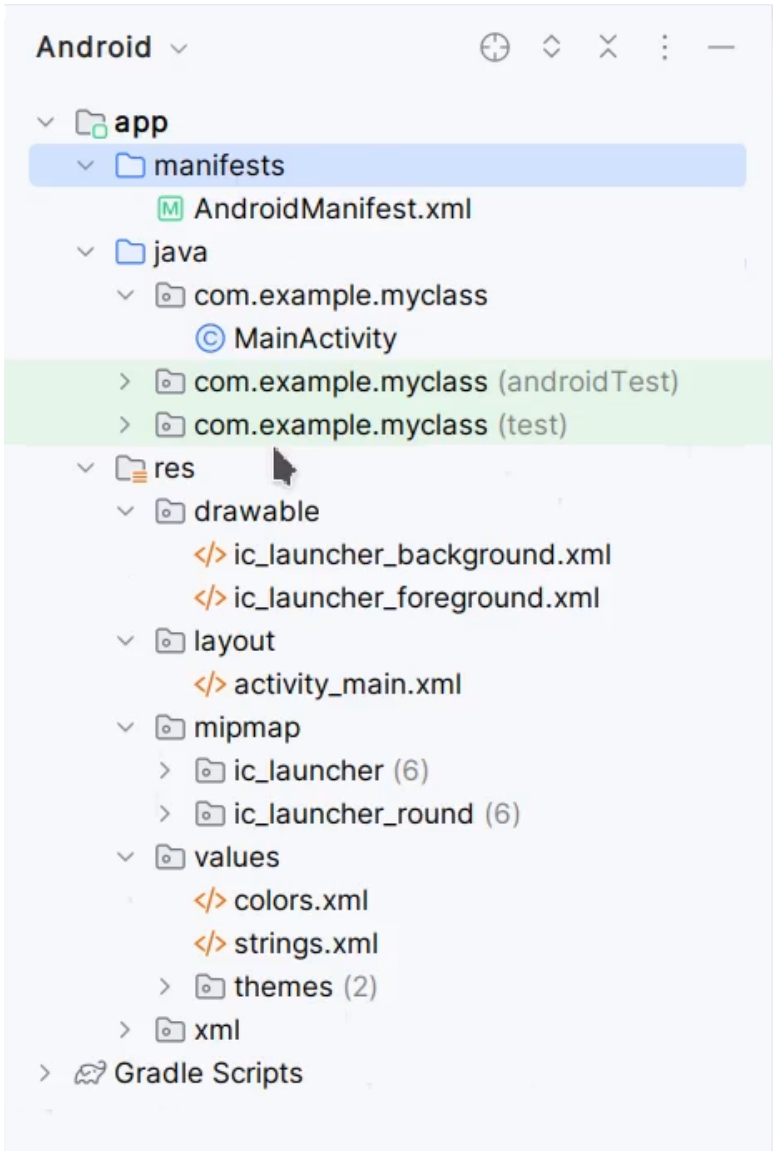
Android Application Development

The Android Studio IDE

Installation via snap

```
sudo snap install android-studio --classic
# /home/shrivatsa/Android/Sdk
```

Android Studio Project structure



manifest folder

- Contains AndroidManifest.xml file which is the first file to be loaded whenever we call the app.
  This file will have the informations about the icons, permissions, the app name and says which file has to be opend first i.e the
  `MainActivity.java` file's information will be inside manifest.
- It contains permission information required by the app. ex: permission to access calendar, camera etc.
- App icon information.
- Title of the app
- Which screen has to be loaded when we start the app.
- The AndroidManifest.xml file calls the java file into the background which in turn calls the activity_main.xml file (the frontend file)
- <manifest> </manifest> is the root tag of the manifest file

res folder

- The resources folder contains list of all resources the app needs to run like the colour code, the themes etc.
- The R represents the resources folder in the ActivityMain.java file.

- The resources folder saves the app icon info inside the drawable folder and activity_main.xml which has front end information is also stored in res folder.

mipmap folder

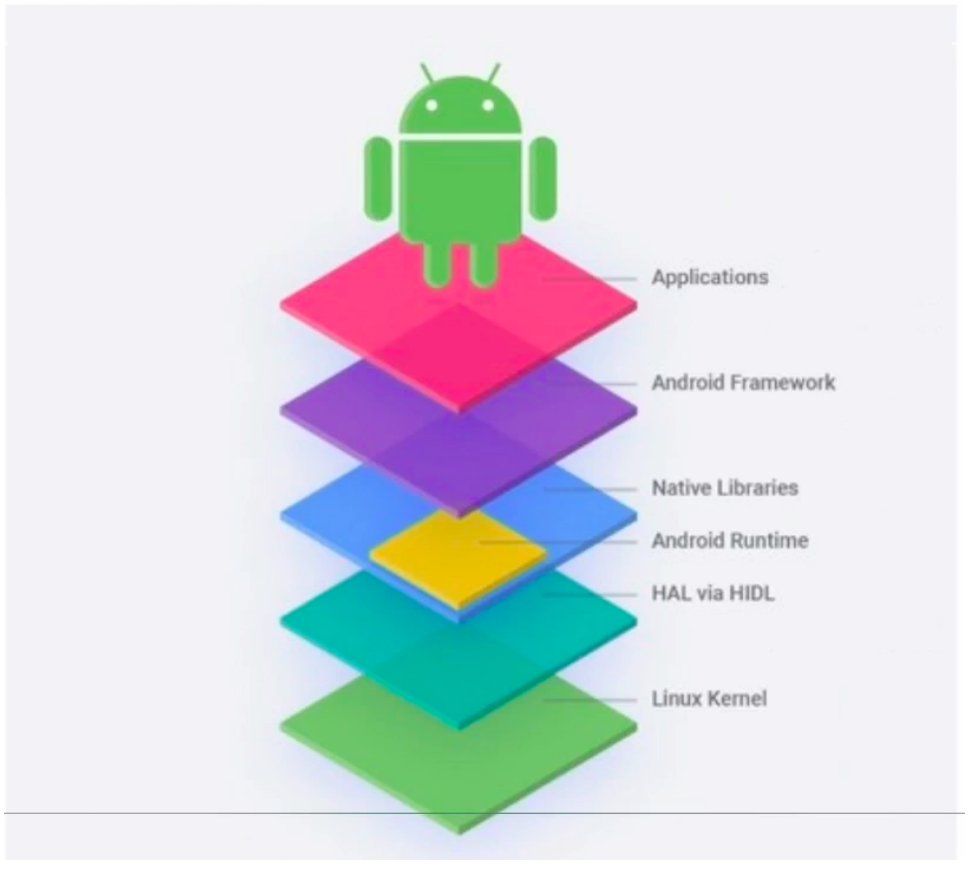
- This folder contains information about the icon size of the application so in order to fit screen size of different phones. It also store the shape info of the launcher.
- This folder also contains info about weather the icon must be rounded of square.

Gradle

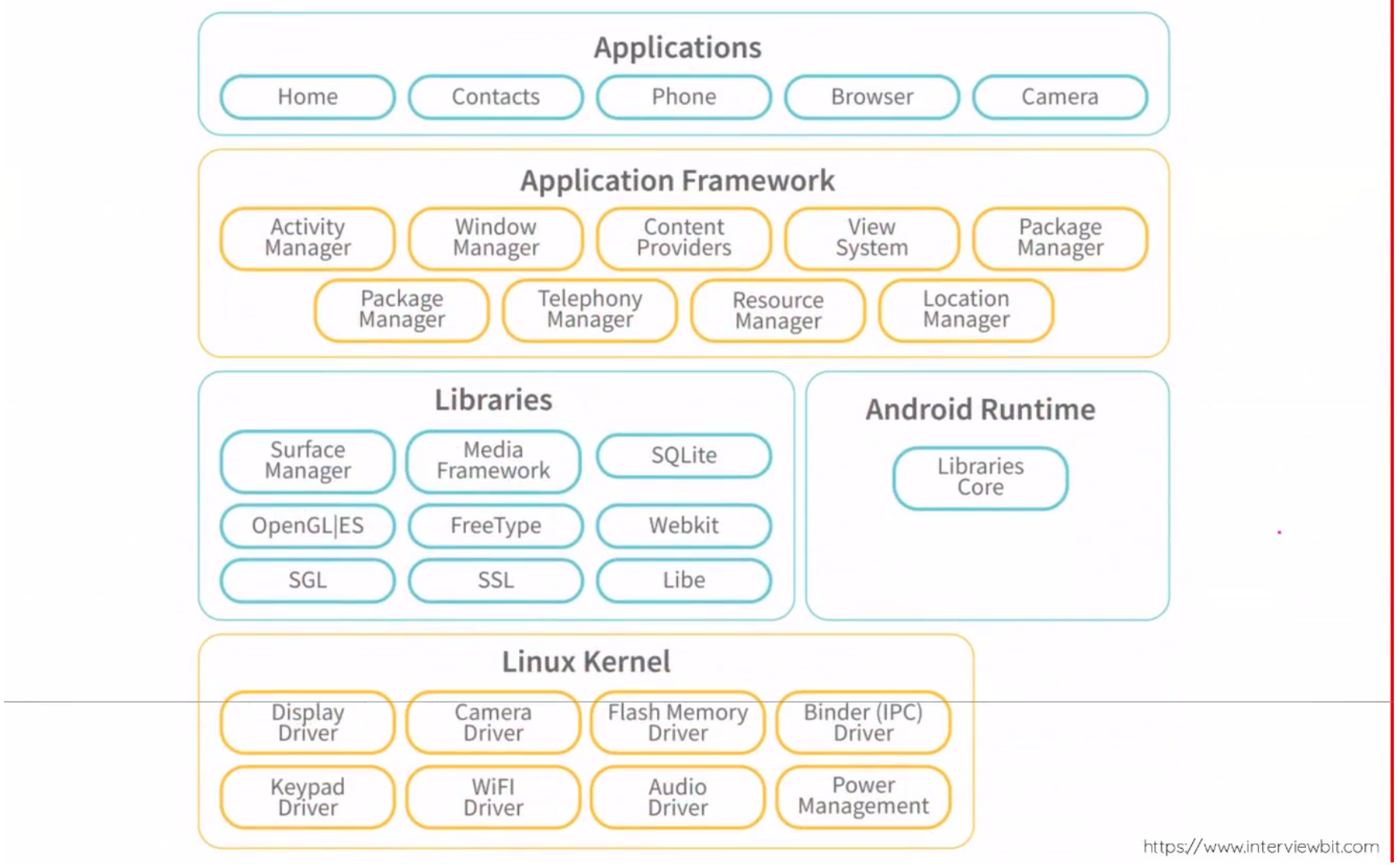
The gradle script is going to run every time and give the app as a package (APK). It cheks for the compatability of the OS of the phone in which the app is installed and then say wether it’s really possibe to install the apk in the phone or not. This is one of the reson why we specify the minimum SDK before we build the app and Gradle reads this info and comapres the SDK of the the phone in which the app is installed and says weather it’s really possible to run the app on the phone.

Android Architecture

The android architecture consits of different layers where each layer communicates with the layer below it and supports the layer above it just like the ISO OSI model in computer networks .



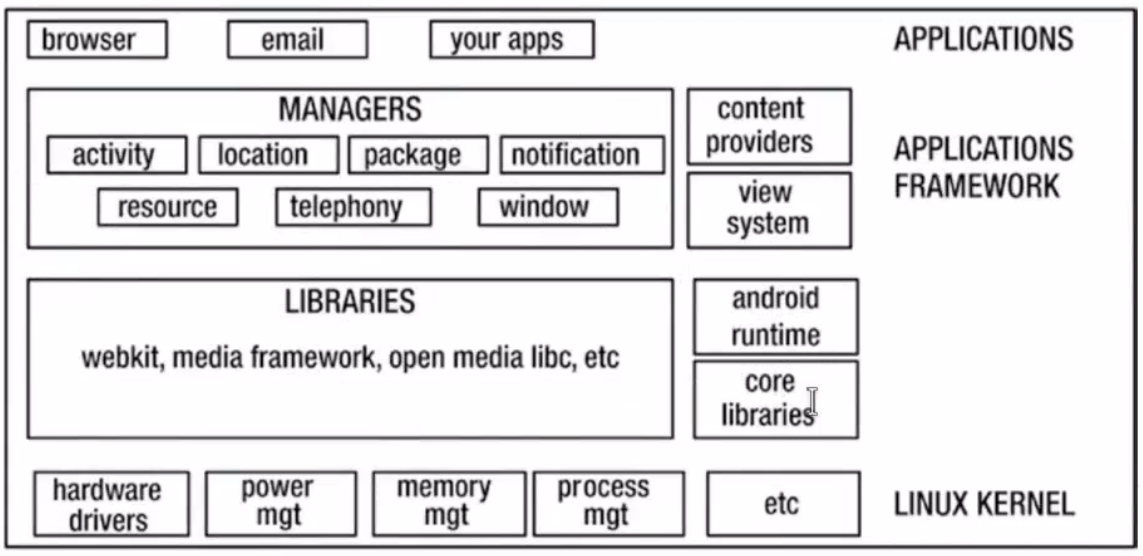
Android Architecture



1. **Application Layer**
    a. Consist of Native Apps, Third Party Apps(Installed) and Progressive Web Apps(PWA) all of which are visible to the user.
    b. The native apps also consist of Home app or the Home Screen which is technically also an app and the browser app which can be both native and third party
2. **Android Framework**
    a. It was also called as the “Java Framework” earlier. This layer serves the APIs required to run the application.
    b. It consists of components like Activity Manager which is responsible for orchestrating different functionalities of an app like for example while watching a video on youtube if we get a call or a text message it notifies us about it.
    c. Telephony Manager manages notifications and calls while the phine is in different modes like airplane made or do not disturb.
3. **Native Libraries**
    a. Consists of core libraries written in native C and C++ which provide Graphics, 2D rendering engines, SQLite database etc. It’s responsible for “under the hood” working of the android app.
    b. Provide native libraries to support the APIs like for example all the contacts are stores in SQLite Database in the phone and the Telephony Manager API fetch the contact from the database and gives it to the contacts app, this is also how one layer supports the other.
    c. One of the major services of library is the SSL(Secure Socket Layer) which handles and secures all the encryption and user information in your device. It doesn’t reveal the user information to the external world(Third party app or websites) untill and unless the user explicitly does so. This service sends data to the Linux Kernel in 0’s and 1’s i.e bits.
4. **Android Runtime(ART)**
    a. It was earlier called as the Dalvik Virtual Machine(DVM) and acts like a virtual machine
    b. It also contains core librares written in Java and Kotlin and which are by default present in the Android Runtime Engine and which support to execute the VMs.

Note: Both the ART and Native Libraries are called as **System Runtime Layer**

1. **HAL via HIDL(Hardware Abstraction Layer via HAL Interface Description Language)**
2. **Linux Kernel**
   Becoz of this kernel we get the flexblity to run the app on the Computer also



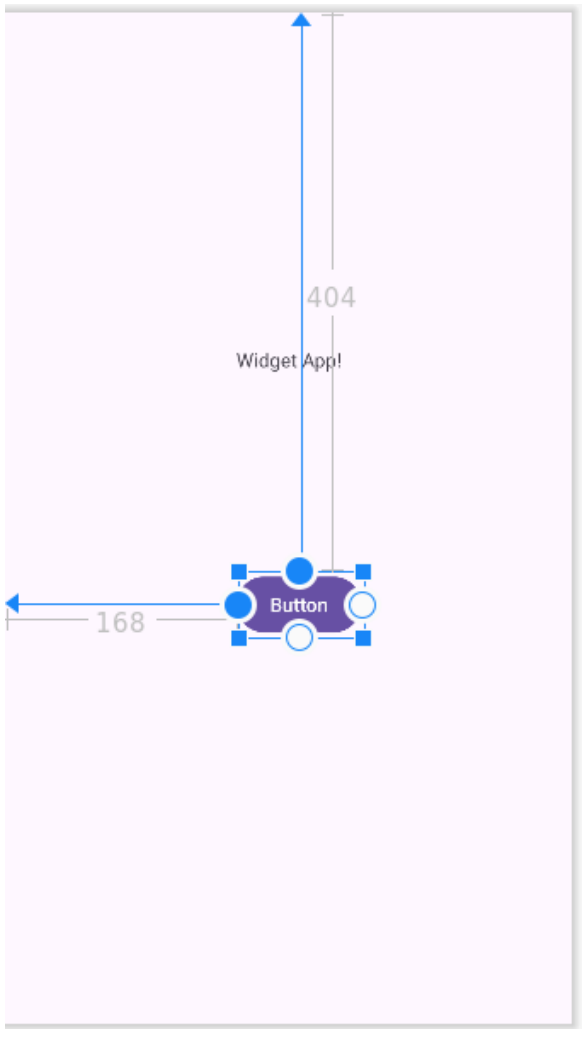
Updated Android Architecture as per textbook

Adding Buttons and Widgets

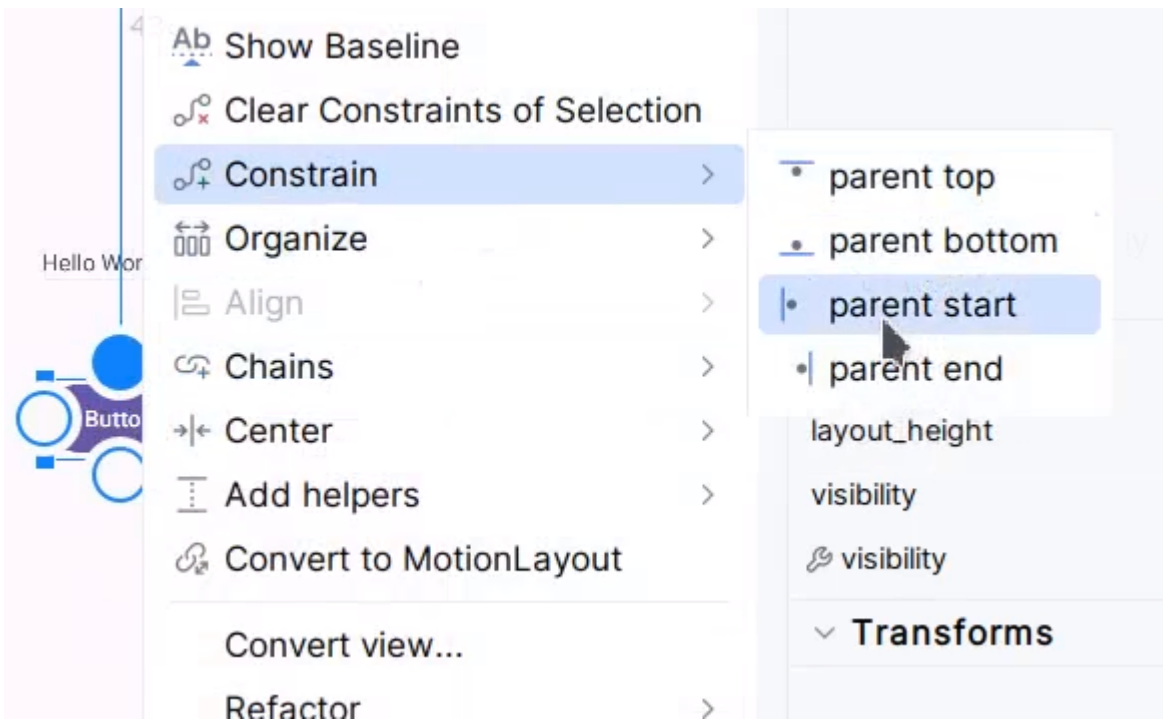
For every component and widget we have to specify the constraint layout or else the component will occupy the corner of the screen i.e (0,0) position.

3 Methods to set the constraint layout

1. Select the concentric circles and drag it to top of the window then drag to adjust the position.



2. Left click on the button and select `Constrain` > `parent stand`



3 steps to connect your front end to back end

1. Specify an Id in the front end(XML)
2. Declare component in the back end(Java Code)
3. Typecast(link) the component between the frontend and the backend

Note

In android studio there is an Id for each component(button, textview etc) and also for the the constraint layout.

Connecting the frontend(XML) with backend(Java)

Basic Structure of Android Program(Backend)

```
package in.shrivatsa.widgetsandlayouts;

import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.CompoundButton;
import android.widget.ImageButton;
import android.widget.Switch;
import android.widget.Toast;

import com.google.android.material.snackbar.Snackbar;

public class MainActivity extends AppCompatActivity {

    // declaration of different components(views)
    Button toastButton, snackbarButton;
    Switch controlSwitch;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

- **public class MainActivity:** This line declares a Java class named `MainActivity`, which serves as the main entry point for the Android application. The class is marked as "public" to allow it to be accessed from other classes.
- **extends AppCompatActivity:** `AppCompatActivity` is a base class provided by the Android framework. It is an extension of the standard `Activity` class and provides additional support for modern Android features, such as the Action Bar. By extending `AppCompatActivity`, your activity inherits functionality that makes it compatible with older Android versions.

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

- **@Override:** This annotation indicates that the method below it is intended to override a method declared in the superclass. In this case, it's overriding the `onCreate` method from the `AppCompatActivity` class.
- **protected void onCreate(Bundle savedInstanceState):** The `onCreate` method is a crucial part of the Android activity lifecycle. It is called when the activity is first created. The `Bundle savedInstanceState` parameter is used to restore the activity's previous state if it was destroyed and recreated by the system. This bundle can contain information about the activity's state, such as user interface data.

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

- **super.onCreate(savedInstanceState):** This line calls the `onCreate` method of the superclass (`AppCompatActivity`). It's essential to call the superclass method first to perform any necessary initialization before executing the custom logic in the `onCreate` method.

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

- **setContentView(R.layout.activity_main):** This method is used to set the user interface for the activity. It defines which layout resource file should be inflated and displayed on the screen. In this case, it inflates the layout defined in the `activity_main.xml` file. The `R.layout.activity_main` is a reference to the unique identifier of the layout resource.

In summary, the code snippet represents the skeleton of an Android activity. The `MainActivity` class extends `AppCompatActivity`, overrides the `onCreate` method, and sets the content view to the layout defined in `activity_main.xml`. This layout will contain the visual components and structure of the user interface for the activity.

Views in Android

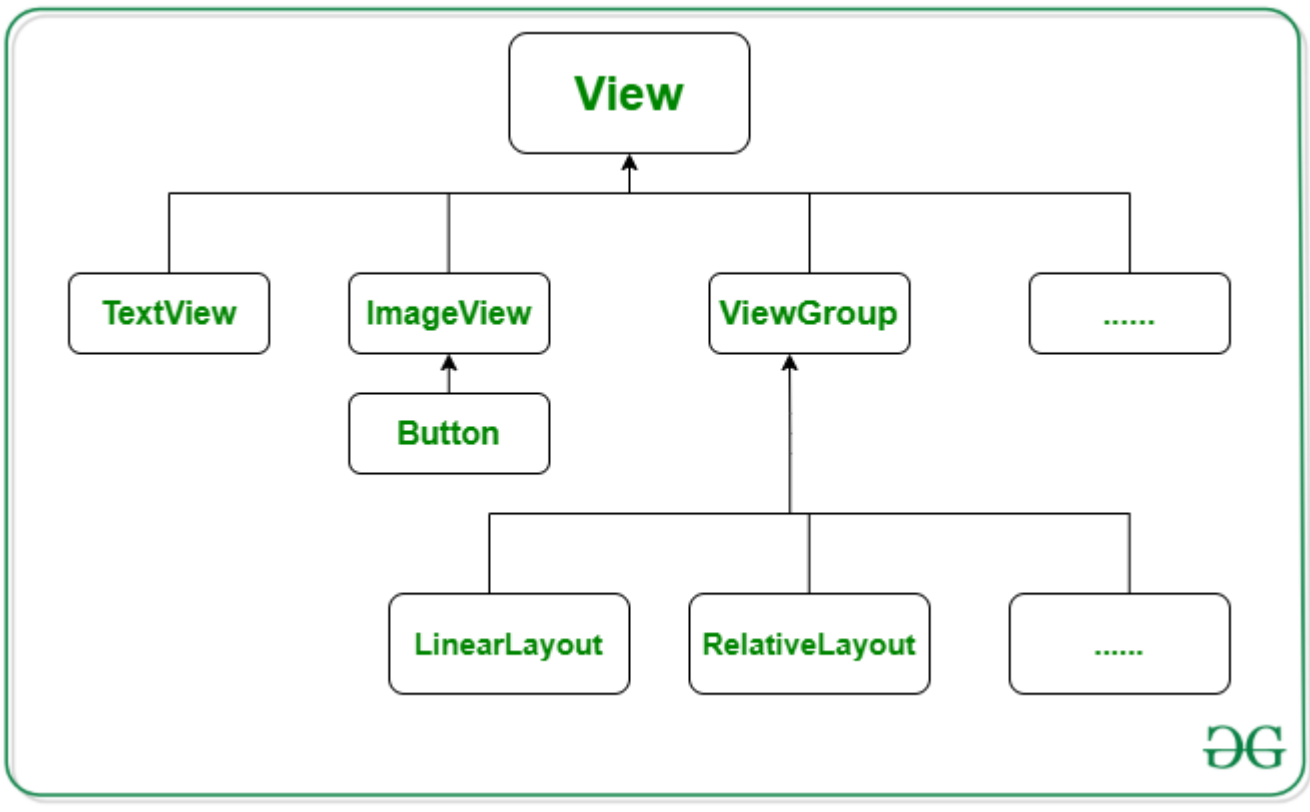
View

View is a basic building block of UI (User Interface) in android. A view is a small rectangular box that responds to user inputs. Eg: EditText, Button, CheckBox, etc.
to import views into the program use the following statement

`Import androidx.view.Views`

ViewGroup is an invisible container of other views (child views) and other **ViewGroup.** Eg: **LinearLayout** is a **ViewGroup** that can contain other views in it. **ViewGroup** is a special kind of view that is extended from View as its base class. ViewGroup is the base class for layouts. As the name states View is singular and the group of Views is the **ViewGroup.** In simple terms, a view is a user interface feature that we interact with when we use an app, such as a button, editing text and images, and so on. **Android.view** has a child class called View. Observe While the View group is the container that houses all of these views as well as many other **ViewGroup** such as linear or Frame Layout. For example, if we design and use the **LinearLayout** as the root feature, our main layout would be the LinearLayout. Within it, we can add another view category (i.e. another LinearLayout) and several other views such as buttons or **TextViews**.

It is the superclass for all the GUI components in android. For example, the EditText class is used to accept the input from users in android apps, which is a subclass of View, and another example of the TextView class which is used to display text labels in Android apps is also a subclass of View.



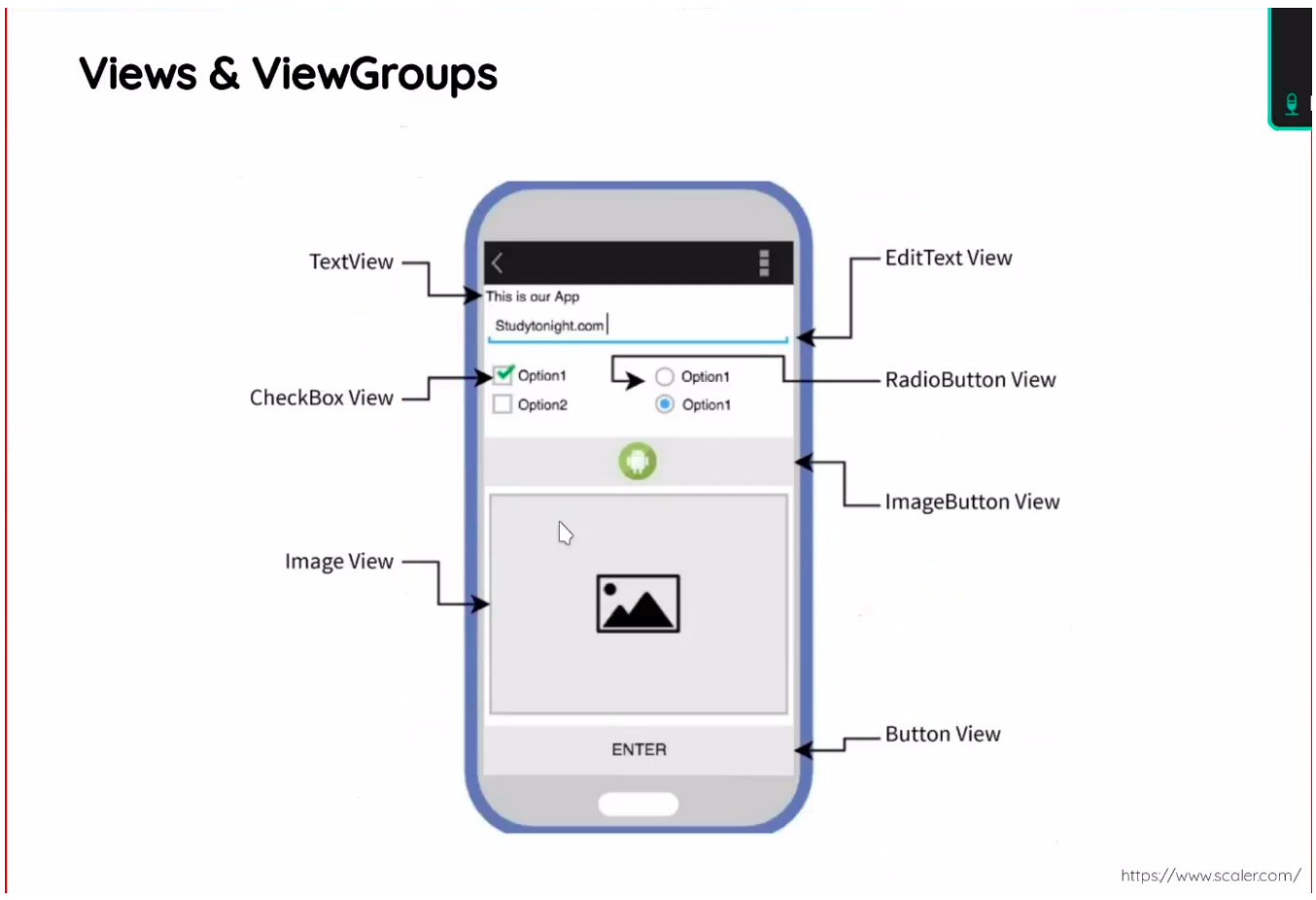
View Vs ViewGroup

View in android programming is something that can be seen. Views enables user to perform an event on them like click, scroll etc and they also accept to the user interaction
for example: Button, Image, TextView, Clendar etc

ViewGroup acts like a container and cannot be seen
for example: Linear Layout, Constriant Layout, Frame Layout, Table

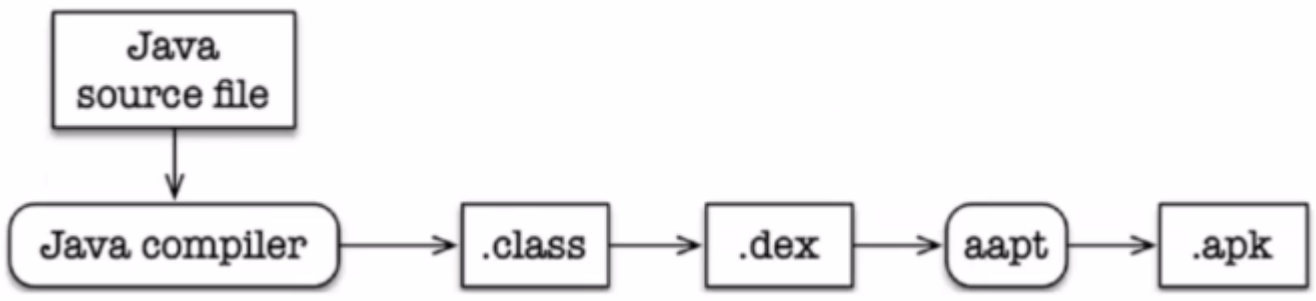
View objects are often called widgets and can be one of many subclasses, such as Button or TextView. The ViewGroup objects are usually called layouts and can be one of many types that provide a different layout structure, such as LinearLayout or ConstraintLayout.

Note: We can place Views and ViewGroup inside of a ViewGroup but the vice versa is not possible



Illustrstion of Different Views and ViewGroups in Android

Android Compilation Process



Compilation process of an Android App

1. The android activity file written in Java is compiled by the Java compiler which looks for errors if any and converts it to `.class` file
2. The compliled class file is converted to a file with extension `.dex` which is a Dalvik Executable file and is read by the Android Runtime Environment(ART which in earlier days was called Dalvik Virtual Machine)
3. Now the system genrates a file with an extension `.aapt` which stands for Android Assests Packaging Tool responsible for packaging all the things under the `res` folder including the drawables, the XML file and the icons.
4. At last all of the files generated in above steps are packaged into a single file called Android Application Package file with extension `.apk`

Thread programming in Android

In Android, there are two types of threads: **the main thread** and **worker threads**. The main thread is the thread that **runs the user interface (UI)** and **handles all of the user interactions**. Worker threads, on the other hand, are used to **perform background tasks** that may take some time to complete. By using worker threads, the application can continue to respond to user interactions on the main thread without getting blocked.

`runUiThread` method allows you to execute code on the main thread from a worker thread

```
new Thread(new Runnable() {
    public void run() {
        // Perform long-running task here, such as downloading a file
        // ...

        // Once the task is complete, update the UI
        runOnUiThread(new Runnable() {
            public void run() {
                // Update UI here, such as displaying the downloaded file
                // ...
            }
        });
    }
}).start();
```

Thread Handler

A Handler allows you to send and process `Message` and Runnable objects associated with a thread's `MessageQueue`

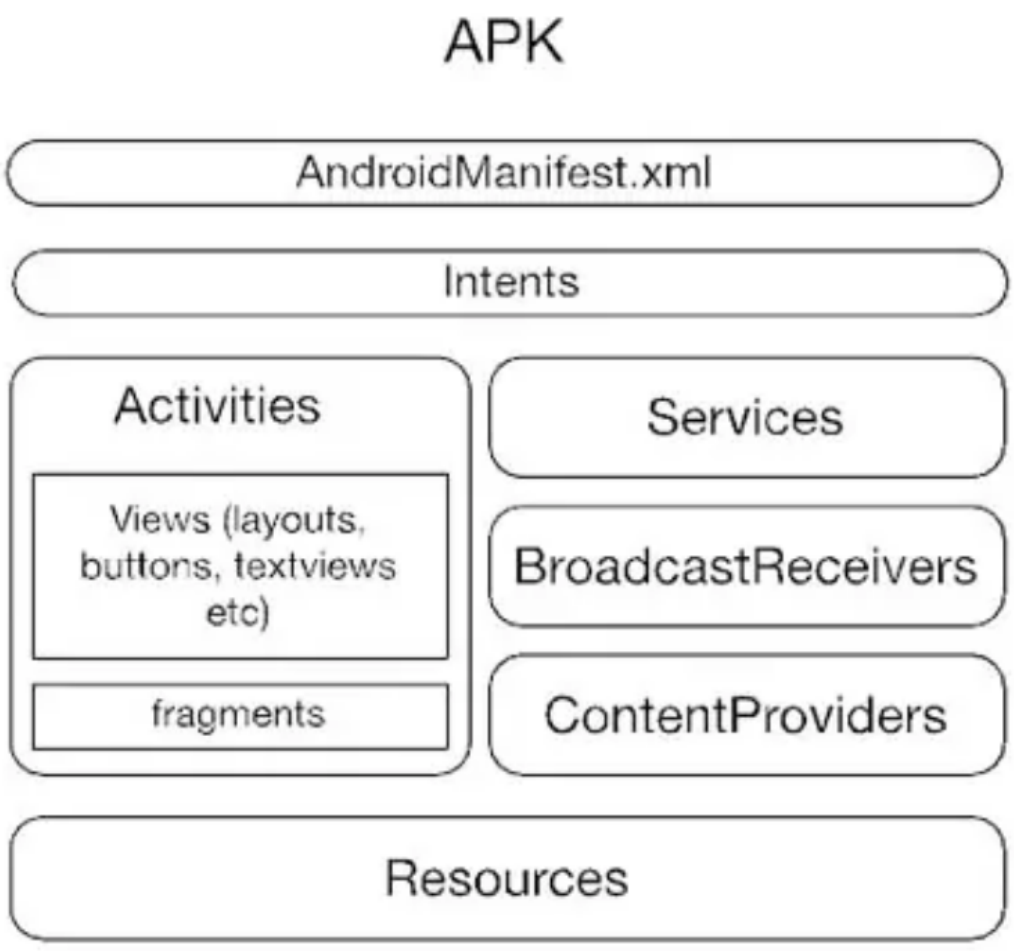
There are two main uses for a Handler:

(1) To schedule messages and runnables to be executed at some point in the future;

(2) To enqueue an action to be performed on a different thread than your own.

Scheduling messages is accomplished with the `post(Runnable)`, `postAtTime(java.lang.Runnable, long)`, `postDelayed(Runnable,Object,long)`, `sendEmptyMessage(int)`, `sendMessage(Message)`, `sendMessageAtTime(Message, long)`, and `sendMessageDelayed(Message, long)` methods.

Components of an Android App and Activity Life Cycle



Logical Representation of an Android App

Intents

Intent is a communication which will navigate from one activity to the other. We can also call it as a seamless(neat) way of action since there is a swift drift from one activty to the other

Fragment

A Fragment is a small piece of code that can be resued again and again. Usually we use fragments to execute repetetive tasks instead of hardcoding it everytime

Services

A service enables the user to get something and pass something to it. Some examples of services are Flashlight, Location, Bluetooth, Camera, WIFI,

Broadcast Recievers

One activity triggering the other activity. For example when battery drops below 20% the power saving mode tells us to plug in the phone to charge.

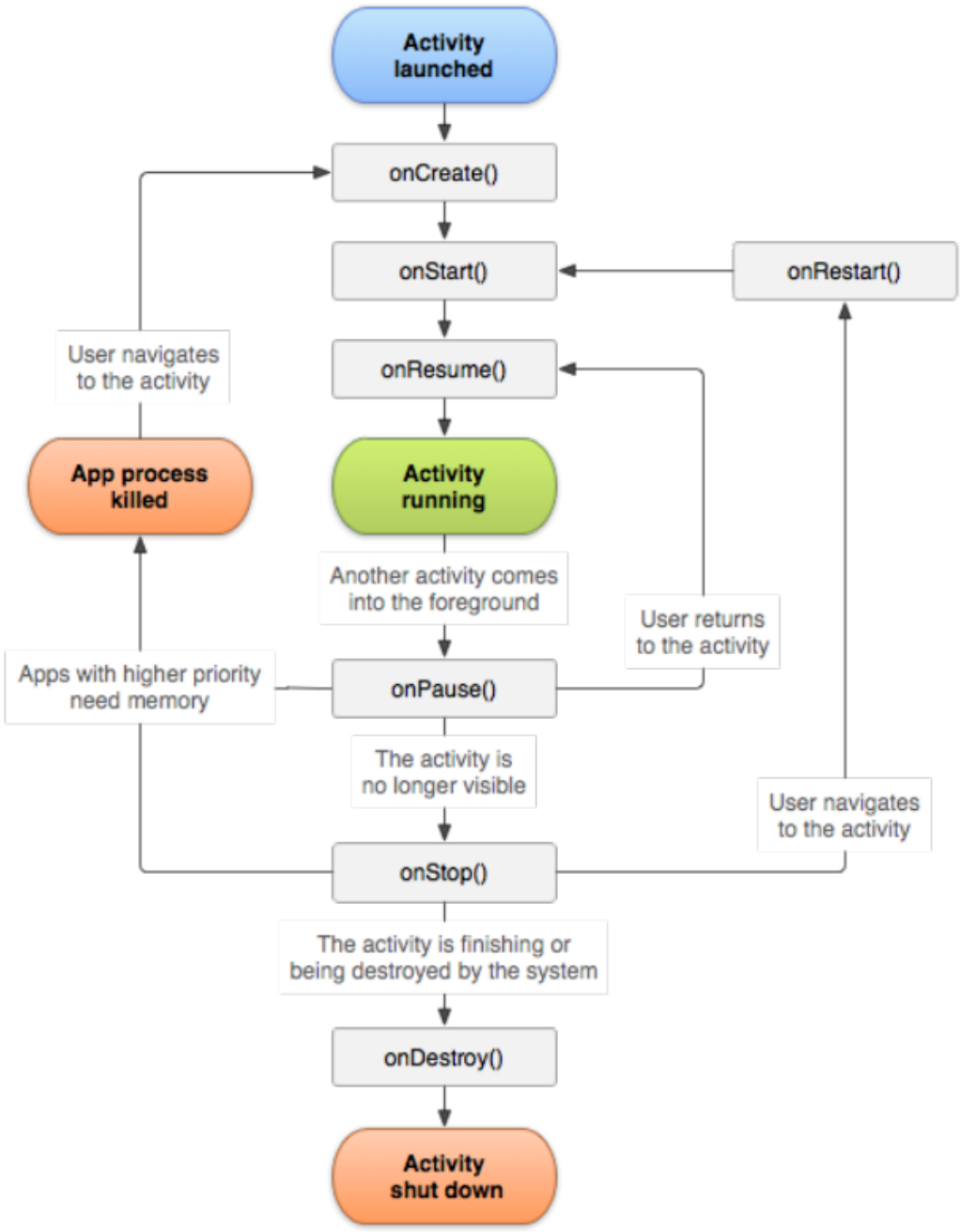
Resources

Involves the phone’s resources like the hardware, drivers, APIs and Database which stores the contact info present inside the phone

Question may be asked as

1. Give the composition of the android project
2. What makes up an android app
3. Logical representatio of an app

Android Application Life Cycle



An android app can be in one of the four states

1. Activity Launched
2. App Process Killed
3. Activity Running
4. Activity Shut Down

The above four sates can be achived by 6 activity callbacks
to uderstand this let’s take an example of an app like netflix or Amazon Prime

1. `onCreate()`
   Is called when an activity is lauched. Like for example when we launch an app from the home screen of the phone
2. `onStart()`
   When we enter the app and give provide our login credentials(username and password) that is when onStart is called. It’s when the app starts to run
3. `onResume()`
   When we exit(not close) from the app temporarily and switch to another app and come back, onResume is called. Example: While watching a movie on Netflix we get a notification from watsapp, we go check watsapp and come back to Netflix to continue watching movie. Once this state is called the app will be in the running state.
4. `onPause()`
   This callback is called when an app is running in the foreground and another activity comes into the foreground. Then the previous activity is put to hold state temporaily. Example : we are watching a video on youtube and a call pops up suddenly in front.
5. `onStop()`
   When we completely close the app onStop is called. Example: You have finished watching the movie and have closed netflix. Even though we have closed this app we may still recieve notifications from it since the app is still installed in our phone.

6. `onDestroy()`
   Is when you completely delete the app.

Apart from these six callbacks we also have `onRestart()` which is not considered as a callback since it always maps to `onStart()`