FIT2081 Final Test Notes

- Second, create the DAO, and define all the query functions you want to have/use.
- Third, the room Database must be defined, it may also contain one or more tables.

- Workshop Quiz

Week 8

- Workshop Quiz

Week 9

- Workshop Quiz

Week 10

- Workshop Quiz

Week 11

- Workshop Quiz

Week 12

- Why you should look after your private key
- Workshop Quiz

Week 1

There are 3 fundamental types of mobile apps, with each of them requiring different development skills, platforms and their own capabilities, advantages, and disadvantages:

- Native Apps:
    - The app's compiled code runs directly on the device platform, and is built using the SDK tools recommended by the platform vendor (Android: Android Studio, Apple: Swift).
    - They require relatively technically sophisticated developers to develop. And the developers will have complete access to the device's features.
    - Usually more fluid, graphically superior, faster, and are less exposed to security risks of a website.
- Mobile Web Apps:
    - A website designed for smart device displays and is accessed by a device's browser. May employ HTML/CSS3 that makes the application to look and feel like a native app.
    - Requires abundant, relatively less sophisticated (and cheaper) Web developers, and this "app" will function the same way across all platforms.
    - Usually different look and feeling to native apps, though HTML5 allows development of a sophisticated UI, it'll still feel different.
    - Most of the time, won't be able to function offline.
    - They're subject to the security risks of websites.
    - They do not have platform fees (Google Play, App Store), and have unrestricted distribution.
    - Updates are instantaneous across all platforms
- Hybrid Apps:
    - Written using a language and development environment other than the recommended languages for the platform but ran as a native app. There are two types:
        - A thin native app shell containing a Web application. It'll contain a shell containing a native component that interacts with the platform's web rendering engine. (Amazon Mobile)
        - A cross compiler to convert code into a native app that is executable for each required target platform. (FireMonkey, Xamarin, Appcelerator)
    - Requires abundant, relatively less sophisticated, and cheap web developers, and the app usually functions the same way across all platforms.

Terms:

- SDK = Software Development Kit: A bundle of all the software components necessary to develop and deploy on a given development platform.
- Class Library/Application Programing Interface (API): Code that we do not write but we can call/execute to perform common yet complicated tasks. Usually are included in huge libraries and can perform a wide variety of tasks from creating a responsive UI to interacting with hardware like the GPS/accelerometer.
- Integrated Development Environment (IDE): A software environment that contains necessary and useful extra features to aid developers in creating applications (Smart debuggers, Refactoring support in IntelliJ/Android Studio).

Week 2

Operating System: Collection of software that manages computer hardware resources and provides common services for computer programs.

Android: Linux kernel OS, Android Software Stack, lightweight, designed for touchscreen mobile devices, and open source.

The Linux kernel provides a multitasking execution environment allowing multiple processes to execute concurrently. Therefore, each android application runs a process directly on the Linux kernel.

API Level: An integer value that uniquely identifies the framework API version offered by the android platform.

Fragmentation: The inability of audiences to access or use the application due to a multitude of reasons:

- Device Fragmentation: The device is too old, too weak or has dimensions that cannot run the application (TVs).
- API Level Fragmentation: Different Android devices utilize different hardware and are run by different companies which have their own version of android. This means that there may be incompatibilities in the subversions of the Android, or bugs that are unavoidable in older android versions. This is a huge issue because not all devices can access the latest API, and if they utilize the latest API, it can still have differences from phone to phone. This is because lots of devices use Android, and Android is an open-source platform.

Forward Compatibility: Old apps running on new platform versions. Android apps are generally forward compatible with new APIs since most of the changes are additive in nature.

Backward Compatibility: New apps running on old platform versions. Android Apps aren't necessarily backward compatible with older versions of Android the App was intended for.

Activities:

- A single, standalone module of application functionality which usually occupies a single UI screen and the functionalities within it. (So the atoms of an android instance are activities not apps).
  -They have lifecycles and can be partially hidden, fully hidden, foreground or destroyed.

Services:

- Processes that are run in the background and have no user interface. They run in the background to perform long running operations or remote processes.
- Can be started/managed by Activities, Broadcast Receivers or others.

Content Provider:

- A mechanism to share data between applications. Any application can provide other applications access to its data by implementing a content provider, including but not limited to the ability of: add, remove or querying the data. This is provided that the application accessing the content provider has the necessary permissions to do so.
- They are useful for reading and writing data to the app. And they manage a shared set of app data, and this data can be kept in the web, file system, SQLITE database. Other apps can query or even modify the data.
- Provides an abstraction from the underlying data source.
- They are activated by a request from a Content Resolver, which will handle direct transactions with the content provider.

Content Resolver:

- The single, global instance in the application that provides access to your and other applications' content providers. It accepts requests from clients, and resolves these requests by directing them to the content provider with a distinct authority.
- They include the ability to CRUD (Create, Read, Update, Delete).
- They provide an abstraction from the application's Content Providers.

Broadcast Receivers:

- The mechanism in which applications are able to respond to Broadcast Intents, they must be registered by an application and configured with an Intent Filter to indicate the types of Broadcasts the app is interested to.
- When a matching intent is broadcasted, the receiver will be invoked by the Android runtime, regardless of whether the application that has the receiver is running or not.
- They are given 5 seconds to complete any required tasks, and they operate in the background with no UI.

Intents:

- 3 of the 4 component types being activities, services, and broadcast receivers are activated by an asynchronous message called an intent. Intents bind individual components together at runtime, and they act as messengers that request action from other component(s).
- They are created with the intent object.
- Activity Intents: The mechanism by which one activity can launch another activity/service and implement the flow through the activities that make up an application.
- Broadcast Intents: A system wide intent that is sent out to all applications that have an interested Broadcast Receiver.

Manifest File: An XML file that will detail all of an apps components, capabilities, and more. They include a declaration of all components in the application. If a component isn't declared, then the system can't see it. It will also:

- Identify any user permissions the app requires.
- The minimum API level used by the app.
- Declares the hardware and software features used by the app.
- Declares libraries the app needs.

Resources Folder: Contains strings, images, audio, videofonts, colors that may be used in the UI. We should also utilize the resources folder by creating different resources for different device configurations (landscape, portrait, language).

Context: An abstract class that allows access to application specific resources and classes. The Android/Application Context may be used in the application code to gain access to the application resources at runtime.

Workshop Quiz:

1. Assume you are developing an application that should be available in two different languages (English and French).
    a. Develop an application for each language separately is the best solution. Do you agree?? Justify your answer.
    b. b) If not, what alternative approach would you recommend and why?
2. Suppose you are developing an Android app that needs to keep a method (function) running in the background. Briefly explain how would you implement this function?
    a. List two similarities between Activities and Services.
    b. What is common between Activity, Service, and Broadcast Receiver?
3. Consider an app compiled against API level 29 on a device running Android 11 (API Level 30)
    a. This usually works. Why?
    b. How could this go wrong?
    c. Can it be fixed without changing the API level of the app and the device? Explain.

1. No, I do not agree as it is not the best solution available. That is because creating two separate applications will be inefficient as changes made to the "main" version of the app will need to be replicated in the version for the other language, and it is a waste of time/cost/space to develop the app separately.

    b. Android provides the ability to modify the Resources folder and allows us to plug in alternative resources for different configurations, such as different languages. We can simply provide another XML file for resources in other languages. By defining UI strings in the XML, we can simply translate the string into other languages and save the strings in separate files. Meaning, we can insert different files for different languages to the resource directory and the Android system will automatically apply the appropriate language according to the user's language setting. This way, it also allows a better experience for the user as they do not need to download separate apps just to change languages.

2. Using an intent, use it to initiate a Service. A service is appropriate since it can run independently after the app starts it, even when the app's activity is no longer in focus. That's because the nature of a service is to run in the background, as it has no User Interface.

    b. They're both initiated by Intents. Both of them can be started by Activities through such use of an Intent.

    c. Activities, services and broadcast receivers are activated by an asynchronous message called an intent.

3. Because apps in Android are generally forward compatible which means that older applications can usually run on newer APIs. That's because changes to the Android API are generally additive in nature, which means that all the necessary framework to run the older app are usually present in the newer Android platform.

    b. If the older app makes use of features/libraries that was present in the older API the app was developed on, and was later removed in newer versions of the API.

    c. Isolate features from the older app that makes use of libraries or features from the 29th Level API and remove them. This way that the newer device can run the app as it does not have dependencies that are unique to the older Android version. Otherwise, the app would likely be need to be updated and compiled in the new API.

Week 3

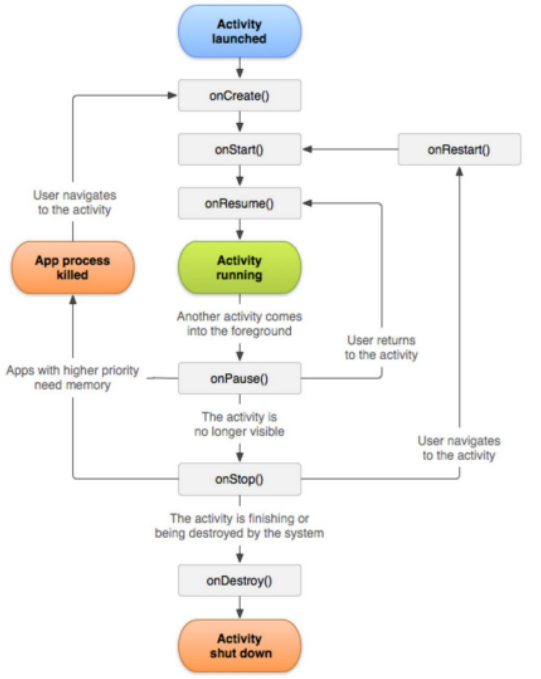
Android will manage resources, and they may free up memory which will (in the process) kill processes if it's required. Processes are ranked by their highest-ranking active component, and some processes may depend on each other. Processes that has a foreground activity cannot be killed, but partially hidden or fully hidden activites can be.

Activities go in and out of their possible lifecycle states as a result of user interaction or OS process kills.

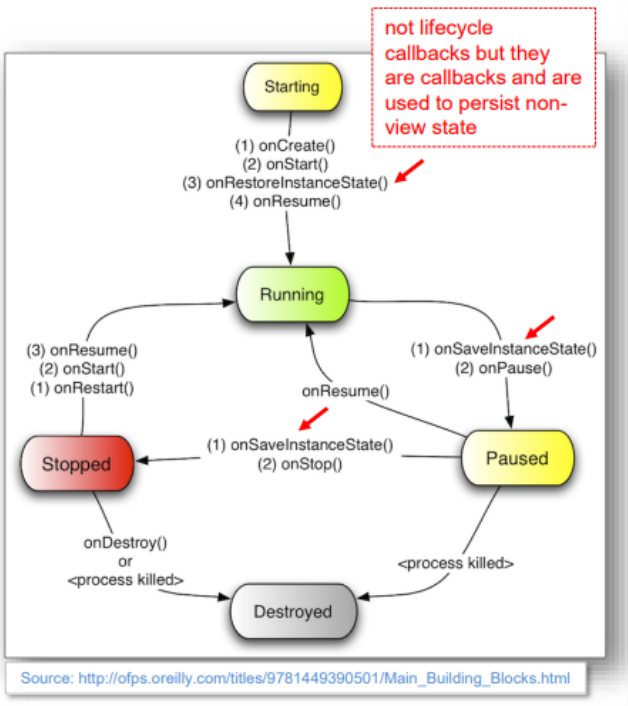
Activity Lifecycle

To keep things clear in your mind say Processes are killed by the OS but Activities are Destroyed by the OS or by User actions.

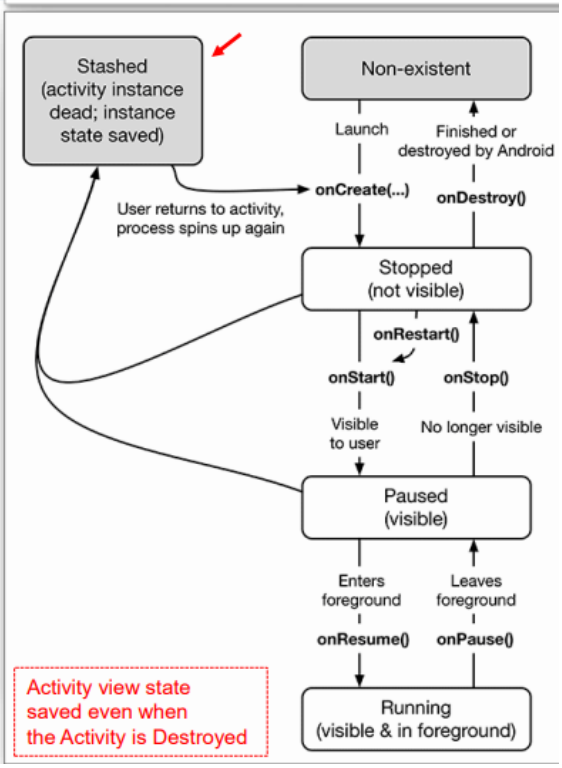
- Activity States
  - foreground, partially hidden, fully hidden, destroyed
- Lifecycle Callbacks
  - onCreate(), onStart(), onResume(), onPause(), onStop(), and onDestroy()
- Lifecycle Loops
  - Partially hidden
    - Navigate back before destroyed (recents, home)
  - Fully hidden
    - Navigate back before destroyed (recents, home)
  - Process Destroyed
    - App relaunched
- "Back" destroys an Activity!



Source: Android Programming – The Big Nerd Ranch Guide (Phillips, B. et al)



Source: http://ofps.oreilly.com/titles/9781449390501/Main_Building_Blocks.html



Within lifecycle callback methods, developers can declare how the activity behaves when the user leaves and re enters the activity. This can be done by overriding the method. Yet super() must always be called and must always begin callback overrides, because super will perform a lot of standard procedures required by any activity, failure to call the super will cause the callback method to misbehave.

Activity Datas:

- Activity Instance Data:
    - View data (the state of the views in the layout)
    - Non-view data (activity instance variables
    - They will involve reading and writing to the device's volatile memory.
- Persistent Data
    - Data that exists across app destroys, they can be made app private, but not activity private. And they will be writing and ready from the device's non volatile memory.

Persistence Across Events. But what events?

- Device configuration event (rotating device)
- App is uninstalled from the device
- Activity is partially or fully hidden (not killed)
- Activity is destroyed using the back button

How to save/restore activity instance states:

onSaveInstanceState:

- Not called if user indicates they are done with the Activity
- Must call super to let Android save view-state (Not required in the syntax)
- Must code to manually save non-view state in Bundle

onCreate:

- Must call super to let Android restore view-state (Syntax requirement)
- Must code to manually restore previously saved non-view state from Bundle

onRestoreInstanceState:

- Only called if the Bundle is Non null
- Must call super to let Android restore view-state it previously saved (Not required in the syntax)
- Must code to manually restore previously saved non-view state from Bundle

onDestroy:

- Not called when the app is destroyed by the Android system to free up resources.

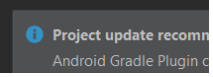
The parameters (A bundle) in each of the 3 callbacks is a shared bundle object that lets Android to save/restore an activity's view state. But Android will automatically auto save the view state of an activity (except if finish() and the back key is pressed).

How to save data to non-volatile memory:

- Key and Value Sets: (Using shared preferences)
- SQL Databases

Using SharedPreferences:

```
public void loadPref(View view){
    sharedPref = getPreferences(Context.MODE_PRIVATE);
    if (sharedPref.getString( s: "preference", clearString).equals("True")){
        title.setText(sharedPref.getString( s: "titlePref", clearString));
        year.setText(sharedPref.getString( s: "yearPref", clearString));
        country.setText(sharedPref.getString( s: "countryPref", clearString));
        genre.setText(sharedPref.getString( s: "genrePref", clearString));
        cost.setText(sharedPref.getString( s: "costPref", clearString));
        keywords.setText(sharedPref.getString( s: "keywordsPref", clearString));
        actorNumber.setText(sharedPref.getString( s: "actorNumberPref", clearString));
        Toast.makeText(getApplicationContext(), text: "Preferences loaded successfully", Toast.LENGTH_LONG).show();
        return;
    }
    Toast.makeText(getApplicationContext(), text: "Preferences doesn't exist", Toast.LENGTH_LONG).show();
}

public void savePref(View view){
    SharedPreferences.Editor editor = sharedPref.edit();
    editor.putString( s: "titlePref", title.getText().toString());
    editor.putString( s: "genrePref", genre.getText().toString());
    editor.putString( s: "keywordsPref", keywords.getText().toString());
    editor.putString( s: "yearPref", year.getText().toString());
    editor.putString( s: "countryPref", country.getText().toString());
    editor.putString( s: "costPref", cost.getText().toString());
    editor.putString( s: "actorNumberPref", actorNumber.getText().toString());
    editor.putString( s: "preference", s1: "True");
    editor.apply();
```



Workshop Quiz

1. Suppose you have an activity with an instance variable (class level) of type float called ‘fees’. Your task is to save and restore the value of ‘fees’ each time the activity changes its orientation without saving and restoring all the view data. Implement the required code.

```
public class MainActivity extends AppCompatActivity {
    private float fees;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    //your code goes here
}
```

2. Write a piece of code that stores the following key-value pairs asynchrony in a shared preferences file named “s12022.fit2081”
    a. week:3
    b. task: Q2
    c. isCode:true
3. What is the effect of implementing onBackPressed() without calling super.onBackPressed()?
4. Briefly explain the scenarios where the following callback methods do not get invoked during Activity lifecycle.
    a. onSaveInstanceState()

b. onRestoreInstanceState()
c. onDestroy()

1.
```
public class MainActivity extends AppCompatActivity {
    private float fees;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    protected void onSaveInstanceState(@NonNull Bundle outState) {
        outState.putFloat("fees", fees);
    }

    @Override
    protected void onRestoreInstanceState(@NonNull Bundle savedInstanceState) {
        fees = savedInstanceState.getFloat("fees", 0.0f);
    }
}
```

2.
```
SharedPreferences sp = getSharedPreferences("s12022.fit2081", Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sp.edit();
editor.putInt("week", 3);
editor.putString("task", "Q2");
editor.putBoolean("isCode", true);
editor.apply();
```

3. Super calls are necessary for the function to actually behave like they're supposed to. Usually we override a callback function if we want the method to perform extra behaviors should the conditions for the callback method be met, but the super call is usually left there. Should this super call be removed from onBackPressed(), it will likely mean that the activity won't be destroyed at all and we won't be taken to the previous activity (Should it exist). Meaning, it's as if the back button wasn't pressed at all, although any custom implementation we added to the onBackPressed() will still run.

4. a. Not called if the user is finished with the activity. This can happen if the user exits the app using a back button, or the finish() method is invoked, which will kill the current activity.

   b. If the app is killed (using the back button), or if the Bundle is Null (Which means there's nothing to restore).

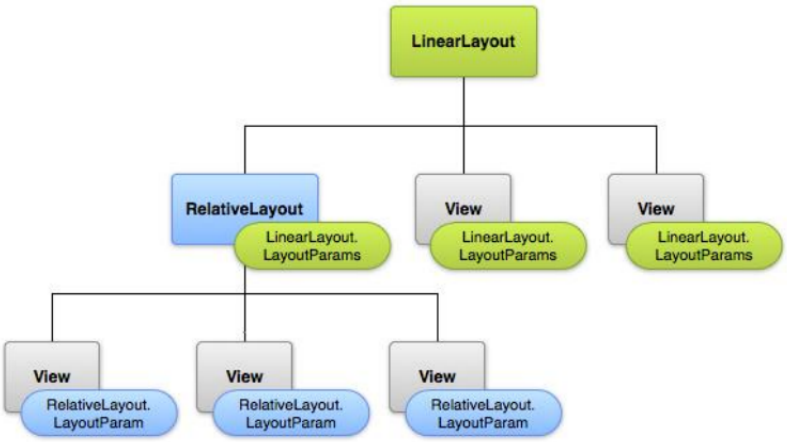
   c. If the app is killed by Android, or if the app's configuration does not change to the point where onDestroy() needs to be invoked (no change in orientation).

Week 4:

Android UIs are made up of views.

View: Represents the basic building block for UI components, it occupies a rectangular area on the screen and can handle events as well. It is the base class for widgets which are used to create interactive UI components like buttons or EditText.

ViewGroup: The base class for layouts and they are invisible containers that hold views. They allow views to be nested. And they also define the ViewGroup.LayoutParams class which serves as the base class for layout parameters. Since ViewGroup is a View, it can be contained by another ViewGroup. So nesting can be done to optimise each part of the UI. But it's strongly recommended by Google not to do so for efficiency and clarity reasons.



Layout: Defines the visual structure for a UI, such as an activity's UI or app widget. And they can be declared in two ways:

- Declare the UI elements in XML, they provide a straightforward XML vocabulary that corresponds to the View and its subclasses.
- Instantiate layout elements at runtime.

Declaring the UI in the XML enables us to better separate the presentation of the application from the code that controls its behavior. If the UI's description is separated from the application code, it means we can modify or adapt it without needing to modify the source code or recompile. It's also much easier to visualize the structure of the UI, which will make it easier to debug problems.

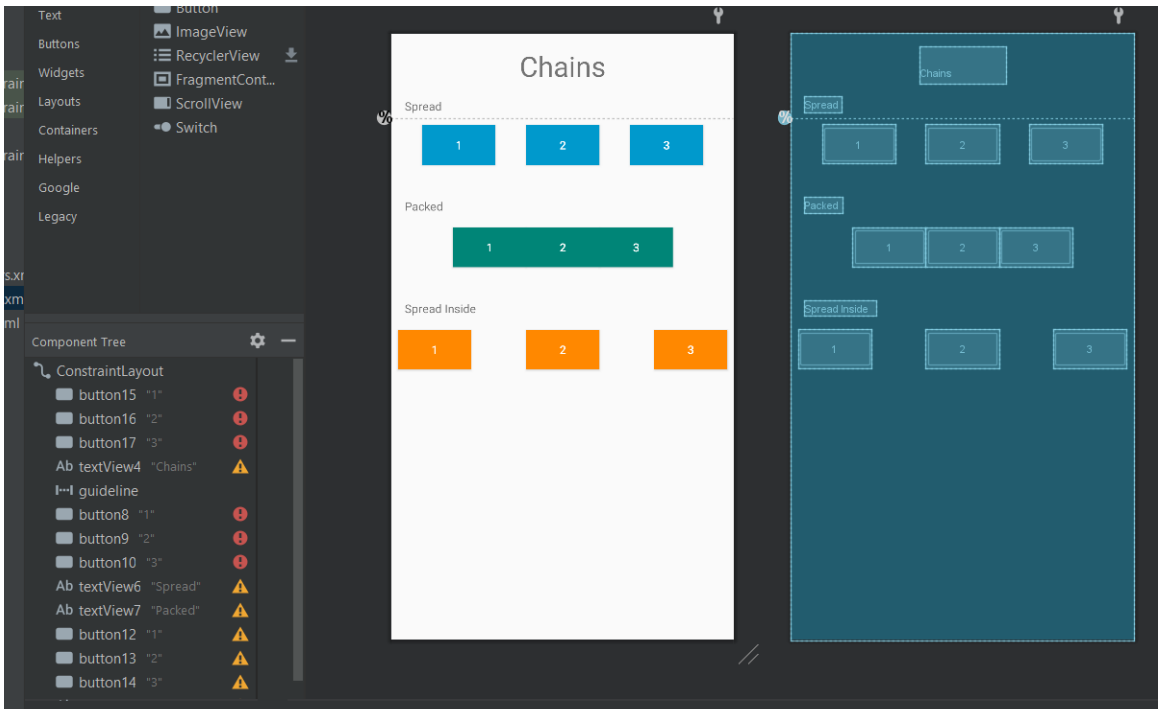
ConstraintLayout: Views are attached to the layout sides or horizontal and vertical guidelines. They act as "springs" and they can expand/collapse depending on configuration/orientation the device is being displayed on. The tension on the springs holding the view between two endpoints can be biased towards one end or the other by a percentage slider.

Important Note: Professional Developers prefer creating a custom listener compared to onClick to separate presentation and logic.

Style: A collection of attributes that specify the look and format for **A** view or window. It can specify attributes like height, padding, font color/size, background color and is defined in an XML resource that is separated from the XML that specifies the layout.

Style can be inherited, and the parent attribute in the style element lets you inherit attributes from an existing style and define only the attributes you want to change or add.

Theme: A style applied to an **ENTIRE** activity or app rather than an individual view. When a style is applied as a theme, every view in the activity or app applies each style attribute that it supports. This is efficient in applying styles accross the board compared to setting them one by one.



Action Bar: It was the default theme before the Lollipop version, and not a widget. But after Lollipop it's no longer part of the default theme, as developers are expected to work with a designated toolbar widget instead.

```
Toolbar toolbar = findViewById(R.id.toolbar);
setSupportActionBar(toolbar);
```

Setting a toolbar can be done by using the setSupportActionBar() method. It is preferred to use a Toolbar as the AppBar and not the ActionBar. Which means we should be picking a theme that DOES contain ".NoActionBar" as opposed to otherwise.

Broadcast Intents: Intent objects that are broadcast via sendBroadcast()/sendStickyBroadcast()/sendOrderedBroadcast() methods of the activity class. It provides a messaging and event system between applications, and can be used by the Android system to notify interested apps about key system events (such as connecting Bluetooth, low battery).

When an intent is created, it must include an action string and optional data, done so in a key and value pair using the putExtra() method.

```
Intent intent = new Intent();
intent.setAction("intent.filter.data");
```

```
intent.putExtra("key", "SomeData");

sendBroadcast(intent);
```

In order to listen to specific broadcasts, an application needs to register broadcast receivers. And it can be done by extending the BroadcastReceiver class and then overriding the onReceive() method.

```
class MyBroadCastReceiver extends BroadcastReceiver {

@Override

public void onReceive (Context context,Intent intent) { }

}
```

The second parameter (Intent) is the intent where you can find all the data items.

```
String theData= intent.getExtras().getString("key");
```

We can register the receiver by doing so:

```
registerReceiver(new MyBroadCastReceiver(),new IntentFilter("intent.filter.data"))
```

The first parameter is an instance of MyBroadCastReceiver while the second parameter is the string that is used in declaring the broadcast intent. (That we want to intercept).

Example: SMS Receiver

Create a custom Broadcast receiver:

```
public class SMSReceiver extends BroadcastReceiver {

@Override

public void onReceive(Context context, Intent intent){}

}
```

Then retrieve the list of messages from the intent inside the onReceive() method.

```
SmsMessage[] messages= Telephony.Sms.Intents.getMessagesFromIntent(intent);
```

Then iterate through to the SMS body and number:

```
public class SMSReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        SmsMessage[] messages = Telephony.Sms.Intents.getMessagesFromIntent(intent);
        for (int i = 0; i < messages.length; i++) {
            SmsMessage currentMessage = messages[i];
            String senderNum = currentMessage.getDisplayOriginatingAddress();
            String message = currentMessage.getDisplayMessageBody();
            Toast.makeText(context, "Sender: " + senderNum + ", message: " + message,
Toast.LENGTH_LONG).show();
        } }
```

But to read SMS, it must request permissions to do so:

```
ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.SEND_SMS,
Manifest.permission.RECEIVE_SMS, Manifest.permission.READ_SMS}, 0);
```

And these requests should be added to the manifest as well:

```
   < uses-permission android:name="android.permission.RECEIVE_SMS">
    < uses-permission android:name="android.permission.READ_SMS">
    < uses-permission android:name="android.permission.SEND_SMS">
    < uses-permission android:name="android.permission.WRITE_SMS">
```

Example: Phone Call Listener:

```
public class MyCallsReceiver extends BroadcastReceiver {
    Context self;
    static TelephonyManager telephonyManager;

    public void onReceive(Context context, Intent intent) {
        self = context;
        if (telephonyManager == null) {
            telephonyManager = (TelephonyManager)
context.getSystemService(Context.TELEPHONY_SERVICE);
            MyPhoneStateListener PhoneListener = new MyPhoneStateListener();
            telephonyManager.listen(PhoneListener, PhoneStateListener.LISTEN_CALL_STATE);
        }
    }

    private class MyPhoneStateListener extends PhoneStateListener {
        public void onCallStateChanged(int state, String incomingNumber) {
            if (state == 1) {
                showToast("Number:=" + incomingNumber);
            }
        }
    }
}
```

These permissions should also be added:

```
List wantedPermissions = new ArrayList<>();
wantedPermissions.add(Manifest.permission.CALL_PHONE);
wantedPermissions.add(Manifest.permission.READ_CALL_LOG);
wantedPermissions.add(Manifest.permission.READ_PHONE_STATE);
wantedPermissions.add(Manifest.permission.PROCESS_OUTGOING_CALLS);
ActivityCompat.requestPermissions(this, wantedPermissions.toArray(new
String[wantedPermissions.size()]), 0);
    < uses-permission android:name="android.permission.READ_PHONE_STATE" />
    < uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS"/>
```

```
< uses-permission android:name="android.permission.READ_CALL_LOG"/>
```

Workshop Quiz

1. When does the horizontal bias in constraint layout become redundant? (3m)
    a. Explain one difference and one similarity between Theme and Style in Android. (4m)
2. Update the following activity to catch the broadcast sent by the method onBtnClickHandler @line 12 and show the embedded messages (see line 15) in a toast. (10m)

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //your code goes here
    }

    //your code goes here

    public void onBtnClickHandler(View view){
        Intent msg=new Intent();
        msg.setAction("S12022W4");
        msg.putExtra("KEY","Catch me if you can!!!!");
        sendBroadcast(msg);
    }
}
```

3. Given the code below, briefly explain the reason behind declaring the input parameter 'param' as of type 'View'. (6m)

```
public void onBtnClickHandler(View param){
  // some code goes here
}
```

1. The horizontal bias becomes redundant, if we also implement a horizontal chain linked between views. As we would be adjusting the bias on the same axis. A chain depends on it's relative positions to other Viewobjects. Or if the size of the View matches the size of the layout/available space, which means that the horizontal bias becomes redundant since it cannot be moved to the left or the right.

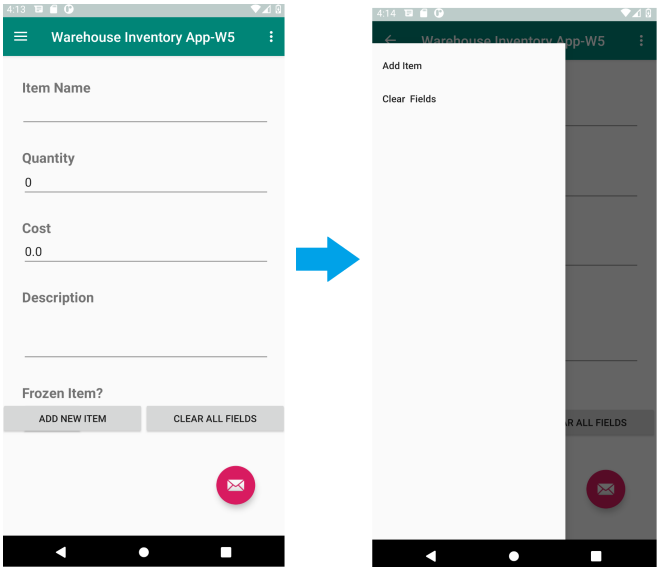
   b. Both are collection of attributes that can change the look, format and aesthetics of a single View/Window. The look involves color, font, size, etc. The main difference is that the attributes of a style will only apply to that single View/Window, while a theme is a collection of styles grouped together that can change the look and feel of the entire view as well as it's children.

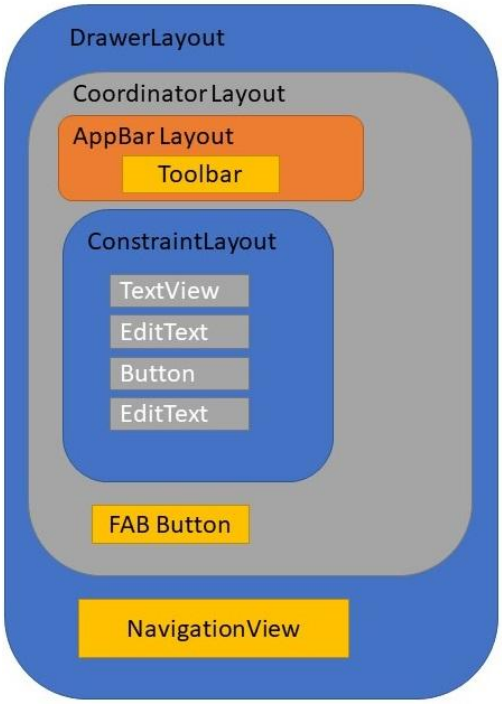
2.
```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        registerReceiver(new MyBroadCastReceiver(),new IntentFilter("S12022W4"))
    }

    public class MyBroadCastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive (Context context,Intent intent) {
       Toast.makeText(this, intent.getStringExtra("KEY") ,Toast.LENGTH_SHORT
).show();
    }
    }


    public void onBtnClickHandler(View view){
        Intent msg=new Intent();
        msg.setAction("S12022W4");
        msg.putExtra("KEY","Catch me if you can!!!!");
        sendBroadcast(msg);
    }
}
```

3. The view parameter indicates the View object that triggers this specific onBtnClickHandler callback. It is necessary so that the function is able to know which button is actually being clicked. This way, we can perform operations, or provide extra behavior to said button after being clicked. View itself is chosen as the class type because it is a generic class that is extended by many components. This is a form of polymorphism in Java.

Week 5

DrawerLayout: Acts as a top level container for window content that allows for interactive drawers to be puddle out from one or both vertical edges of the window. The positioning and layout is controlled using the android:layout_gravity attribute on child views corresponding to which side of the view you want the drawer to emerge from (left/start or right/end). Only one drawer view is available for each vertical edge of the window, if more is configured to appear from one vertical edge, then an exception will be thrown.





A layout of type CoordinatorLayout which contains:

- A layout of type AppBarLayout that wraps a toolbar
- A constraint layout that represents the main white area that will include all the text views, buttons, and etc

A navigation view that represents the drawer that can be pulled from the left or right:

- A menu resource file that will contain the options to be displayed within the navigation drawer.

Instantiating a Coordinator Layout

Add these lines to the build.gradle:

```
implementation 'androidx.appcompat:appcompat:1.1.0'
implementation 'com.google.android.material:material:1.1.0'
implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
```

Add a new layout resource file in the layout folder with the following details:

- file name: activity_drawer_main.xml (or any name you prefer)
- root element: androidx.drawerlayout.widget.DrawerLayout

Inside the drawer layout, two items must be included:

- First, a layout that should contain the toolbar, a constraint layout and a floating action button.
  - file name: app_bar_main (or any name you prefer)
  - root element: androidx.coordinatorlayout.widget.CoordinatorLayout
  - And the file we just created should be included in the drawerlayout with the following details:
  - `<include`
  - `layout="@layout/app_bar_main"`
  - `android:layout_width="match_parent"`
  - `android:layout_height="match_parent" />`
- Second, a navigation drawer should be added to the drawer layout with the following details in the resource file:
  - `<com.google.android.material.navigation.NavigationView`
  - `android:layout_width="wrap_content"`
  - `android:layout_height="match_parent"`
  - `android:id="@+id/nav_view"`
  - `android:layout_gravity="start"`
  - `android:fitsSystemWindows="true"`
  - `app:menu="@menu/nav_menu" />`

The attribute "`android:layout_gravity="start"`" indicates that the position of the drawer is on the left side of the screen, and the attribute "`app:menu="@menu/nav_menu"`" represents the menu file nav_menu.

Creating a Resource Menu file

1. Under the res folder, create a new Android Resource Directory named and with the type "menu".
2. Right click the menu folder, and select New-->Menu Resource File with the file name "nav_menu"
3. In the nav_menu, add these details:
4.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/item_id_1"
        android:title="Item One" />
    <item
        android:id="@+id/item_id_2"
        android:title="Item Two" />
</menu>
```

The attribute "`android:id="@+id/item_id_1"`" represents the ID for the menu item. While the "`android:title="Item One"`" sets the title for the menu item.

The details in the CoordinatorLayout will look like the following:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.coordinatorlayout.widget.CoordinatorLayout
    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="edu.monash.navdrawer2081student.MainActivity">

    <com.google.android.material.appbar.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/AppTheme.AppBarOverlay">

        <androidx.appcompat.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            app:popupTheme="@style/AppTheme.PopupOverlay" />

    </com.google.android.material.appbar.AppBarLayout>

    <include layout="@layout/content_main"/>


</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

And a FAB may also be included:

```
<com.google.android.material.floatingactionbutton.FloatingActionButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/fab"
    android:layout_margin="45dp"
    android:layout_gravity="bottom|end"
    app:srcCompat="@drawable/ic_add_black_24dp" />
```

Where the "`android:layout_margin="45dp"`" specifies the margins around the FAB button, the "`android:layout_gravity="bottom|end"`" specifies the location of the FAB button, which means it's being placed in the bottom right corner of the screen, and "`app:srcCompat="@drawable/ic_add_black_24dp"`" specifies the icon used for the FAB button.

The drawer and the toolbar should be hooked in the onCreate, and this is what needs to be written:

```
Toolbar toolbar = findViewById(R.id.toolbar);
setSupportActionBar(toolbar);
DrawerLayout drawer = findViewById(R.id.drawer_layout);
ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
        activity: this, drawer, toolbar, "Open navigation drawer", "Close navigation drawer");
drawer.addDrawerListener(toggle);
toggle.syncState();

NavigationView navigationView = findViewById(R.id.nav_view);
navigationView.setNavigationItemSelectedListener(this);
```

The 5 arguments needed in ActionBarDrawerToggle are:

1. Current Activity context
2. Drawer Layout Variable
3. Toolbar Variable
4. Drawer open description message via resource string
5. Drawer close description message via resource string

and the navigationView.setNavigationItemSelectedListener(this) means we are using a custom implementation so that the drawer's items can respond to our clicks. And it has the following implementation:

```
class MyNavigationListener implements NavigationView.OnNavigationItemSelectedListener {
    @Override
    public boolean onNavigationItemSelected(@NonNull MenuItem item) {
        // get the id of the selected item
        int id = item.getItemId();
        if (id == R.id.item_id_1) {
            // Do something
        } else if (id == R.id.item_id_2) {
            // Do something
        }
        // close the drawer
        drawerlayout.closeDrawers();
        // tell the OS
        return true;
    }
}
```

The FAB should also have it's own custom listener with the following implementation:

```
FloatingActionButton fab =  findViewById(R.id.fab);

fab.setOnClickListener(new View.OnClickListener() {

   @Override

   public void onClick(View view) {

       saveData();

       Toast.makeText(getApplicationContext(), "Toast! " + title.getText().toString() + "
has been saved!", Toast.LENGTH_LONG).show();

   }

});
```

A snackbar can also be implemented with the following code:

```
Snackbar.make(view, "Replace with your own action",
Snackbar.LENGTH_LONG).setAction("Action", null).show();
```

The listener is an anonymous class of type View.OnClickListener, so if the user clicks the FAB button, the callback custom onClick will get executed.

The Snackbar itself is a widget that provides a brief feedback about an operation through a message at the bottom of the screen, like the Toast, it can disappear automatically (with time, or even being swiped of the screen) and it provides the user with an opportunity to have a clickable that can be programmed to run an action.

Creating an Options Menu file

1. Add a new menu resource file under the Menu folder directory in the resources, it will be named "options_menu"
2. Add two items to the menu file with the following implementation:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/action_settings"
        android:title="Settings" />
    <item
        android:id="@+id/action_Logout"
        android:title="Logout" />
</menu>
```

Two methods must be implemented which are the onCreateOptionsMenu and onOptionsItemSelected

This method will be invoked to inflate the menu list, and it will accept the input as a reference to the menu object that should be passed to the inflator.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.options_menu, menu);
    return true; }
```

And when the menu option is clicked, the callback below will get invoked and it will receive an input as a reference to the menu item that was clicked. The return true is used to indicate an event has been consumed. So that the default action of calling the associated item's runnable or sending a message to it's handler isn't called.

```
@Override
public boolean onOptionsItemSelected(@NonNull MenuItem item) {
    int id=item.getItemId();
    if (id == R.id.action_Logout) {
        Toast.makeText(this,"Logout...",Toast.LENGTH_SHORT).show();
    } else if (id == R.id.action_settings) {
        Toast.makeText(this,"Settings...",Toast.LENGTH_SHORT).show();
   }
    return true; }
```

Make Layout Scrollable:

Simply wrap the constraint layout in a ScrollView

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView 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:fillViewport="true"
    android:layout_height="match_parent">

//Constraint Layout details
</ScrollView>
```

Floating Action Button: It should be used to perform a single, obvious and most common action.

Snackbar: Is a panel that appears at the bottom of the screen containing a message and an optional action button (can be Undo). And this panel can be made to disappear after some time or require the user to swipe it away. The difference with Toast is that Toasts will always disappear by themselves and has no action button.

The Snackbar.make() returns a snackbar object and the snackbar's set action is invoked on this object to specify both the text on it's action and the listener object that contains the function for the action for when it's clicked.

They (like Toasts) are made and displayed in code (not in the layout XML).

ListView: A view that groups several items and displays them in a vertically scrollable list, they usually source content from an array or database. The content is sourced from the associated ListAdapter.

The way to setup a ListView (can be done in onCreate) is:

1. Get a reference to the ListView
2. Instantiate an adapter, that will specify the data source (for simplicity, it'll be an ArrayList)
3. Plug the adapter into the ListView using the ListView's setAdapter(...) method which will take a ListAdapter as its only parameter.

One Adapter that can be used is an ArrayAdapter<T> which is a generic Java Class type. The T must be replaced with a specific reference type.

We can use the adapter's add(ItemToAdd) and remove(indexToRemove) to add and remove items to the ListView.

Example: adapter.remove(-1) to remove last item in the index.

Then, after every data insertion or deletion, we should use the adapter's notifyDataSetChange() to ensure the ListView is refreshed to update the latest data.

Workshop Quiz

1. What are the roles of ArrayAdapter's second and third parameters?
2. Suppose you have a button with id="btn" that is linked to an event handler method "onClickBtnHandler() at the design time using XML. Write a piece of code that links the same button to another handler that is responsible for calling a method called "executeAlternativeBtnHanlder()".
3. Assume you have a right-to-left drawer in your activity, write a java statement that closes it.
4. Assume you have an activity with a listview (id="s12022) in it. Develop a piece of code that adds the following items to the listview: (10m)
    a. Week 5
    b. FIT2081

1. The second parameter of the ArrayAdapter constructor specifies an integer which is the resource ID of a layout that contains a TextView used to instantiate Views and it will be used for rows in your ListView,. While the third parameter is the list to be supplied to the ArrayAdapter. This adapter will take the third parameter as the source of data to later supply to the List View.

2.
```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   Button btn = findViewById(R.id.btn);
   btn.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view) {
            executeAlternativeBtnHandler();
      }
  });
}
```

3. In the onCreate:

```
NavigationView navigationView = findViewById(R.id.nav_view);
navigationView.setNavigationItemSelectedListener(this);
```

Outside onCreate:

```
@Override
public boolean onNavigationItemSelected(MenuItem item) {
DrawerLayout drawer = findViewById(R.id.drawer_layout);
drawer.closeDrawer(GravityCompat.END);
return true;
}
```

or

if (customDrawer.isDrawerOpen(GravityCompat.END)) {

customDrawer.closeDrawer(GravityCompat.END);

}

This is the correct one

4. ListView listView = findViewById(R.id.s12022);

   ArrayList<String> list = new ArrayList<String>();

   ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, list);

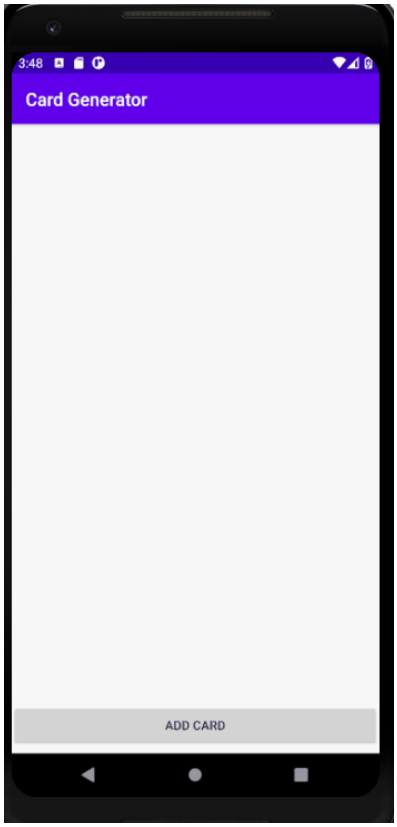
   listView.setAdapter(adapter);

   adapter.add("Week 5");

   adapter.add("FIT2081");

   adapter.notifyDataSetChanged();

Week 6



RecyclerView: A flexible ViewGroup for displaying a large data set into a limited window. The purpose is to allow information to be presented in the form of a scrollable list. It's significantly more efficient than a ListView in how it manages the View that make up a list, essentially reusing existing views that makeup the list items as they scroll off the screen instead of creating new ones (which is why it's called RecyclerView.

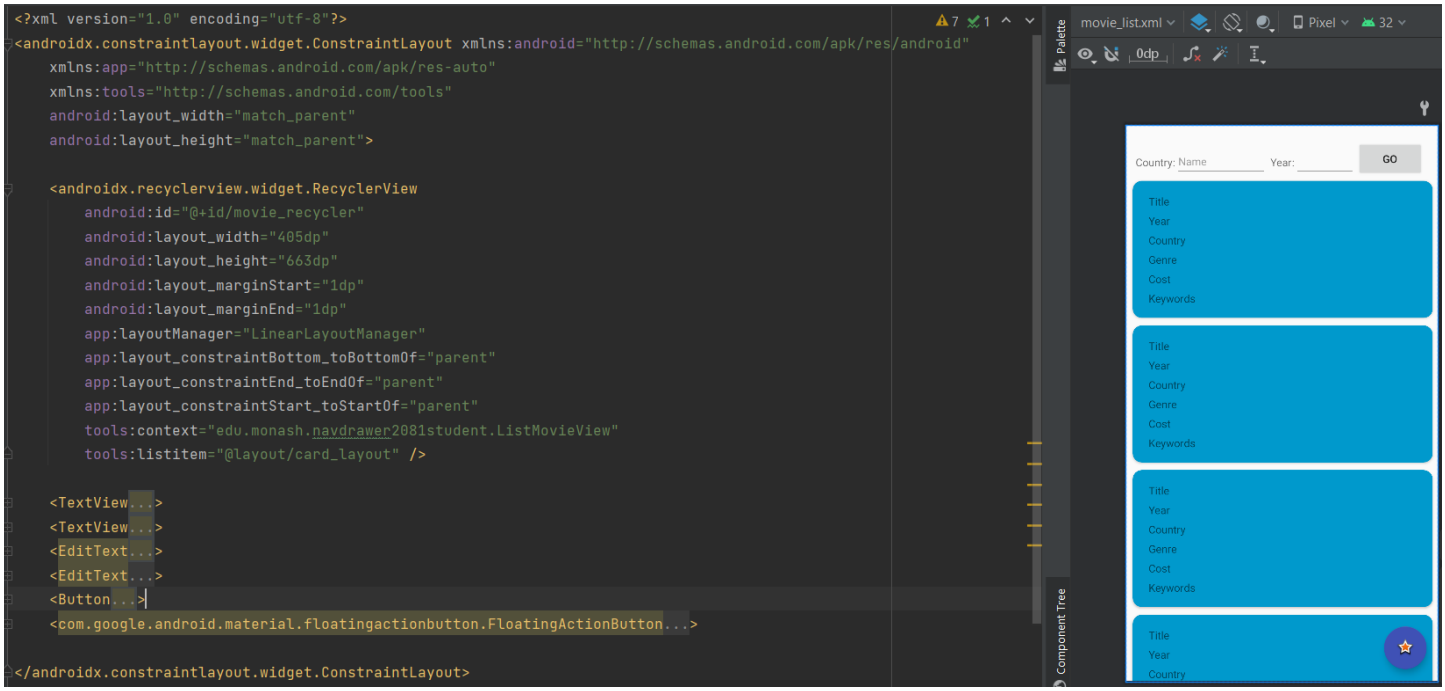
RecyclerView is created as an improvement of ListView since it reuses cells while scrolling up/down and it decouples the list from it's container, so you can put list items easily at run time in the different containers with the setting layoutManager. They also support both vertical and horizontal scrolling, and integrates animations for adding, updating and removing items.

However, the ListView supports onItemClickListener. And they support the usage of default adapters, contain a header/footer and it's generally simpler to code.

CardView: A user interface view that allows information to be presented in groups using the card metaphor. And cards are usually shown in lists using an instance of RecyclerView and can be configured to appear with shadow effects, rounded corners and other aesthetic effects. The UI layout to be presented with a CardView instance is defined within an XML layout resource file and loaded into the CardView at runtime. The CardView layout can contain a layout of any complexity using the standard layout managers like RelativeLayout and LinearLayout.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:id="@+id/card_view"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="5dp"
    card_view:cardBackgroundColor="@android:color/holo_blue_dark"
    card_view:cardCornerRadius="12dp">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="180dp"
        android:padding="16dp">
        <TextView...>
        <TextView...>
        <TextView...>
        <TextView...>
        <TextView...>
        <TextView...>
        <TextView...>
        <TextView...>
        <TextView...>
        <TextView...>
        <TextView...>
        <TextView...>
    </RelativeLayout>
</androidx.cardview.widget.CardView>
```

The RecyclerView below is contained in a constraint layout as the primary container.




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

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/movie_recycler"
        android:layout_width="405dp"
        android:layout_height="663dp"
        android:layout_marginStart="1dp"
        android:layout_marginEnd="1dp"
        app:layoutManager="LinearLayoutManager"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        tools:context="edu.monash.navdrawer2081student.ListMovieView"
        tools:listitem="@layout/card_layout" />

    <TextView...>
    <TextView...>
    <EditText...>
    <EditText...>
    <Button...>
    <com.google.android.material.floatingactionbutton.FloatingActionButton...>

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

In order to use a RecyclerView, then we need to implement a RecyclerView.Adapter. The adapter is created as a subclass of the RecyclerView.Adapter class and must (at a minimum) implement the following methods:

- **getItemCount**(): This method must return a count of the number of items that are to be displayed in the list. This is important as the amount of items/cards instantiated will be determined by the value returned by getItemCount. So if a list/database/array actually has 100 items and getItemCount only returns 10, then only 10 if the source's items will be displayed.

```
@Override
public int getItemCount() {
    if (movies == null){
        return 0;
    }
    return movies.size();
}
```

- **onCreateViewHolder**(): The method creates and returns a ViewHolder object initialized with the view that is to be used to display the data. The view is typically created by inflating the XML layout file.

```
@NonNull
@Override
public MovieRecyclerAdapter.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.card_layout, parent,  attachToRoot: false);
    return new ViewHolder(v);
}
```

- **onBindViewHolder**(): This method passes the ViewHolder object created by the onCreateViewHolder() method and an integer value indicating the “index” or the position of the list item that is about to be displayed. Contained within the ViewHolder object is the layout assigned by the onCreateViewHolder() method. The purpose of this method is to populate the views in the layout with the text/graphic/information that corresponds to the specific item, at the index specified by the the second parameter position.

```
@Override
public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
    holder.titleV.setText(movies.get(position).getTitle());
    holder.genreV.setText(movies.get(position).getGenre());
    holder.keywordsV.setText(movies.get(position).getKeywords());
    holder.yearV.setText(String.valueOf(movies.get(position).getYear()));
    holder.countryV.setText(movies.get(position).getCountry());
    holder.costV.setText(String.valueOf(movies.get(position).getCost()));
}
```

-

The holder specifies the viewholder object created by the onCreateViewHolder(), the position refers to the index of the item from the source.

Here are the other methods in the custom MyRecyclerAdapter class:

```
private List<Movies> movies;
public void setData(List<Movies> inputMovies){this.movies = inputMovies;}

public class ViewHolder extends RecyclerView.ViewHolder {
    public TextView titleV;
    public TextView genreV;
    public TextView keywordsV;
    public TextView yearV;
    public TextView countryV;
    public TextView costV;

    public ViewHolder(@NonNull View itemView) {
        super(itemView);
        titleV = itemView.findViewById(R.id.card_title_ans);
        genreV = itemView.findViewById(R.id.card_genre_ans);
        keywordsV = itemView.findViewById(R.id.card_keywords_ans);
        yearV = itemView.findViewById(R.id.card_year_ans);
        countryV = itemView.findViewById(R.id.card_country_ans);
        costV = itemView.findViewById(R.id.card_cost_ans);

    }
}
```

setData() is set to pass the data to the adapter (the list/database which can be called the source).

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

    countryTextView = findViewById((R.id.countryFilterTV));
    yearTextView = findViewById((R.id.yearFilterTV));

    recyclerView = findViewById((R.id.movie_recycler));
    layoutManager = new LinearLayoutManager( context: this);  //A RecyclerView.LayoutManager impleme
    recyclerView.setLayoutManager(layoutManager);

    movieVM = new ViewModelProvider( owner: this).get(MovieViewModel.class);
    adapter = new MovieRecyclerAdapter();
    recyclerView.setAdapter(adapter);

    movieVM.getMovies().observe( owner: this,newData->{
        adapter.setData(newData);
        adapter.notifyDataSetChanged();
    });
}
```

This is how to pass data to the RecyclerAdapter, note that this is using a SQLITE Database, so the ViewModel will need to be observed, and only then can the data be sent to the adapter and notified.

Fragments: If we want to use an activity with two tasks that have two separate XMLs, Java Logic and can be reused in other activities, then a Fragment is the way to go.

They represent a behaviour or portion of the UI in a FragmentActivity. Multiple fragments can be combined in a single activity to create a multi-paned UI to be reused in multiple activities. They can be seen as modular sections of an activity, each with their own lifecycle, and which receives their own input events. (Like a subactivity that can be reused in different activities).

Fragments vs Activities:

- An activity is an application component that represents the full screen, while a fragment is a portion of the UI in an activity.
- An activity **may** contain 0 or more fragments.
- Fragments can be reused for multiple activities/
- Fragments cannot exist independently, and should always be a part of an activity.
- There can be multiple fragments occupying the screen at a time, but there can only be one activity occupying a screen at a time

The onAttach() method on the right is invoked when the fragment is associated with the activity (the Activity is passed here.

The onCreate() is called by the system when creating the fragment. Essential components of the fragment should be initialized that will want to be retained when the fragment is paused/stopped then resumed.

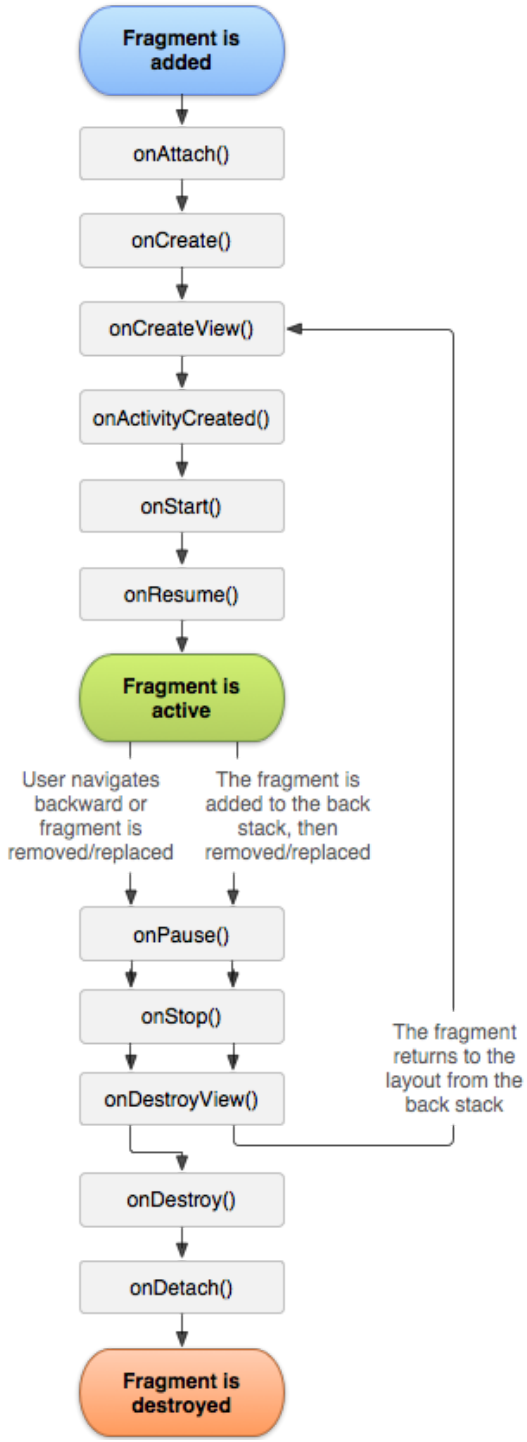
The onCreateView() is called by the system when it's time for the fragment to draw its UI for the first time. To draw the fragment's UI, the view must be returned that is the root of the fragment's layout.

onActivityCreated() is called when the activity's onCreate() method has been returned. And onStart() is called once the fragment is visible.

onPause() is called by the system if the user leaves the fragment (doesn't mean destroyed). And this is usually where changes that should persist beyond the current session must be committed.

onDestroyView() is invoked when the view hierarchy associated with the fragment is being removed.

onDetach() is invoked when the fragment is being disassociated from the activity



All About JSON

JSON: JavaScript Object Notation is a lightweight data-interchange format that allows humans to easily read and write data that is easy for machines to parse and generate. Data can also be nested in the JSON Format. It is incredibly simple serialised data, and 3 basic units can be inter-nested to any depth: (Object, Array, Literal values) JSON's basic types are: Number (int, real or float), String, boolean, Array, Object and null

Example:

```
{
   "name":"Tim",
   "age":23,
   "address":"Melbourne",
   "units":[
      "FIT1051",
      "FIT2095",
      "FIT2081"
   ]
}
```

JSON has the following syntax:

- Data is in key/value pairs.
- Data is separated by commas.
- Objects are held in curly braces.
- Square brackets hold arrays

Example:

```
{
   "firstname":"Tim",
   "lastname":"John",
   "websites":[
      {
         "description":"Company",
         "url":"http://company.com",
         "live":false
      },
      {
         "description":"School",
         "url":"http://school.com",
         "live":true
      }
   ]
```

```
}
```

In Java, Objects can be converted to their JSON representation. Here's how:

Add the dependency:

```
dependencies {
  implementation 'com.google.code.gson:gson:2.8.6'
}
```

Now create a new instance of Gson:

```
Gson gson = new Gson();
```

And let's say we have an array of objects:

```
ArrayList<Item> db = new ArrayList<>();
db.add(item1);
db.add(item2);
db.add(item3);
```

Now we convert the ArrayList into a String in JSON format

```
String dbStr = gson.toJson(db);
```

and we can also convert it back to Json.

```
Type type = new TypeToken<ArrayList<Item>>() {}.getType();
db = gson.fromJson(dbStr,type);
```

Workshop Quiz

1. Rewrite the class Week6Adapter such that it becomes a RecyclerView Adapter that shows the content of the array list 'names' in a recycler view using the card layout week6_name_card.xml. The adapter must bind the two attributes of the PersonName class into the two text views that are declared in the card layout.

   Week6Adapter.java class

   ```
   public class Week6Adapter {
       ArrayList<PersonName> names = new ArrayList();
       public Week6Adapter(ArrayList<PersonName> PersonName) {
           this.names = names;
       }
   }
   ```

   PersonName.java class

   ```
   public class PersonName {
       private String firstName;
       private String lastName;

       public PersonName(String firstName, String lastName) {
           this.firstName = firstName;
   ```

```
            this.lastName = lastName;
        }

        public String getFirstName() {
            return firstName;
        }

        public String getLastName() {
            return lastName;
        }
    }
```

week6_name_card.xml

```
<androidx.cardview.widget.CardView
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:id="@+id/name_card"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="5dp"
    card_view:cardBackgroundColor="@android:color/holo_blue_dark"
    card_view:cardCornerRadius="12dp">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="16dp">
        <TextView
            android:id="@+id/firstname_id"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentTop="true"
            android:layout_marginStart="58dp"
            android:layout_marginTop="8dp"
            android:layout_toEndOf="@+id/card_id"
            android:textSize="30sp" />

        <TextView
            android:id="@+id/lastname_id"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentStart="true"
            android:layout_alignParentTop="true"
```

```
            android:layout_marginStart="8dp"
            android:layout_marginTop="8dp"
            android:textSize="30sp" />
    </RelativeLayout>
</androidx.cardview.widget.CardView>
```

2. Briefly explain two differences between Fragments and Activities.
3. Briefly explain the data type and the role of param1, param2, and param3 in the following statement:

```
getSupportFragmentManager().beginTransaction().replace(param1,param2).addToBack
Stack(param3).commit();
```

1.
```
public class Week6Adapter {

    ArrayList<PersonName> names = new ArrayList();

    public Week6Adapter(ArrayList<PersonName> PersonName) {

        this.names = names;

    }


    @NonNull

    @Override

    public Week6Adapter.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
viewType){

      View v LayoutInflater.from(parent.getContext()).inflate(R.layout.name_card,
parent, false);

      return new ViewHolder(v);

      }


      @Override

      public void onBindViewHolder(@NonNull ViewHolder holder, int position) {

            holder.fName.setText(names.get(position).getFirstName());

            holder.lName.setText(names.get(position).getLastName());

      }

      @Override

      public int getItemCount() {

            return names.size();

      }


      public class ViewHolder extends RecyclerView.ViewHolder {

            public TextView fName;
```

```
        public TextView lName;
    public ViewHolder(@NonNull View itemView) {
        super(itemView);
        fName = itemView.findViewById(R.id.firstname_id );
        lName = itemView.findViewById(R.id.lastname_id );
    }
}
```

2. An activity may contain a fragment, but a fragment itself is a portion of the UI in an activity, so it does not contain an activity.
   An activity can exist independently without a fragment, while a fragment cannot exist independently as it is part of an activity.
3. Parameter 1: An integer that will contain the ID of the Container View which identifies the container whose fragment(s) do we want to replace.
   Parameter 2: A Fragment object, which will be the new fragment which will replace the existing fragment contained in the Container we identifies in parameter 1. This parameter should never be Null.
   Parameter 3: An optional String (that can be Nullable), that we can use to name the state of the back stack.

   Note: LayoutInflater Instantiates a layout XML file into its corresponding View objects.

Week 7

Database: An organized collection of structured information/data that is typically stored electronically in a computer system. It is usually controlled by a Database Management System (DBMS), and Android uses SQLite as it's DBMS.

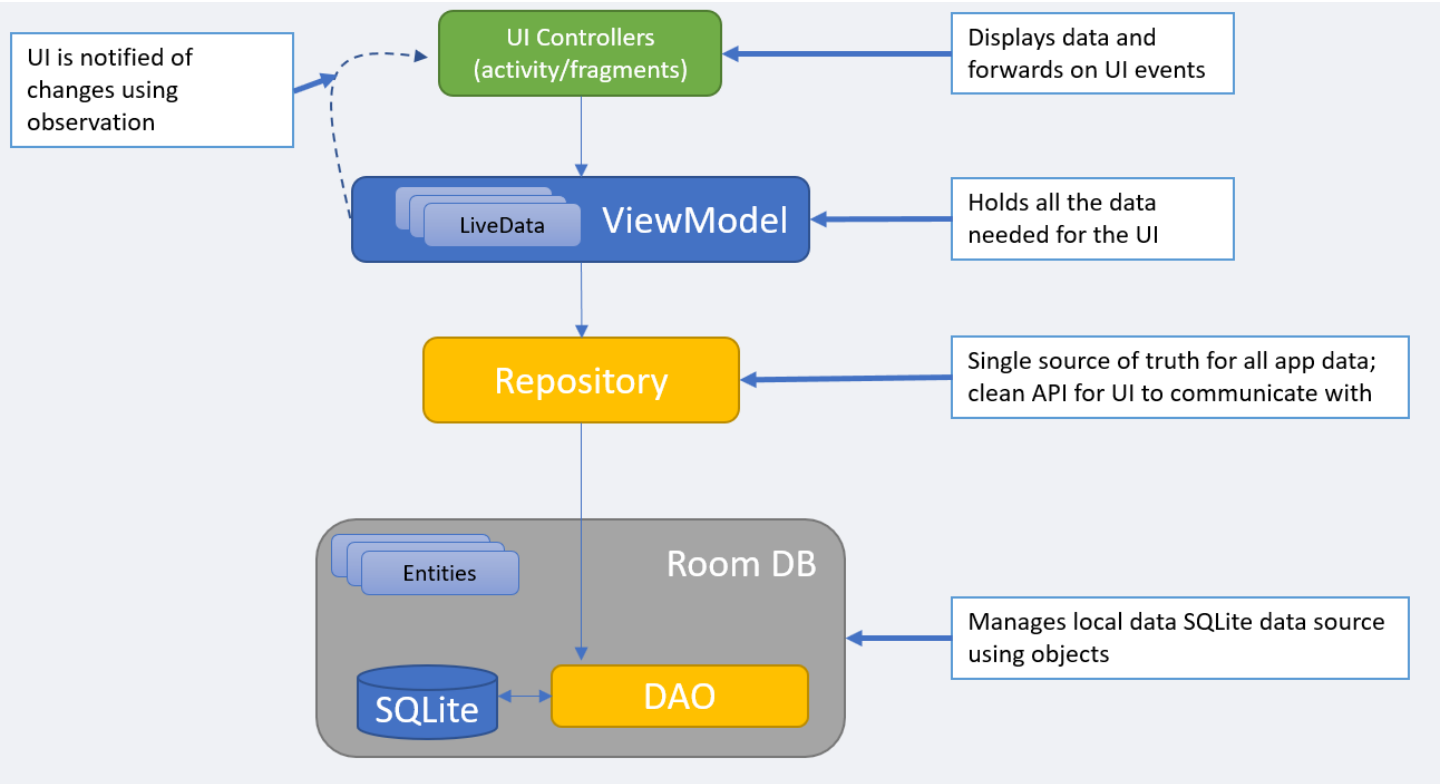
SQLite uses SQL (Structured Query Language) to communicate, and SQL is a language that allows access and manipulation of databases, they can be used to execute tasks such as adding/retrieving/updating data in the database. And the index that acts as the PK cannot be NULL.

- Tables consist of rows and columns
- Each column has a name
- Each row has the same set of columns
- Each cell stores one value only
- The data in a column have the same data type
- The table has a column that works as an index (CustomerID). This column is unique and cannot contain NULL values. This column is called the 'primary key'.

To get all customers who live in Germany, then simply:

```
SELECT * FROM Customers WHERE Country='Germany';
```

SQLite Database: Is a relational DB,S that is used by Android to store relational data.



DAO: Data Access Object is the main class in the form of an interfacewhere you define your database interactions. They usually include a variety of query methods.

Entities: Each entity represents one table in the database.

Room Database: The room database object provides the interface to the underlying SQLite database, and they're in the form of an abstract class.

Repository: The class that contains all the necessary code for directly handling all data sources used by the application. This avoids the need for the UI controller & the ViewModel to contain code that directly accesses sources such as the Database.

ViewModel: Provides the data for a specific UI component, such as a fragment or activity and contains data handling business logic to communicate with the repo. They're designed to store and manage UI-related data in a lifecycle conscious way. They allow data to survive configuration changes such as screen rotations. It's easier and more efficient to separate out view data ownership from the UI controller logic.

LiveData: A data holder that allows a value to become observable. An observable object can notify other objects when changes to its data occur. This solves the problem of ensuring the UI always matches the ViewModel's data.

How to create a Database

First create the entity, this acts as the table in the database.

```
package edu.monash.navdrawer2081student.provider;

import androidx.annotation.NonNull;

import androidx.room.ColumnInfo;

import androidx.room.Entity;

import androidx.room.PrimaryKey;


@Entity(tableName = "movies")

public class Movies {

   public static final String TABLE_NAME = "movies";


   @PrimaryKey(autoGenerate = true)

   @NonNull

   private int id;

   @ColumnInfo(name = "Title")

   private String title;

   @ColumnInfo(name = "Year")

   private Integer year;

   @ColumnInfo(name = "Country")

   private String country;

   @ColumnInfo(name = "Cost")

   private Integer cost;

   @ColumnInfo(name = "Genre")

   private String genre;

   @ColumnInfo(name = "Keywords")

   private String keywords;

   @ColumnInfo(name = "Bookmark")

   private String bookmark;
```

```
   public Movies(String title, Integer year, String country, Integer cost, String genre, String
keywords, String bookmark) {
       this.title = title;
       this.year = year;
       this.country = country;
       this.cost = cost;
       this.genre = genre;
       this.keywords = keywords;
       this.bookmark = bookmark;
   }

   public String getTitle() { return title; }
   public Integer getYear() { return year; }
   public String getCountry() { return country; }
   public Integer getCost() { return cost; }
   public String getGenre() { return genre; }
   public String getKeywords() { return keywords; }
   public String getBookmark() { return bookmark; }
   public int getId() { return this.id; }
   public void setId(int id){ this.id = id; }
}
```

The @Entity line is required to define the class as a room entity, which also specifies the table name.

The @PrimaryKey makes the ID as a primary key for the current table and @NonNull ensures that this column cannot be saved without a value.

Second, create the DAO, and define all the query functions you want to have/use.

```
package edu.monash.navdrawer2081student.provider;
import androidx.lifecycle.LiveData;
import androidx.room.Dao;
import androidx.room.Insert;
import androidx.room.Query;
import java.util.List;

@Dao
public interface MovieDao {
   @Query("SELECT * from movies")
   LiveData<List<Movies>> getAllMovies();

   @Insert
   void addMovie(Movies movies);
```

```
    @Query("DELETE from movies")
    void removeAllMovies();

    @Query("DELETE from movies where id=(SELECT MAX(id) from movies)")
    void removeLastMovie();

    @Query("DELETE from movies where bookmark=:bookmarks")
    void removeBookmark(String bookmarks);

    @Query("select * from movies where country=:country")
    LiveData<List<Movies>> getCountry(String country);

    @Query("select * from movies where year=:year")
    LiveData<List<Movies>> getYear(int year);

    @Query("select * from movies where year=:year and country=:country")
    LiveData<List<Movies>> getCountryYear(int year, String country);

    @Query("select * from movies where bookmark=:bookmarks")
    LiveData<List<Movies>> getFavorites(String bookmarks);
}
```

The @Dao line is required to consider the interface as a DAO.

The @Query annotation provides the select SQL statement that should be executed when the method below is invoked, it is important to specify the correct type for the method just below the Query as the build process will fail otherwise.

The LiveData method allows us to observe any changes to the database.

The @Insert annotation inserts the object that is passed through the method addMovie.

Third, the room Database must be defined, it may also contain one or more tables.

```
package edu.monash.navdrawer2081student.provider;
import android.content.Context;
import androidx.room.Database;
import androidx.room.Room;
import androidx.room.RoomDatabase;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
```

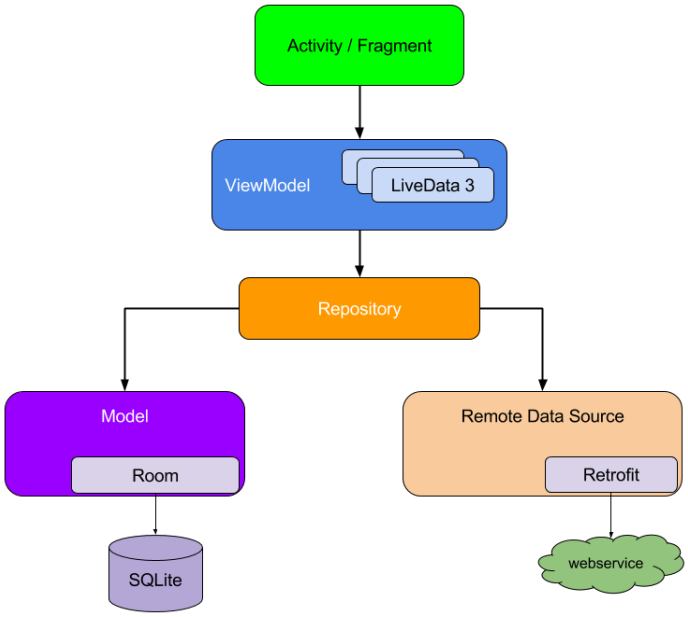
```
@Database(entities = {Movies.class}, version = 3)
public abstract class MovieDatabase extends RoomDatabase {
   public static final String MOVIE_DB_NAME = "movie_database";
   public abstract MovieDao movieDao();
   private static volatile MovieDatabase instance;
   static final ExecutorService dbWriter= Executors.newFixedThreadPool(4);

   public static MovieDatabase getMovieDB(Context context){
       if(instance == null){
           synchronized (MovieDatabase.class){
               if(instance == null){
                   instance = Room.databaseBuilder(context.getApplicationContext(),
MovieDatabase.class, MOVIE_DB_NAME).fallbackToDestructiveMigration().build();
               }
           }
       }
       return instance;
   }
}
```

The @Database line is required to consider the current class as a Room DB, and it specifies the list of entities and the current version. The version is required for upgrading/downgrading the current scheme.

The databaseWriteExecutor instance will be used by the repo to execute DAO methods.

The getDatabase method returns a reference to the current database instance if it isn't null, otherwise, it will make a new instance using Room.databaseBuilder() which needs as inputs the context, the Room Database class (itself), and the name for the database that has been declared above.

Fourth, the repository must be created. The repository itself is a Java class that provides a clean and easy API so that application can access multiple data sources.




```
package edu.monash.navdrawer2081student.provider;
import android.app.Application;
import androidx.lifecycle.LiveData;
import java.util.List;
```

```
public class MovieRepository {
   private MovieDao movieDao;
   private LiveData<List<Movies>> allMovies;
   MovieDatabase db;

   public MovieRepository(Application application) {
       db = MovieDatabase.getMovieDB(application);
       movieDao=db.movieDao();
       allMovies = db.movieDao().getAllMovies();
   }
   LiveData<List<Movies>> getAllMovies() {
       return allMovies;
   }
   void insertMovie(Movies movies){
       MovieDatabase.dbWriter.execute(()->{movieDao.addMovie(movies);});
   }
   void deleteAll(){
       MovieDatabase.dbWriter.execute(()->{movieDao.removeAllMovies();});
   }
   void deleteBookmarks(String bookmark){
       MovieDatabase.dbWriter.execute(()->{movieDao.removeBookmark(bookmark);});
   }
   void deleteLast(){
       MovieDatabase.dbWriter.execute(()->{movieDao.removeLastMovie();});
   }
   LiveData<List<Movies>> getMovieCountry(String country){
       return db.movieDao().getCountry(country);
   }
   LiveData<List<Movies>> getMovieYear(int year){
       return db.movieDao().getYear(year);
   }
   LiveData<List<Movies>> getMovieCountryYear(int year, String country){
       return db.movieDao().getCountryYear(year, country);
   }
   LiveData<List<Movies>> getFavorites(String bookmark){
       return db.movieDao().getFavorites(bookmark);
   }
}
```

The Repo will have the DAO and LiveData as class variables. And they will be initialised in the constructor. Methods that are related to data modification in the database will also employ databaseWriteExecutor to execute the SQL statement.

Fifth, is the initiation of the ViewModel. It is used when a fragment/activity needs special data or a different way to retrieve the data.

```
package edu.monash.navdrawer2081student.provider;
import android.app.Application;
import androidx.annotation.NonNull;
import androidx.lifecycle.AndroidViewModel;
import androidx.lifecycle.LiveData;
import java.util.List;

public class MovieViewModel extends AndroidViewModel{
   private MovieRepository repo;
   private LiveData<List<Movies>> movies;

   public MovieViewModel(@NonNull Application application) {
       super(application);
       repo = new MovieRepository(application);
       movies = repo.getAllMovies();
   }

   public LiveData<List<Movies>> getMovies() {
       return movies;
   }

   public void insertMovie(Movies movie){
       repo.insertMovie(movie);
   }

   public void removeAll(){
       repo.deleteAll();
   }
   public void removeLast(){
       repo.deleteLast();
   }
   public void removeBookmarks(String bookmark){
       repo.deleteBookmarks(bookmark);
   }

   public LiveData<List<Movies>> getCountry(String country){
       return repo.getMovieCountry(country);
   }

   public LiveData<List<Movies>> getYear(int year){
       return repo.getMovieYear(year);
```

```
    }

    public LiveData<List<Movies>> getMovieCountryYear(int year, String country){
        return repo.getMovieCountryYear(year,country);
    }

    public LiveData<List<Movies>> getFavorites(String bookmark){
        return repo.getFavorites(bookmark);
    }
}
```

In the onCreate() of the MainActivity, we can simply instantiate the movieViewModel by doing so:

```
movieViewModel = new ViewModelProvider(this).get(MovieViewModel.class);
```

And in order to insert a Movie object, we can simply:

```
Movies movie = new Movies(title.getText().toString(),
Integer.parseInt(year.getText().toString()), country.getText().toString(),
Integer.parseInt(cost.getText().toString()), genre.getText().toString(),
keywords.getText().toString(), bookmark.getText().toString());

movieViewModel.insertMovie(movie);
```

The following dependencies should also be added:

```
dependencies {
    def room_version = "2.4.2"

    implementation "androidx.room:room-runtime:$room_version"
    annotationProcessor "androidx.room:room-compiler:$room_version"
    // optional - RxJava2 support for Room
    implementation "androidx.room:room-rxjava2:$room_version"
    // optional - RxJava3 support for Room
    implementation "androidx.room:room-rxjava3:$room_version"
    // optional - Guava support for Room, including Optional and ListenableFuture
    implementation "androidx.room:room-guava:$room_version"
    // optional - Test helpers
    testImplementation "androidx.room:room-testing:$room_version"
    // optional - Paging 3 Integration
    implementation "androidx.room:room-paging:2.5.0-alpha01"
}
```

Workshop Quiz

1. Assume you have a table named ‘phones’ with the following attributes:
    - Id: int (primary key)
    - maker: varchar(20)
    - screenSize: int
    - price: float

    Write a piece of code that represents a method in a Dao interface that is part of a Room database implementation. If the method gets invoked, all phones with ‘screenSize’ less than 9 inches and prices greater than 1500 must be deleted. (Note: suggest a name for your method)

2. Briefly explain the roles of the following annotation:
    a. @Database
    b. @Entity
3. In the following piece of code, what is the role of 'LiveData'?

```
public LiveData<List<Customer>> getAllCustomers() {
return mAllCustomers;
}
```

4. Briefly explain the roles of Repository and ViewModel layers in a Room database.

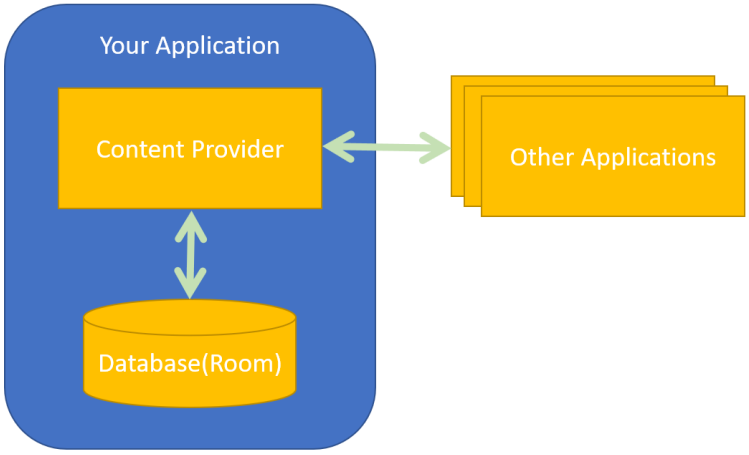
```
1. @Dao
public interface PhonesDao{
   @Query("DELETE from phones WHERE screenSize < 9 AND price > 1500")
   void removeHighRange();
}
```

2. **a**. The @Database annotation is required so Android Studio considers the current class as a room Database. as it also specifies the list of entities in the database as well as it’s current version. The list of entities will specify the number of tables this Room Database wants to hold. **b**. The annotation will specify the entity as a Room table, which tells Android Studio that the current class is a valid Room table that can be referred by other Database classes. The annotation also includes the line of the database.
3. The LiveData is a data holder class that is observable and is always up to date. Unlike a normal List data structure, we cannot just access the elements in the List directly, and instead we should use observe which will allow us to access the slice of data at a time period. In the piece of code, the LiveData means that the function will return the data type of LiveData, which will contain all the customers in the customer table in a live mode, which requires the LiveData to be observed before it can be accessed.
4. The duty of the repository is to contain all the logical aspect of the database operations, and it provides a clean and easy method to access up to multiple data sources. So the application can access multiple data sources, without handling the logical side by directly accessing the database. This allows the UI controller and the ViewModel to not need to directly interact with the database, and instead just invoke the methods of the repo.

    The ViewModel provides the data to the UI, and will act as the intermediary between the Repository class and the UI. It will also contain the data-handling business logic to communicate with the model. It is responsible for further simplifying the interactions between the UI and the Repository to make the objects more easily manageable and presentable.

Week 8

In order to export data from one application to another, we must implement the content provider class on top of the Room database. We do not need to implement the Content Provider class if we want the data to stay private and not be shared.



In order to access the content provider of an application, the access is provided by a URI (Universal Resource Identifier) defined by the Content Provider.

We can create one by right clicking the desired folder in Android Studio -> Other -> ContentProvider

URI Authority: A unique name that identifies the content provider. No two content providers with the same authority can exist on the same device.

After adding the Content Provider class to the project, provider details will automatically be added to the manifest.

```
<provider
   android:name=".provider.MovieContentProvider"
   android:authorities="fit2081.app.william"
   android:enabled="true"
   android:exported="true" />
```

And then in the ContentProvider application, the following methods need to be implemented:

- onCreate(): called by the provider during initialization.
- query(Uri, String[], Bundle, CancellationSignal): returns data to the caller
- insert(Uri, ContentValues): inserts new data into the content provider
- update(Uri, ContentValues, String, String[]): updates existing data in the content provider
- delete(Uri, String, String[]): deletes data from the content provider
- getType(Uri): returns the MIME type of data in the content provider

onCreate():

```
@Override
public boolean onCreate() {
```

```
    db = MovieDatabase.getMovieDB(getContext());

    return true;

}
```

query(): Used to query the database, and return data to the caller with the following parameters:

- Uri: maps to the table name
- projection: list of columns that should be included in each row
- selection: Is a string that represents the where
- Selection Arguments: an array of strings represents values that should be embedded in the selection statement.
- Sort order: A string that indicates whether to sort the data in ascending order

For example, lets have the following SQL statement:

```
SELECT title, year, country FROM movies WHERE country = 'Japan' AND year = 2016 ORDER
BY title DSC;
```

Argument	SQL Value
URI	"fit2081.app.william" or "fit2081.app.william/movie_database" or "fit2081.app.william/tv_show"
Projection	[“Title”, “Year”, “Country”] or [“title”, “year”, “country”]
Selection	“where country = ‘Japan’ and year = 2016” or “where country=? and year =?”
Selection Argument	null or [“Japan”, “2016”]
Sort Order	“title DSC” or null for default

```
@Override

public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String
sortOrder) {

    SQLiteQueryBuilder builder = new SQLiteQueryBuilder();

    builder.setTables(Movies.TABLE_NAME);

    String query = builder.buildQuery(projection, selection, null, null, sortOrder, null);

    final Cursor cursor = db.getOpenHelper().getReadableDatabase().query(query, selectionArgs);

    return cursor;

}
```

Cursor is used as a return type to the query method. It’s an interface that provides random read-write access to the result set returned by a database query. It’s a data structure (storage) that holds one or more rows retrieved from a database, and it contains methods to move the cursor to the next/previous/first/last row.

insert(): inserts a new row into the database and returns a new URI with the inserted row ID.

```
@Override
```

```
public Uri insert(Uri uri, ContentValues values) {
   long rowId = db.getOpenHelper().getWritableDatabase().insert(Movies.TABLE_NAME, 0, values);

   return ContentUris.withAppendedId(CONTENT_URI, rowId);
}
```

contentValues is a special data structure used to hold the data of one row only. It's used to send data to the database. The format used in contentValues is a key-value pair format. And the keys in the content values are the table's column names.

Yet while Content Values can hold the data of one row only, Cursor may hold data for multiple rows.

delete() is a method used to delete one or more rows from the database. The return value is the number of rows that are affected by the operation, and the method has similar arguments to query.

```
@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
   int deletionCount;
   deletionCount = db.getOpenHelper().getWritableDatabase().delete(Movies.TABLE_NAME, selection,
selectionArgs);

   return deletionCount;
}
```

update() also has similar arguments to the insert method, and it returns the number of rows affected by the operation.

```
@Override
public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs) {
   int updateCount;
   updateCount = db.getOpenHelper().getWritableDatabase().update(Movies.TABLE_NAME, 0,
values, selection, selectionArgs);

   return updateCount;
}
```

Notice that for queries it uses getReadableDatabase(), and for update/insert/delete it uses getWriteableDatabase(), well they are self explanatory.

getType(): It will tell you what type of data the content provider will provide you with.

To access the Content Provider from other applications, you won't need to implement the room database or content provider, instead we should use the Content Resolver. Content Resolver is used to access the content provider, and it contains a set of methods that mirror those of the content provider.

The code below gets the number of Movies in the database:

```
public void getMovieNumber (View view){
   Uri uri= Uri.parse("content://fit2081.app.william");
```

```
    Cursor result= getContentResolver().query(uri,null,null,null, null);
    movieNumber = findViewById(R.id.movieNumber);

    if (result == null) {
        movieNumber.setText("-");
    }
    else {
        movieNumber.setText(String.valueOf(result.getCount()));
    }
}
```

And to insert:

```
public void insertMovie (View view){
    Uri uri= Uri.parse("content://fit2081.app.william");
    ContentValues values = new ContentValues();
    values.put("Title","Shanghai Night");
    values.put("Year",1969);
    values.put("Country","China");
    values.put("Cost",10000);
    values.put("Genre", "Comedy");
    values.put("Keywords","action-comedy");
    Uri uri2= getContentResolver().insert(uri,values);
}
```

Workshop Quiz

1. Suppose you are developing an application that uses the 'Room' database to manage its local database. The app also exports its data using a 'Content Provider'. Briefly explain how the Content Provider class uses the Room database to insert a new record?.
2. Develop a piece of code that instantiates a URI matcher for an Android application with Content Provider Authority: monash.fit.movie and add roles for the following patterns:
    a. monash.fit.movies/movies
    b. monash.fit.movies/movies/129
    c. monash.fit.movies/movies/Harry
3. Develop a piece of code that inserts a new record in a database that is available in another application and can be accessed via the authority "fit2081.s12022.week8". The new record, which is detailed below, must be saved in a table named 'customers'.
    a. 'name':'Zena'
    b. 'age':34
    c. 'address':'Perth'

1. Content Provider is able to provide Resolvers with Data from multiple sources, and not just the SQLite Database. But since our application is using the Room Database, we are going to implement a Content Provider on top of

such database. This is done by first initializing the Database by getting the instance of the database in the constructor of the Content Provider. Then the other functions in the Content Provider will simply perform operations on the database instance that was created. Functions like insert, update and delete will invoke the getWritableDatabase from the database instance they created before, and will perform operations on the database (since they're writing data). While functions like query will use a Query builder and will invoke the getReadableDatabase from the database instance they created before. However, unlike in the activity/fragments, we do not use the ViewModel to perform CRUD operations on the database, and we will be accessing the database directly. That is because if we are using the ViewModel/DAO/Repository, we will need to make multiple custom functions and queries to match the arguments passed into the query/update/insert function, which will make it inefficient.

2.
```
private static final int movies = 1;
private static final int 129 = 2;
private static final int Harry = 3;

private static UriMatcher createUriMatcher() {

    final UriMatcher uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
    final String authority = CONTENT_AUTHORITY;

    //sUriMatcher will return code 1 if uri like authority/tasks
    uriMatcher.addURI(authority, "monash.fit.movies/movies", movies);
    uriMatcher.addURI(authority, "monash.fit.movies/movies/129", 129);
    uriMatcher.addURI(authority, "monash.fit.movies/movies/Harry", Harry);

    return uriMatcher;
}
```

3.
```
Uri uri= Uri.parse("fit2081.s12022.week8/customers");
ContentValues values = new ContentValues();
values.put("name", "Zena");
values.put("age", 34);
values.put("address", "Perth");
Uri uri2 = getContentResolver().insert(uri, values);
```

Week 9

Google Maps: A set of API classes and interfaces group together in the com.google.android.gms.maps package, and they're part of Google APIs and not Android's. Which means beyond a certain usage point, then it is required to pay for the extra features. The main class of this API is GoogleMap, and every methods related to the map uses the GoogleMap as the entry point.

Web Services are like function calls made using HTTP. They are a way of CRUDing remote data.

WebView is just a View widget that displays Web pages. It does not include any features of a fully developed web browser, like navigation controls or an address bar, and instead it's usually done only to show a single web page. A common scenario for using WebView is when we want to provide information in the app that will have routine updates (without the need for recompiling and resubmitting in the app store).

By doing some settings and doing some coding, a WebView can gain many of the features of a web page rendered by a browser, and can even enable JavaScript, page navigation/history, adding an address bar, and can offer very exciting native/mobile web app hybrid capabilities.

Executor: Enables proper and easy use of the UI thread, as it allows the usage of background operations and publish results on the UI thread.

MapsActivity: The UI layout is a single fragment element (id = map). The class called com.google.android.gms.maps.SupportMapFragment inflates the map image into the XML fragment and then supports all the functionality expected from a Google Map. Another component of the Google Maps API is an interface called OnMapReadyCallback. And by implementing this interface, we promise to listen and react to it's only event which signifies the map is ready for use (onMapReady). In the onMapReady event handler, we can initialize the map as required and set up listeners to make the map interactive.

MapsActivity extends AppCompatActivity for maximum backward compatibility, and it will also implement onMapReadyCallback.

onMapReadyCallback: A callback interface for when the map is ready to be used. Once an instance of this interface is set on a MapFragment or MapView object, the onMapReady method is triggered when the map is ready to be used and will provide a non-null instance of Google Map.

In the onCreate():

Geocoder: A class used for geocoding and reverse geocoding. Geocoding itself is the process of transforming an address or other location descriptions into a (latitude, longitude) coordinate. While reverse geocoding is the process of transforming a (latitude, longitude) coordinate into a partial address.

getSupportFragmentManager() is used by the app to acquire the support library's fragment class rather than the Android framework's fragment class. SupportMapFragment itself extends Fragment, and it is a Map component in an app. This fragment is the simplest way to place a map in an application. It's a wrapper around a view of a map to automatically handle the necessary life cycle needs, and this component can be added to an activity's XML layout file.

The SupportMapFragment's getMapAsync method will specify the object which will listen for an onMapReady event with it's first and only parameter.

The MapsActivity class therefore implements the OnMapReadyCallback which includes a single method header onMapReady in which a response to the map being ready can be handled. A reference to the map is passed into the onMapReady event handler so the map can be accessed and manipulated.

onMapReady(GoogleMap reference): The map is now ready for use and we have the reference to the map we just created to make sure the functionality of the GoogleMap can be accessed.

Some initialization of the GoogleMap is performed, like setting up a listener containing an event handler like onMapClick to handle clicks on the map, and the onMapClick is set up in the usual compressed syntax.

The onMapClick's event handler's input parameter is a LatLng object called "point", which will contain the latitude and longitude values of the click being registered on the map.

```
@Override
public void onMapReady(GoogleMap googleMap) {
   googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
   googleMap.getUiSettings().setMapToolbarEnabled(true);
   LatLng melbourne = new LatLng(-37.814, 144.96332);
   googleMap.moveCamera(CameraUpdateFactory.newLatLng(melbourne));

   googleMap.setOnMapClickListener(new GoogleMap.OnMapClickListener() {
       @Override
       public void onMapClick(LatLng point) {
           //save current location
           String msg;
           boolean actionFlag;
           String selectedCountry = "";

           List<Address> addresses = new ArrayList<>();
           try {
               addresses = geocoder.getFromLocation(point.latitude, point.longitude, 1);
           }
           catch (IOException e) {
               e.printStackTrace();
           }

           if (addresses.size() == 0) {
               msg = "No Country at this location!! Sorry";
               actionFlag = false;
           }
           else {
               android.location.Address address = addresses.get(0);
               selectedCountry = address.getCountryName();
               msg = "Do you want more details about " + address.getCountryName() + "?";
               actionFlag = true;
           }
           Snackbar.make(mapFragment.getView(), msg, Snackbar.LENGTH_LONG).setAction("Details",
(actionFlag) ? (new ActionOnClickListener(selectedCountry)) : null).show();
       }
   });
```

```
}
```

What the code above is doing:

```
addresses = geocoder.getFromLocation(point.latitude, point.longitude, 1);
```

Using the geocoder class' getFromLocation method to translate the given latitude and longitude into a partial address. The third parameter, means to return the first address that was obtained from using the getfromlocation method.

```
android.location.Address address = addresses.get(0);

selectedCountry = address.getCountryName();
```

The addresses.get(0) line is used to get the first (and only) address in the addresses list.

And from the address, we would get the country name using address.getCountryName().

```
Snackbar.make(mapFragment.getView(), msg, Snackbar.LENGTH_LONG).setAction("Details", (actionFlag) ?
(new ActionOnClickListener(selectedCountry)) : null).show();
```

This line basically takes country if it exist and provides a call to action if the user wants to view the country details or not.

The getFromLocation can generate IOException (and others), and IOException is a sub class of Exception but not of Runtime Exceitpion which makes it a Java checked exception. This means by syntax, it should be inclosed in a try/catch block or an error will be present.

Executors Class:

Retrieving data from the Web in the main UI thread can cause problems as we'll lock up the UI. So we should be performing those threads in the background by using the executor class.

In order to have multi threaded applications that runs some tasks in the background and update the UI, we must have at least two pointers/references, in which the first pointer references to the background thread and the second one points at the UI thread.

ExecutorService executor = Executors.newSingleThreadExecutor();

In the statement above, we will create a reference to a single threaded executor, the executor that is going to run our tasks in the background.

The statement below is the second pointer which is a reference to handler that uses the UI thread to update the UI elements and will be used to publish the results of the executor on the UI thread.

Handler uiHandler=new Handler(Looper.getMainLooper());

executor.execute(() -> {

//your task goes here

//Now to update the UI elements

uiHandler.post(()->{

}

});

Workshop Quiz

1. What is the role of the class GeoCoder in an Android application uses Google maps?
    a. What permission does WebView require to work and where to place it?
2. Develop a piece of code that is responsible for executing a method called 'getWeek9Pages()' asynchronously. The method 'getWeek9Pages()' returns a string value that should be displayed in a text view with id="week9PageID".
3. Briefly explain the role of the statement in line 2 in the following piece of code.
    a. What will happen if you delete it?

```
1        WebView webView=findViewById(R.id.week9_webview);
2        webView.setWebViewClient(new WebViewClient());
3        webView.loadUrl("https://en.wikipedia.org/wiki/Australia");
```

1. Geocoder is a class that is provided by the Android API, in which it has methods related to perform tasks related to Geocoding, and reverse Geocoding. Geocoding itself is the process of transforming an address into points/coordinates of Latitude and Longitude. While Reverse Geocoding is the process of transforming coordinates of Latitude and Longitude into a (partial) address

   In the context of Google Maps, it can be used for users to enter addresses and then display such address in the map, done in order to show users the rough estimate of the location, which means it will perform Geocoding. Users can also enter latitude and longitude points and reverse geocoding will be performed to show the user the point of the map as well as the location data of such points.

   b. It needs the permission to access the Internet. As to access the World Wide Web, we need an active internet connection.

   <uses-permission android:name="android.permission.INTERNET" />

2. ExecutorService executor = Executors.newSingleThreadExecutor();

   Handler uiHandler=new Handler(Looper.getMainLooper());

   TextView week9 = findViewById(R.id.week9PageID);

   executor.execute(() -> {

   String str = getWeek9Pages();

   uiHandler.post(()->{ week9.setText(str); }

   });

3. The role of line 2 is to set the WebView to have a WebViewClient, that's why a WebViewClient was initialised. The WebView Client itself is a class that will receive various notifications and requests of the WebView object. The WebViewClient is able to handle requests from the WebView to enable JavaScript, to route and other features.

   b. If we do delete the second line, then the URL we specified in loadUrl will still be opened. However the URL will be opened in our default browser, and not inside the WebView. This means we will exit the app, and the WebView we've created will be rendered useless since it won't be used.

Week 10

Gesture: A gesture is a sequence of touch events, and each touch event comes with x & y coordinates. And a gesture starts with the touch down event, and the system will track the finger position and will end with a touch up event.

Touch events can be intercepted by a view by overriding the onTouchEvent() or registering a custom onTouchListener() and the implementation of the onTouch() callback method.

In each of these methods, they may have a parameter called MotionEvent. MotionEvent is an object used to report movement events. It may hold absolute or even relative movements and their data.

Type of events:

- MotionEvent.ACTION_DOWN: This event is generated when the first touch on the view occurs
- MotionEvent.ACTION_UP: This event is generated when the touch is lifted from the screen.
- MotionEvent.ACTION_MOVE: Any motion of touch between the DOWN and UP events will be represented by the move.

onTouchEvent() can be overridden in the MainActivity, as demonstrated by the following:

```
public class MainActivity extends AppCompatActivity {
    private static final String DEBUG_TAG = "WEEK10_TAG";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        int action = event.getActionMasked();
        switch(action) {
            case (MotionEvent.ACTION_DOWN) :
                Log.d(DEBUG_TAG,"Action was DOWN");
                return true;
            case (MotionEvent.ACTION_MOVE) :
                Log.d(DEBUG_TAG,"Action was MOVE");
                return true;
            case (MotionEvent.ACTION_UP) :
                Log.d(DEBUG_TAG,"Action was UP");
                return true;
            default :
                return false;
        }
    }
}
```

This implementation will mean that whenever the screen is touched, then the LogCat will notify that an action appeared. This isn't specific to any view, and will be affected by any touch on the screen. The getactionmasked returns an int that signifies the type of the motionevent.

The implementation below allows users to listen to a specific view instead of the entire layout:

```
public class MainActivity extends AppCompatActivity {

    private static final String DEBUG_TAG = "WEEK10_TAG";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        View view=findViewById(R.id.my_layout);
        view.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View v, MotionEvent event) {
                int action = event.getActionMasked();
                switch(action) {
                    case (MotionEvent.ACTION_DOWN) :
                        Log.d(DEBUG_TAG,"Action was DOWN");
                        return true;
                    case (MotionEvent.ACTION_MOVE) :
                        Log.d(DEBUG_TAG,"Action was MOVE");
                        return true;
                    case (MotionEvent.ACTION_UP) :
                        Log.d(DEBUG_TAG,"Action was UP");
                        return true;
                    default :
                        return false;
                }
            }
        });
    }

}
```

With a compressed syntax, we registered a custom onTouchListener specific to the view.

Using MotionEvent.getX()/getY() will return the pixel coordinates relative to the specific view, while MotionEvent.getRawX()/getRawY() will return the pixel coordinates relative to the **whole screen**.

```
public class MainActivity extends AppCompatActivity {
    TextView actionType;
    TextView getXY;
    TextView getRawXY;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        View view=findViewById(R.id.frame_layout_id);
        actionType=findViewById(R.id.action_type);
        getXY=findViewById(R.id.get_x_y_id);
        getRawXY=findViewById(R.id.get_raw_x_y_id);
        view.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View v, MotionEvent event) {
                int x=(int)event.getX();
                int y=(int)event.getY();
                int rawX=(int)event.getRawX();
                int rawY=(int)event.getRawY();
                getXY.setText(x+","+y);
                getRawXY.setText(rawX+","+rawY);

                int action = event.getActionMasked();
                switch(action) {
                    case (MotionEvent.ACTION_DOWN) :
                        actionType.setText("Down");
                        return true;
                    case (MotionEvent.ACTION_MOVE) :
                        actionType.setText("MOVE");
                        return true;
                    case (MotionEvent.ACTION_UP) :
                        actionType.setText("UP");
                        return true;
                    default :
                        return false;
                }
            }
        });
    }
}
```

Workshop Quiz

1. Given the following excerpt of code, briefly explain when the method onTouchEvent() gets invoked. (5m)

```
@Override
 public boolean onTouchEvent(MotionEvent event) {
     Log.d(TAG,"onTouchEvent: "+event.getActionMasked());
     return true;
 }

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

     layout=findViewById(R.id.layout);
     outer=findViewById(R.id.view_out);
     inner=findViewById(R.id.view_inner);
     layout.setOnTouchListener(new View.OnTouchListener() {
         @Override
         public boolean onTouch(View view, MotionEvent motionEvent) {
             Log.d(TAG,"Layout: "+motionEvent.getActionMasked());
             return layoutFlag;
         }
     });
     outer.setOnTouchListener(new View.OnTouchListener() {
         @Override
         public boolean onTouch(View view, MotionEvent motionEvent) {
             Log.d(TAG,"Outer: "+motionEvent.getActionMasked());
             return outerFlag;
         }
     });
     inner.setOnTouchListener(new View.OnTouchListener() {
         @Override
         public boolean onTouch(View view, MotionEvent motionEvent) {
             Log.d(TAG,"Inner: "+motionEvent.getActionMasked());
             return innerFlag;
         }
     });
 }
```

2. Briefly explain the difference between getX() and getRawX() methods that are available in the MotionEvent class. (4m)

3. The following piece of code is designed to show a toast with each Action Up event. But, it shows nothing. Find and explain the problem, then fix it. (7m)

```
view.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        boolean flag=true;
        int action = event.getActionMasked();
        if(action == MotionEvent.ACTION_DOWN){
            //some code goes here
            flag=!flag;
        }
        if(action == MotionEvent.ACTION_UP){
            Toast.makeText(self,"Action UP",Toast.LENGTH_SHORT).show();
            flag=!flag;
        }
        return flag;
    }
});
```

1. As we can see, the layout, inner, and outer Views already have their own custom Listeners declared in onCreate, which means that when a touch is registered to any of those views, their own custom listeners with the onTouch will be invoked. This means that if we touch on any of these views, the onTouchEvent declared outside the onCreate won't be triggered, and their own respective custom Listeners will be invoked instead. Only if we touch the views outside the three declared in the onCreate, will the onTouchEvent be triggered. It will also get invoked if the 3 custom listeners return false.

2. getX() is a method use to get the X coordinate/pixel of the MotionEvent RELATIVE to the view it is invoked from. Meaning that if we created a small square view in the middle of the screen, and we touch the top left point of the small view, getX() will return 0, since it is relative to the view. While getRawX() is a method that gets the X coordinate/pixel of the MotionEvent RELATIVE to the entire screen, which means that we're going to receive the X coordinate with respect to the entire dimensions of the screen. Using the small view example, the getX() will return 0, while the getRawX() will probably return a value of 500 (maybe). Though it is possible for both getX() and getRawX() to be equal.

3.
```
view.setOnTouchListener(new View.OnTouchListener() {
@Override
public boolean onTouch(View v, MotionEvent event) {
      boolean flag=false;
      int action = event.getActionMasked();
      if(action == MotionEvent.ACTION_DOWN){
            Toast.makeText(self,"Action DOWN",Toast.LENGTH_SHORT).show();
```

```
                flag=!flag;
        }
        if(action == MotionEvent.ACTION_UP){
                Toast.makeText(self,"Action UP",Toast.LENGTH_SHORT).show();
                flag=!flag;
        }
        return flag;
}
});
```

The problem here is that the flag is first initialized as true, and the subsequent code inside the if statement is flag = !flag which would negate the value of flag from true to false. This means that the onTouch will then return false as a whole. Returning false for this onTouch would indicate to the parent view that the touch has not been consumed, and that this view is not interested in receiving subsequent motion events beyond the ACTION_DOWN. Since the view indicates that it's not interested in the subsequent motion events beyond ACTION_DOWN, then the subsequent MotionEvents won't be registered to that view.

Other fixes that can be done is placing a Toast to make text in the if ACTION_DOWN to indicate through a Toast that the ACTION_DOWN has occurred. And we can also set flag to return false after the event ACTION_UP is done to make sure that the events being consumed will stop at ACTION_UP.

Week 11

A multi touch gesture occurs when more than one pointer/finger touches the screen, and each finger within a gesture is kept track using the pointer's index and ID. The ID for a pointer is unique, will not change during the gesture's lifetime and is generated once the pointer touches the screen and joins the gesture. While the index for a pointer is relative and will change as fingers come and leave the screen. The MotionEvent object saves all the pointer's data in a special array and uses indices to access the pointer's entries. And these entries might shift up if pointers leave the screen.

Event	Description
ACTION_DOWN	The first pointer (finger) touches the screen. The motion event object contains the initial starting location.
ACTION_POINTER_UP	A non-primary (secondary) pointer leaves the screen
ACTION_POINTER_DOWN	A non-primary (secondary) pointer touches the screen
ACTION_MOVE	A motion happened to primary or non-primary pointer
ACTION_UP	The last pointer leaves the screen

Method	Description
getPointerCount()	The current number of pointers (fingers) on the screen
getPointerId(int pointerIndex)	get the pointer id associated with a particular pointer data index in the current gesture
findPointerIndex(int pointerId)	find the pointer index for the given id
getX(int pointerIndex)	find the x coordinate for the given pointer index
getY(int pointerIndex)	find the y coordinate for the given pointer index

Gesture Detector classes are used to detect common gestures through a set of motion events. And there are three steps to make them work:

- Create an instance of the Gesture Detector Class (your own implementation through extending/implementing)
- Implements the required methods
- Intercepts the touch events and pass them to the gesture detector.

These overridden callback methods like to return booleans, so what does it mean? Returning true for the callback method means that it informs the parent that the event **has been consumed** and is **ready to accept** further events following from the current gesture. It means if an onDown returns true, then whatever happens after that like onFling or onDoubleTap will be consumed. However, if it returns false, then the indicates that the event **is not consumed** and it is **not interested** in the remainder of the gesture, meaning if onDown returns false, then any subsequent events want be registered to that specific view.

Mainly, the two classes of detectors GestureDetector, and ScaleGestureDetector:

Class	Inerface	Methods	Description
GestureDetector	GestureDetector	onDown(MotionEvent e)	Notified when a tap occurs with the down MotionEvent that triggered it.
		onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY)	Notified of a fling event when it occurs with the initial on down MotionEvent and the matching up MotionEvent. e1 is the first event (touch down) e2 is the motion event that triggered the current event velocityX: The velocity of this event (fling event) along the X-axis velocityY: The velocity of this event (fling event) along the Y-axis
		onLongPress(MotionEvent e)	Notified when a long press occurs with the initial on down MotionEvent that trigged it.
		onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY)	Notified when a scroll occurs with the initial on down MotionEvent and the current move MotionEvent. e1 is the first event (touch down) e2 is the current event distanceX: the distance between e2 and the previous event (not e1) along the X-axis distanceX: the distance between e2 and the previous event (not e1) along the Y-axis

		onShowPress(MotionEvent e)	The user has performed a down MotionEvent and not performed a move or up yet.
		onSingleTapUp(MotionEvent e)	Notified when a tap occurs with the up MotionEvent that triggered it.
GestureDetector	OnDoubleTapListener	onDoubleTap(MotionEvent e)	Notified when a double-tap occurs.
		onDoubleTapEvent(MotionEvent e)	Notified when an event within a double-tap gesture occurs, including the down, move, and up events.
		onSingleTapConfirmed(MotionEvent e)	Notified when a single-tap occurs.
ScaleGestureDetector	OnScaleGestureListener	onScale(ScaleGestureDetector detector)	Responds to scaling events for a gesture in progress.
		onScaleBegin(ScaleGestureDetector detector)	Responds to the beginning of a scaling gesture.
		onScaleEnd(ScaleGestureDetector detector)	Responds to the end of a scale gesture.

onFling() vs onScroll():
onFling requires **some velocity** in the movement, like swiping to unlock the phone. They also have differences in parameter where in onScroll you will have the distanceX/distanceY and in onFling you will have velocityX/velocityY. onScroll() is invoked when moving your finger with **normal speed**, such as when scrolling a list or dragging and dropping. Also, onFling will be called **ONCE** at the end of the gesture, while onScroll will be called **MULTIPLE TIMES** as you move your finger on the screen.

If we want to really implement the GestureDetector's methods in the MainActivity, this is how to implement it:

```
private GestureDetectorCompat mDetector;

private ScaleGestureDetector mScaleDetector;

mDetector = new GestureDetectorCompat(this, this);

mScaleDetector = new ScaleGestureDetector(this, this);
```

And to implement the required methods:

```
public class MainActivity extends AppCompatActivity implements
GestureDetector.OnGestureListener, GestureDetector.OnDoubleTapListener {

}


   @Override
   public boolean onDown(MotionEvent e) { return false; }


   @Override
   public void onShowPress(MotionEvent e) { //Custom Implementation }


   @Override
   public boolean onSingleTapUp(MotionEvent e) { return false; }


   @Override
   public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float
distanceY) { return false; }


   @Override
   public void onLongPress(MotionEvent e) { //Custom Implementation }


   @Override
   public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY)
{ return false; }


   @Override
   public boolean onSingleTapConfirmed(MotionEvent e) { return false; }


   @Override
   public boolean onDoubleTap(MotionEvent e) { return false; }


   @Override
   public boolean onDoubleTapEvent(MotionEvent e) { return false; }


   @Override
```

```
    public boolean onScale(ScaleGestureDetector detector) { return false; }

    @Override
    public boolean onScaleBegin(ScaleGestureDetector detector) { return true; }

    @Override
    public void onScaleEnd(ScaleGestureDetector detector) { //Custom Implementation }
```

For gesture detectors to work, the onTouch callback must be overridden and the motionevent object MUST be forwarded to the detectors. Such as by implementing this line in the MainActivity:

```
@Override
public boolean onTouch(View v, MotionEvent event) {
        mDetector.onTouchEvent(event));
        mScaleDetector.onTouchEvent(event);
        return true;
    }
```

onScale:

```
    @Override
    public boolean onScale(ScaleGestureDetector detector) {
        tV.setText("onScale");
        return true;
    }

    @Override
    public boolean onScaleBegin(ScaleGestureDetector detector) {
        tV.setText("onScaleBegin");
        return true;
    }

    @Override
    public void onScaleEnd(ScaleGestureDetector detector) {
        tV.setText("onScaleEnd");
    }
```

If onScaleBegin() callback returns false, the two methods onScale() and onScaleEnd() will not be invoked, because it indicates to the parent that it is not interested in the current gesture.

It is much better to simply extend classes like SimpleOnGestureListener, that way we can pick and choose which functions we want to override instead of implementing every single method.

```
MyScaleListener MyScaleListener=new MyScaleListener();
mScaleDetector = new ScaleGestureDetector(this, MyScaleListener);

MyGestureListener myGestureListener = new MyGestureListener();
mDetector = new GestureDetectorCompat(this, myGestureListener);
mDetector.setOnDoubleTapListener(myGestureListener);
```

Where MyGestureListener is a custom GestureListener that extends from GestureDetector.SimpleOnGestureListener

Workshop Quiz

1. Briefly explain two differences between the pointer's ID and index in Android.
2. What will happen if the onTouch() callback returns false instead of true?
3. Update the following piece of code such that it displays a toast that shows the number of double-tap gestures that occurred on the layout 'main_layout' so far.

```
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        View layout=findViewById(R.id.main_layout);
    }
}
```

1. ID is used to track the pointer, while index is used to access a pointer's data from an eventmotion object.

   ID won't change during the pointer's lifespan since it will be uniquely issued to a pointer so it will remain the same unless the pointer is gone, while index for a pointer will change as pointers come and go.

2. It means that the the view/layout that implements such onTouch callback is not interested in any follow up motionevents or gesture beyond the one it receives in onTouch. If it is not interested, then the subsequent motion events/gesture will be thrown to views that are higher in its hierarchy (its parent), and it won't be sent any further motion events. Subsequent motion events can be double tap, action up, etc.

3.
```
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity implements View.OnTouchListener{
    private int doubleTaps = 0;
    private GestureDetectorCompat mDetector;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        View layout=findViewById(R.id.main_layout);
        layout.setOnTouchListener(this);
        MyGestureListener myGestureListener = new MyGestureListener();
        mDetector = new GestureDetectorCompat(this, myGestureListener);
    }
    @Override
    public boolean onTouch(View v, MotionEvent event){
      mDetector.onTouchEvent(event);
      return true;
    }
    private class MyGestureListener extends GestureDetector.SimpleOnGestureListener{
      @Override
      public boolean onDoubleTap(MotionEvent e){
            doubleTaps += 1;
            Toast.makeText(getApplicationContext(), "Double Taps: " +
String.valueOf(doubleTaps), Toast.LENGTH_SHORT).show();
            return false;
      }
      }
```

Week 12

Applications should be built into APKs (Android Package Kit) to be run, which is the package file format used by the Android OS for distribution and installation of apps. Source code files are compiled into virtual machine codes in the APK, and compilation units and resources are linked. And all APKs should be signed with a debug or release key.

Instant Run: In Android Studio 2.3, we can use Instant Run which significantly reduces the time it takes to update the app with changes to the code/resource. If the target device runs a minimum of Android 5.0 (API 21), Apply Changes can be applied to push certain code and resource changes to the running app without building a new APK, sometimes without even restarting the current activity.

Cold Swap: Changes to the structural code, such as the addition of a new method or a change to the signature of existing methods will require the app to be restarted.

Hot Swap: Occurs when the code within an existing method is changed. The new method implementation is used the next time it is called by the app. It occurs instantaneously and if configured correctly, it will be accompanied with a toast.

Warm Swapping: When changes are made to the resource file of the project (layout change or string/color modification). A warm swap involves restarting the current running activity, and the screen will flicker as the activity restarts. A toast will be reported in the activity to indicate success.

The normal Run/Debug buttons are always available when changes made should force an app restart, however using the apply changes button provides a faster workflow for changes that are incremental.

To enable Instant Run: Settings -> Build, Execution, Deployment -> Instant Run

However, this is device specific and is not totally robust at this time.

Gradle: An advanced build toolkit to automate, and manage the build process. It compiles app resources, source code and packages them into an APK that can be tested/deployed/signed/distributed, and allows us to define custom build configurations.

Each build configuration can define its own set of code and resources, while reusing parts common to all app versions.

Build Types: It define certain properties that Gradle uses when building and packaging your app. They're usually configured for different stages of the development process. For example: The debug build type enables debug options and has the APK signed wit ha debug key, while the release version would've had polished features (not experimental), compressed and are signed with the release key. At least one build type must be defined (Android Studio automatically has debug & release types built)

Examples: Debug/Release

Product Flavors: Represents different versions of the app that may be released, like the free/paid/demo versions of the app. You can customize product flavors to use different code & resources, while sharing/reusing common parts across all versions. They are optional and they must be manually created.

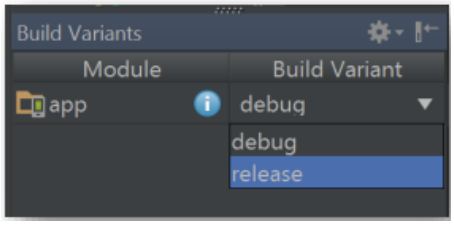
Example: Demo/Lite/Full

Build variants: The number of build variants is the (Number of Product Flavors * Build Types). They're a cross product of build type and product flavor, and is the configuration Gradle uses to build the app. Although build variants cannot be configured directly, the build types and product flavors can be configured.

Example: Demo/Debug, Demo/Release, Full/Debug, Full/Release

Default Run/Debug Configurations: When a project is first created, Android Studio will make a default run/debug configuration for the main activity based on the Android App template. To run/debug the project, one run/debug configuration should be defined, which is why it's not recommended to delete the default configuration.

Default Build Types: Build types can be created and configured in the build.gradle file (inside the Android block). When a new module is created, Android Studio automatically creates the debug and release build types for you. Although the debug build types doesn't appear in the build configuration file, Android Studio sets it to "Debuggable: True". This allows the debugging of the app on secure android devices and configures the APK signing with a debug key.



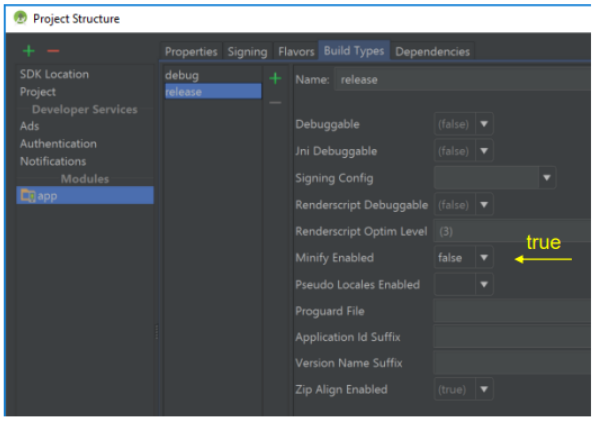
How to switch Build Variants: View -> Tool Windows -> Build Variants

Both Run & Debug Toolbar Buttons produces a debug build type if that is the currently selected build variant, and it is the default type in any new project. Clicking the Run toolbar button will build and launch the debug build variant which will NOT engage with the debugger, while clicking the Debug toolbar button will build and launch a debug build variant which WILL engage with the debugger.

All Apps will need to be digitally signed to successfully run on a device/emulator. If a debug version is being built, then the Android Studio will automatically sign the app with a debug key.

ProGuard: Used optionally when building the APK process (which is the release variant). They're used to perform several optimizations, verification, and obfuscation tasks, and are able to improve the efficiency and reduce the size of an application. They're able to shrink the code by detecting and safely removing unused classes, fields, methods, and attributes from the app as well as its dependencies. Resources that aren't used are also removed, including resources from code that aren't used.

- Enabling ProGuard (Minify Enabled = true)
  - File → Project Structure → app → release → Build Types → MinifyEnabled
  - There is more. It can get complicated! Also see Gradle module level file.



Before the App is released by Android Studio to the App Store, a private/public key pair must be generated to sign the app by you. Key pairs are stored in a key store file.

To sign the app:

- Build Generate Signed APK.
- "Generate Signed APK" wizard displays
- The wizard includes
  - Creating a new keystore or accessing an existing keystore

- Generating a new key pair or accessing an existing key pair
- Generating the signed APK ready for upload to a store

Key Pair: A public-key certificate, it's also known as a digital/identity certificate which contains the public key of a public/private key pair. The owner of the certificate holds the corresponding private key.

Keystore: A binary file that contains one or more private keys. When you sign an APK for release through Android Studio, you can choose to generate a new keystore and private key or use existing ones. It's also preferable to choose a strong password for your keystore, and they must be kept in safe/secure place.

The default debug keystore and certificate is created and stored in: $HOME/.android/debug.keystore, and the self-signed key used to sign the debugging APK has an expiration duration of 365 days. Amd when the key expires, a build error will occur. This can be fixed by simply deleting the debug.keystore. The next time the debug build type is built and run, the build tools will regenerate a new keystore, and debug key. The app must be run as building alone won't regenerate the keystore and debug key.

Why you should look after your private key

- If someone uses your private key, your authoring identity and the trust of the users are compromised
- If a third party takes your key without your knowledge/permission, such party could sign and distribute apps that maliciously replace your authentic apps or corrupt them. They can modify the app using your identity to scam users, steal identity or to destroy user's systems.
- Losing your private key would mean that you won't be able to sign all future version of your app. If your key is stolen/misplaced, you will not be able to publish new updates to your existing app as you cannot regenerate a previously generated key. This effectively means the app will not receive any more support/bug fixes/feature updates in the Google Play since no one is able to access the key to provide new updates.

Signing:

When the contents of the app's APK are signed, the contents of the app's APK are hashed (by being fed to cryptographic hashed function). The hash is encrypted using the developer's private key. The signed hash + the id of the hashing function + public part of the signing key are added to the APK as a signing block

Verifying:

- Extract the signing block (defined earlier)
- Contents of the APK are hashed to create a calculated hash
- The sent signed hash is decrypted using the sent public key
- And the calculated, sent hash are compared and if equal, verify:
    - If the owner of the public key sent the APK
    - The integrity of the APK is confirmed (as any change would cause a difference in the hash)

Cryptographic Hash Function: A special class of a hash function that has certain properties which makes it suitable for use in cryptography.

Properties:

- Deterministic, so the same message will always result in the same hash
- Quick to compute the hash value for any given message
- It is infeasible to generate a message from it's hash value except by trying all possible combinations/messages.

- A small change to the message will change the hash value, to the extent where the new hash value appears uncorellated with the old hash value
- It is impossible to find two different messages with the same hash value.

Workshop Quiz

1. You are developing an Android application that represents a calculator. You are planning to release three versions: free, paid, and demo. Your application lifecycle consists of two tiers (stages): development and production. List all the build variants of your application.
2. What are the consequences if the private key of an app is lost but not compromised (i.e. no one else acquires it)?
3. Briefly explain the role of ProGuard in Android.

1. Build variants = Build types * Product flavors

   2 * 3 = 6

   Free/Development

   Free/Production

   Paid/Development

   Paid/Production

   Demo/Development

   Demo/Production

2. When a private key is lost, it means we cannot sign future updates of the app for release. And this will mean that we're unable to publish new updates that may be required by the application (security, features, bugs). And although no one else possesses the private key, this will mean the application won't be able to receive any future updates in the Google Play Store, as we won't be able to regenerate a previously generated key.

3. ProGuard is an optimization tool available in Android Studio, it's able to compress/shrink the size of an application and make it more space efficient. It is done by detecting and removing parts of the code (like functions, classes and fields) that are never used, as well as unused dependencies. Resource files that are never referenced by the application code will also be removed.

Study tips:

- Please review ALL of the WS quizzes. They're the bread and butter of the final exam.
- Review the lecture slides again, and try to get a feel of what each week is all about.
- Review the code you did for the labs again, they will refresh your memory on how to do the topics for each week.
- Try to memorize the syntax for code questions from the workshop quizzes. There's going to be questions regarding the syntax for some functionalities.

Extra Resources:

https://quizlet.com/au/410119668/fit2081-flash-cards/

https://quizlet.com/212154865/fit2081-week-3-flash-cards/