Week 1

Terminology:

- Software Development Kit (SDK)
    - Software Components
    - Develop & Deploy on given development platform
- Class Library (API)
    - Application Programming Interface
    - Call & execute to perform common yet complicated tasks
- Integrated Development Environment (IDE)
    - Software environment
    - Contain tools required to develop application

Native Application

- Compile code run directly on devices' platform
- Built using SDK tools & languages

Mobile Web Application

- Website
- Built using website development technology
- Designed for smart device display
- Accessed by device's browser
- Dynamic ongoing development
- Eg. Debug clients' code on clients' device using Chrome remote debugging tools

Hybrid Application

- Use language and development environment other than the recommended languages for the platform
- Deployed as a Native App
- Usually have 3rd party support
- Main types:
    - Thin native app shell containing Web App
    - Cross compiler used to convert code into Native App executable for required target platform

Comparison

Native	Mobile Web	Hybrid
Scarce, technically sophisticated, costly developers	Abundant, less sophisticated, cheaper web developers	Abundant, less sophisticated, cheap developers
-	Write Once	Write Once
Complete access to the device's features	Access to device features depends on the quality, completeness and cross browser consistency of the browser's Native API bridge	Access to device features depends on the quality, completeness and cross browser consistency of the 3rd party browser's Native API bridge
Faster speed	Slower speed	Slower speed
No security risks of web apps	Security risks present	No security risks of web apps
Lengthy approval process, fees and content restriction	-	Lengthy approval process, fees and content restriction
Delayed &/ consistent update / maintenance	Instant update / maintenance	Delayed &/ consistent update / maintenance

Native = MobileWeb + Native

Week 2

Android:

- Operating System (OS)
  - Collection of software
  - Manage computer hardware resources
  - Provide common service for computer programs
- Lightweight
- Open Source
  - Free, source code available for modification and redistribution
- Designed for touchscreen mobile devices
- Architecture
  - Linux Kernel + Software Stack
  - Linux Kernel
    - Multitasking execution environment
    - Allow multiple process execution
    - Advantage:
      - True of a single VM

Google:

- Purchased from Android Inc.
- Open Source
- Create & release new versions of Android
  - Eg. update software stack, Linux Kernel
- Maintain largest Android App Store (Google Play)

Android Ecosystem

- Interdependent, evolving components that enabled the creation and distribution of Android Application
  - Components:
    - Hardware manufacturers
    - IDEs creators
    - Distribution channels
    - App developers

SDK Platform

- Entire SDK
- Include version of API

API Level

- Integer value
- Identify unique framework API revision offered by a version of the Android platform
- Framework API
  - Application use to interact with underlying Android system
  - Consist:
    - Packages and Classes core set
    - Set of XML elements and attributes for declaring a manifest file
    - Set of XML elements and attributes for declaring and accessing
    - Set of Intents
    - Set of permissions that applications can request and permission enforcements included in the system

Fragmentation

- API divided or fragmented into multiple versions, variations, or implementations
- Occurs when different versions or variations of an API are released or maintained separately, leading to a fragmented ecosystem of APIs
- Porting Android to specific hardware = time- and resource-consuming process for device manufacturers
- Prioritise their newest devices, leave older ones behind
  - Older smartphones are frequently not updated

if the manufacturer decides it is not worth their time, regardless of whether the phone is capable of running the update
- Additional delays can be introduced by wireless carriers

Forward Compatibility
- Old apps running on new platform versions
- Essential because of OTA (over the air) platform updates

Backward Compatibility
- New apps (using new features?) running on old platform versions

Activities
- Single, standalone module of application functionality
- Correlates directly to a single UI screen and its corresponding functionality
- Lifecycles → Partially or Fully hidden or killed by OS

Services
- Processes that run in the background
- No UI
- Started / Managed by Activities, Broadcast Receivers or other Services
- Ideal Situation:
    - Application needs to continue performing tasks but does not necessarily need a user interface to be visible to the user

Content Providers
- Data sharing between applications
- App1 provide App2 access to App1's underlying data through content providers
- Access to the data is provided via a Universal Resource Identifier (URI) defined by the Content Provider
- Data can be shared in the form of a file or an entire SQLite database
- The Content Providers currently available on an Android system may be located using a Content Resolver

Content Resolver
- Single, global instance in your application that provides access to your content providers
- Accepts requests from clients, and resolves these requests by directing them to the content provider with a distinct authority
- Content Resolver stores a mapping from authorities to Content Providers
- The Content Resolver does not know the implementation of the Content Providers it is interacting
- CRUD methods
    - Create
    - Read
    - Update
    - Delete

Content Providers & Receivers
- Whereas the Content Resolver provides an abstraction from the application's Content Providers, Content Providers provide an abstraction from the underlying data source (i.e. a SQLite database)

Broadcast Receivers
- Application respond to Broadcast Intent
    - System-wide broadcast announcements
- Registered by an application and configured with an Intent Filter to indicate the types of

broadcast in which it is interested
- Broadcast Receivers operate in the background and do not have a user interface

Intents
- Created with an Intent object
    - Explicit or Implicit
- Activities, services and broadcast receivers activated by intent
- Bind individual components to each other at runtime
- Explicit Intents: Request the launch of a specific activity by referencing the activity by class name
- Implicit Intents: Stating either the type of action to be performed or providing data of a specific type on which the action is to be performed

Activity Intent
- Activity1 is able to launch another [activity2 or service] and implement the flow through the activities that make up an application

Broadcast Intent
- System-wide intent that is sent out to all applications that have registered an “interested” Broadcast Receiver
- Eg. Indicate change in device status
    - Power on / off
    - Screen on / off
    - Charging or not

Inter App Activation
- Unique
- Any app can start another app’s component = starts the process for that another app
- Deliver a message to the Android system that specifies your intent to start a particular component. The system then activates the component for you
    - App1 CANNOT DIRECTLY activate App2 component
    - Android system can DIRECTLY activate App2 component

Content Providers
- NOT activated by Intents
- Activated when targeted by a request from a Content Resolver
- Content resolver handles all direct transactions with the content provider
    - The component that's performing transactions with the provider:
        - NO direct transactions
        - CALLS methods on the Content Resolver object

Manifest
- XML file
    - App components details
        - Eg. Capabilities
- Declaration of all components in the application
- Primary Task:
    - Inform the system about the app's components
- Other tasks:
    - Identifies any user permissions the app requires
    - Declares the minimum API Level required by the app
    - Declares hardware and software features used or required by the app
    - Declares API libraries the app needs to be linked against

Resources

- Android application package [APK] will also typically contain a collection of resource files
- Strings, images, fonts and colours that appear in the user interface together with the XML representation of the user interface layouts
- Easy to update various characteristics of your app without modifying code
- Optimise your app for a variety of device configurations
    - Provide alternative resources for different device configurations
    - Eg.
        - Defining UI strings in XML = translate the strings into other languages and save those strings in separate files
        - Based on a language qualifier that you append to the resource directory's name and the user's language setting, the Android system applies the appropriate language strings to your UI

Context

- Interface to global information about an application environment
- Implementation is provided by the Android system
- Access to application-specific resources, classes and up-calls for application-level operations
    - Eg. Launching activities

Week 3

Mobile OS:

- Android System Key responsibility:
    - Manage limited resources effectively so that both the operating system and the applications running on it remain responsive to the user at all times
- Android is given full control over the lifecycle and state of both the processes in which the applications run
- Each running Android application is viewed by the OS as a separate process
- If the OS identifies resources on the device are reaching capacity = OS will terminate processes to free up memory
    - Consider priority and state of all currently running processes
    - Importance hierarchy
    - Processes host applications and applications made of components
        - Rank processes by their highest ranking active component
        - Lowest ranking terminated/killed first
    - Activities ranked by lifecycle states
        - UI current interacting activity > Partially hidden activity > Fully hidden activity

Activity Lifecycles

- 4 possible Activity Lifecycle states:
    - Foreground
    - Partially hidden
    - Fully hidden
    - Destroyed

- "Back" destroys an Activity
- State transition due to User interaction and/or OS termination
    - When a process is killed all its app's components are destroyed in including its Activities

Stacks
- Activities arranged in back stack in the order where each activity is opened
- New activity opened is on top of stack (more focus)
- Previous activity remains in the stack, but is stopped. When an activity stops, the system retains the current UI state
- "Back" button → destroyed activity (pop from stack), previous activity resume
- When all activities are removed from the stack, the task no longer exists

Complications
- 1 back stack task can include Activities from different apps
- finish() same as back button
- Sleep phone = blank activity

Activity Lifecycle Callbacks
- onCreate(), onStart()
- onResume(), onPause()
- onStop(),onDestroy()

Activity Lifecycle Loops
- Partially hidden
- Fully hidden
- Process Destroyed

Lifecycle Callbacks
- Within the lifecycle callback methods, you can declare how your activity behaves when the user leaves and re-enters the activity
- Call super in all Lifecycle callbacks
    - Lifecycle callback overrides their super call

More Activity Callbacks
- onSaveInstanceState(…)
- onRestoreInstanceState(…)
- Appropriate transitions between Activity states
- Saving the state of an Activity

Activity Instance State
- View / Non-view
- Activity Instance Data:
    - View:
        - State of all Views in the Activity's layout
    - Non-View:
        - Activity instance variables
- Persistent data
    - Persistent across app uses implies across app destroys
    - Associated with multiple uses of the Activity separated by any amount of time
    - This type of data can be made App private but not Activity private

Activity Data & Persistence Across Events
- Type of events:
    - Partially / Fully Hidden Activity
        - Activity is still in memory, no state data (view or non-view) is lost
    - Activity destroyed by "back" button
        - Activity code containing the Activity's finish() method to execute
        - What Android system thinks: Activity's

view-state is no longer required
        - Persistent data remains in its last updated state in non-volatile memory
    - Device configuration
        - Eg. Rotating Device
            - Android OS destroy then relaunch Activity, no killing
    - App is uninstalled from the device
        - All of an app's data including any persistent data stored on the device's nonvolatile memory is lost

Saving/Restoring Activity Instance State

- onSaveInstanceState (Bundle outState)
    - Not called if user indicates they are done with the Activity
    - Must call super to let Android save view-state
    - Must code to manually save non-view state in Bundle
- onCreate (Bundle savedInstanceState)
    - Must code to manually restore previously saved non-view state from Bundle
    - Must call super to let Android save view-state
- onRestoreInstanceState (Bundle savedInstanceState)
    - Only called if the Bundle is non-null
    - Must call super to let Android restore view-state it previously saved
    - Must code to manually restore previously saved non-view state from Bundle

Saving/Restoring Persistent Activity Data

- Subclass instance of Activity
    - Execute its parent's lifecycle callbacks at the appropriate times if you don't override them
    - If override, manually call their super if aim to execute in addition to their override's code
- Auto-saving/restoring of the view-state of an Activity is done in Activity's onCreate(...), onSaveInstanceState(...) and onRestoreInstanceState(...)
    - Parameter: Shared Bundle object Android use to save/restore an Activity's view-state
        - Manually save and restore an Activity's non-view state
- To auto-save/restore their state, View must have:
    - Unique ID
        - Tag their saved state in the bundle
    - saveEnabled property set to true
- Android will auto-save the view-state of an Activity
    - If destroyed, expect state to be maintained
        - Unless OS kill for resource
    - If these methods are overridden in an Activity subclass and these overrides do not call their super Android's auto-save/restore of view-state will not occur

- Android will NOT save the view-state of an Activity
    - If user indicate they're done with the activity = aim to destroy activity
        - Eg. "back" key
        - OS don't call onSaveInstanceState

Saving/Restoring Persistent Activity Data Options:

- 3 main options:
    - Saving Key-Value Sets
        - Shared preferences file
    - Saving Files
        - Basic file
    - Saving Data in SQL Databases
        - SQLite database, read & write structured data
- Involves an editor and a commit/apply action
- Restore current persistent values from a SharedPreferences file to instance variables in onCreate or if nothing to restore set default values
- Update these instance variables holding current persistent values as required during Activity operation
- Save the values of these instance variables holding current persistent values to their SharedPreferences file in onPause

Tracing Lifecycles:

- Use of Log.x to send trace statement to Logcat

Week 4 Part I

Views

- Hierarchy of View objects
- View:
    - android.view.View
    - Basic building block for UI components
    - Responsible for drawing and event handling
    - Widget Base class
    - Create interactive UI components
        - Eg. buttons
    - Layouts: Invisible containers holding other Views, ViewGroups and define layout properties

ViewGroups

- android.view.ViewGroup
- Subclass
- Base class for layouts & view containers
- Defines ViewGroup.LayoutParams class
    - Layout parameter base class
    - Allow nested views
        - Hierarchy representation

LayoutParams

- Used by views to indicate how to lay out views
- Base LayoutParams class" Describe view size (width & height)
- There are subclasses of LayoutParams for different subclasses of ViewGroup
    - Eg. AbsoluteLayout has its own subclass of LayoutParams which adds an X and Y value

Creating UIs

- Layout defines the visual structure for UI
- Declare layouts (2-ways):
    - Declare UI elements in XML: Straightforward XML vocabulary that

corresponds to the View classes and subclasses
- Instantiate layout elements at runtime: Create and manipulate View and ViewGroup objects programmatically

Java and XML
- Choosing between Java and XML for declaring UI elements depends on the specific requirements and preferences of the project
- Java:
  - Allows for dynamic instantiation of View objects at runtime
  - Provides more flexibility in terms of programmatic control and behaviour of the UI
  - Offers direct access to the methods and properties of UI elements
  - Allows for more complex logic and customization
- XML:
  - Enables better separation of UI presentation from application code
  - Allows for easier modification and adaptation of UI without modifying source code and recompiling
  - Supports creating different layouts for various screen orientations, screen sizes, and languages
  - Provides a visual structure of the UI, making it easier to debug issues

View Properties
- Belong to the View object's Class

View LayoutParams
- Belong to the Class of the Layout a View is contained by

Layouts Types
- View Group
  - Several Layout direct subclasses
    - Eg. Coordinator Layout, Frame Layout, Grid Layout, Linear Layout, Relative Layout
  - Several Layout indirect subclasses
    - Eg. Table Layout
  - Layout type to use depends on UI Requirement
  - ViewGroup is a View a ViewGroup can be contained by another ViewGroup
- View Containers
  - Many direct and indirect View container subclasses that can be part of UI's View hierarchy
    - Eg. Toolbar

ConstraintLayout
- Direct subclass found in the support library
- Compatible with Android 2.3 (API level 9) and higher
- ConstraintLayout contains a nested class ConstraintLayout.LayoutParams descended fromViewGroup.LayoutParams
  - Contains mostly instance variables whose values allow a View to specify its size, position and behaviour

Styles, Themes, Material Design

- Style:
  - Collection of attributes that specify the look and format for a View or window
  - Specify attributes
    - Eg. Font size
  - Separate design from content
- Themes:
  - Style applied to an entire Activity or app rather than just individual View
  - When a style is applied as a theme, every view in the activity or app applies each style attribute that it supports
  - Define a set of attributes in one place but these attributes are automatically applied throughout the app
- Material Design:
  - Comprehensive guide for visual, motion and interaction design across platforms and devices
  - Components and functionality available in Android 5.0 [Lollipop] (API level 21) and above
  - Android provides the following elements for you to build material design apps:
    - New theme
    - New widgets for complex views
    - New APIs for custom shadows and animations
  - Compatibility complications
    - Material themes only work with Activities that are subclass of Activity which of course has limited backward compatibility

Action Bar / AppBar

- App bar/Action bar is a special kind of toolbar
- Used for branding, navigation, search and actions
- App Bar = Action Bar
- UI design concept
- NOT UI component
  - But have toolbar
  - Have ActionBar class
    - Address App Bar/Action Bar for an Activity's UI
- Pre-Lollipop:
  - Part of the default theme
  - Not a widget
- Post-Lollipop:
  - NOT part of the default theme
  - A widget
- DON'T use ActionBar as your App Bar
  - Use a theme which does NOT contain ".NoActionBar"
  - Action bar appear without any coding but getSupportActionBar() is used to manipulate it
  - API level differences
    - Features added incrementally
- DO use Toolbar as your App Bar
  - Use a theme which DO contain ".NoActionBar"
  - Use android.support.v7.widget.Toolbar and setSupportActionBar (Toolbar toolbar) to set the Toolbar to act as the Action Bar for this Activity window
  - No API level differences

Week 4 Part II

Broadcast Intents

- Intent objects that are sent out to multiple components within an application or different applications
- Used to communicate messages or events between different parts of an application
- Android OS use broadcast intents to notify applications about important system events
- Create a broadcast intent:
  - Specify an action string that describes the purpose of the intent.
  - Include additional data in key-value pairs using the putExtra() method. More information provided along with the intent

```
Intent intent = new Intent();
intent.setAction("intent.filter.data");
intent.putExtra("key", "SomeData");
sendBroadcast(intent);
```

Broadcast Receiver

- Base class for code that receives and handles broadcast intents sent by Context.sendBroadcast(Intent)
- If an application needs to listen for specific broadcasts, it has to register broadcast receivers
- Done by extending the BroadcastReceiver class and overriding the onReceive() method
  - 2nd param: Intent used to find all the data item

```
class MyBroadCastReceiver extends
BroadcastReceiver {
@Override
public void onReceive (Context
context,Intent intent) {
    }
}
```

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

```
MyBroadCastReceiver myBroadCastReceiver =
new MyBroadCastReceiver();
```

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

  - Register broadcast receiver using registerReceiver():
    - 1st param: Instance of the broadcast receiver
    - 2nd param: Action string that is used when declaring broadcast intent

SMS Receiver's BroadcastReceiver

```
public class SMSReceiver extends
BroadcastReceiver {

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

Week 5

Drawer layout

- Container that holds the content of a window and allows interactive "drawer" views to be pulled out from the sides of the window
- Specify the positioning and layout of the drawer by using the android:layout_gravity attribute on the child views
- Attribute determines the drawer will emerge from the left or right side of the view
- 1 drawer view for each side of the window
- If your layout tries to configure more than one drawer view for a side, an error will occur when running the app

AppBar Layout

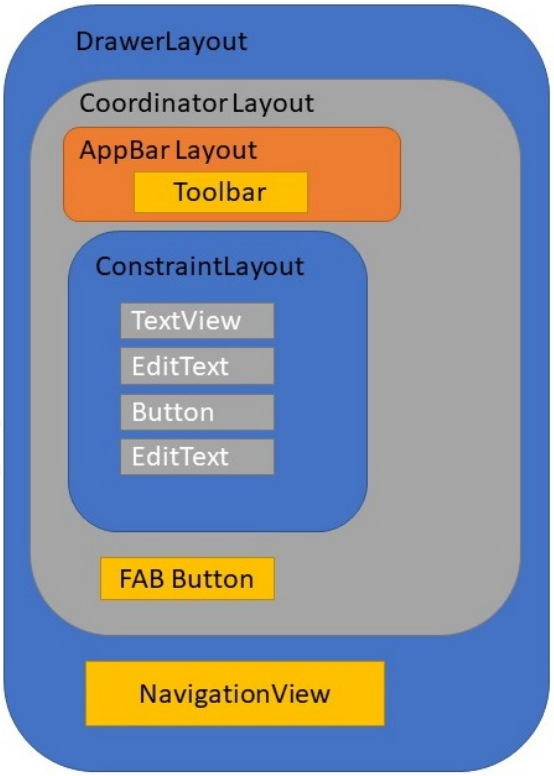
- Vertical LinearLayout that implements many of the features of material designs app bar concept
- Eg. scrolling gestures

Toolbar

- Generalisation of action bars for use within application layouts
- Toolbar may be placed at any arbitrary level of nesting within a view hierarchy
- An application may designate a Toolbar as the action bar for an Activity using the setActionBar() method
- The toolbar supports a more focused feature set than ActionBar
- From start to end, a toolbar may contain a combination of the following optional elements:
    - Navigation button
        - May be an Up arrow, navigation menu toggle, close, collapse, done or another glyph of the app's choosing
    - Branded logo image
    - A title and subtitle
    - One/many custom views
    - An action menu

CoordinatorLayout

- General-purpose container
- Coordinating interactive behaviours between children




```
<?xml version="1.0" encoding="utf-8"?>
<androidx.drawerlayout.widget.DrawerLayout
xmlns:android="http://schemas.android.com/
apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/drawer_layout"

xmlns:app="http://schemas.android.com/apk/
res-auto">

    <include
        layout="@layout/app_bar_layout"

android:layout_width="match_parent"
```

```
android:layout_height="match_parent" />


<com.google.android.material.navigation.Na
vigationView

android:layout_width="wrap_content"

android:layout_height="match_parent"
        android:id="@+id/nav_view"
        android:layout_gravity="start"
        app:menu="@menu/nav_menu" />
</androidx.drawerlayout.widget.DrawerLayou
t>
```

Line	Content
8-11	Include the app_bar_layout in the current one
13-18	Add NavigationView to the layout
14-15	Width and height of the Navigation view specified
17	Specifies the location of the navigation view
18	Specifies the location that lists this navigation view's menu items

```
<?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 code above represents a menu with two items represented by an ID and title
- ID used later by the navigation view events handlers

App_bar_layout items

- AppBarLayout wraps a Toolbar to give material design specs

```
<com.google.android.material.appbar.AppBar
Layout

android:layout_width="match_parent"

android:layout_height="wrap_content">

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

android:layout_width="match_parent"

android:layout_height="?attr/actionBarSize
" />

</com.google.android.material.appbar.AppBa
rLayout>
```

- Reference (include statement) to the main content layout

```
<include
        layout="@layout/activity_main"

android:layout_width="match_parent"

android:layout_height="match_parent"

app:layout_behavior="@string/appbar_scroll
ing_view_behavior"
    />
```

- Last line → stop overlapping toolbar
- Floating Action Button (FAB)

```
<com.google.android.material.floatingactio
nbutton.FloatingActionButton
        android:id="@+id/fab"

android:layout_width="wrap_content"

android:layout_height="wrap_content"

android:layout_gravity="bottom|end"
        android:layout_margin="45dp"

app:srcCompat="@drawable/baseline_add_24"
     />
```

Line	Content
5	Specifies the location of the FAB button
6	Margin between the FAB button and the activities edges
7	Specifies the icon of the FAB button

Hook Drawer layout and Toolbar

- Reference drawer layout and toolbar:

```
drawerlayout =
findViewById(R.id.drawer_layout);

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

- Use toolbar as action bar:

```
setSupportActionBar(toolbar);
```

- Actual hooking:

```
ActionBarDrawerToggle toggle = new
ActionBarDrawerToggle(
                this, drawerlayout,
toolbar, R.string.navigation_drawer_open,
R.string.navigation_drawer_close);
drawerlayout.addDrawerListener(toggle);
toggle.syncState();
```

- Strings resources declaration:

```
<resources>
    <string
name="navigation_drawer_open">Open</string
>
    <string
name="navigation_drawer_close">Close</stri
ng>
</resources>
```

Navigation View Events Listener

- Listen to events from the navigation view:
  - Implement the 'OnNavigationItemSelectedListener' interface (see line 1)
  - Retrieve ID of the selected item (see line 5)
  - Compare the selected id with the ids we declared in the navigation menu file (see lines 7-11)
  - Execute the required behaviour (lines 8 and 10)
  - Return true/false that tells Android whether we consumed the event or not (see line 15)

```
   class MyNavigationListener implements
NavigationView.OnNavigationItemSelectedLis
tener {
        @Override
        public boolean
onNavigationItemSelected(@NonNull MenuItem
item) {
            // get the id of the selected
item
            int id = item.getItemId();

            if (id ==
R.id.add_item_menu_id) {
                // Do something
            } else if (id ==
R.id.clear_fields_menu_id) {
                // Do something
            }
            // close the drawer
            drawerlayout.closeDrawers();
            // tell the OS
            return true;
        }
    }
```

Listen to FAB Events

1. Create reference to FAB button

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

2. Set listener for FAB button

```
fab.setOnClickListener(new
View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Snackbar.make(view, "Replace with
your own action", Snackbar.LENGTH_LONG)
                .setAction("Action",
null).show();
    }
});
```

- User clicks or taps on the fab button, the callback method "onClick" will be executed
- The method shows a Snackbar from the bottom of the screen

Snackbar

- Brief feedback about an operation through a message at the bottom of the screen
- Disappear automatically
- Can be swiped off the screen

Making Snackbar

- 1st param of Snackbar.make(...) method is a starting point for the SnackBar to walk up the UI's View tree looking for a CoordinatorLayout to be its parent
    - Lift the FAB up out of the way when shown
- Snackbar.make(...) returns a SnackBar object. Snackbar's setAction() method is invoked on this object to specify the text on its action button and the listener object containing the event handler for when the action button is clicked
- Snackbar's setAction() method returns the Snackbar object it was invoked on. This returned Snackbar object is displayed using Snackbar's show() method

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

Options Menus ⋮

- Accessible menu from device display
- Allow inclusion of other application options beyond those included in the App's UI

Create Options Menus

1. Add a new Menu Resource File

```
<?xml version="1.0" encoding="utf-8"?>
<menu
xmlns:android="http://schemas.android.com/
apk/res/android">
    <item
        android:id="@+id/option_item_id_1"
        android:title="Option One" />
    <item
        android:id="@+id/option_item_id_2"
        android:title="Option Two" />
</menu>
```

2. In MainActivity.java, implement methods onCreateOptionsMenu() and onOptionsItemSelected()

```
@Override
    public boolean
onCreateOptionsMenu(Menu menu) {

getMenuInflater().inflate(R.menu.options_m
enu, menu);
        return true;
    }
```

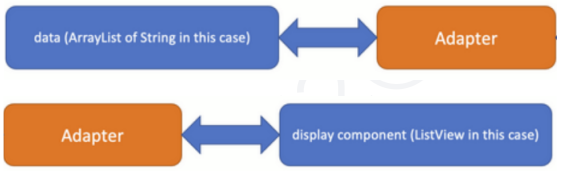
- onCreateOptionsMenu method responsible for inflating the menu file

```
@Override
    public boolean
onOptionsItemSelected(@NonNull MenuItem
item) {
        int id = item.getItemId();

        if (id == R.id.option_item_id_1) {
            // Do something
        } else if (id ==
R.id.option_item_id_1) {
            // Do something
        }
        // tell the OS
        return true;
    }
```

- onOptionsItemSelected responsible for handling the options menu events

List Views

- Components can be created and added to a layout in code or XML
- Implements a view showing items in a vertically scrolling list where the items come from the ListAdapter associated with this View



Setting up a ListView (see MainActivity's onCreate):

1. Create the ArrayList (see line 1)
2. Get reference to the ListView (see line 13)
3. Instantiate adapter (see line 15)
4. Plug adapter into the ListView using the ListView's setAdapter(...) (see line 16)

- ListView's setAdapter(...) method takes a ListAdapter as its only parameter

```
    ArrayList<String> listItems = new
ArrayList<>();
    ArrayAdapter<String> adapter;
    private ListView myListView;


    @Override
    protected void onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_main);
        Toolbar toolbar =
findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        myListView =
findViewById(R.id.list_view);

        adapter = new
ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1,
listItems);
        myListView.setAdapter(adapter);
```

- 'list_view' is the ID of the listview UI element that is placed in the layout:

```
      <ListView
          android:id="@+id/listView"
          android:layout_width="0dp"
          android:layout_height="0dp">
      </ListView>
```

Week 6

Recycler Views

- Flexible view group for providing a limited window into a large data set
- Purpose: Allow information to be presented to the user as a scrollable list
- Efficient in managing the views that make up a list
- Reusing existing views that make up list items as they scroll off the screen instead of creating new ones
- Use a subclass of RecyclerView.Adapter for providing views that represent items in a data set

Card View

- UI view that allows information to be presented in groups using a card metaphor
- Shown in lists using a RecyclerView instance and may be configured to appear with shadow effects and rounded corners
- Defined within an XML layout resource file and loaded into the CardView at runtime
- Contain a layout of any complexity using the standard layout managers such as 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.co
m/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:co
lor/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/suit_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/card_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>
```

- CardView works as a top-level container that has a relative layout as a child element
- Inside the relative layout, 2 text views to display our data

RecyclerView.Adapter

- Created as a subclass of the RecyclerView.Adapter class and must implement the following methods:
  - getItemCount()
  - onCreateViewHolder()
  - onBindViewHolder()
- getItemCount(): Return a count of the number of items to be displayed in the list
- onCreateViewHolder(): Creates and returns a ViewHolder object initialised with the view that is used to display the data
  - View is typically created by inflating the XML layout file
- onBindViewHolder(): Passes the ViewHolder object created by the onCreateViewHolder() method and an integer value indicating the list item that is about to be displayed
  - Contained within the ViewHolder object is the layout assigned by the onCreateViewHolder() method
  - Responsible for populating the views in the layout with the text and graphics corresponding to the specified item and returning the object to the RecyclerView
- Implementation of the RecyclerAdapter:

```
package com.fit2081.recyclercard;

import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

import androidx.annotation.NonNull;
import
androidx.recyclerview.widget.RecyclerView;
```

```
import java.util.ArrayList;

public class MyRecyclerAdapter extends
RecyclerView.Adapter<MyRecyclerAdapter.Vie
wHolder> {

    ArrayList<Item> data = new
ArrayList<Item>();

    public void setData(ArrayList<Item>
data) {
        this.data = data;
    }

    @NonNull
    @Override
    public ViewHolder
onCreateViewHolder(@NonNull ViewGroup
parent, int viewType) {
        View v =
LayoutInflater.from(parent.getContext()).i
nflate(R.layout.card_layout, parent,
false); //CardView inflated as
RecyclerView list item
        ViewHolder viewHolder = new
ViewHolder(v);

Log.d("week6App","onCreateViewHolder");
        return viewHolder;
    }

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

holder.cardTv.setText(data.get(position).g
etCard());

holder.suitTv.setText(data.get(position).g
etSuit());

Log.d("week6App","onBindViewHolder");

    }

    @Override
    public int getItemCount() {
        return data.size();
    }

    public class ViewHolder extends
```

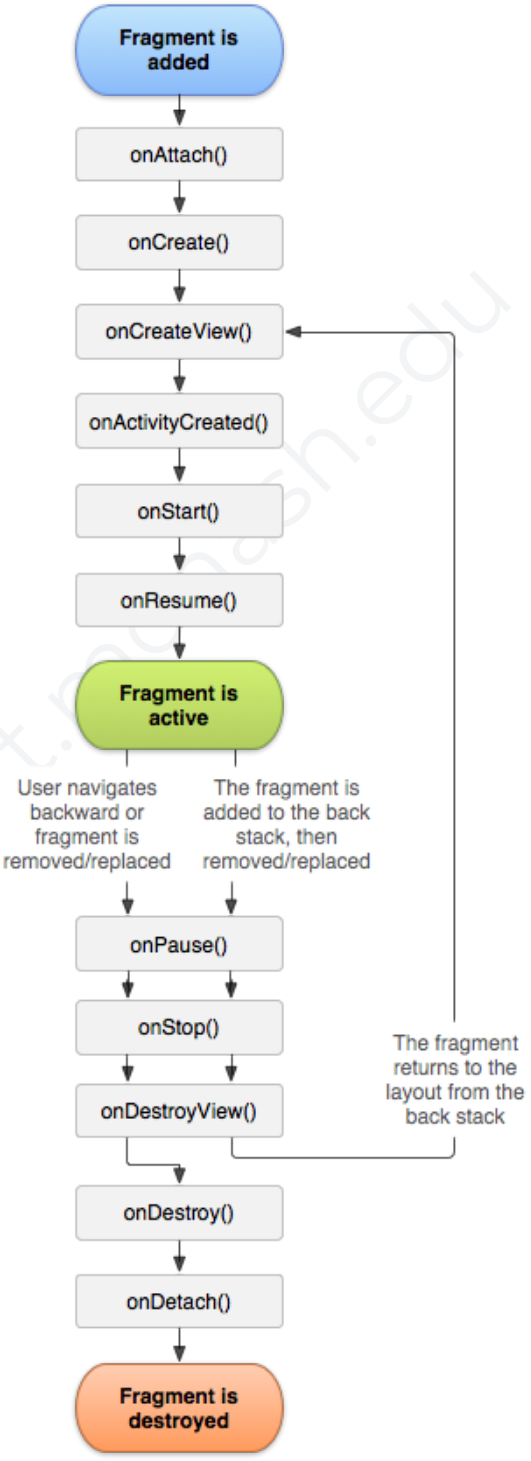
```
RecyclerView.ViewHolder {
        public TextView suitTv;
        public TextView cardTv;

        public ViewHolder(@NonNull View
itemView) {
            super(itemView);
            suitTv =
itemView.findViewById(R.id.suit_id);
            cardTv =
itemView.findViewById(R.id.card_id);
        }
    }
}
```

- Data is passed to the adaptor through the method setData
- methodÂ onCreateViewHolder inflates the card's layout
    - passes v, which is a reference to the card layout to the contractor of ViewHolder local class
- Line 28 method returns the view holder object that will be an input to the following method onBindViewHolder
- Latter method, we receive two parameters, a viewer holder and a position
- Data at that position retrieved and place it in that view holder

Each time the data gets changed, adapter should be notified by calling adapter.notifyDataSetChanged() in order to update the recycler view

Fragments

- Represents a behaviour or a portion of the user interface in a FragmentActivity
- Combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities
- Have own lifecycle, receives own input events and adds / removes while activity runs



- onAttach() method is invoked when the fragment has been associated with the activity
- onCreate() The system calls this when creating the fragment

- onCreateView() The system calls this when it's time for the fragment to draw its user interface for the first time
- onActivityCreated() Called when the activity's onCreate() method has returned
- onStart() This method is called once the fragment gets visible
- onPause() The system calls this method as the first indication that the user is leaving the fragment
- onDestroyView()method is invoked when the view hierarchy associated with the fragment is being removed
- onDetach() method is invoked when the fragment is being disassociated from the activity

```
package edu.monash.fit2081.fragments;

import
android.support.v7.app.AppCompatActivity;
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);
    }

    public void handleBtn1(View view){

getSupportFragmentManager().beginTransacti
on().replace(R.id.frag1,new
Fragment1()).addToBackStack("f1").commit()
;

    }
    public void handleBtn2(View view){

getSupportFragmentManager().beginTransacti
on().replace(R.id.frag1,new
Fragment2()).addToBackStack("f2").commit()
;

    }
    public void handleBtn3(View view){

getSupportFragmentManager().beginTransacti
on().replace(R.id.frag1,new
Fragment3()).addToBackStack("f3").commit()
;

    }
}
```

This specific statement is responsible for replacing the current fragment with a new one:

```
getSupportFragmentManager().beginTransacti
on().replace(R.id.frag1, new
Fragment1()).addToBackStack("f1").commit()
;
```

- getSupportFragmentManager: Returns FragmentManager used to create transactions for adding, removing or replacing fragments
- beginTransaction: Starts a series of edit operations on the Fragments
- .replace: Replaces the current fragment with a new fragment of type Fragment1 on the layout with id R.id.frag1
- addToBackStack: Adds this transaction to the back stack
    - Transaction will be remembered after it is committed and will reverse its operation when later popped off the stack

JSON

- Lightweight data-interchange format
- Easy for humans to read and write
- Easy for machines to parse and generate
- Eg.

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

Syntax Rules:

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

Nested Data

- Nested Arrays and objects can be in JSON format

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

Gson

- Used to convert Java Objects into their JSON representation
- Convert a JSON string to an equivalent Java object
- Work with arbitrary Java objects including pre-existing objects that you do not have source code of
- Provides simple toJson() and fromJson() methods to convert Java objects to JSON and vice-versa
- Add Gson dependencies:

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

- Create new Gson instance:

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

- Convert the ArrayList object into a string:
  (ArrayList name is db)

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

- Push the string into your database (or shared preferences):

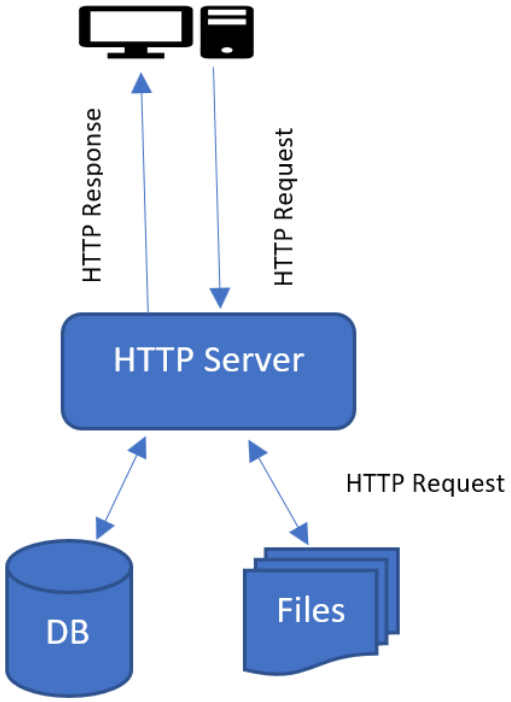
```
SharedPreferences.Editor edit = sP.edit();
edit.putString(DB_KEY, dbStr);
edit.apply();
```

- Convert the string (JSON) back to the ArrayList data type:
  - fromJson converts the first parameter's data type into the type specified in the second parameter

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

HTTP

- HyperText Transfer Protocol
- HTTP server: Program that serves data to clients using the HTTP protocol



- Client from a web browser sends a request to a web server
- Server accepts the request and generates a response using its local database or files
- Server sends back its response to the client
- Client's browser displays/renders the response

Volley

- 3rd party library
- Send the request
- Process the response

```
RequestQueue queue =
Volley.newRequestQueue(this);
```

Volley

- Create the request and place it later in the queue:

```
JsonObjectRequest stringRequest = new
JsonObjectRequest(Request.Method.GET, url,
null, new Response.Listener<JSONObject>()
{
            @Override
            public void
onResponse(JSONObject response) {
                // When the response
arrives from the server
                //process it here
            }
        }, new Response.ErrorListener() {

            @Override
            public void
onErrorResponse(VolleyError error) {
                // if an error occurred,
this callback method will get executed
            }
        });
```

- onResponse: Invoked if the response (data) arrives
- onErrorResponse: Called when error occurs
- 2nd param: String representing the URL of the endpoint
- Add the string to the queue:

```
queue.add(stringRequest);
```

Extract data from the JSONObject

- Key to retrieve the value
- Eg. 'response' contain JSON object with their keys

```
String name=response.getString("name");
String unit=response.getString("unit");
int year=response.getInt("year");
JSONObject
dataObj=response.getJSONObject("data");
int mark=dataObj.getInt("mark");
double ratio=dataObj.getDouble("ratio");
```

- Keys are name, unit, year, data
- Data keys are mark, ratio

Week 7

Database

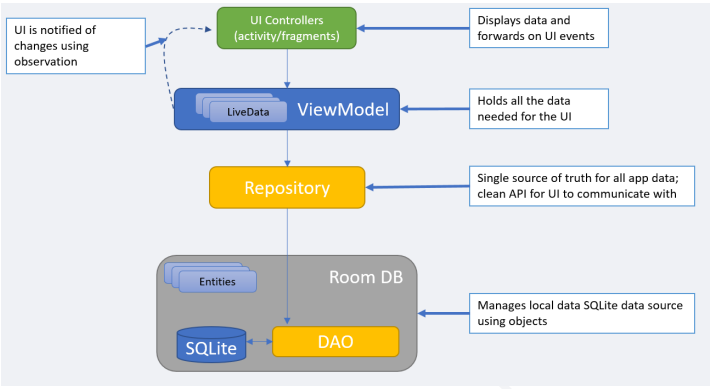
- Organised collection of structured information or data, typically stored electronically in a computer system
- Controlled by a database management system (DBMS)
- Android uses SQLite as a database management system

SQL

- SQLite uses SQL language (Structured Query Language)
- Access and manipulate databases
- SQL Statements add or retrieve data from a database
- Query, manipulate, and define data and provide access control

Table

- 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
    - Unique
    - Cannot be NULL values
    - 'Primary key' (PK)



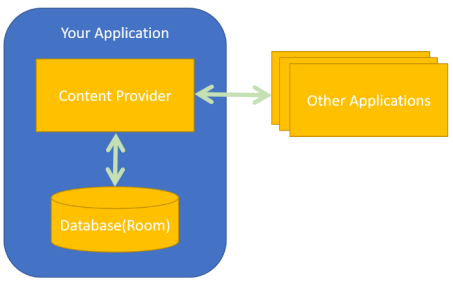
- SQLite database: Relational database management system that is used by Android to store relational data
- DAO: Data Access Objects are the main classes where you define your database interactions
    - Include a variety of query methods
- Entities: 1 entity = 1 table in the database
- Room database: Provides the interface to the underlying SQLite database
- Repository: A class that contains all of the code necessary for directly handling all data sources used by the application
    - Avoids the need for the UI controller and ViewModel to contain code that directly accesses sources such as databases or web services
- 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 model
- LiveData: Observable object can notify other objects when changes to its data occur, solving the problem of ensuring that the UI always matches the data within the ViewModel

Entities

```
package com.fit2081.rooms.provider;

import androidx.annotation.NonNull;
import androidx.room.ColumnInfo;
import androidx.room.Entity;
import androidx.room.PrimaryKey;

@Entity(tableName = "customers")
public class Customer {
    @PrimaryKey(autoGenerate = true)
    @NonNull
    @ColumnInfo(name = "customerId")
    private int id;
    @ColumnInfo(name = "customerName")
    private String name;
    @ColumnInfo(name = "customerAddress")
    private String address;

    public Customer(String name, String
address) {
        this.name = name;
        this.address = address;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public String getAddress() {
        return address;
    }

    public void setId(@NonNull int id) {
        this.id = id;
    }

}
```

Line	Content
8	Annotation @Entity is required to define the class as a Room Entity. It also specifies the table name
10	Annotation '@PrimaryKey' makes the id as a primary key for the current table. '@NonNull' ensures that this column will not be saved without a value
'Customers' has three attributes: id, name, and address. Each attribute has an annotation 'ColumnInfo' that specifies the column name in the database	

Ignore this:



Room Database

```
package com.fit2081.rooms.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 = {Customer.class},
version = 1)
public abstract class CustomerDatabase
extends RoomDatabase {

    public static final String
CUSTOMER_DATABASE_NAME =
"customer_database";

    public abstract CustomerDao
customerDao();

    // marking the instance as volatile to
ensure atomic access to the variable
    private static volatile
CustomerDatabase INSTANCE;
    private static final int
NUMBER_OF_THREADS = 4;
    static final ExecutorService
databaseWriteExecutor =

Executors.newFixedThreadPool(NUMBER_OF_THR
EADS);

    static CustomerDatabase
getDatabase(final Context context) {
        if (INSTANCE == null) {
            synchronized
(CustomerDatabase.class) {
                if (INSTANCE == null) {
                    INSTANCE =
Room.databaseBuilder(context.getApplicatio
nContext(),

CustomerDatabase.class,
CUSTOMER_DATABASE_NAME)
                            .build();
                }
            }
        }
        return INSTANCE;
    }

}
```

Line	Content
10	'@Database' annotation is required to consider the current class as a Room database. List of entities and the current version is specified. Version is required for upgrading or downgrading the current scheme
20	'databaseWriteExecutor' instance will be used by the repository to execute the DAO methods
23	'getDatabase' method returns a reference to the current database instance if it is not null else new instance is created using 'Room.databaseBuilder()', which needs as input the context, a reference to the Room Database class, and a name for the database

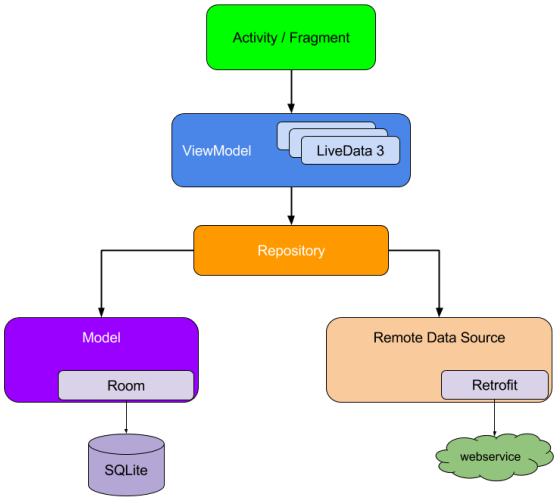
DAO

```
package com.fit2081.rooms.provider;

import androidx.lifecycle.LiveData;
import androidx.room.Dao;
import androidx.room.Insert;
import androidx.room.Query;

import java.util.List;

@Dao
public interface CustomerDao {

    @Query("select * from customers")
    LiveData<List<Customer>>
getAllCustomer();

    @Query("select * from customers where
customerName=:name")
    List<Customer> getCustomer(String
name);

    @Insert
    void addCustomer(Customer customer);

    @Query("delete from customers where
customerName= :name")
    void deleteCustomer(String name);

    @Query("delete FROM customers")
    void deleteAllCustomers();
}
```

Line	Content
10	Annotation '@Dao' is required to consider the interface as a DAO
13	Query annotation provides the select SQL statement that should be executed when the method 'getAllCustomer()' is invoked. Select statement at line 13 retrieves all the records in table 'customers' (see entity@line 8) as it does not have the 'WHERE' clause
14	Output of 'getAllCustomer' method is LiveData which allows us to observe any changes to the database
16	Contains a 'WHERE' clause. Retrieves all the customers with a name equal to a value that is provided as an input parameter to the method 'getCustomer'. Please note the colon (:) which is required to specify the name of the variable
19	'@Insert' annotation inserts the object that is passed through the method 'addCustomer'
22	Delete query deletes all the rows that have names equal to the input parameter to the method 'deleteCustomer'
25	Deletes all the rows (empty the table) if a call to method 'deleteAllCustomers' occurs

Room Repository

- Provides an easy and clean API
- Access different data sources
- Manage 2 different data sources:
  - Local SQLite database
  - Remote data source




```
package com.fit2081.rooms.provider;
import android.app.Application;
import androidx.lifecycle.LiveData;

import java.util.List;

public class CustomerRepository  {

    private CustomerDao mCustomerDao;
    private LiveData<List<Customer>>
mAllCustomers;

    CustomerRepository(Application
application) {
        CustomerDatabase db =
CustomerDatabase.getDatabase(application);
        mCustomerDao = db.customerDao();
        mAllCustomers =
mCustomerDao.getAllCustomer();
    }
    LiveData<List<Customer>>
getAllCustomers() {
        return mAllCustomers;
    }
    void insert(Customer customer) {

CustomerDatabase.databaseWriteExecutor.exe
cute(() ->
mCustomerDao.addCustomer(customer));
    }

    void deleteAll(){

CustomerDatabase.databaseWriteExecutor.exe
cute(()->{

mCustomerDao.deleteAllCustomers();
        });
    }
}
```

Line	Content
9	Declares a reference to the Dao interface which will be used to execute the database operations
10	Defines an array list that is used to hold a copy of data
12	Creates a reference to the database that will be used to access the Dao interface. The Dao interface will be used later (line@15) to get the list of customers
20	Inserts a new row (object) into the database. It uses the 'databaseWriteExecutor' to access the database and execute the insert SQL statement
24	Uses the 'databaseWriteExecutor' to execute the delete statement

ViewModel

- Retrieval of special data
- Retrieval of data by different method

```
package com.fit2081.rooms.provider;

import android.app.Application;

import androidx.annotation.NonNull;
import
androidx.lifecycle.AndroidViewModel;
import androidx.lifecycle.LiveData;

import java.util.List;

public class CustomerViewModel extends
AndroidViewModel {
    private CustomerRepository
mRepository;
    private LiveData<List<Customer>>
mAllCustomers;

    public CustomerViewModel(@NonNull
Application application) {
        super(application);
        mRepository = new
CustomerRepository(application);
        mAllCustomers =
mRepository.getAllCustomers();
    }

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

    public void insert(Customer customer)
{
        mRepository.insert(customer);
    }
    public void deleteAll(){
        mRepository.deleteAll();
    }

}
```

MainActivity.java

- Create an instance of the view model:

```
private CustomerViewModel
mCustomerViewModel;

mCustomerViewModel = new
ViewModelProvider(this).get(CustomerViewMo
del.class);

mCustomerViewModel.getAllCustomers().obser
ve(this, newData -> {
            adapter.setCustomers(newData);

adapter.notifyDataSetChanged();
            tv.setText(newData.size() +
"");
        });
```

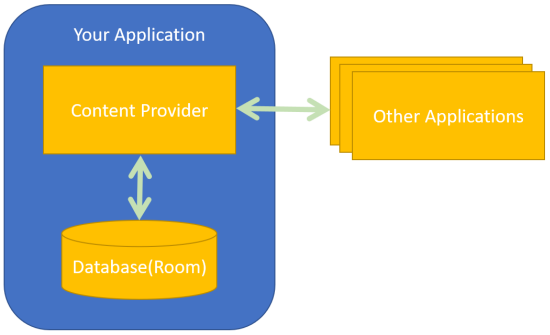
Line	Content
3	ViewModelProvider (An utility class that provides ViewModels for a scope) used to get access to the view model
4	Invokes the getAllCustomers to get the list of customers getAllCustomers Output: LiveData, caller have to observe Observe method: Invokes the callback method which is provided in the second parameter each time the data gets changed. The new updates can be accessed via the input variable 'newData'

Dependencies to add:

```
Implementation
'androidx.room:room-common:2.5.1'
Implementation
'androidx.room:room-runtime:2.5.1'
annotationProcessor
'androidx:room:room-compiler:2.5.1'
```

Week 8

Content Provider



- A class that helps applications to share their data with other applications on the same device
- Access to data in database via Universal Resource Identifier (URI) defined by Content Provider
- URI Authority
    - Unique name to identify CP
    - Hence can't have 2 CP with same authority value on same device
- Required methods:
    1. onCreate(): called to initialise provider
    2. query(Uri, String[], CancellationSignal): return data to caller
        a. Uri: maps to the table name
        b. Projection: to project the list of column to be included
        c. Selection: string representing the where clause
        d. Selection arguments: string array representing values that should be embedded in the selection argument
        e. Sort order: sort data in asc/desc order
    3. insert(Uri, ContentValues): insert new data to CP
    4. update(Uri, ContentValues, String,
    5. String[]): update existing data
    6. delete(Uri, String, String[]): delete data
    7. getType(Uri): return the data's MIME type

```
select name,age,salary from users where age>25 and salary<1500 order by salary ASC;
```

Argument	SQL value
Uri	content_authority/users
projection	["name","age","salary"]
Selection	case 1: "where age>25 and salary<1500" case 2: "where ag
Selection Argument	for case 1: null for case 2: ["25","1500"]
sort order	"salary ASC" or null for default

- Cursor
    - Interface to provide random read-write access to the result set returned by a database query
    - A data structure that holds one or more rows retrieved from a database
    - Contains methods to move cursor to next row, previous, first, or last row
- Content Values
    - Special data structure to hold data of one row only
    - Key-value pair format
- FAQ:
    1. Difference between Content Values and Bundle:
        a. Key in CV are the table's column name

2. Difference between CV and Cursor:
    a. CV can hold data of one row only, Cursor may contain data for multiple rows

- UriMatcher()
    - Method used to create new instance of the UriMatcher class
    - Provide each URI unique code to be used by CP's methods to identify table to use / retrieve requested ID
    - '#' → integer value representing an ID
- Content Resolver
    - Object containing a set of methods that mirror those of the Content Provider (CRUD) to access Content Provider

Week 9

Google Maps

- A set of Google API classes and interfaces grouped together

Web Services

- Function calls made using HTTP
- GET, POST, PUT, DELETE (R, C, D, U) data operations

Web View

- View widget to display web pages
- Default: show a web page
- When to use WebView:
    - When you want to provide info in an application that might need to update frequently without editing, recompiling and resubmitting to app stores.

Thread

- To avoid locking the main UI thread
- Eg: multi-threaded app running tasks in the background, and update th eUI
- Require at least two pointers/references
- First pointer references the background thread; second points at the UI thread
- Executor run tasks in background
- Handler publish result of executor on UI thread

JSON

- Lightweight data-interchange format
- Easy for humans to read and write
- Easy for machines to parse and generate
- Syntax:
    - Data in name/value pairs
    - Commas separate data
    - Curly bracket hold objects
    - Square bracket hold arrays
    - No single quotes, no round brackets

Week 10

Gesture

- A sequence of touch events (x and y coordinates)
- Touch down -> move -> up

Motion Event

- The object to report movement event (mouse, finger)
- The key to obtain info about motion events, eg: location of the touch

Types of Events

- MotionEvent.ACTION_DOWN: the first touch
- MotionEvent.ACTION_UP: touch is lifted from screen
- MotionEvent.ACTION_MOVE: any motion of touch between up and down events

NOTE

- getActionMasked(): methods returning an integer number representing the type of the current event
- onTouch() return true, means successfully handled the touch event; else return false, give control back to parent touch event
- getX(), getY() obtain the absolute coordinate value relative to the view
- getRawX(), getRawY() are the coordinate relative to device screen

Week 11

Multi-Touch

- Happens when more than one pointer (finger) touches the screen
- Use pointer's index and ID to keep track each pointer within a gesture
    - Each pointer gets a unique ID during the gesture's lifetime and it's used to track the pointers within the gesture
    - MotionEvent obj saves all the pointers' data in a special array and uses indices to access the pointers entries
    - Pointers' indices can be changed (shift up) when a pointer leaves the screen, ID does not

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

- To detect common gestures through a set of motion events
- Steps:

1. Create an instance of the Gesture Detector class
2. Implements the required methods
3. Intercept the touch events and pass them to the gesture detector

FAQ:

1. Some methods return boolean values, what does that mean
    a. To inform the parent that the event has been consumed and is ready to accept further events from current gesture
    b. If return false, means the event is not consumed and it is not interested in the remainder of the gesture
2. Difference between onFling() and onScroll() callbacks
    a. onFling() needs velocity in movement (eg: swipe to unlock screen)
    b. onScroll() is invoked when the touch moves with normal speed (scroll a list)
    c. onFling() will only be called once at the end of the gesture; onScroll() will be called multiple times continuously as the touch moves on the screen
3. What happens when onScaleBegin() callbacks return false
    a. onScale and onScaleEnd will not be invoked, as the onScaleBegin indicates that it is not interested in the current gesture anymore
4. How to get the pinch size at the end of the gesture
    a. Using detector.getCurrentSpan()
5. What if want to implement a subset of the callback
    a. Implement a class that extends the convenience class (a class that contains a subset of the interface callbacks)

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.

Week 12

Building APKs

- What needs to be done:
  - Source code files compilation to compilation units
  - Link compilation units and resources
  - Signing APK with debug / release key store
  - Various optimisations
- Rebuilding??
  - Instant Run
    - Reduce time taken to update app with code and resource changed
    - Settings → Build, Execution, Deployment → Instant Run
  - Warning
    - Device Specific
    - X totally robust at this time
- Gradle
  - Advanced build toolkit, to automate & manage the build process
  - Allow to define flexible custom build configurations
  - Each build configuration define its own set of code & resources while reusing parts common to all versions of app
- Build Types
  - Define certain properties that Gradle uses when building & packaging the app
  - Typically configured for different stages of the development lifecycle
  - Eg. Debug, Release
  - X product flavours then Build types = Build variant
- Product Flavours
  - Different versions of app that release to users
    - Eg. Free / Paid
  - Customise product flavours to use different code and resources, while sharing and reusing the parts common to all versions
  - Eg. Different features, Demo / Full, Branding
  - Optional, manual creation required
- Build Variants
  - Cross product of a build type & product flavour
  - Gradle use this configuration to build the apps
  - Eg. Demo / Debug, Full / Release
- Android Studio do default work
  - Default Run / Debug Configuration
    - Android Studio creates default run/debug configuration for the main activity based on Android App template
    - To run/debug, must have >= 1 run/debug configuration defined
  - Default Build Types
    - Create and configure build types in the module-level build.gradle file
    - Create new module = Android auto-creates debug & release build types

- Setting Build Type of modules for release
    - View → Tool Windows → Build Variants
        - Switch debug → release
    - Debug Keystore
        - All apps need to be digitally signed to successfully upload to an emulator / device
        - Android Studio auto-configure new projects with debug keystores
    - Don't confuse release and debug build types and run and debug toolbar buttons

Toolbar buttons	Build APK according to current selected build variant the push to specified target and launch it
Run toolbar button	Build & launch debug build variant that **will not** engage with debugger
Debug toolbar button	Build & launch debug build variant that **will** engage with debugger

- ProGuard
    - Used (optionally) in the APK build process (release variant)
        - Perform several optimisation/verification and obfuscation tasks
        - Improve efficiency
        - Reduce size

Signing APK
- Public/Private Key Pair
    - Generate private/public key pair to sign the app before upload to Google Play Store
    - Store key pairs in a key store file
- Private Key
    - Key pair
        - Public-key certificate
        - 'Fingerprint' that uniquely associates APK to you and your corresponding private key
        - Contains the public key of a public / private key pair
    - Keystore
        - Binary file that contains one or more private keys
        - When sign APK for release using Android Studio can choose to generate new keystore & private key or use keystore & private key already have
- Look after Private Key
    - If other people use your key:
        - Your authoring identity and the trust of the user are compromised
        - They could sign and distribute apps then maliciously replace your authentic apps or corrupt them or steal user data
    - Private key is required for signing all future versions of the app. If a key is lost or misplaced, you cannot publish updates to existing app

    - Can't regenerate a previously generated key
- Debug Key
    - When running or debugging projects from the IDE, Android Studio automatically signs APK with a debug certificate generated by the Android SDK tools
    - Debug certificate expiry:
        - Expiration date of 365 days from its creation date
        - When certificate expires, build error will occur
        - Delete debug.keystore file to fix error
        - Next time you build and run the debug build type, build tools will regenerate a new keystore and debug key

Submitting APKs to Google Play App Store

- Submitting APKs to the Google Play app store
    - Once "Generate Signed APK" Wizard Finishes, a release APK will have been created ready for submission to the Google Play App Store
    - Create a "Google Play Developer Console Account"
    - Publish
        - Complete pre-launch checklist
        - In developer console, click publish button
        - Little non-technical app reviews
    - Publish New Versions
        - Updated APK signed using same key pair used in initial submission
        - Up version number
    - Analysing APK
        - Select APK file to be analysed
        - Analyse component size & file structure
        - Track size blowouts
    - Technical Details
        - Signing

Hash app's APK contents
Encrypt hash using private key of the app's developer's public / private key pairs
Signed hash + hashing function ID + public key of the signing key are added to APK as signing block

        - Verifying

Extract signing block	
Hash APK contents to create calculated hash	
Decrypt the sent, signed hash with the sent public key	
Compare calculated & sent hash If equal, verify:	
APK sent by owner of public key	Confirm integrity of APK

    - Signing & Public / Private keys
        - Quick, Easy key pair generation
        - Private key must be closely guarded
        - Anyone can have the public key
        - Message signed with private key
        - APK not encrypted

        - Computationally costly
        - Used hash instead
    - Signing with private key = Successfully decrypted with corresponding public key

Cryptographic Hash Function
- Special class of hash function that has certain properties which makes it suitable for use in cryptography
- Required Properties:
    - Deterministic (results in the same hash)
    - Quick to compute the hash value
    - Small change to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value

Task to prepare for release
- Gather materials and resources
    - Collect various supporting elements
        - Cryptographic keys for app signing
        - App icon
        - End-user licence agreement
    - Must digitally sign with a certificate, as required by the Android platform
- Configure app for release
    - Concise overview of the recommended configuration changes to the source code, resources files and app manifest before releasing your app Build your app for release
- Build app for release
    - Build APK
    - Signing & Optimising APK
- Prepare external servers & resources
    - Remote server: crucial to ensure that the server is secure and appropriately configured for production use
    - Important when implementing in-app billing and verifying signatures on a remote server
    - App retrieves from remote server or real-time service
        - Eg. Content feed
        - Essential to verify that the content provided is up-to-date & suitable for production usage
- Test app for release
    - Essential to ensure proper functionality across realistic device & network environments
    - Recommended to test on both handset-sized and tablet-sized devices to ensure the correct sizing of UI elements & satisfactory performance & battery efficiency