(binding.root)
```

Figure 183. Creating a property for binding

```
override fun onBackPressed() {
    // if the drawer is open, close it
    if (binding.drawerLayout.isDrawerOpen(GravityCompat.START)) {
        binding.drawerLayout.closeDrawer(GravityCompat.START)
    } else {
        // otherwise, let the super class handle it
        super.onBackPressed()
    }
}
```

Figure 184. Overriding `onBackPressed()`

5.8.6 Add Logic to Handle Menu Items

The menu looks lovely now, but it does not **do** anything yet. Now the time has come to handle the menu actions. We need to implement another listener for that.

To implement the menu navigation:

```
13  class MainActivity : AppCompatActivity(), View.OnClickListener,
14          NavigationView.OnNavigationItemSelectedListener {
15
16      override fun onNavigationItemSelected(item: MenuItem): Boolean {
17          TODO( reason: "Not yet implemented")
18      }
```

Figure 185. Implementing NavigationView.OnNavigationItemSelectedListener

1. Change `MainActivity` to also implement the interface `NavigationView.OnNavigationItemSelectedListener`

```
67      override fun onNavigationItemSelected(item: MenuItem): Boolean {
68          // TODO: navigate to photo activity!
69          when(item.itemId) {
70  //              R.id.nav_photo ->
71          }
72          binding.drawerLayout.closeDrawer(GravityCompat.START)
73          // returning true marks the item as selected
74          return true
75      }
```

Figure 186. Implementing the Selection Listener

2. Override the `onNavigationItemSelected` method as shown in Figure 186.
3. Now we need to set the listener on the NavigationView. Add the code shown in Figure 187.

The menu item doesn't have anything to do yet – the photo activity will be created next.

```kotlin
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ActivityMainWithNavDrawerBinding.inflate(layoutInflater)
    setContentView(binding.root)

    binding.imgSb1.setOnClickListener(this)
    binding.imgSb2.setOnClickListener(this)
    binding.imgSb3.setOnClickListener(this)
    binding.imgSb4.setOnClickListener(this)
    binding.imgSb5.setOnClickListener(this)
    binding.imgSb6.setOnClickListener(this)

    setSupportActionBar(binding.navToolbar)
    supportActionBar?.setDisplayHomeAsUpEnabled(true)
    supportActionBar?.setDisplayShowHomeEnabled(true)
    var toggleOnOff = ActionBarDrawerToggle( activity: this,
        binding.drawerLayout, binding.navToolbar,
        R.string.navigation_drawer_open,
        R.string.navigation_drawer_close)
    binding.drawerLayout.addDrawerListener(toggleOnOff)
    toggleOnOff.syncState()

    binding.navView.bringToFront()
    binding.navView.setNavigationItemSelectedListener(this)
}
```

Figure 187. Setting the Listener

5.9 Working with Camera Intents and Permissions

We are now going to add an activity that allows us to open the camera from within our application. We will take a photo and store display the photo in our app.

5.9.1 Creating the User Interface

Create a new Empty activity and call it `CoffeeSnapsActivity`.

Add the following layout to your new activity:

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.drawerlayout.widget.DrawerLayout 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=".CoffeeSnapsActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <androidx.appcompat.widget.Toolbar
            android:id="@+id/nav_toolbar"
```



```
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:theme="@style/ThemeOverlay.AppCompat.Light"
            app:popupTheme="@style/ThemeOverlay.AppCompat.Light"/>

        <FrameLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"/>

    </LinearLayout>

    <com.google.android.material.navigation.NavigationView
        android:id="@+id/nav_view"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        app:headerLayout="@layout/nav_header_layout"
        app:menu="@menu/navigation_menu">
    </com.google.android.material.navigation.NavigationView>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <ImageView
            android:layout_width="189dp"
            android:layout_height="189dp"
            android:layout_gravity="center_horizontal"
            android:src="@drawable/starsuckslogo">
        </ImageView>

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:fontFamily="@font/hind_guntur_light"
            android:textColor="@color/starsucksGreen"
            android:textSize="20dp"
            android:padding="5dp"
            android:text="SMILE :-)">
        </TextView>

        <androidx.cardview.widget.CardView
            android:layout_width="270dp"
            android:layout_height="167dp"
            android:layout_gravity="center_horizontal"
            android:layout_marginTop="20dp">

            <ImageView
                android:id="@+id/img_cameraImage"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:src="@mipmap/ic_launcher_round">
            </ImageView>

        </androidx.cardview.widget.CardView>

        <com.google.android.material.floatingactionbutton.FloatingActionButton
            android:id="@+id/photoFab"
            android:layout_marginTop="20dp"
            android:layout_gravity="center_horizontal"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:clickable="true"
            android:src="@drawable/ic_menu_photo"
```



```
            app:backgroundTint="@color/starsucksRed">
        </com.google.android.material.floatingactionbutton.FloatingActionButton>

    </LinearLayout>

</androidx.drawerlayout.widget.DrawerLayout>
```

Now we can update the navigation code in the `MainActivity` to open our new activity:

```
override fun onNavigationItemSelected(item: MenuItem): Boolean {
    when(item.itemId) {
        R.id.nav_photo -> openIntent(applicationContext, order: "",
                                     CoffeeSnapsActivity::class.java)
    }
    binding.drawerLayout.closeDrawer(GravityCompat.START)
    // returning true marks the item as selected
    return true
}
```

Figure 188. Opening the Activity

5.9.2 Taking a Photo using an Intent

We can now add the code to `CoffeeSnapsActivity` that takes a photo and displays it using the `ImageView`.

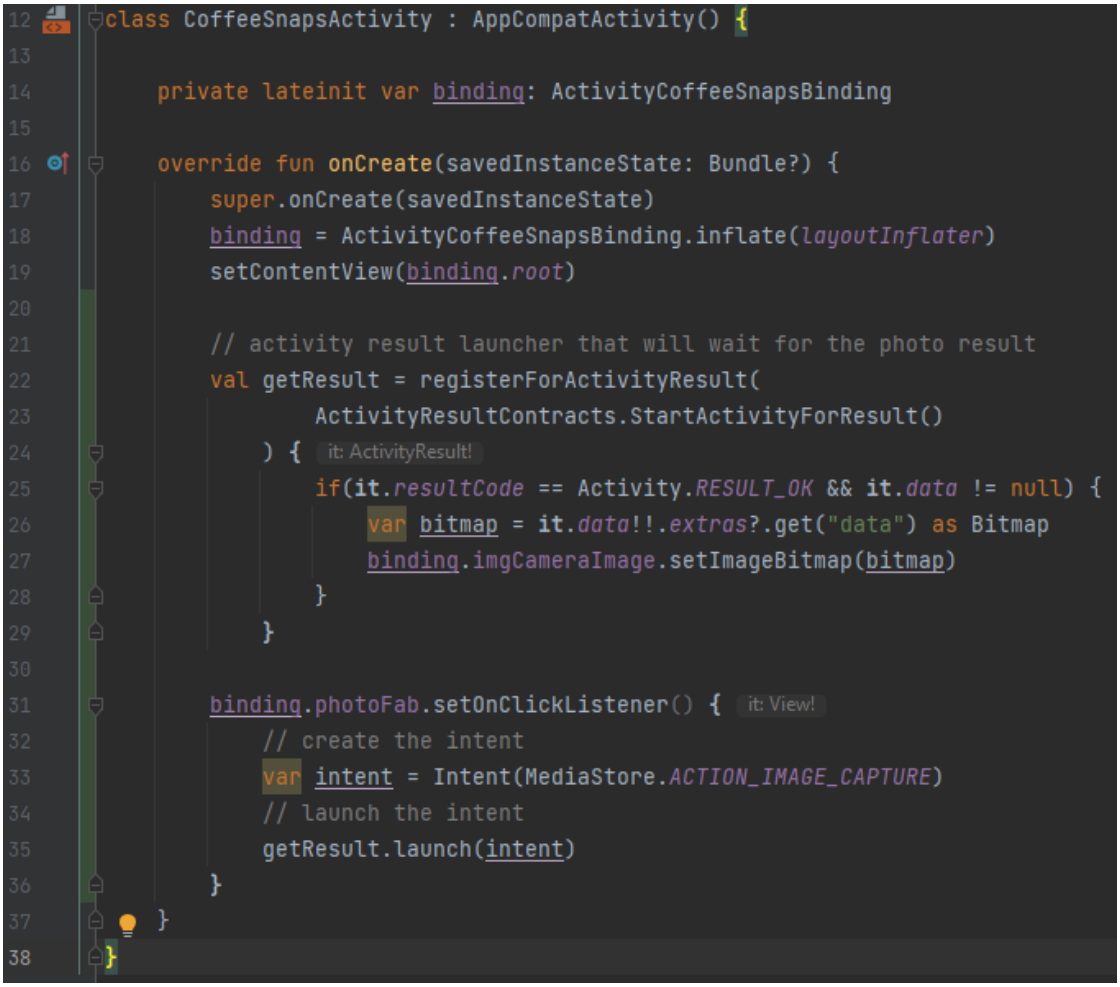
There are two ways of taking a photo in Android: using an intent or making use of a camera API such as CameraX. Since we are working with Intents right now, we will stick to that for a start. But if you want more control, use the CameraX API instead. (Ndonga, 2021)

```
class CoffeeSnapsActivity : AppCompatActivity() {

    private lateinit var binding: ActivityCoffeeSnapsBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityCoffeeSnapsBinding.inflate(layoutInflater)
        setContentView(binding.root)
    }
```

Figure 189. Access components using View Binding

Use View Binding to get hold of the components. Now we can add an `OnClickListener` for the `FloatingActionButton` that will make use of an `Intent` to take the photo.

Many examples that you will find online will make use of the method `startActivityForResult`. But if you look closely at the code, you will notice that Android Studio will warn that the method is deprecated. Read more about the new way of calling an intent in (Sharma, 2021)



```
class CoffeeSnapsActivity : AppCompatActivity() {

    private lateinit var binding: ActivityCoffeeSnapsBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityCoffeeSnapsBinding.inflate(layoutInflater)
        setContentView(binding.root)

        // activity result launcher that will wait for the photo result
        val getResult = registerForActivityResult(
                ActivityResultContracts.StartActivityForResult()
            ) { it: ActivityResult!
                if(it.resultCode == Activity.RESULT_OK && it.data != null) {
                    var bitmap = it.data!!.extras?.get("data") as Bitmap
                    binding.imgCameraImage.setImageBitmap(bitmap)
                }
            }

        binding.photoFab.setOnClickListener() { it: View!
            // create the intent
            var intent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
            // launch the intent
            getResult.launch(intent)
        }
    }
}
```

Figure 190. Adding the OnClickListener and result launcher

Taking a photo is an asynchronous request. We start an activity, that will be handled by something outside our app, and we will be notified later when the process is complete. The activity result launcher (here called `getResult`) is responsible for handling the response that is returned to us.

If we use the ACTION_IMAGE_CAPTURE intent, we don't need to ask for any permissions. (Murphy, 2020)

Note that there are some issues with Android 11 (API level 30 or later) with using this method. If the user disabled the built-in camera, this would not work as expected. Read more in (Murphy, 2020) So, let us change our implementation to use CameraX instead.

5.9.3 Asking for Permissions

If we used a Camera API instead, we would need to request the CAMERA permission. Let's have a quick look at that.

Permission Group	Permissions
android.permission-group.CALENDAR	• android.permission.READ_CALENDAR • android.permission.WRITE_CALENDAR
android.permission-group.CAMERA	• android.permission.CAMERA
android.permission-group.CONTACTS	• android.permission.READ_CONTACTS • android.permission.WRITE_CONTACTS • android.permission.GET_ACCOUNTS
android.permission-group.LOCATION	• android.permission.ACCESS_FINE_LOCATION • android.permission.ACCESS_COARSE_LOCATION
android.permission-group.MICROPHONE	• android.permission.RECORD_AUDIO
android.permission-group.PHONE	• android.permission.READ_PHONE_STATE • android.permission.CALL_PHONE • android.permission.READ_CALL_LOG • android.permission.WRITE_CALL_LOG • com.android.voicemail.permission.ADD_VOICEMAIL • android.permission.USE_SIP • android.permission.PROCESS_OUTGOING_CALLS
android.permission-group.SENSORS	• android.permission.BODY_SENSORS
android.permission-group.SMS	• android.permission.SEND_SMS • android.permission.RECEIVE_SMS • android.permission.READ_SMS • android.permission.RECEIVE_WAP_PUSH • android.permission.RECEIVE_MMS • android.permission.READ_CELL_BROADCASTS
android.permission-group.STORAGE	• android.permission.READ_EXTERNAL_STORAGE • android.permission.WRITE_EXTERNAL_STORAGE

Figure 191. Dangerous Permissions

Android has a built-in permission system that deals with Normal and Dangerous permissions (there are other types of permissions which are beyond the scope of this module manual). Normal permissions should not affect the user's privacy and are granted without requesting the user's explicit permission. Dangerous permissions can affect a user's privacy and needs the user to allow the functionality the permission protects before it can be used.

You can read more about Android Permissions here: https://developer.android.com/guide/topics/permissions/overview/ [Accessed 17 November 2022].

It is important to know which the dangerous permissions are, so you can ask the user if you need to access those features. Android considers the permissions in Figure 191 as dangerous.

For an example of requesting permissions, read https://www.simplifiedcoding.net/android-request-permission-at-runtime-example/ [Accessed 17 November 2022].

5.9.4 Taking a Photo using CameraX

This section is based on information from (Ndonga, 2021).

The first thing we need to do is add dependencies to the project. We are going to make use of a library that is not included by default in an Android app. The latest versions of the CameraX libraries are list in (Android Open Source Project, 2022f). We are going to use the latest stable release, which is 1.1.0 at the time of writing of this manual. Note that we are copying the dependencies from the Groovy tab.

Open the build.gradle file (the one for the Module, in our case StarSucks.app). The first thing we need to check is the minimum API version. Find the property under android > defcaultConfig > minSdkVersion and change it to 26 if it is less than that.

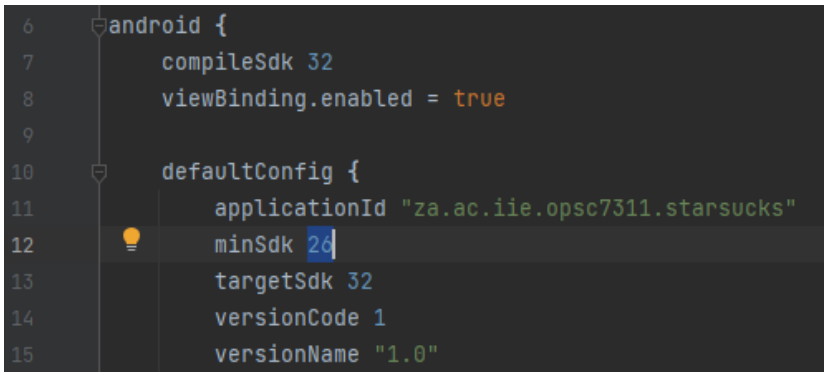


Figure 192. Changing the minSdkVersion to 26

Find the dependencies section – should be at the bottom of the file.


```
dependencies {

    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.appcompat:appcompat:1.5.1'
    implementation 'com.google.android.material:material:1.6.1'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'

    // CameraX core library using the camera2 implementation
    def camerax_version : String  = "1.1.0"
    implementation "androidx.camera:camera-camera2:${camerax_version}"
    implementation "androidx.camera:camera-lifecycle:${camerax_version}"
    implementation "androidx.camera:camera-view:${camerax_version}"
}
```

Figure 193. CameraX dependencies

Add the dependencies shown in Figure 193. And then click **Sync Now** at the top of the file so Gradle can download the necessary files.

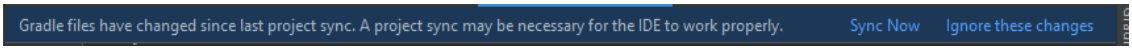


Figure 194. Run the Gradle Sync

The next thing we need to do, is to ask for permission to use the camera. Now we will no longer be able to get away without that.

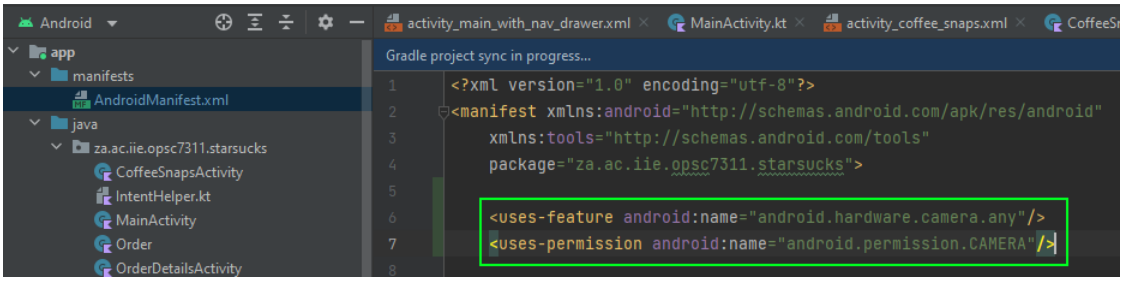


Figure 195. Permissions in the AndroidManifest.xml file

Under the manifests folder, open the AndroidManifest.xml file, and add the two lines shown in Figure 195.

Now we are ready to implement the new camera features. First, remove all the intent code that we added before. Then, we change the `ImageView` where we displayed the photo we took before, with a `PreviewView` as shown in Figure 196.

Run the app at this point. When you go to the coffee snaps activity, you should see a black square where the image was before.

```
59        <androidx.cardview.widget.CardView
60            android:layout_width="270dp"
61            android:layout_height="167dp"
62            android:layout_gravity="center_horizontal"
63            android:layout_marginTop="20dp">
64
65            <androidx.camera.view.PreviewView
66                android:id="@+id/img_cameraImage"
67                android:layout_width="match_parent"
68                android:layout_height="match_parent">
69            </androidx.camera.view.PreviewView>
70
71        </androidx.cardview.widget.CardView>
```

Figure 196. Changing to PreviewView

Now we can add the code to start up the camera and display the preview of the image on the activity. First, add two new properties:

```
19  class CoffeeSnapsActivity : AppCompatActivity() {
20
21      private lateinit var binding: ActivityCoffeeSnapsBinding
22      private lateinit var cameraProviderFuture: ListenableFuture<ProcessCameraProvider>
23      private lateinit var cameraSelector: CameraSelector
```

Figure 197. Adding new properties

Then add the below function to start the camera. Note that the example code from (Ndonga, 2021) needs to be updated to reflect our names of our controls. So, carefully include the function as shown here in Figure 198.

```
46      // With thanks to the tutorial at
47      // https://www.section.io/engineering-education/how-to-implement-camerax-api-in-android/
48      private fun startCamera(){
49          cameraProviderFuture = ProcessCameraProvider.getInstance( context: this)
50          cameraSelector = CameraSelector.DEFAULT_BACK_CAMERA
51          cameraProviderFuture.addListener({
52              val cameraProvider = cameraProviderFuture.get()
53              val preview = Preview.Builder().build().also{  it: Preview
54                  it.setSurfaceProvider(binding.imgCameraImage.surfaceProvider)
55              }
56              try {
57                  cameraProvider.unbindAll()
58                  cameraProvider.bindToLifecycle( lifecycleOwner: this, cameraSelector, preview)
59              } catch (e: Exception) {
60                  Log.d( tag: "CoffeeSnapsActivity",  msg: "Use case binding failed")
61              }
62
63          }, ContextCompat.getMainExecutor( context: this))
64      }
```

Figure 198. Function to start the camera

And finally, we can add the code to ask for camera permissions and start the camera preview.

```
25    override fun onCreate(savedInstanceState: Bundle?) {
26        super.onCreate(savedInstanceState)
27        binding = ActivityCoffeeSnapsBinding.inflate(layoutInflater)
28        setContentView(binding.root)
29
30        // Ask for camera permission in case it wasn't granted already
31        val cameraProviderResult = registerForActivityResult(
32            ActivityResultContracts.RequestPermission()) { permissionGranted->
33                if (permissionGranted) {
34                    // initialise the camera
35                    startCamera()
36                } else {
37                    Toast.makeText( context: this@CoffeeSnapsActivity,
38                        text: "Cannot take a photo without camera permissions",
39                        Toast.LENGTH_SHORT)
40                }
41            }
42        // start the camera
43        cameraProviderResult.launch(android.Manifest.permission.CAMERA)
44    }
45
```

Figure 199. Ask for permissions and start the preview

Now run the app again. You will see that you are prompted to allow the app to take pictures when you open the coffee snaps activity. And if you allow it, the preview of the camera's view will be displayed. Go ahead, try it out!

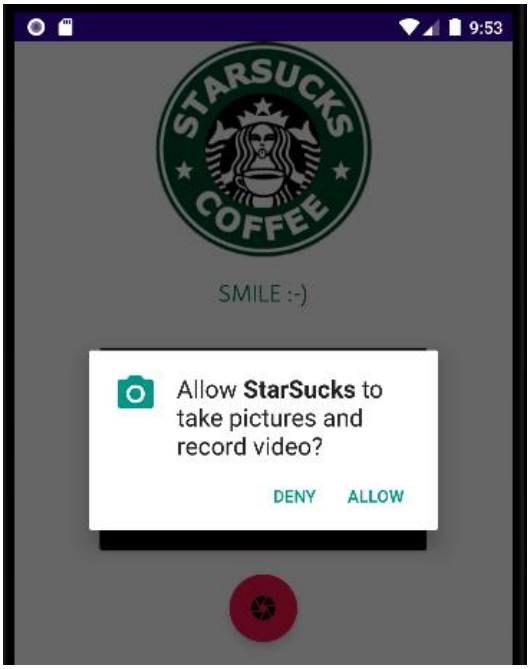


Figure 200. App now asks for photo permission

We don't have to leave the app to take the photo, which is very convenient. But what we are currently seeing is just the preview. Let's add code to take the photo and save it to a file.


```
27      private var imageCapture: ImageCapture? = null
28      private lateinit var imgCaptureExecutor: ExecutorService
```

Figure 201. Add two more properties

```
35          // Create the executor
36          imgCaptureExecutor = Executors.newSingleThreadExecutor()
```

Figure 202. Initialise the executor

```
54  // With thanks to the tutorial at
55  // https://www.section.io/engineering-education/how-to-implement-camerax-api-in-android/
56  private fun startCamera(){
57      cameraProviderFuture = ProcessCameraProvider.getInstance( context: this)
58      cameraSelector = CameraSelector.DEFAULT_BACK_CAMERA
59      cameraProviderFuture.addListener({
60          val cameraProvider = cameraProviderFuture.get()
61          val preview = Preview.Builder().build().also{  it: Preview
62              it.setSurfaceProvider(binding.imgCameraImage.surfaceProvider)
63          }
64          imageCapture = ImageCapture.Builder().build()
65          try {
66              cameraProvider.unbindAll()
67              cameraProvider.bindToLifecycle( lifecycleOwner: this, cameraSelector, preview,
68                  imageCapture)
69          } catch (e: Exception) {
70              Log.d( tag: "CoffeeSnapsActivity",  msg: "Use case binding failed")
71          }
72
73      }, ContextCompat.getMainExecutor( context: this))
74  }
```

Figure 203. Update the startCamera function

```
55      // Add the code to the button to take the photo
56      // Based on https://developer.android.com/training/camerax/take-photo and
57      // https://www.section.io/engineering-education/how-to-implement-camerax-api-in-android/
58      binding.photoFab.setOnClickListener() {  it: View!
59          val outputFileOptions = ImageCapture.OutputFileOptions.Builder(
60              File(externalMediaDirs[0], child: "Coffee_${System.currentTimeMillis()}")).build()
61          imageCapture?.takePicture(outputFileOptions, imgCaptureExecutor,
62              object : ImageCapture.OnImageSavedCallback {
63                  override fun onError(error: ImageCaptureException) {
64                  }
65                  override fun onImageSaved(outputFileResults: ImageCapture.OutputFileResults) {
66                      Log.d( tag: "CoffeeSnapsActivity",
67                          msg: "Photo saved to ${outputFileResults.savedUri}")
68                  }
69              })
70      }
```

Figure 204. Add code to the button to take the photo

When we click the button, the location of the photo that was taken is logged. Not super exciting. Let us add an `ImageView` where we can display the captured photo.

```
<androidx.cardview.widget.CardView
    android:layout_width="270dp"
    android:layout_height="167dp"
    android:layout_gravity="center_horizontal"
    android:layout_marginTop="20dp">

    <ImageView
        android:id="@+id/img_SavedPhoto"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:src="@drawable/ic_launcher_background"
        ></ImageView>
</androidx.cardview.widget.CardView>
```

Figure 205. Add an ImageView where we can display the photo

Now we can add a line of code to display the image that we already have the `Uri` for.

```
override fun onImageSaved(outputFileResults: ImageCapture.OutputFileResults) {
    Log.d( tag: "CoffeeSnapsActivity",
         msg: "Photo saved to ${outputFileResults.savedUri}")
    binding.imgSavedPhoto.setImageURI(outputFileResults.savedUri)
}
```

Figure 206. Add code to display the photo

However, if we do this, the app unceremoniously crashes. If we look at the Run window in Android Studio, we see this exception:

```
E/AndroidRuntime: FATAL EXCEPTION: pool-1-thread-1
    Process: za.ac.iie.opsc7311.starsucks, PID: 1002
    android.view.ViewRootImpl$CalledFromWrongThreadException: Only the original thread that created a view hierarchy can touch its views.
        at android.view.ViewRootImpl.checkThread(ViewRootImpl.java:7286)
        at android.view.ViewRootImpl.requestLayout(ViewRootImpl.java:1155)
        at android.view.View.requestLayout(View.java:21926)
```

Figure 207. Exception when trying to display the image

If you look closely at the code that we used to capture the photo, you will see that there is an executor involved. That starts the capture process on a different thread. And with Android, just like with most user interface frameworks, you cannot modify the user interface from a different thread. The method `runOnUiThread` will work in this instance. (TutorialKart, 2021)

```
this@CoffeeSnapsActivity.runOnUiThread(java.lang.Runnable {
    binding.imgSavedPhoto.setImageURI(outputFileResults.savedUri)
})
```

Figure 208. Running the code on the user interface thread

And that is it. Now the photo will get displayed on the `ImageView`. Go ahead and try it out!

5.10 Adding a Launcher Icon

There is one last thing to be done for the app's user interface to be polished – adding a launcher icon for the app. If you look in the list of apps installed in the emulator (see Figure 209), you will see that our app still has the default icon. While that is functional, it certainly doesn't look professional.

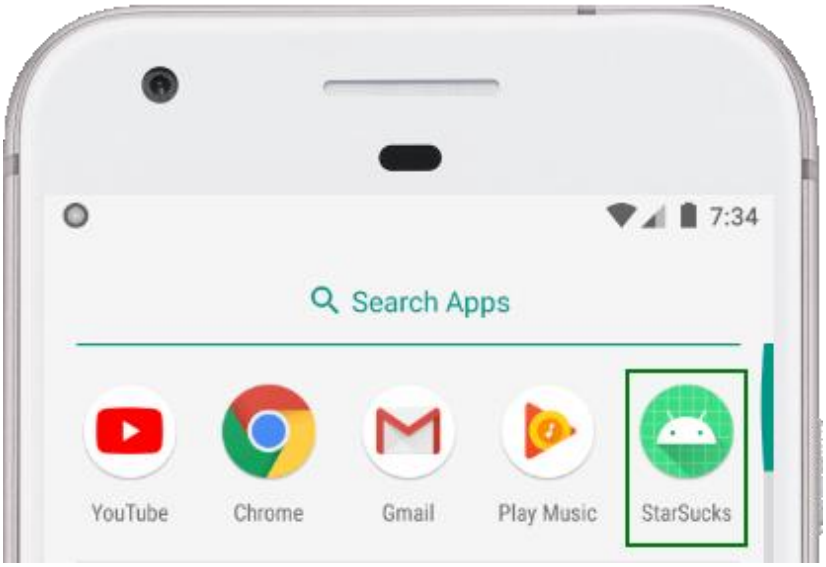


Figure 209. Default Launcher Icon

To add a launcher icon for the app:

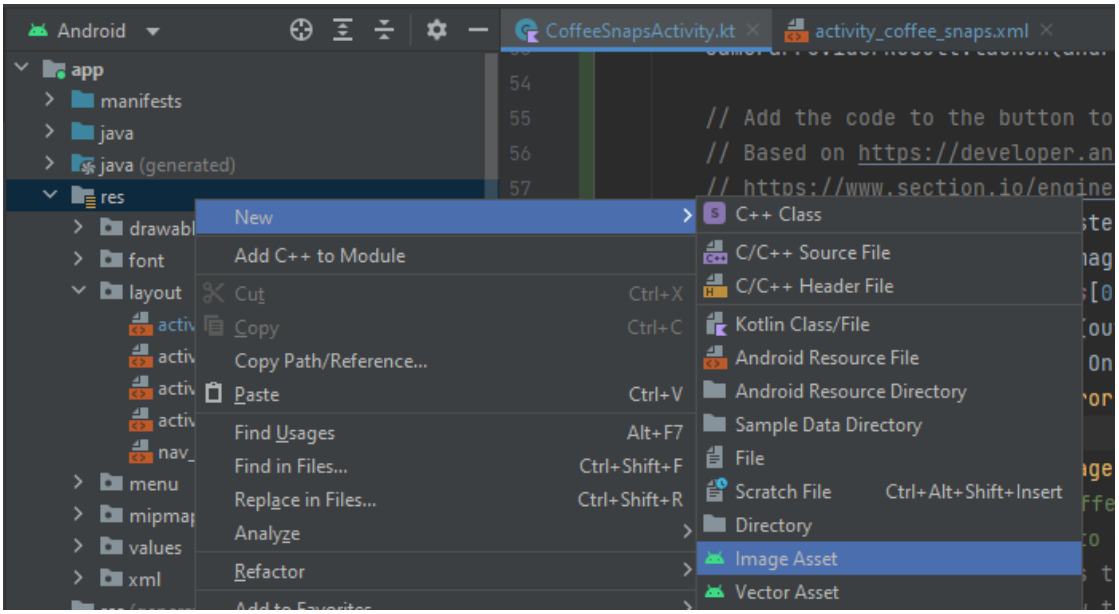


Figure 210. Creating a New Image Asset

1. Right-click the **res** folder, click **New** and then **Image Asset**.

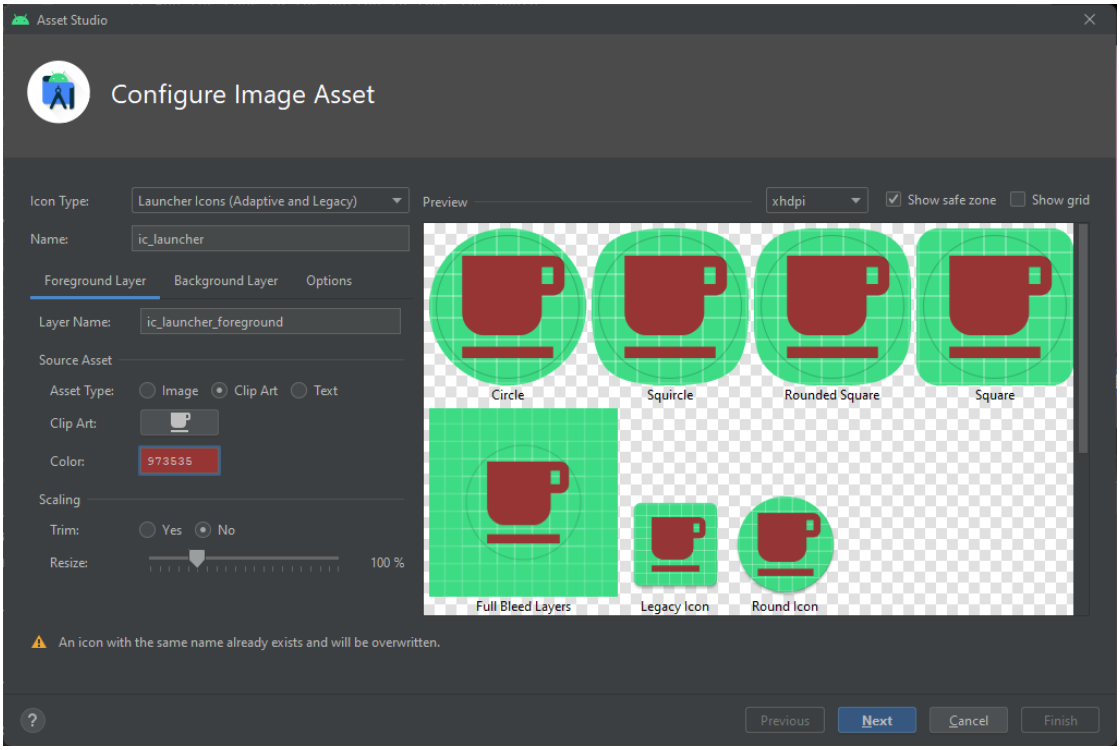


Figure 211. Foreground configuration with the icon called local cafe

2. On the **Foreground Layer** tab, choose a **Clip Art** image.
3. Chose a **colour** for the foreground.
4. Switch to the Background Layer tab.
5. Choose a **colour** for the background.
6. Click **Next**.
7. Click **Finish**.

Tip: If your colour is correct when you run the app, but the icon is wrong, delete the file highlighted in Figure 212 in the **Project** view. Then recreate follow the steps again to create the launcher icon.

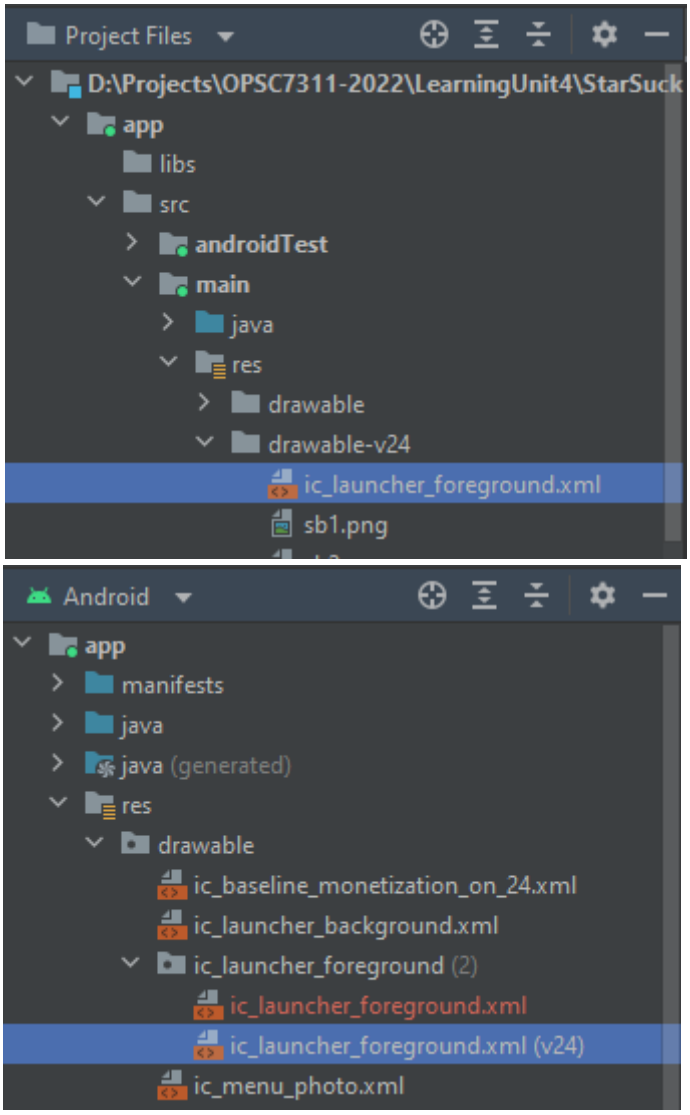


Figure 212. Delete This File

Now the app has its brand-new icon.

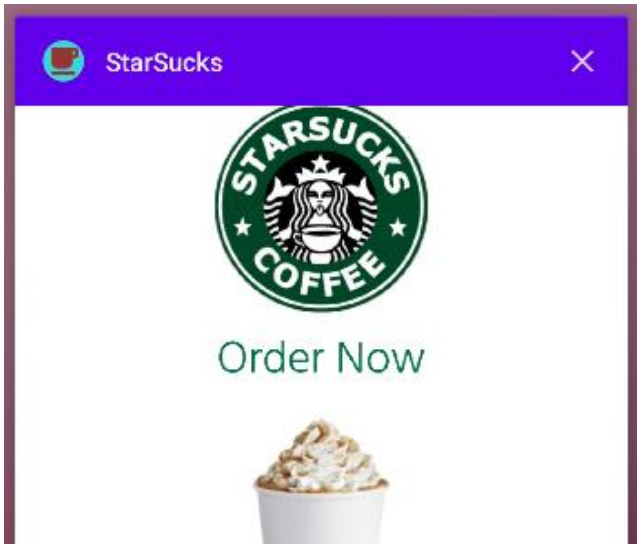


Figure 213. App with the New Icon

6 Recommended Additional Reading

Engel, C., 2017. *4 Ways To Implement OnClickListener On Android.* [Online] Available at: https://medium.com/ @CodyEngel/4-ways-to-implement-onclicklistener-on-android-9b956cbd2928 [Accessed 17 November 2022].

7 Activities

Do the activities that appear on Learn.

8 Revision Exercises

Create a small quiz application where the user is asked a series of questions about famous landmarks (use an `ImageView` to display these) and they must correctly identify them using `EditTexts`. Once the user has answered all the questions, they should be taken to a second activity that displays their score.

9 Solutions to Revision Exercises

Compare your code to the example StarSucks code from the GitHub repository. You can also check to see if your application is displaying the images correctly and calculating the scores correctly.

Learning Unit 5: Modern Data Management Techniques	
Learning Objectives: • Explain the difference between Firebase and a traditional SQL database. • Describe the advantages of using Firebase. • Explain why an application would need to be authorized to access a Firebase database. • Create a connection between an application Firebase. • Explain the purpose of JSON in a mobile development. • Create a Firebase database. • Create code to read data from the Firebase database. • Create code to write data to the Firebase database.	**My notes**
Material used for this learning unit: • GitHub repository: LearningUnit5	
How to prepare for this learning unit: • Make sure that you have the GitHub source code available and that your Android Studio is up to date.	

1 Introduction

In all the learning units so far, we have built apps that only store their data in memory. And while that can be quite enough, depending on the app, we usually need long term storage as well. In this learning unit we will explore writing data to a NoSQL database.

2 NoSQL Databases

2.1 Firebase

Firebase is a mobile application development platform owned by Google since 2014. The platform has a range of products which are very useful for mobile application development. You can find out more about the available products here: https://firebase.google.com/. You will probably find that you end up using most of them.

There are two database products in Firebase: the Realtime Database and Cloud Firestore. We will look at both options.

2.2 Firebase Real-Time Database

The Firebase Realtime Database does exactly what one would expect. It stores and retrieves user data in real time (milliseconds). You can read more about the Realtime Database here: https://firebase.google.com/products/realtime-database/ [Accessed 17 November 2022].

The Firebase Realtime Database is also a NoSQL database. That stores data in a JSON format. We will look a bit deeper into that next.

2.2.1 Understanding NoSQL

NoSQL stands for **Not only SQL** and is a different approach to database design. You would have used mostly Relational Database Management Systems (RDBMS) such SQL Express and MySQL until now. You have probably normalised and related a database and in turn cried a little. SQL relies on schemas and highly structured data.

NoSQL accommodates a wide variety of data models and have only been around since the early 2000s. The introduction of cloud and mobile computing created scenario's where the need for scalability and processing speed greatly outweighed the need for a structured carefully planned database. There are different types of NoSQL databases, which is beyond the scope of this course – you can read more about them here: https://searchdatamanagement.techtarget.com/definition/NoSQL-Not-Only-SQL [Accessed 17 November 2022].

Firebase's Realtime Database is a document database that stores JSON documents. Now we probably need to chat about what JSON is.

2.2.2 Understanding JSON

JSON is an acronym for Java Script Object Notation. It is a text based, human readable data interchange format used to pass data between database and applications, between different applications etc. You will learn a lot more about JSON when you code up an application that uses a REST- API. An easy way to think about it is that JSON gives a structure to save our data in that is understood by a wide variety of technologies. This makes it very easy of us to send data in this format to other applications. Web API's send large sets of data to all sorts of clients in JSON format.

Let's look at the example shown in Figure 214, from the OpenWeather API documentation. We are going to copy and paste the sample JSON into a JSON formatter to check if it is Valid JSON, but also to get a better understanding of what we are seeing in Figure 214.

Navigate to the following website: https://jsonformatter.curiousconcept.com/ and copy the sample JSON from https://openweathermap.org/current into the JSON formatter and click on Process.

API response:


```
{"coord": { "lon": 139,"lat": 35},
  "weather": [
    {
      "id": 800,
      "main": "Clear",
      "description": "clear sky",
      "icon": "01n"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 289.92,
    "pressure": 1009,
    "humidity": 92,
    "temp_min": 288.71,
    "temp_max": 290.93
  },
  "wind": {
    "speed": 0.47,
    "deg": 107.538
  },
  "clouds": {
    "all": 2
  },
```


Figure 214. OpenWeather API Response (OpenWeather, n.d.)

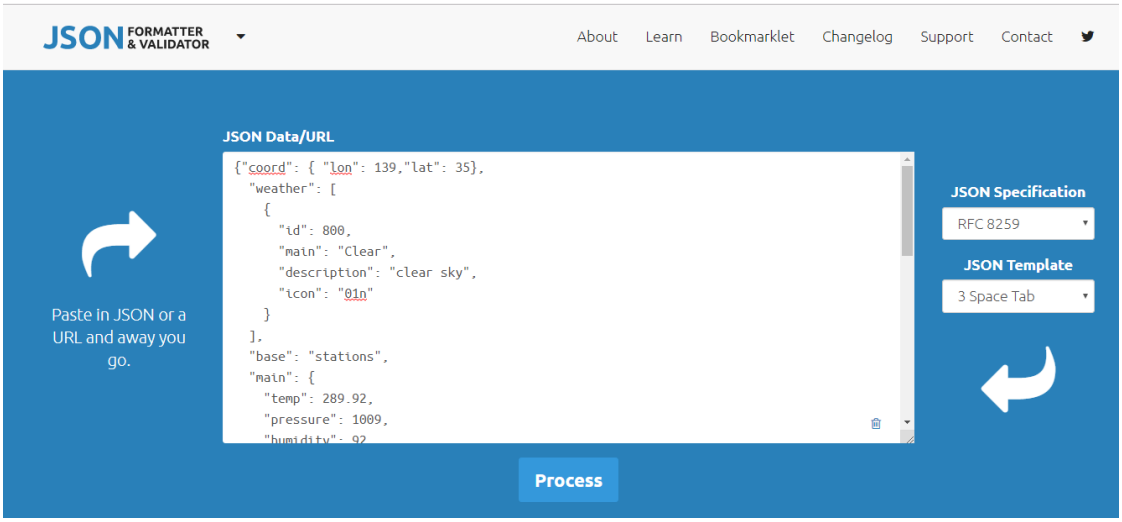


Figure 215. Processing the JSON Response

You should get the following output.

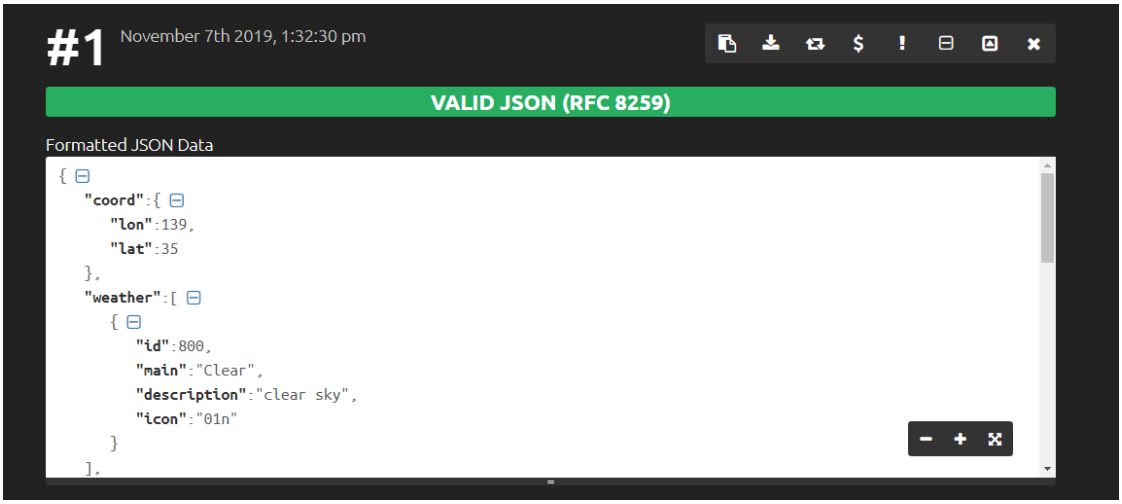


Figure 216. Processing Complete

Let's look at what we can learn from this. Collapse all the entries (click on the [-]) and you should see the following data structure.

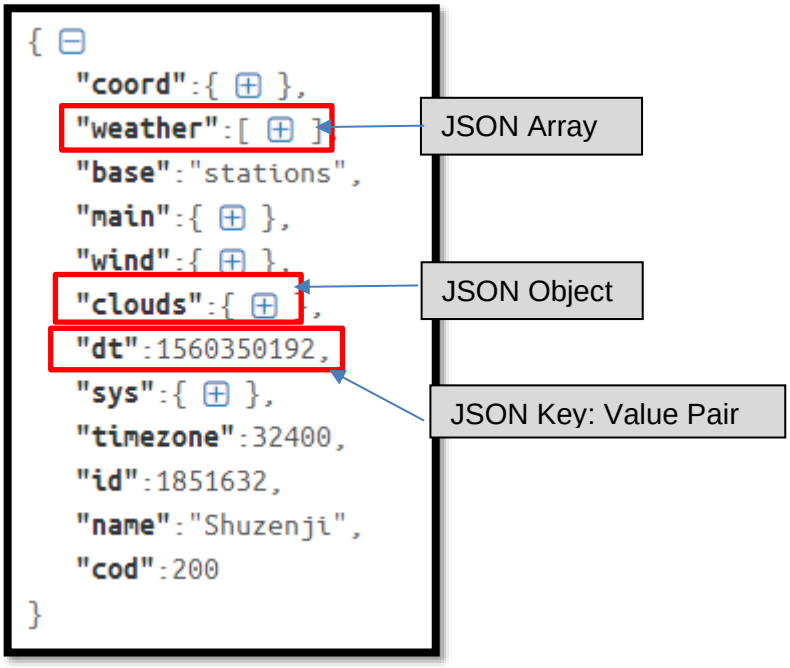


Figure 217. JSON Structure

JSON Objects: A JSON Object is a collection of key:value pairs, objects can also hold other objects and JSON Arrays. I have expanded the main JSON object below and you can see it holds key value pairs for temperature, pressure, and humidity.

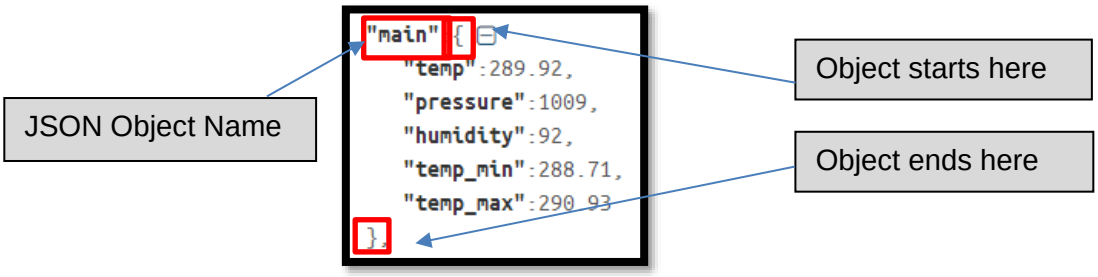


Figure 218. JSON Object

JSON Array: JSON Arrays holds an ordered list of values.

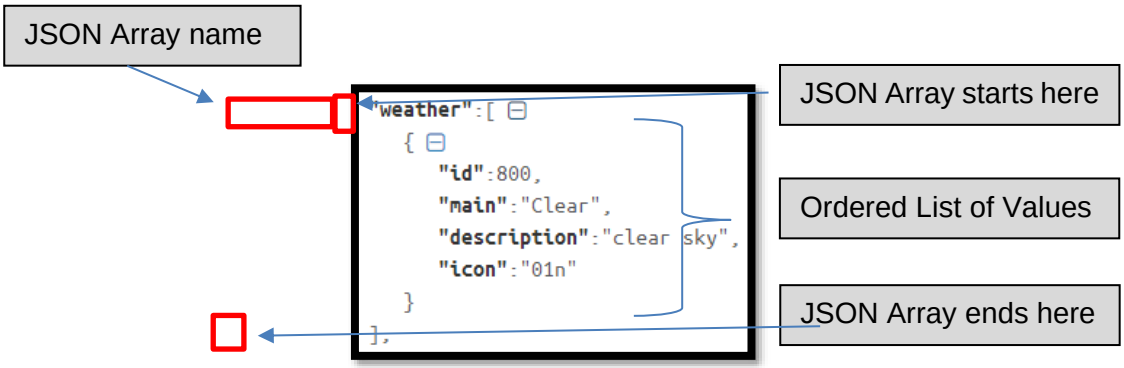


Figure 219. JSON Array

If we look at some of my previous data stored on a Firebase Realtime Database, we will see that it looks like the image below.

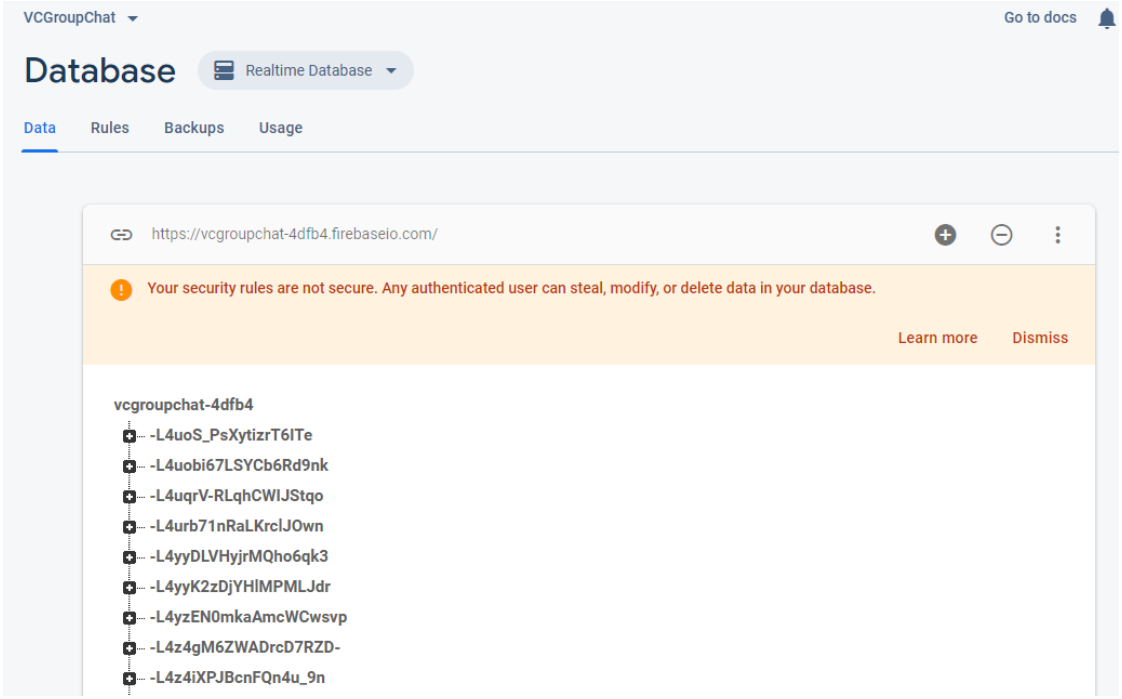


Figure 220. Firebase Realtime Database Data

We can export the data from Firebase into a JSON file by click **Export JSON**.

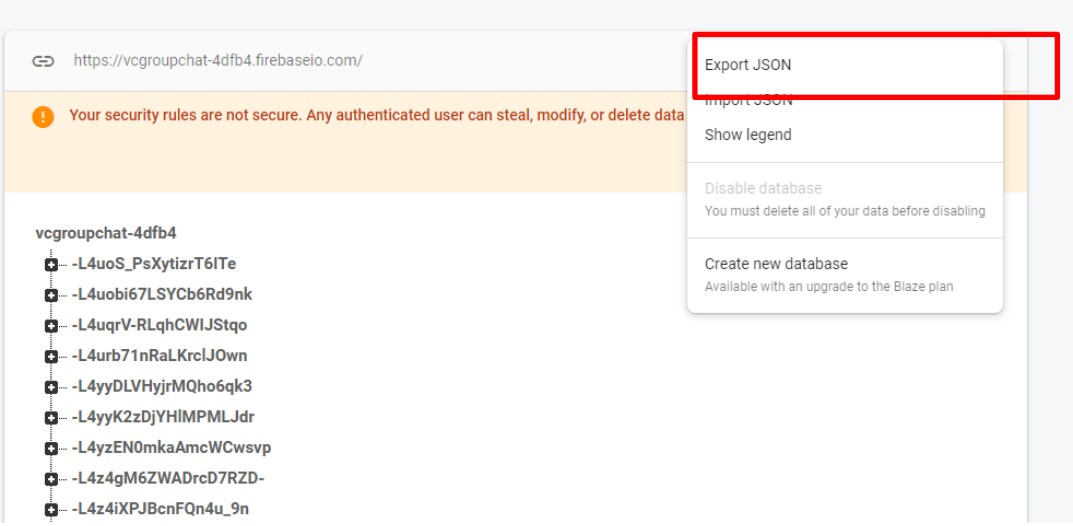


Figure 221. Exporting JSON from Firebase

We can then copy and paste the JSON in the downloaded .json file into a JSON formatter. We will see the following:

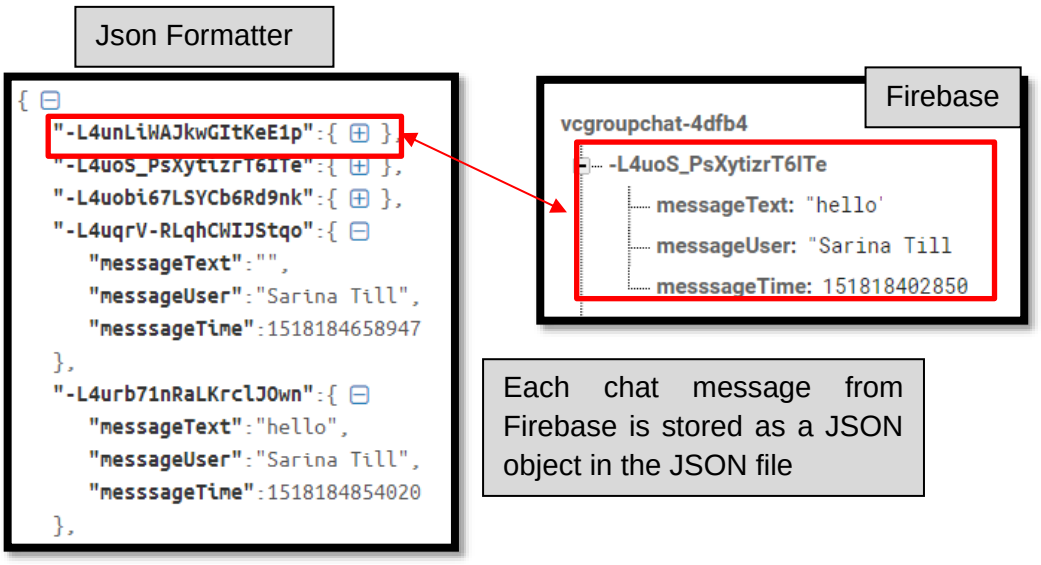


Figure 222. Firebase Data in JSON Format

You can learn more about JSON and the related syntax here: https://medium.com/omarelgabrys-blog/json-in-a-nutshell-7d638dfea7cc [Accessed 17 November 2022].

2.3 Cloud Firestore

Cloud Firestore is a newer NoSQL database. The data structure, based on documents, are more flexible than the one used by the Realtime Database. (Google, 2022)

Choosing between the Realtime Database and Cloud Firestore for a real app is quite complex. There are lots of difference between the two, some quite subtle. But you will

find good guidelines here: https://firebase.google.com/docs/database/ rtdb-vs-firestore [Accessed 17 November 2022].

Now that you have a better understanding of NoSQL and JSON and Firebase Realtime and Cloud Firestore, lets add Firebase to our Android Project. We will start with the Realtime Database.

Note: You will need a GMAIL account for this – create one if you don't have one.

3 Connect an App to Firebase

Adding Firebase is incredibly easy. Android Studio has all the tools to automate the process.

Note: Follow steps 1 to 8 on the sample project in the `StarsucksRealtime` folder in the GitHub repository too, to configure the app to run against your account. Also remember to configure the database for public access, as described later in this section.

To add Firebase to the project:

1. Click on **Tools** and select **Firebase** from the menu.

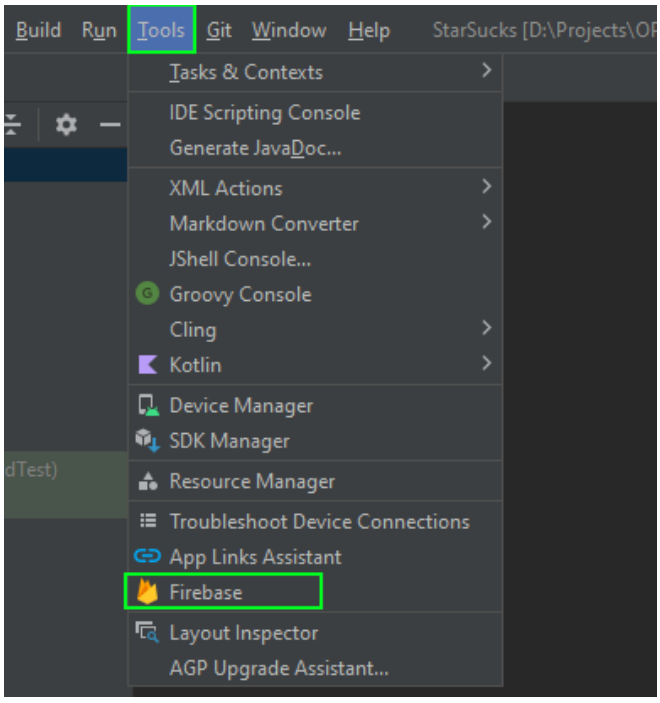


Figure 223. Accessing Firebase from the Tools Menu

2. This will open the **Firebase** assistant panel to the right of your screen. Here you will see all the available Firebase products.
3. Find the **Realtime Database** and click on the arrow to expand it.

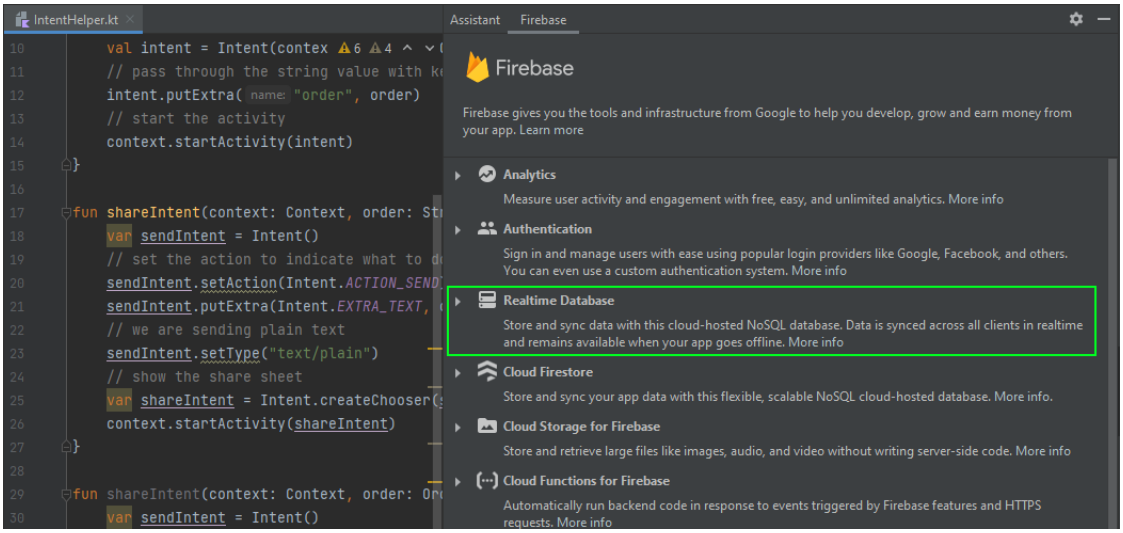


Figure 224. Realtime Database

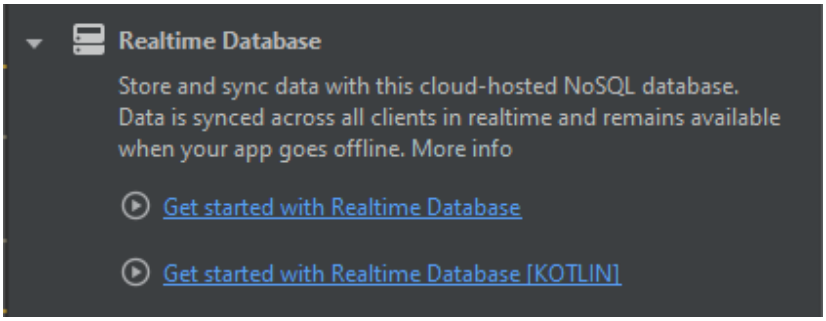


Figure 225. Get started with Realtime Database

4. Click the **Get started with Realtime Database [KOTLIN]** link. This will change the panel to look like Figure 226.

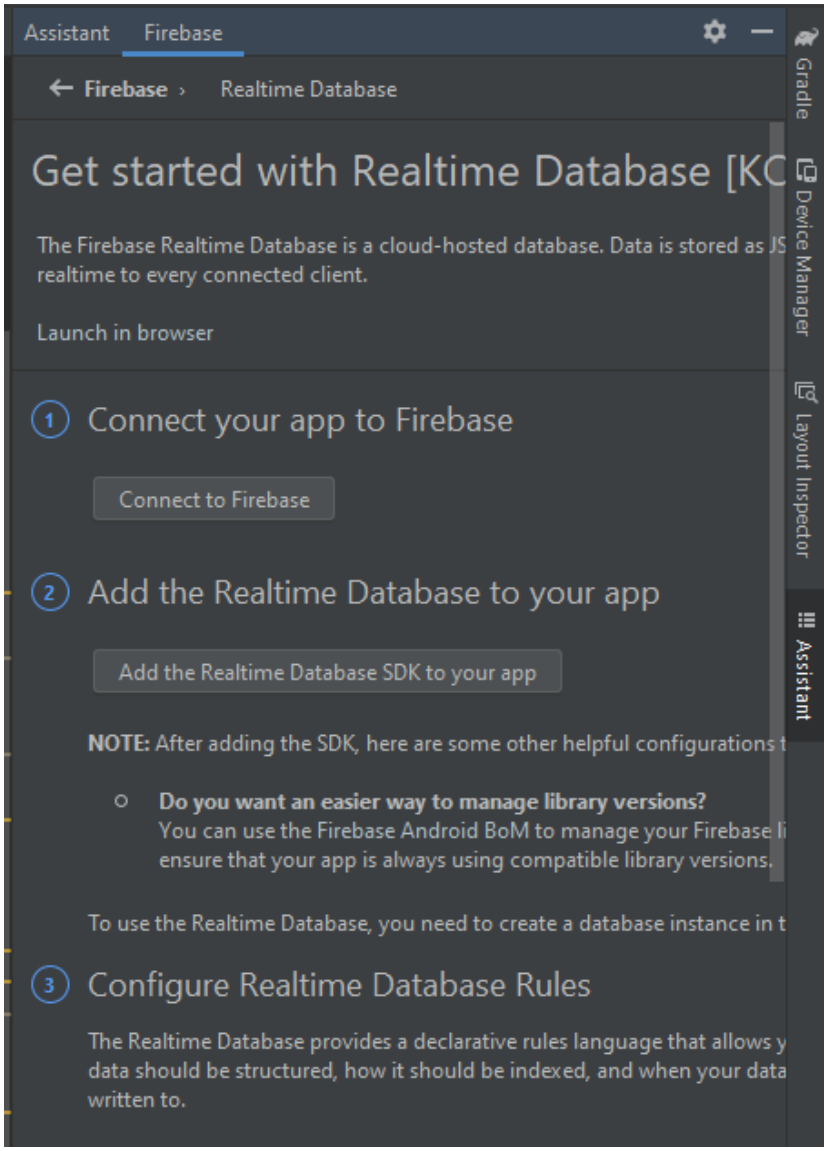


Figure 226. Get started with Realtime Database

5. Click **Connect to Firebase**.
6. You will be prompted to log in with your Gmail account in your browser at this point. Once you are logged in, you will see the **Firebase Console** web UI.
7. Create a **new project** – call it StarSucks.
8. When prompted, click **Connect**.

You will see the following confirmation message appear in when you return to Android Studio.

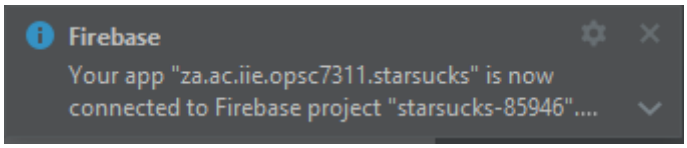


Figure 227. Firebase Successfully Connected

We now need to add the Firebase Realtime database dependencies to our app. These are class libraries that contains all the built-in logic that we would need to work with Firebase.

Gradle can throw some scary errors at this stage – but don't worry, it is usually just a version conflict. You can just Google for the correct version of Firebase to use with your Max API and change the entry in Gradle. It should work fine 99% of the time.

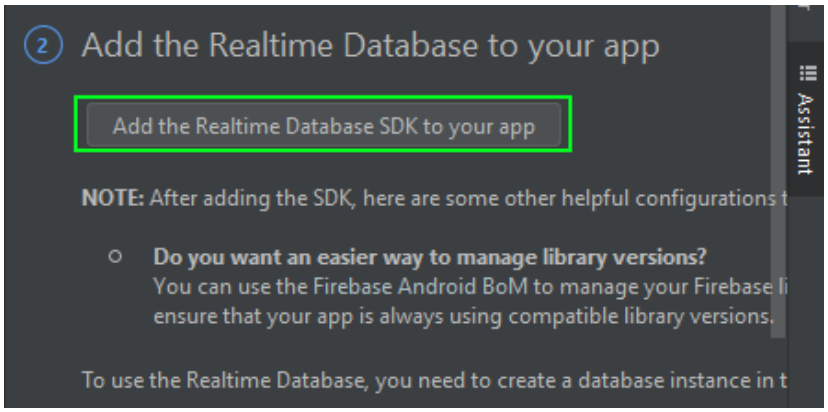


Figure 228. Adding the Dependencies

Click **Add the Realtime Database to your app**.

Android Studio will ask you to confirm that you want to add the dependencies to the project. It will also list the dependencies that will be added to both your build.gradle and app/build.gradle files (project and app (module level)). Click **Accept Changes** to continue.

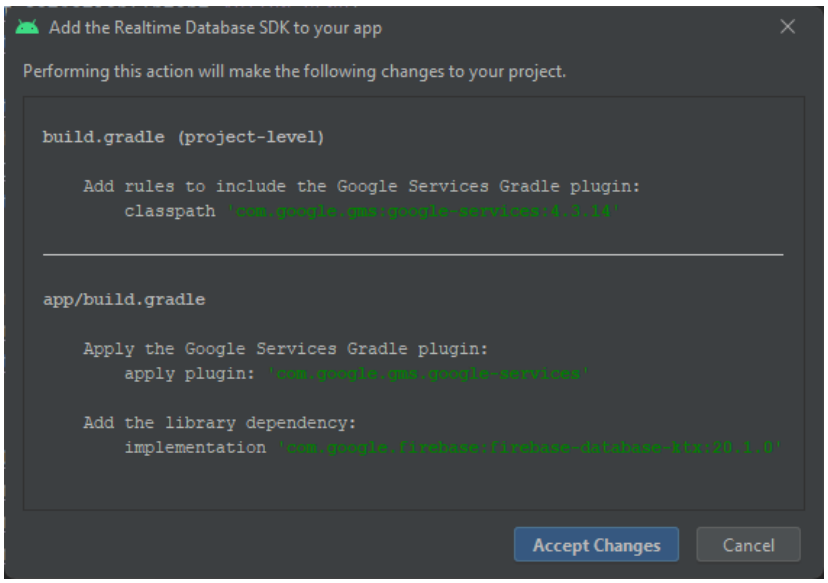


Figure 229. Accepting Dependency Changes

Gradle will sync and download all the dependency files.

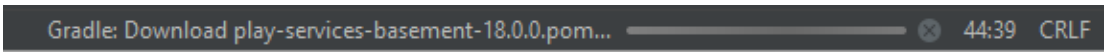


Figure 230. Gradle Sync in Progress

You will see the following confirmation message if your dependencies were set up without any errors.

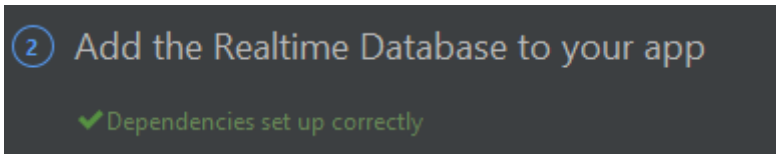


Figure 231. Dependencies Set Up Successfully

You can however only breathe easily when you have a successful sync and Gradle build.

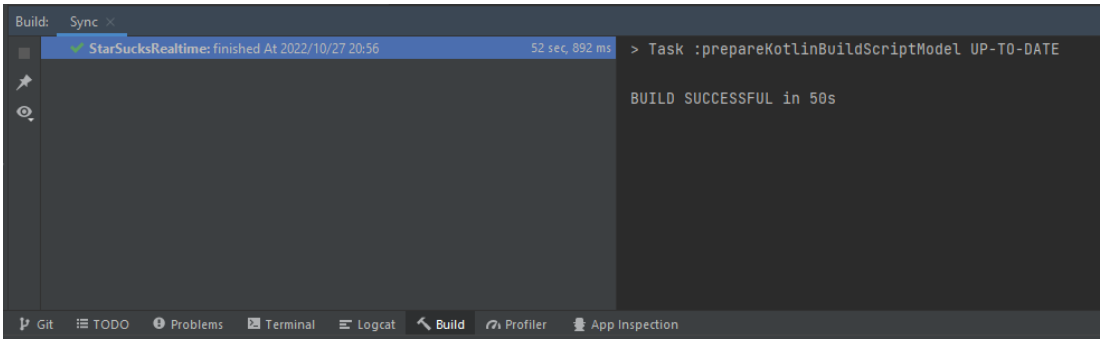


Figure 232. Successful Gradle Build

We can now go and open our Gradle files to inspect the changes made to them. Navigate to **Gradle Scrips** and click on **build.gradle (Module:StarSucks.app)** first.

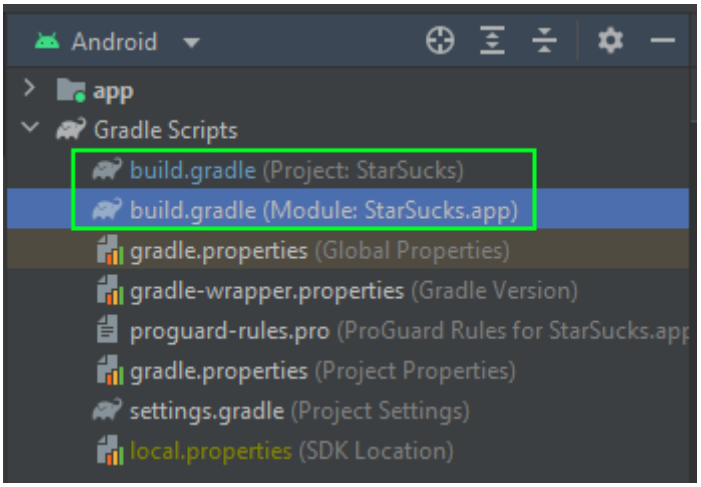


Figure 233. Gradle Files

You will notice that the following entry was added to the Gradle script file.


```
plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
    id 'com.google.gms.google-services'
}

android {
    compileSdk 32
    viewBinding.enabled = true

    defaultConfig {
        applicationId "za.ac.iie.opsc7311.starsucks"
        minSdk 26
        targetSdk 32
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard
        }
    }
```

Figure 234. Google Services Plugin Added

If you scroll down further, you will find that the dependency has also been added

```
dependencies {

    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.appcompat:appcompat:1.5.1'
    implementation 'com.google.android.material:material:1.6.1'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    implementation 'com.google.firebase:firebase-database-ktx:20.1.0'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'

    // CameraX core library using the camera2 implementation
    def camerax_version : String = "1.1.0"
    implementation "androidx.camera:camera-camera2:${camerax_version}"
    implementation "androidx.camera:camera-lifecycle:${camerax_version}"
    implementation "androidx.camera:camera-view:${camerax_version}"
}
```

Figure 235. Realtime Database Dependency

This matches up perfectly with the dependencies asked for earlier (see Figure 229). You will find the remaining entry in the build.gradle (Project: StarSucks) script file:

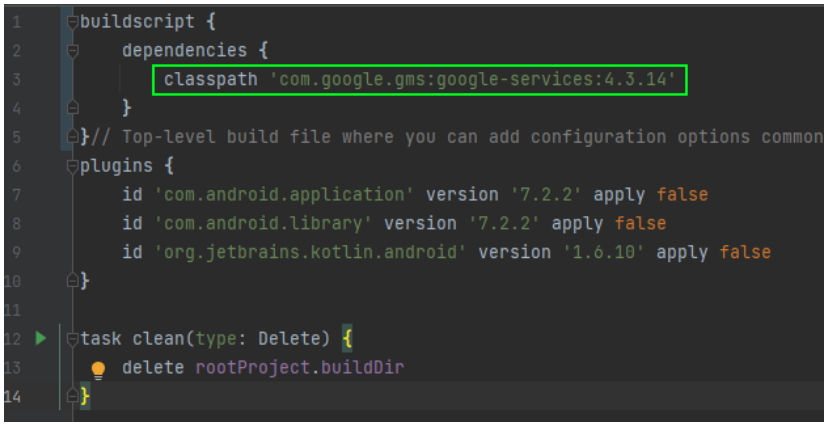


Figure 236. Project Script File

Good, our dependencies are ready to go. We can now configure our database rules. The default security settings for the Realtime Database is to only allow read and writes to authenticated members (using Firebase Authentication). We want to change that while we test our app. Just remember to change it back once you are ready to deploy your application. This is fine for testing, but a **bad idea for the security of your app once it enters the production environment**.

Go back to the Firebase panel. You will find the **Configure Firebase Database Rules** section there. Click on the link for **configure your rules for public access**.

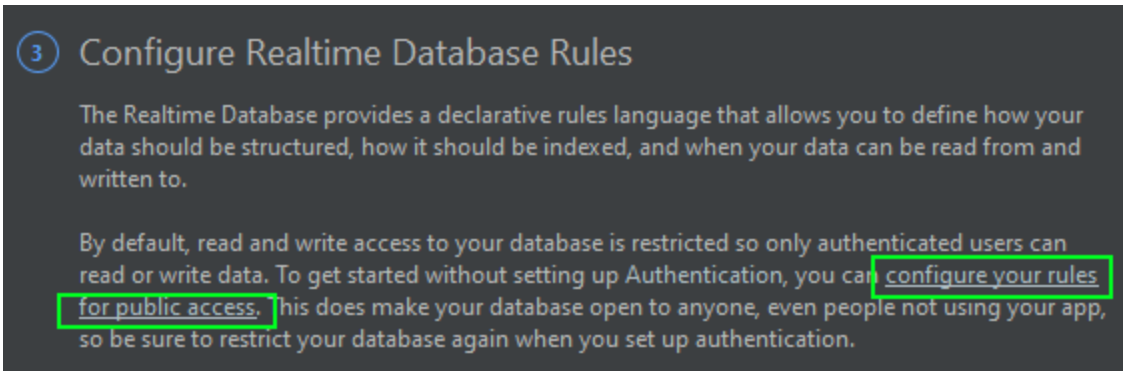


Figure 237. Configure for Public Access

You need to log into your Firebase console to set your database settings. You can access the console here: https://console.firebase.google.com/u/0/ [Accessed 17 November 2022].

Find the tile for your app and click on it.

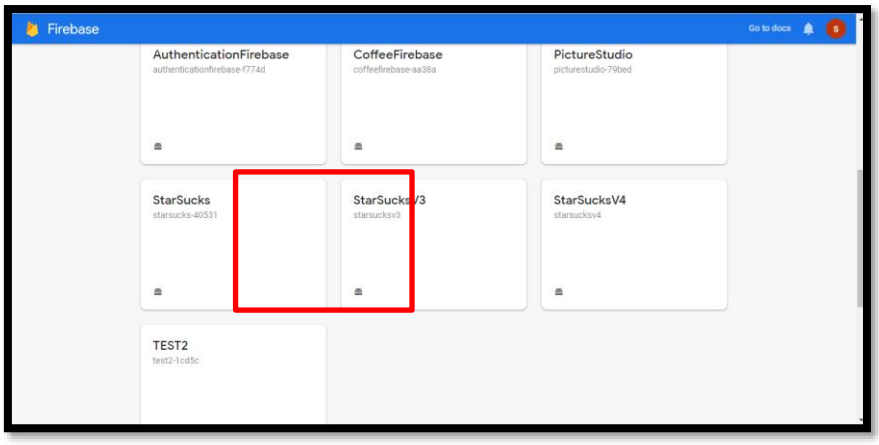


Figure 238. Select the Project

Your project will open, you can go ahead and click on **Realtime Database** in the Project Overview panel to the left of the screen, under the **Build** category.

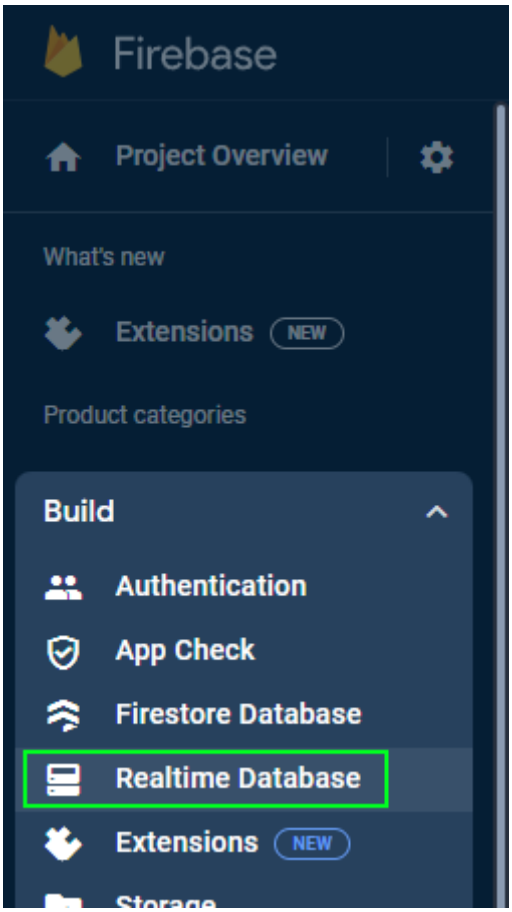


Figure 239. Accessing Database Settings

Click **Create Database**.

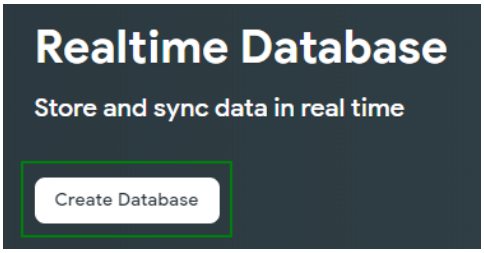


Figure 240. Creating the Database

The first option that appears asks you to choose a location for the database. South Africa is not available as an option, so choose europe-west1 as the next-best option and click **Next**.

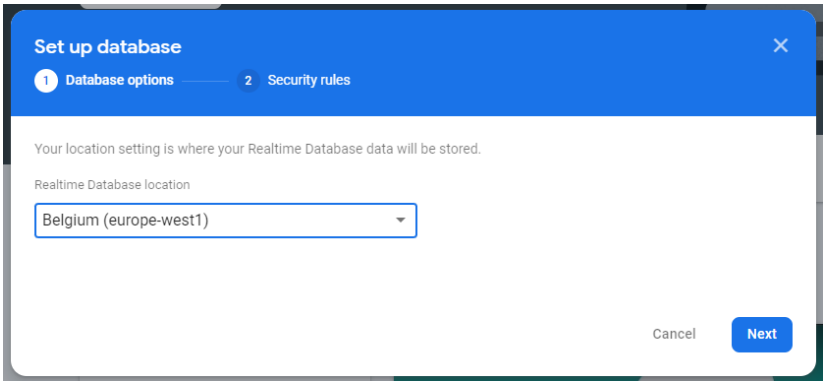


Figure 241. Database Location

The security rules for Realtime Database window will appear. Click **Start in test mode**.

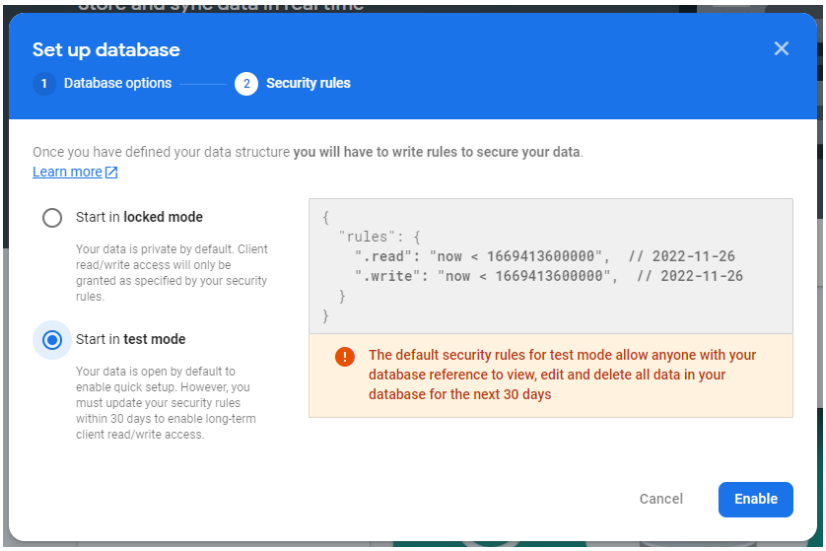


Figure 242. Switching to Test Mode

Click **Enable**.

This will change your read and write rules to true – you can now write to this database without first authenticating.
Your database will be created and ready for use:

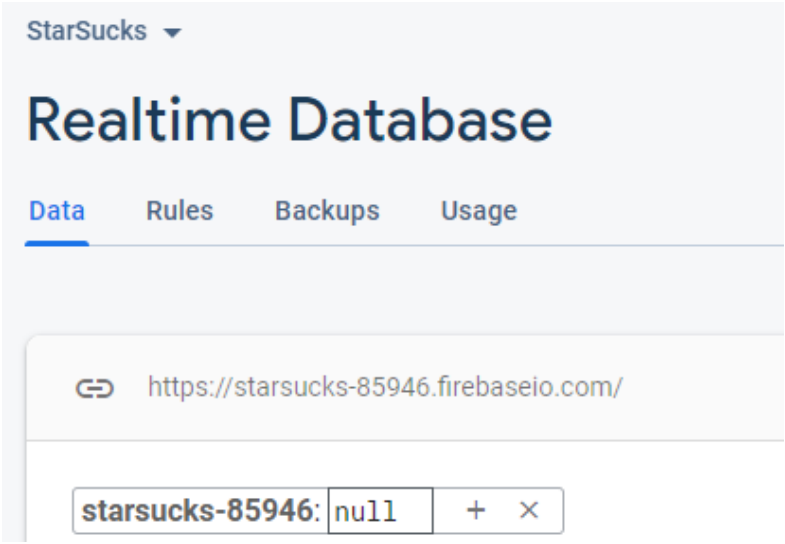


Figure 243. Database Created

We are now ready to start adding the logic to our application so that we can write to the Firebase Realtime Database from our application. Firebase provided a guided tutorial in the Firebase panel within Android Studio. We will now follow that tutorial.

4 Firebase Data Storage

We are now ready to add the logic to our application so that we can push data up to Firebase Realtime Database. We will be using our Data Transfer Object (DTO) that we create using the model we build. This is just the `Order` class. This class contains the properties and constructors that reflect the structure our data will stored in on the Realtime Database. It also allows us to pull down the JSON data and convert it into an object.

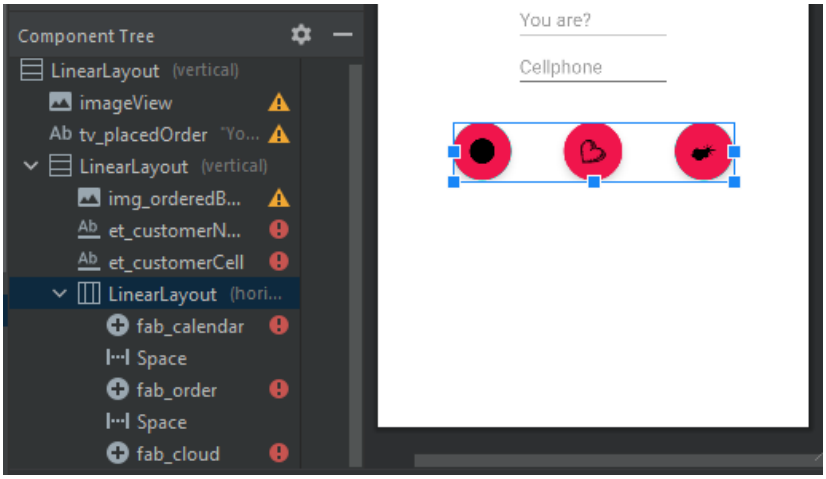


Figure 244. New Components in Order Details

Let's add the new floating action buttons shown in Figure 244 to our app. The images smallcloud.png and date.png can be found in the assets folder for leaning unit 4 in the repository.

4.1 Adding a Date Picker

You will recall that we declared a date field in the `Order` class – this is because we would like to pass the order date to our database.

Add the following code to the `OnClickListener` of your `FloatingActionButton` for the calendar (`fabCalendar`).

```
binding.fabCalendar.setOnClickListener() {  it: View!
    // Create a calendar to get today's date
    val datePickerCalendar = Calendar.getInstance()
    val year = datePickerCalendar.get(Calendar.YEAR)
    val month = datePickerCalendar.get(Calendar.MONTH)
    val day = datePickerCalendar.get(Calendar.DAY_OF_MONTH)

    // Show a date picker, starting from today's date
    val listener = object : DatePickerDialog.OnDateSetListener {
        override fun onDateSet(view: DatePicker?, year: Int, month: Int, dayOfMonth: Int) {
            order.orderDate = "${year}-${month+1}-${day}"
        }
    }
    var ordersDatePicker = DatePickerDialog( context: this@OrderDetailsActivity,
                                listener, year, month, day)
    ordersDatePicker.show()
}
```

Figure 245. Date Picker Code

Note that the months returned by the date picker are from 0 to 11. (Android Open Source Project, 2022g) So, to get a month that makes sense to users, we add 1.

4.2 Writing Data to the Database

We next need to add a `FirebaseDatabase` instance and a `DatabaseReference` to our `OrderDetailsActivity`. You can copy and paste over the Firebase sample code provided in the Firebase panel.

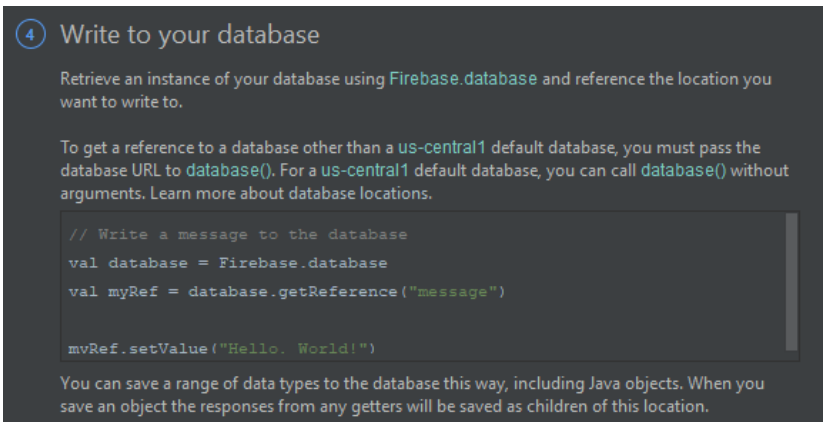


Figure 246. Firebase Example Code

```
class OrderDetailsActivity : AppCompatActivity() {
    var order = Order()
    val database = Firebase.database
    // add orders to the path
    val starSucksRef = database.getReference( path: "orders")
```

Figure 247. Database Reference

Now we add the `OnClickListener` to the `fabCloud` button to write the data to the database.

```
binding.fabCloud.setOnClickListener() {  it: View!
    order.customerName = binding.etCustomerName.text.toString()
    order.customerCell = binding.etCustomerCell.text.toString()

    // check that no data is missing
    if (!order.customerName.isNullOrBlank() && !order.customerCell.isNullOrBlank() &&
            !order.orderDate.isNullOrEmpty() && !order.productName.isNullOrBlank()) {

        // add the order to the list of orders
        starSucksRef.push().setValue(order)
    } else {
        // message to display to the user if something is missing
        Toast.makeText( context: this@OrderDetailsActivity,
            text: "Please complete all fields", Toast.LENGTH_SHORT).show()
    }
}
```

Figure 248. Writing to the Database

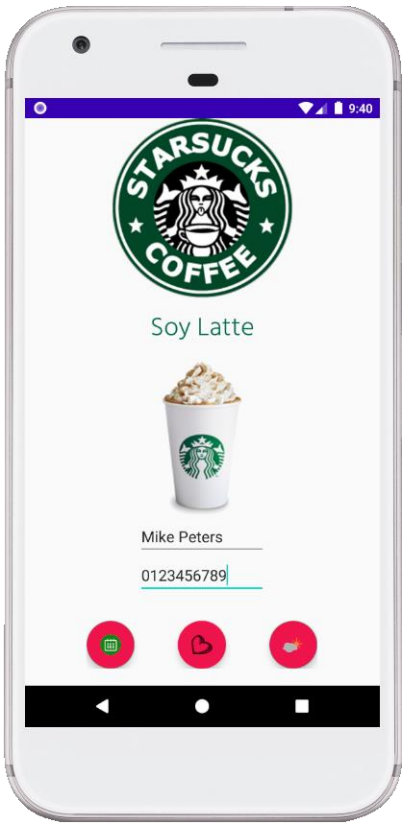


Figure 249. Testing the Database Writing

Let's try it out. We enter Mike Peters and his cell number. We next select the date and click the cloud button. We should see the corresponding values in our Realtime Database. So, let's go look in the Firebase console.

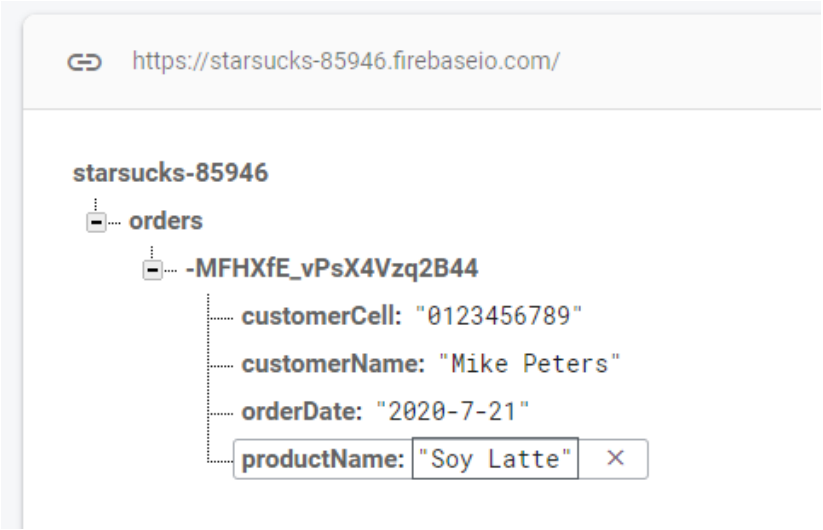


Figure 250. Data in the Realtime Database

And there it is – our very first entry!

4.3 Retrieving our Data from Realtime Database and displaying it in a ListView

We would like to be able to view our order history by pulling these values out of our Realtime Database and displaying them in a `ListView`. We need to add a new activity to do this. Go ahead and add a new empty activity called `OrderHistoryActivity`.

Create the following UI for the `OrderHistoryActivity`.

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

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <ImageView
            android:id="@+id/img_starsucksLogo"
            android:layout_width="207dp"
            android:layout_height="177dp"
            android:layout_gravity="center_horizontal"
            android:src="@drawable/starsuckslogo"></ImageView>
```

```
        <TextView
            android:id="@+id/tv_lblOrderHistory"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:fontFamily="@font/hind_guntur_light"
            android:text="Your Order History"
            android:textColor="@color/starsucksGreen"
            android:textSize="20dp"></TextView>

        <ListView
            android:id="@+id/lstv_OrderHistory"
            android:layout_width="match_parent"
            android:layout_height="match_parent"></ListView>
    </LinearLayout>
</LinearLayout>
```

We are now ready to add the logic to retrieve our data from the Realtime Database. We need to add our `DatabaseReference` and `FirebaseDatabase` references here too. Make sure to pass in the same path as the path you used when you pushed your values up to Firebase - “orders” in our case.

```
class OrderHistoryActivity : AppCompatActivity() {
    val database = Firebase.database
    // add orders to the path
    val starSucksRef = database.getReference( path: "orders")

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val binding = ActivityOrderHistoryBinding.inflate(layoutInflater)
        setContentView(binding.root)
    }
}
```

Figure 251. Starting with the Logic

Also we will be using View Binding again as shown in Figure 251.

We can now use our Database reference by adding a `ValueEventListener` to it. The `ValueEventListener` will fire every time we add a value to the Firebase Realtime Database, but it will also fire once after we add it. We will pull down a `DataSnapshot` every time the listener fires. We will then ask the `DataSnapshot` to pull down all its children.

```
// Read from the database
starSucksRef.addValueEventListener(object: ValueEventListener {

    override fun onDataChange(snapshot: DataSnapshot) {
        var list = mutableListOf<Order>()

        // iterate over the children in the list
        for (pulledOrder in snapshot.children) {
            val order : Order? = pulledOrder.getValue(Order::class.java)
            if (order != null) {
                list.add(order)
            }
        }

        // create the adapter to display the items
        var orderAdapter = ArrayAdapter( context: this@OrderHistoryActivity,
            android.R.layout.simple_list_item_1, list)
        binding.lstvOrderHistory.adapter = orderAdapter
    }

    override fun onCancelled(error: DatabaseError) {
        Toast.makeText( context: this@OrderHistoryActivity,
             text: "Error reading from database", Toast.LENGTH_SHORT).show()
    }
})
```

Figure 252. Value Event Listener

And the last thing that we still need to do is to override the `toString()` method for class `Order`.

```
override fun toString(): String {
    return "Customer Name: ${customerName}\n" +
            "Product Name: ${productName}\n" +
            "Order Date: ${orderDate}"
}
```

Figure 253. Overriding the toString() Method

4.4 Adding the Order History to the Navigation Drawer

Now we have a great activity, but we still need to add it to the navigation drawer so we can see it in action. Add it to:

- navigation_menu.xml
- MainActivity.onNavigationItemSelected

Then we can run the app and view the order history.

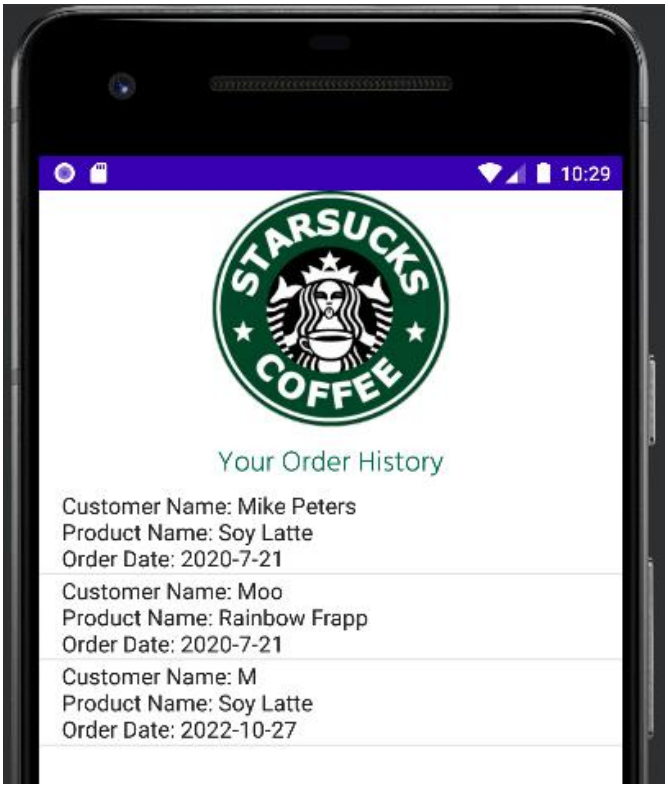


Figure 254. Order History in the App

Our writing to and reading from the database is working using the Realtime Database. Now, let us do the same using Cloud Firebase.

4.5 Writing and Retrieving Data using Cloud Firestore

Now we are working in the other copy of the sample source code, in the `StarSucksFirestore` folder.

When connecting our app to Firebase, we choose **Cloud Firestore** this time. Follow the same process to connect and add the dependencies.

In the Firebase console, under **Build** select **Firestore Database**, and click **Create database**.

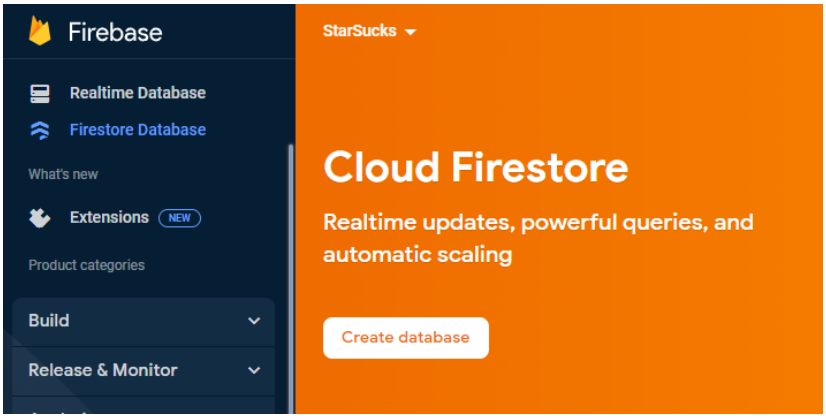


Figure 255. Creating a Cloud Firestore database

Start the database in test mode and select a European region. Now we are ready to add the code to use Cloud Firestore.

In the `OrderDetailsActivity`, we need an instance of the database. Then we can write to the database.

```
14  class OrderDetailsActivity : AppCompatActivity() {
15      var order = Order()
16      val db = Firebase.firestore
```

Figure 256. Get a database instance

```
68          // add the order to the list of orders
69          db.collection( collectionPath: "orders")
70              .add(order)
71              .addOnSuccessListener { documentReference ->
72                  Log.d( tag: "Order Details",
73                      msg: "DocumentSnapshot added with ID: ${documentReference.id}")
74              }
75              .addOnFailureListener { e ->
76                  Log.w( tag: "Order Details",  msg: "Error adding document", e)
77              }
```

Figure 257. Write to the database

And if we look in the console, we can spot our very first entry in the Cloud Firestore.

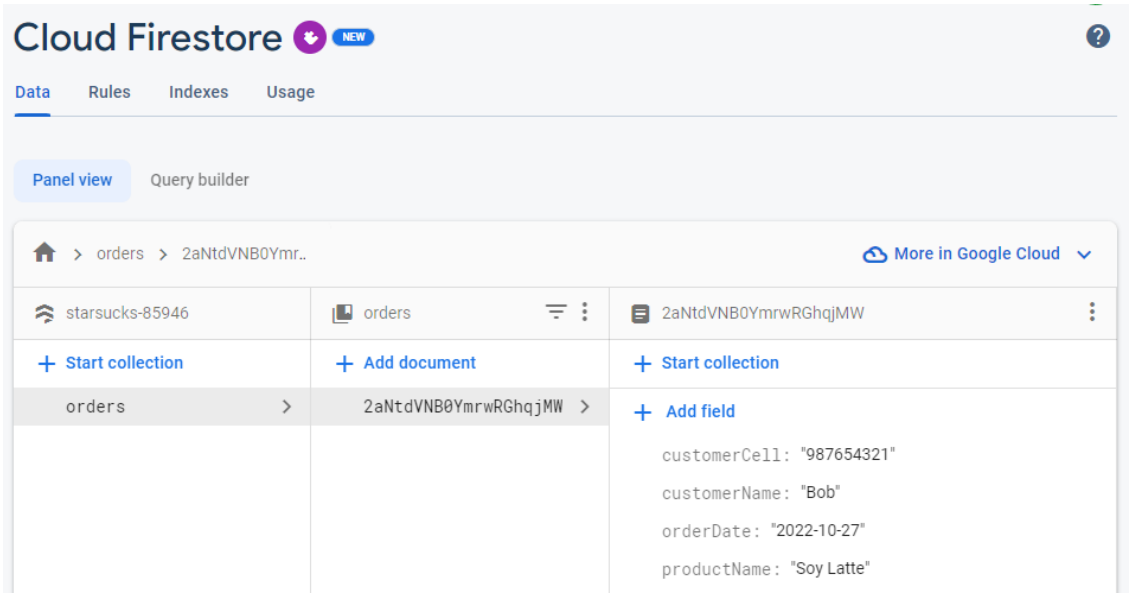
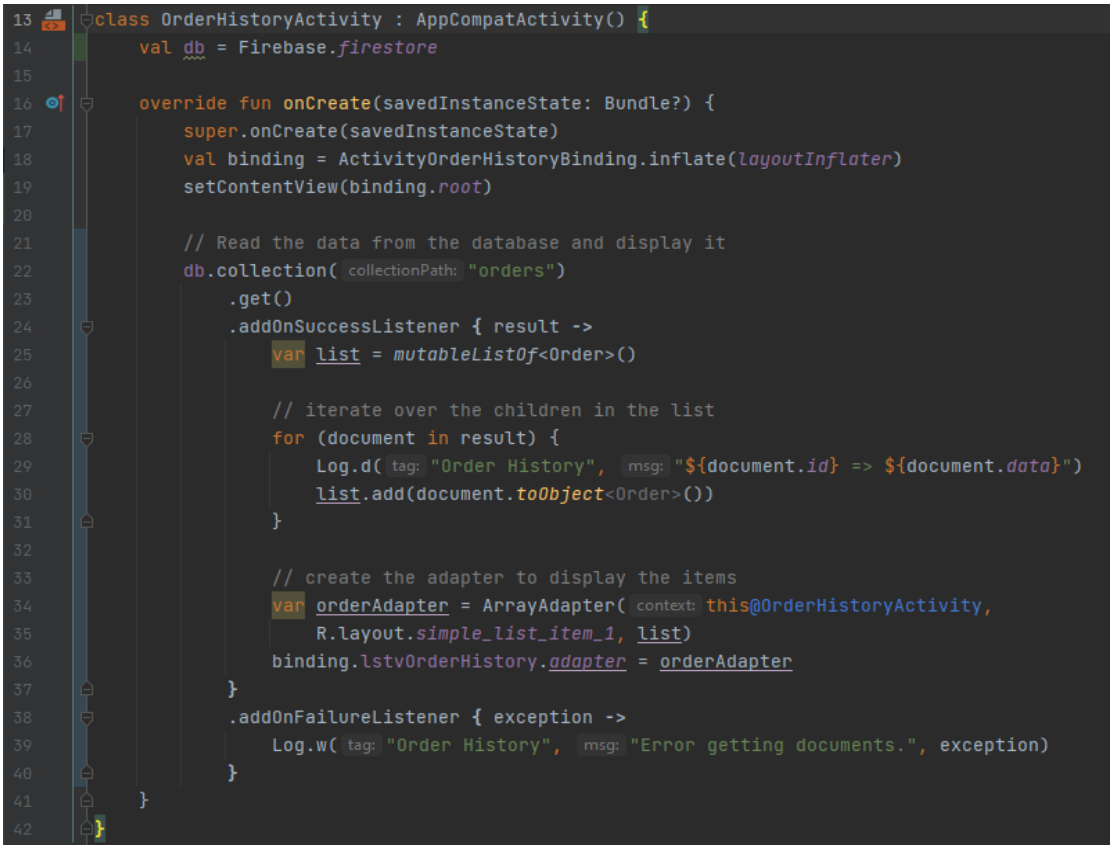


Figure 258. Entry in Cloud Firestore



```
class OrderHistoryActivity : AppCompatActivity() {
    val db = Firebase.firestore

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val binding = ActivityOrderHistoryBinding.inflate(layoutInflater)
        setContentView(binding.root)

        // Read the data from the database and display it
        db.collection( collectionPath: "orders")
            .get()
            .addOnSuccessListener { result ->
                var list = mutableListOf<Order>()

                // iterate over the children in the list
                for (document in result) {
                    Log.d( tag: "Order History", msg: "${document.id} => ${document.data}")
                    list.add(document.toObject<Order>())
                }

                // create the adapter to display the items
                var orderAdapter = ArrayAdapter( context: this@OrderHistoryActivity,
                    R.layout.simple_list_item_1, list)
                binding.lstvOrderHistory.adapter = orderAdapter
            }
            .addOnFailureListener { exception ->
                Log.w( tag: "Order History", msg: "Error getting documents.", exception)
            }
    }
}
```

Figure 259. Reading the Data from Cloud Firestore

Read more about accessing data from Cloud Firestore in (Google, 2022b)

5 Recommended Additional Reading

Read about the best practices for setting up Firebase projects here: https://firebase.google.com/docs/projects/dev-workflows/general-best-practices [Accessed 17 November 2022].

6 Activities

Do the activities that appear on Learn.

7 Revision Exercises

Create a similar application (to the example in this learning unit) that allows a user to order from a different fast-food brand. Extend the app to have a login and registration functionality.

8 Solutions to Revision Exercises

Download and review all the Starsucks apps on the GitHub Repo. Make sure your order app can login and register using Firebase. Check that the registered users are displayed online in the Firebase database.

Bibliography

Abu Experience, 2017. *10 Mobile UX Design Principles You Should Know.* [Online] Available at: http://uxbert.com/10-mobile-ux-design-principles/ [Accessed 17 November 2022].

Agrawal, S., n.d.. *Property, Getter and Setter:Kotlin.* [Online] Available at: https://agrawalsuneet.github.io/blogs/property-getter-and-setter-kotlin/ [Accessed 17 November 2022].

al3c, 2022. *Trying to understand OnClickListener.* [Online] Available at: https://discuss.kotlinlang.org/t/trying-to-understand-onclicklistener/24773/4 [Accessed 17 November 2022].

Android Open Source Project, 2019. *Run your app.* [Online] Available at: https://developer.android.com/training/basics/firstapp/running-app [Accessed 17 November 2022].

Android Open Source Project, 2020b. *Licenses.* [Online] Available at: https://source.android.com/setup/start/licenses [Accessed 15 August 2020].

Android Open Source Project, 2020c. *Run apps on the Android Emulator.* [Online] Available at: https://developer.android.com/studio/run/emulator [Accessed 17 November 2022].

Android Open Source Project, 2020d. *Android Runtime (ART) and Dalvik.* [Online] Available at: https://source.android.com/devices/tech/dalvik [Accessed 17 November 2022].

Android Open Source Project, 2020f. *Build a UI with Layout Editor.* [Online] Available at: https://developer.android.com/studio/write/layout-editor [Accessed 17 November 2022].

Android Open Source Project, 2020g. *Layouts.* [Online] Available at: https://developer.android.com/guide/topics/ui/declaring-layout [Accessed 17 November 2022].

Android Open Source Project, 2020h. *App resources overview.* [Online] Available at: https://developer.android.com/guide/topics/resources/providing-resources [Accessed 17 November 2022].

Android Open Source Project, 2020i. *Build a Responsive UI with ConstraintLayout.* [Online] Available at: https://developer.android.com/training/constraint-layout [Accessed 17 November 2022].

Android Open Source Project, 2020j. *Activity.* [Online] Available at: https://developer.android.com/reference/android/app/Activity [Accessed 17 November 2022].

Android Open Source Project, 2022. *Android 13 Beta.* [Online] Available at: https://developer.android.com/about/versions/13 [Accessed 17 November 2022].

Android Open Source Project, 2022b. *Meet Android Studio.* [Online] Available at: https://developer.android.com/studio/intro [Accessed 17 November 2022].

Android Open Source Project, 2022c. *Run apps on a hardware device.* [Online] Available at: https://developer.android.com/studio/run/device [Accessed 17 November 2022].

Android Open Source Project, 2022d. *About Android App Bundles.* [Online] Available at: https://developer.android.com/guide/app-bundle [Accessed 17 November 2022].

Android Open Source Project, 2022e. *Context.* [Online] Available at: https://developer.android.com/reference/android/content/Context [Accessed 17 November 2022].

Android Open Source Project, 2022f. *CameraX.* [Online] Available at: https://developer.android.com/jetpack/androidx/releases/camera#groovy [Accessed 17 November 2022].

Android Open Source Project, 2022g. *DatePickerDialog.OnDateSetListener.* [Online] Available at: https://developer.android.com/reference/android/app/DatePickerDialog.OnDateSetListener[Accessed 17 November 2022].

Anon., 2018. *File:Java programming language logo.svg.* [Online] Available at: https://en.wikipedia.org/wiki/File:Java_programming_language_logo.svg [Accessed 17 November 2022].

Ayub, N., 2017. *Android Linear Layout Example.* [Online] Available at: https://javatutorial.net/android-linear-layout-example [Accessed 17 November 2022].

Ayub, N., 2017b. *Android Relative Layout Example.* [Online] Available at: https://javatutorial.net/android-relative-layout-example [Accessed 17 November 2022].

Baeldung, 2022. *Calling a Kotlin Function After a Delay.* [Online] Available at: https://www.baeldung.com/kotlin/call-function-after-delay [Accessed 17 November 2022].

Bhatnagar, M., 2018. *Magic lies here - Statically vs Dynamically Typed Languages.* [Online] Available at: https://medium.com/android-news/magic-lies-here-statically-typed-vs-dynamically-typed-languages-d151c7f95e2b [Accessed 17 November 2022].

Chen, J., 2020. *Android Operating System.* [Online] Available at: https://www.investopedia.com/terms/a/android-operating-system.asp [Accessed 17 November 2022].

Doan, T. H., 2020. *Utils class in Kotlin.* [Online] Available at: https://proandroiddev.com/utils-class-in-kotlin-387a09b8d495 [Accessed 17 November 2022].

Engel, C., 2017. *4 Ways To Implement OnClickListener On Android.* [Online] Available at: https://medium.com/@CodyEngel/4-ways-to-implement-onclicklistener-on-android-9b956cbd2928 [Accessed 17 November 2022].

Ernst, M., 2018. *Version control concepts and best practices.* [Online] Available at: https://homes.cs.washington.edu/~mernst/advice/version-control.html [Accessed 17 November 2022].

Foundation, K., 2022c. *Generics: in, out, where.* [Online] Available at: https://kotlinlang.org/docs/generics.html [Accessed 17 November 2022].

G., N., 2022. *Android: Market Share & Other Stats for 2022.* [Online] Available at: https://techjury.net/blog/android-market-share/ [Accessed 17 November 2022].

GeeksforGeeks, 2018. *Anonymous Inner Class in Java.* [Online] Available at: https://www.geeksforgeeks.org/anonymous-inner-class-java/ [Accessed 17 November 2022].

Google, 2014. *File:Android robot 2014.svg.* [Online] Available at: https://commons.wikimedia.org/wiki/File:Android_robot_2014.svg [Accessed 17 November 2022].

Google, 2022b. *Get data with Cloud Firestore.* [Online] Available at: https://firebase.google.com/docs/firestore/query-data/get-data [Accessed 17 November 2022].

Google, 2022. *Choose a Database: Cloud Firestore or Realtime Database.* [Online] Available at: https://firebase.google.com/docs/database/rtdb-vs-firestore#which_database_does_firebase_recommend [Accessed 17 November 2022].

Google, n.d.. *Applying color to UI.* [Online] Available at: https://material.io/design/color/applying-color-to-ui.html#usage [Accessed 17 November 2022].

Heller, M., 2022. *What is Kotlin? The Java alternative explained.* [Online] Available at: https://www.infoworld.com/article/3224868/what-is-kotlin-the-java-alternative-explained.html [Accessed 17 November 2022].

HowToDoInjava.com, n.d.. *Gradle Tutorial – Installation and Hello World Example.* [Online] Available at: https://howtodoinjava.com/gradle/gradle-tutorial-installation-and-hello-world-example/ [Accessed 17 November 2022].

JetBrains, 2020. *File:Kotlin-logo.svg.* [Online] Available at: File:Kotlin-logo.svg [Accessed 17 November 2022].

Jethro, 2018. *Android naming convention.* [Online] Available at: https://stackoverflow.com/questions/12870537/android-naming-convention [Accessed 17 November 2022].

Komatineni, S., 2014. *Understanding R.java.* [Online] Available at: http://knowledgefolders.com/akc/display?url=displaynoteimpurl&ownerUserId=satya&reportId=2883 [Accessed 17 November 2022].

Kotlin Foundation, 2022. *Basic syntax.* [Online] Available at: https://kotlinlang.org/docs/basic-syntax.html [Accessed 17 November 2022].

Kotlin Foundation, 2022b. *Conditions and loops.* [Online] Available at: https://kotlinlang.org/docs/control-flow.html [Accessed 17 November 2022].

kotlinlang, n.d.. *Kotlin Playground.* [Online] Available at: https://play.kotlinlang.org/ [Accessed 17 November 2022].

Lake, I., 2016. *Layouts, Attributes, and you.* [Online] Available at: https://medium.com/androiddevelopers/layouts-attributes-and-you-9e5a4b4fe32c [Accessed 17 November 2022].

Lee, G., 2017. *Android View Measurement.* [Online] Available at: https://blog.takescoop.com/android-view-measurement-d1f2f5c98f75 [Accessed 17 November 2022].

Leiva, A., 2020. *View Binding: The Definitive way to access views on Android.* [Online] Available at: https://antonioleiva.com/view-binding-android/ [Accessed 17 November 2022].

Murphy, M. L., 2020. *Other Changes of Note.* [Online] Available at: https://commonsware.com/R/pages/chap-other-001.html [Accessed 17 November 2022].

Ndonga, R., 2021. *How to Implement CameraX API in Android Using Kotlin.* [Online] Available at: https://www.section.io/engineering-education/how-to-implement-camerax-api-in-android/ [Accessed 17 November 2022].

nevoski, n.d.. *Iphone Android Stencil.* [Online] Available at: https://pixabay.com/vectors/iphone-android-stencil-smartphone-1459087/ [Accessed 17 November 2022].

OpenWeather, n.d.. *Current weather data.* [Online] Available at: https://openweathermap.org/current [Accessed 17 November 2022].

Programiz, 2022. *Kotlin Constructors.* [Online] Available at: https://www.programiz.com/kotlin-programming/constructors [Accessed 17 November 2022].

Raphael, J., 2020. *Android versions: A living history from 1.0 to 11.* [Online] Available at: https://www.computerworld.com/article/3235946/android-versions-a-living-history-from-1-0-to-today.html [Accessed 17 November 2022].

Rout, A. R., 2022. *Different Types of Activities in Android Studio.* [Online] Available at: https://www.geeksforgeeks.org/different-types-of-activities-in-android-studio/ [Accessed 17 November 2022].

sahilkhoslaa, 2018. *Java | How to create your own Helper Class?.* [Online] Available at: https://auth.geeksforgeeks.org/user/sahilkhoslaa/articles [Accessed 17 November 2022].

Sharma, M., 2021. *StartActivityForResult is Deprecated!.* [Online] Available at: https://www.mongodb.com/developer/languages/kotlin/realm-startactivityforresult-registerforactivityresult-deprecated-android-kotlin/ [Accessed 17 November 2022].

Shaukat, A., 2016. *An Overview Of Polymorphism, Inheritance And Encapsulation In OOP.* [Online] Available at: https://www.c-sharpcorner.com/article/an-overview-of-polymorphism-inheritance-and-encapsulation-in-oop/ [Accessed 17 November 2022].

Sinhal, A., 2017. *Closer Look At Android Runtime: DVM vs ART.* [Online] Available at: https://android.jlelse.eu/closer-look-at-android-runtime-dvm-vs-art-1dc5240c3924 [Accessed 17 November 2022].

Sinicki, A., 2019. *How to install the Android SDK.* [Online] Available at: https://www.androidauthority.com/how-to-install-android-sdk-software-development-kit-21137/ [Accessed 17 November 2022].

TFPP Writer, 2018. *5 Examples Of Successful Rebranding That Will Blow Your Mind.* [Online] Available at: https://thefederalistpapers.org/us/5-examples-successful-rebranding-will-blow-mind [Accessed 17 November 2022].

TutorialKart, 2021. *Only the original thread that created a view hierarchy can touch its views.* [Online] Available at: https://www.tutorialkart.com/kotlin-android/original-thread-created-view-hierarchy-can-touch-views/ [Accessed 17 November 2022].

U.S. Brand Colours, n.d.. *Starbucks Colors.* [Online] Available at: https://usbrandcolors.com/starbucks-colors/[Accessed 17 November 2022].

Wagner, B., Lieben, G., Warren, G. E. & Dykstra, T., 2022. *Properties (C# Programming Guide).* [Online] Available at: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/properties [Accessed 18 October 2022].

Intellectual Property

Plagiarism occurs in a variety of forms. Ultimately though, it refers to the use of the words, ideas or images of another person without acknowledging the source using the required conventions. The IIE publishes a Quick Reference Guide that provides more detailed guidance, but a brief description of plagiarism and referencing is included below for your reference. It is vital that you are familiar with this information and the Intellectual Integrity Policy before attempting any assignments.

Introduction to Referencing and Plagiarism

What is 'Plagiarism'?

'Plagiarism' is the act of taking someone's words or ideas and presenting them as your own.

What is 'Referencing'?

'Referencing' is the act of citing or giving credit to the authors of any work that you have referred to or consulted. A 'reference' then refers to a citation (a credit) or the actual information from a publication that is referred to.

Referencing is the acknowledgment of any work that is not your own, but is used by you in an academic document. It is simply a way of giving credit to and acknowledging the ideas and words of others.

When writing assignments, students are required to acknowledge the work, words or ideas of others through the technique of referencing. Referencing occurs in the text at the place where the work of others is being cited, and at the end of the document, in the bibliography.

The bibliography is a list of all the work (published and unpublished) that a writer has read in the course of preparing a piece of writing. This includes items that are not directly cited in the work.

A reference is required when you:

- Quote directly: when you use the exact words as they appear in the source;
- Copy directly: when you copy data, figures, tables, images, music, videos or frameworks;
- Summarise: when you write a short account of what is in the source;
- Paraphrase: when you state the work, words and ideas of someone else in your own words.

It is standard practice in the academic world to recognise and respect the ownership of ideas, known as intellectual property, through good referencing techniques. However, there are other reasons why referencing is useful.

Good Reasons for Referencing

It is good academic practice to reference because:

- It enhances the quality of your writing;
- It demonstrates the scope, depth and breadth of your research;
- It gives structure and strength to the aims of your article or paper;
- It endorses your arguments;
- It allows readers to access source documents relating to your work, quickly and easily.

Sources

The following would count as ‘sources’:

- Books,
- Chapters from books,
- Encyclopaedias,
- Articles,
- Journals,
- Magazines,
- Periodicals,
- Newspaper articles,
- Items from the Internet (images, videos, etc.),
- Pictures,
- Unpublished notes, articles, papers, books, manuscripts, dissertations, theses, etc.,
- Diagrams,
- Videos,
- Films,
- Music,
- Works of fiction (novels, short stories or poetry).

What You Need to Document from the Hard Copy Source You are Using

(Not every detail will be applicable in every case. However, the following lists provide a guide to what information is needed.)

You need to acknowledge:

- The words or work of the author(s),
- The author(s)'s or editor(s)'s full names,
- If your source is a group/ organisation/ body, you need all the details,
- Name of the journal, periodical, magazine, book, etc.,
- Edition,
- Publisher's name,
- Place of publication (i.e. the city of publication),
- Year of publication,
- Volume number,
- Issue number,
- Page numbers.

What You Need to Document if you are Citing Electronic Sources

- Author(s)'s/ editor(s)'s name,
- Title of the page,
- Title of the site,
- Copyright date, or the date that the page was last updated,
- Full Internet address of page(s),
- Date you accessed/ viewed the source,
- Any other relevant information pertaining to the web page or website.

Referencing Systems

There are a number of referencing systems in use and each has its own consistent rules. While these may differ from system-to-system, the referencing system followed needs to be used consistently, throughout the text. Different referencing systems cannot be mixed in the same piece of work!

A detailed guide to referencing, entitled Referencing and Plagiarism Guide is available from your library. Please refer to it if you require further assistance.

When is Referencing Not Necessary?

This is a difficult question to answer – usually when something is 'common knowledge'. However, it is not always clear what 'common knowledge' is.

Examples of ‘common knowledge’ are:

- Nelson Mandela was released from prison in 1990;
- The world’s largest diamond was found in South Africa;
- South Africa is divided into nine (9) provinces;
- The lion is also known as ‘The King of the Jungle’.
- $E = mc^2$
- The sky is blue.

Usually, all of the above examples would not be referenced. The equation $E = mc^2$ is Einstein’s famous equation for calculations of total energy and has become so familiar that it is not referenced to Einstein.

Sometimes what we think is ‘common knowledge’, is not. For example, the above statement about the sky being blue is only partly true. The light from the sun looks white, but it is actually made up of all the colours of the rainbow. Sunlight reaches the Earth's atmosphere and is scattered in all directions by all the gases and particles in the air. The smallest particles are by coincidence the same length as the wavelength of blue light. Blue is scattered more than the other colours because it travels as shorter, smaller waves. It is not entirely accurate then to claim that the sky is blue. It is thus generally safer to always check your facts and try to find a reputable source for your claim.

Important Plagiarism Reminders

The IIE respects the intellectual property of other people and requires its students to be familiar with the necessary referencing conventions. Please ensure that you seek assistance in this regard before submitting work if you are uncertain.

If you fail to acknowledge the work or ideas of others or do so inadequately this will be handled in terms of the Intellectual Integrity Policy (available in the library) and/ or the Student Code of Conduct – depending on whether or not plagiarism and/ or cheating (passing off the work of other people as your own by copying the work of other students or copying off the Internet or from another source) is suspected.

Your campus offers individual and group training on referencing conventions – please speak to your librarian or ADC/ Campus Co-Navigator in this regard.

Reiteration of the Declaration you have signed:

1. I have been informed about the seriousness of acts of plagiarism.
2. I understand what plagiarism is.
3. I am aware that The Independent Institute of Education (IIE) has a policy regarding plagiarism and that it does not accept acts of plagiarism.
4. I am aware that the Intellectual Integrity Policy and the Student Code of Conduct prescribe the consequences of plagiarism.

5. I am aware that referencing guides are available in my student handbook or equivalent and in the library and that following them is a requirement for successful completion of my programme.
6. I am aware that should I require support or assistance in using referencing guides to avoid plagiarism I may speak to the lecturers, the librarian or the campus ADC/ Campus Co-Navigator.
7. I am aware of the consequences of plagiarism.

Please ask for assistance prior to submitting work if you are at all unsure.

