Mobile Application Development

Unit-1

1). What is android? Explain its features?

Android is an operating system and programming platform developed by Google for mobile phones and other mobile devices, such as tablets. It can run on many different devices from many different manufacturers. Android includes a software development kit (SDK) that helps you write original code and assemble software modules to create apps for Android users.

Features:

1. Open-Source Flexibility

At the core of Android's appeal lies its open-source nature. Unlike its counterparts, Android welcomes collaboration and modification by developers, leading to a dynamic ecosystem that constantly evolves. This flexibility empowers **developers to create customized solutions and users to personalize their devices** to match their preferences.

2. Seamless App Integration

One of Android’s standout features is its seamless integration with a vast array of applications. With the **Google Play Store serving as a treasure trove** of options, users can choose from an extensive selection of apps to enhance their experience. From productivity tools to entertainment platforms, Android ensures that users can tailor their devices to suit their lifestyles seamlessly.

3. Intuitive User Interface

Android’s user interface (UI) has undergone significant refinements over the years, culminating in an intuitive and **user-friendly experience**. The UI design prioritizes accessibility, ensuring that even novice users can navigate the system with ease. With a focus on simplicity and clarity, Android’s UI sets a benchmark for usability across different age groups and demographics.

4. Regular System Updates

The Android operating system is dedicated to **continuous improvement through regular system updates**. These updates not only introduce new features but also **enhance security protocols** and address any potential vulnerabilities. This commitment to keeping devices up-to-date ensures a secure and optimized experience for all users.

5. Google Assistant – Your AI Companion

Among the standout features is the integration of **Google Assistant, an AI-powered virtual companion** designed to streamline tasks and provide real-time information. With **voice commands, users can set reminders, send messages, play music, and even control smart home devices.** The ever-evolving capabilities of Google Assistant make it a valuable addition to the Android ecosystem.

6. Customization Galore

Android grants users the **freedom to customize their devices** extensively. From changing the launcher and theme to installing third-party widgets, the level of personalization is unparalleled. This feature not only reflects individuality but also enhances productivity by tailoring the device's functionality to specific needs.

7. Multi-User Support

Android recognizes the diverse needs of its users, offering multi-user support on devices such as tablets. This feature is particularly **beneficial for families**, as **each member can have a personalized profile, complete with apps and settings**. It ensures a shared device can cater to different preferences without compromising on privacy.

8. Battery Optimization

Efficient battery management is a hallmark of the Android OS. With features like **Adaptive Battery**, the system learns usage patterns to

optimize power allocation to different apps. This translates to **extended battery life**, ensuring that users can stay connected and engaged throughout their day without worrying about running out of power.

9. Wide Range of Hardware

Android's compatibility with a wide range of hardware is a significant advantage. Whether it's a budget-friendly device or a flagship model, the **Android OS seamlessly adapts to varying specifications**. This inclusivity enables users to choose devices that align with their requirements and budget without sacrificing the operating system's quality.

10. Growing Augmented Reality (AR) Capabilities

As technology advances, Android remains at the forefront of **integrating augmented reality into the user experience**. With ARCore, Google's platform for building augmented reality experiences, Android devices can provide immersive and interactive encounters that **bridge the gap between the digital and physical worlds.**

2).Create an application that takes the name from a textbox and shows Hello Message along with the name entered in textbox,when the user clicks the ok button?

activity.xml

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

```
<Button
    android:id="@+id/mybutton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="OK" />

<EditText
    android:id="@+id/editTextName"
    android:hint="Enter Name"
    android:inputType="textMultiLine"
    android:layout_marginTop="10dp"
    android:layout_width="match_parent"
    android:layout_height="40dp"/>
<TextView
    android:id="@+id/txtView"
    android:layout_width="wrap_content"
```

```
        android:layout_height="40dp"
        android:text=""
        android:textSize="20dp">
    </TextView>
</LinearLayout>
```

MainActivity.Java

```
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
```

```
//printing a message on the same activity
public class MainActivity extends Activity {
   /** Called when the activity is first created.
*/
   @Override
   public void onCreate(Bundle
savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);

       EditText editTextName =
findViewById(R.id.editTextName);
       final Button button1 =
(Button)findViewById(R.id.mybutton);
       final TextView tv=(TextView)
findViewById(R.id.txtView);
```

```
        //Implement listener for your button so
that when you click the
        //button, android will listen to it.

        button1.setOnClickListener(new
View.OnClickListener() {
            public void onClick(View v) {
                // Perform action on click
               tv.setText("Hello"+"
"+editTextName.getText());

            }         });
    }
}
```

3).Define Activity? Explain Activity life cycle with a neat sketch?

- An activity is **one screen of an app**.

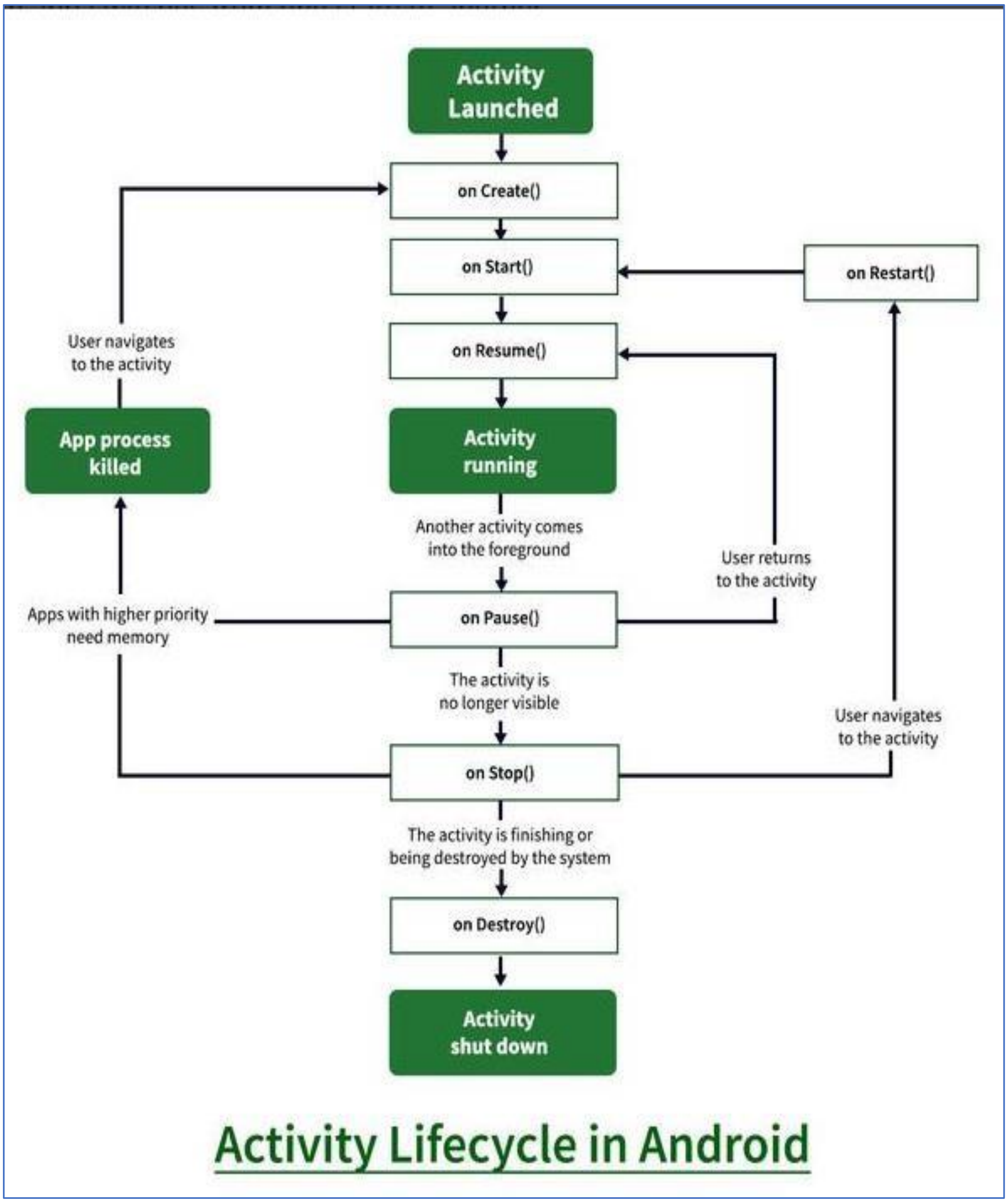
- In that way the activity is **very similar to a window** in the Windows operating system or like **a web page of a website**.
- The most **specific building block of the user interface** is the activity.
- An Android app contains activities, meaning **one or more screens**.
- Examples: **Login screen, sign up screen, and home screen**.

Activity is nothing but **a java class in Android which has some pre-defined functions** which are triggered at different App states **Every application which has UI must inherit it to create a window.**

Whenever we open any application, one of its activity opens, its called the Launcher Activity.

When you create a new project, android studio creates an activity by name "**MainActivity.java**" and its XML file my name "**activity_main.xml**". It is the activity that opens up when an app is opened

Activity Life Cycle:


Activity Launched
on Create()
on Start()
on Restart()
on Resume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
on Pause()
The activity is no longer visible
User navigates to the activity
on Stop()
The activity is finishing or being destroyed by the system
on Destroy()
Activity shut down
Activity Lifecycle in Android

1. onCreate( )

This callback is fired when the system first creates the activity. In onCreate method, all such operations are performed which should be done only once for the entire life of the activity.

This method has a parameter *savedInstanceState* which is a bundle object. It contains activity’s previous saved state. If the activity has never existed before, the value of Bundle object is null.

2. onStart( )

When the activity enters the Started state, the system invokes this callback. The *onStart( )* call makes the activity visible to the user, as the app prepares for the activity to enter the foreground and become interactive. This method can be used to initialize the code that maintains the user interface. After *onStop( )* , if activity restarts then *onStart( )* is called.

onCreate ( ) function is called only once but onStart( ) can be called multiple times , when activity enters the started state.

3. onResume( )

The *onStart( )* method completes very quickly and, as with the Created state, the activity does not stay resident in the Started state. Once this callback finishes, the activity enters the *Resumed* state, and the system invokes the

onResume( ) method.

This is the state in which the app interacts with the user. The app stays in this state until something happens to take focus away from the app. Such an event might be, for instance,

receiving a phone call,

the user's navigating to another activity, or the device screen’s turning off.

When such an event occurs, the activity enters the *Paused* state, and the system invokes the *onPause( )* callback.

4. onPause( )

The system calls this method as the first indication that the user is leaving the activity (though it does not always mean the activity is being destroyed); it indicates that the activity is no longer in the foreground (though it may still be visible if the user is in multi- window mode).

In most case onPause() method called by Android OS when user press Home button (Center Button on Device) to make hide. Activity is not visible to user and goes in background when onPause() method is executed

When the activity moves to the paused state, any lifecycle-aware component tied to the activity’s lifecycle will receive the *onPause( )* event. This is where the lifecycle components can stop any functionality that does not need to run while the component is not in the foreground

onPause( ) method can be used to release system resources, handles to sensors (like GPS), or any resources that affect battery life while your activity is paused and the user does not need them.

5. onStop( )

When your activity is no longer visible to the user, it has entered the *Stopped* state, and the system invokes the onStop( ) callback. Any activity gets stopped in case some other activity takes place of it. For example, if a user was on screen 1 and click on some button and moves to screen 2. In this case Activity displaying content for screen 1 will be stopped.

The system may also call onStop( ) when the activity has finished running, and is about to be terminated.

In the *onStop( )* method, the app should release or adjust resources that are not needed while the app is not visible to the user. It should be used to perform relatively CPU-intensive shutdown operations. So Activity will be in stopped state when hidden or replaced by other activities that have been launched or switched by user

6. onRestart( )

onStart( ) will always be called whenever you enter your Activity just after *onCreate( )* but the *onRestart( )* will only be called before *onStart( )* when your Activity comes from being stopped (passing from *onStop( )*) back to the vision.

7. onDestroy( )

onDestroy( ) is called before the activity is destroyed. The system invokes this callback either because:

1. if user pressed the back navigation button then activity will be destroyed after completing the lifecycle of pause and stop.

2. In case if user press the home button and app moves to background. User is not using it no more and it's being shown in recent apps list. So in this case if system required resources need to use somewhere else then OS can destroy the Activity.

After the Activity is destroyed if user again click the app icon, in this case activity will be recreated and follow the same lifecycle again.

4)Explain android development framework with block diagram?

1. Native Development Frameworks: These frameworks are platform-specific, allowing developers to build apps for a single platform using the platform’s native language and tools. Examples include:

iOS Development (Swift/Objective-C): For Apple devices, Swift and Objective-C are used for native iOS app development. Android Development (Java/Kotlin): For Android, Java and Kotlin are the primary languages for building native apps.

2. Cross-Platform Development Frameworks: These frameworks enable developers to create apps for multiple platforms using a single codebase. Examples include:

React Native: Developed by Facebook, React Native allows developers to build mobile apps for iOS and Android using JavaScript and React. It’s used by companies like Facebook, Instagram, and Airbnb.

Flutter: Created by Google, Flutter is another cross-platform framework using Dart, and it’s known for building beautiful, natively compiled applications. Apps like Alibaba and Google Ads have been developed using Flutter.

3. Hybrid Mobile App Development Frameworks:

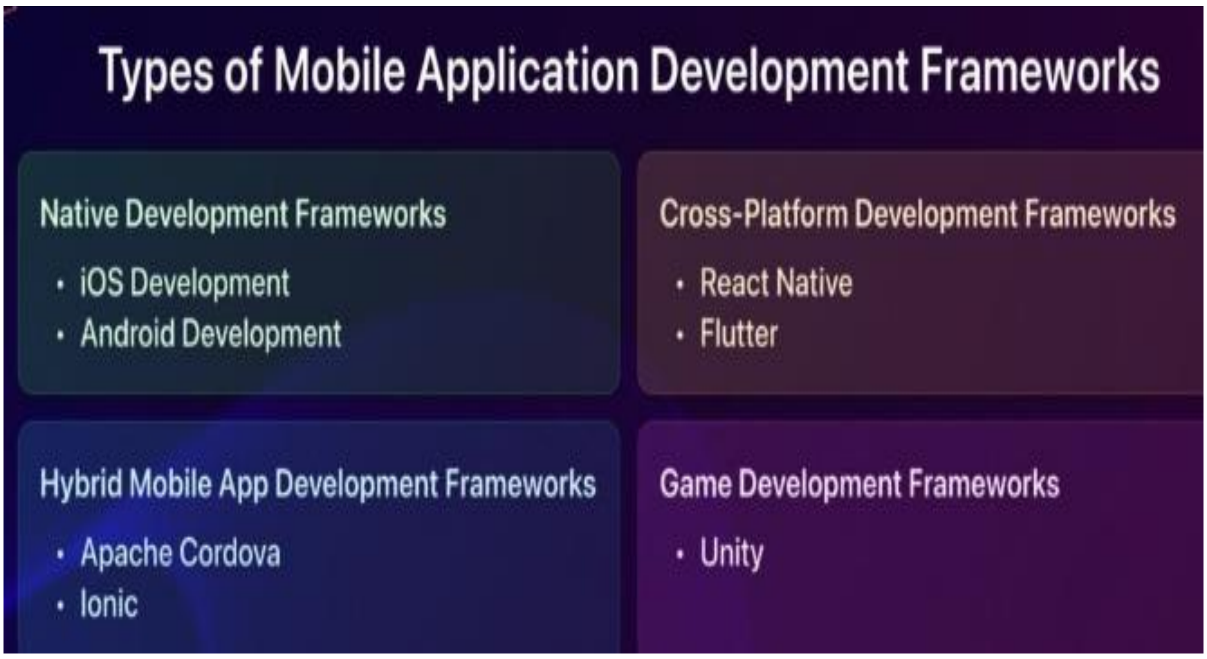
Hybrid mobile app development frameworks combine native and web technologies, usually relying on web views to render content within a native app. Examples include:

Apache Cordova (PhoneGap): It allows developers to use HTML, CSS, and JavaScript to build cross-platform apps. Adobe PhoneGap is a popular variation.

Ionic: Built on top of Cordova, Ionic provides a UI framework and tools for building hybrid apps.

4. Game Development Frameworks: These frameworks are specialized for creating mobile games. Examples include:

Unity: It's a versatile game engine that supports mobile platforms like iOS and Android. Games like "Pokémon GO" and "Super Mario Run" were built using Unity.



Types of Mobile Application Development Frameworks

Native Development Frameworks
- iOS Development
- Android Development

Cross-Platform Development Frameworks
- React Native
- Flutter

Hybrid Mobile App Development Frameworks
- Apache Cordova
- Ionic

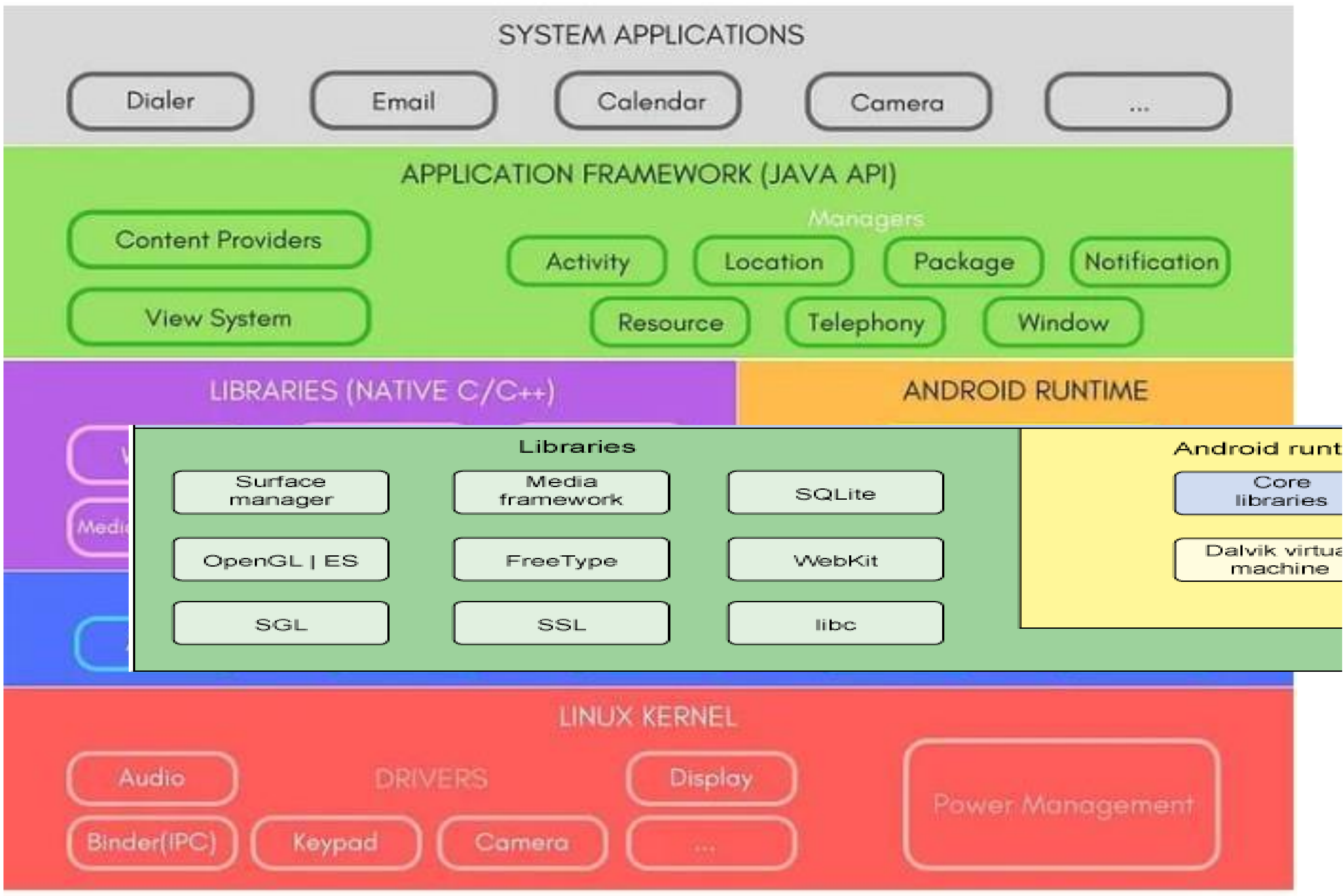
Game Development Frameworks
- Unity

5.Define Android Development framework with block diagram?

Android Architecture

The Android framework is a set of software components that provide the foundation for building Android applications.

It is divided into five layers:



1. <u>Linux kernel</u>

Bottom layer of android operating system is Linux kernel. Android is built on top of Linux 2.6 Kernel and few architectural changes made by Google. The Linux kernel provides essential services such as hardware abstraction, memory management, process management, and device driver support. Android builds upon the robust and secure foundation of Linux, making it a stable and reliable platform.

Key functions of the Linux kernel layer are:

- Hardware Abstraction: The kernel abstracts hardware details, allowing Android to work across a wide range of hardware configurations.
- Memory Management: It manages system memory, ensuring efficient allocation and deallocation of resources.

- Process Management: The kernel oversees the execution of processes and handles multitasking.
- Device Drivers: These drivers enable communication between hardware components and the operating system.

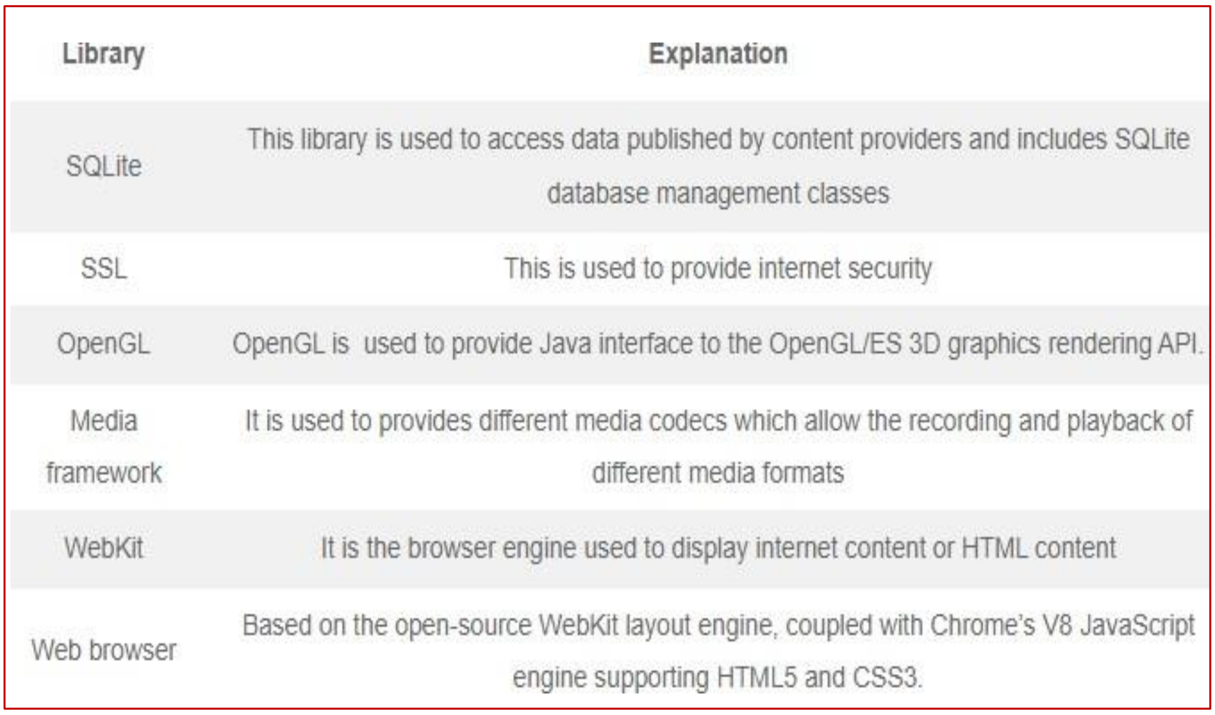
2. **Hardware Abstraction Layer (HAL)**

The Hardware Abstraction Layer (HAL)acts as a bridge between the hardware- specific device drivers and the higher-level Android framework. It ensures that the upper layers of the Android stack remain hardware- agnostic, allowing Android to run on various hardware platforms seamlessly.

The HAL includes libraries and software components that communicate with hardware devices like cameras, sensors, and audio chips. This layer enables Android to support a wide array of devices with different hardware configurations.

3. Libraries and Android Runtime:

Above the HAL, Android uses a combination of libraries and the Android Runtime (ART). It provides the different libraries useful for well functioning of android operating system. The Android component is built using native codes and require native libraries, which are written in C/C++ and most of the libraries are open source libraries. Some of the native libraries are



Library	Explanation
SQLite	This library is used to access data published by content providers and includes SQLite database management classes
SSL	This is used to provide internet security
OpenGL	OpenGL is used to provide Java interface to the OpenGL/ES 3D graphics rendering API.
Media framework	It is used to provides different media codecs which allow the recording and playback of different media formats
WebKit	It is the browser engine used to display internet content or HTML content
Web browser	Based on the open-source WebKit layout engine, coupled with Chrome's V8 JavaScript engine supporting HTML5 and CSS3.

Android Runtime

It provides most important part of android called Dalvik Virtual Machine (DVM). Dalvik Virtual Machine is similar to Java Virtual Machine (JVM) but only difference is that it is designed and optimized for Android. DVM uses core functions of Linux such as memory management and multi threading and enables each android app to run its own process. It basically converts .java file into .Dex format.

4. Android Framework

This layer provides the basic building blocks for Android applications The Android Framework is a collection of Java classes and libraries that provide the core functionalities and building blocks for Android applications. It includes classes for managing the user interface, accessing hardware resources, and communicating with other applications

It consists of various modules:

- Activity Manager: Manages the lifecycle of Android applications and their

components, such as activities and
services.

- Content Providers: Allow apps to share data and access it securely. They act as a bridge between applications
and databases.

- Notification Manager: Handles notifications generated by apps and system events.

- Location Manager: Enables apps to access location-based services.

- Connectivity Manager: Manages network connections, including Wi-Fi and mobile data.

The Android Framework also includes the Application Framework, which allows developers to build custom applications using pre-built components.

5. Applications

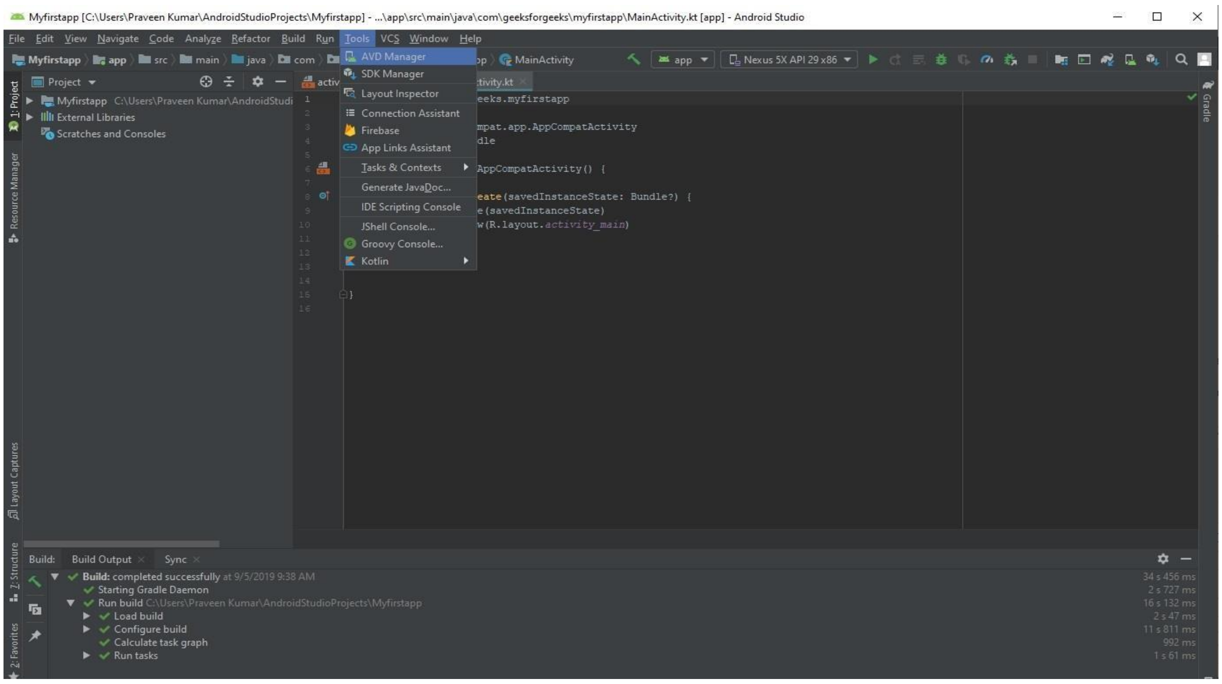
At the top of the Android architecture stack are the applications themselves. These can be pre-installed system apps, such as the phone dialer and messaging app, or third-party apps installed by users from the Google Play Store or other sources.

Android apps are written in Java or Kotlin and interact with the Android Framework through APIs. They leverage the framework's components and services to provide various functionalities to users.

6). How to create AVD`s? Explain the various types of android applications?

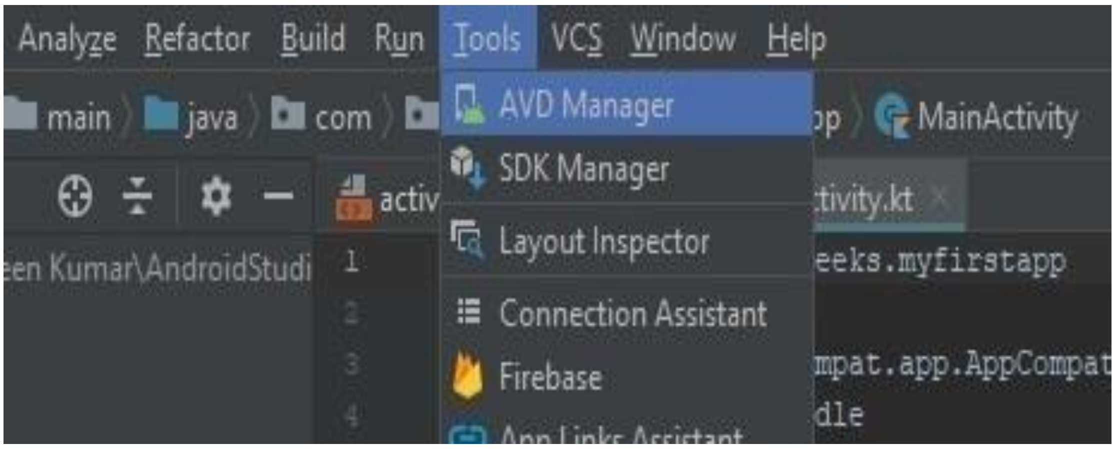
steps to create Android Virtual Device:

Step 1: Go to **Tools** > **AVD Manager**.

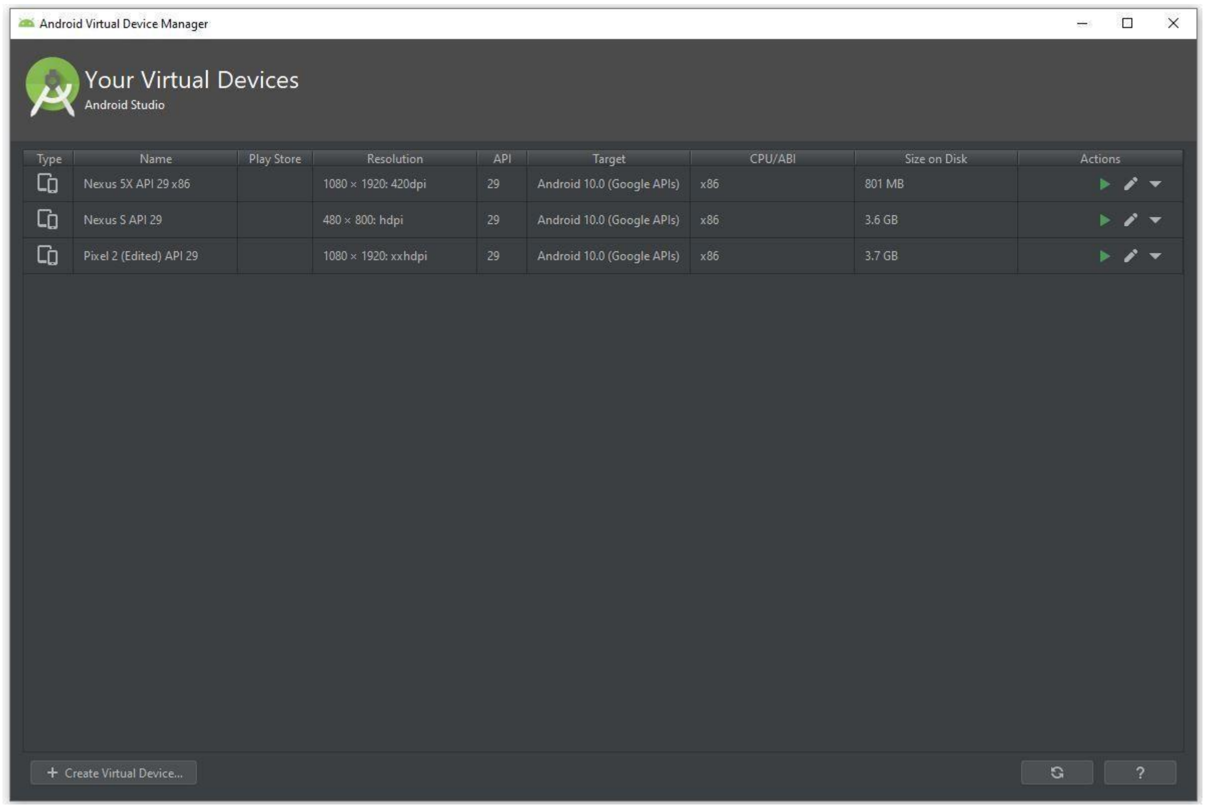

Myfirstapp [C:\Users\Praveen Kumar\AndroidStudioProjects\Myfirstapp] - ...\app\src\main\java\com\geeksforgeeks\myfirstapp\MainActivity.kt [app] - Android Studio
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Myfirstapp app src main java com MainActivity
app Nexus 5X API 29 x86
Project
Myfirstapp C:\Users\Praveen Kumar\AndroidStudi
External Libraries
Scratches and Consoles
AVD Manager
SDK Manager
Layout Inspector
Connection Assistant
Firebase
App Links Assistant
Tasks & Contexts
Generate JavaDoc...
IDE Scripting Console
JShell Console...
Groovy Console...
Kotlin

```
eeks.myfirstapp

mpat.app.AppCompatActivity
dle

AppCompatActivity() {

eate(savedInstanceState: Bundle?) {
e(savedInstanceState)
w(R.layout.activity_main)




}
```

```
Build: completed successfully at 9/5/2019 9:38 AM                          34 s 456 ms
  Starting Gradle Daemon                                                    2 s 727 ms
  Run build C:\Users\Praveen Kumar\AndroidStudioProjects\Myfirstapp        16 s 132 ms
    Load build                                                              2 s 47 ms
    Configure build                                                        11 s 811 ms
    Calculate task graph                                                      992 ms
    Run tasks                                                               1 s 61 ms
```



Step 2: Now click on **Create Virtual Device**.

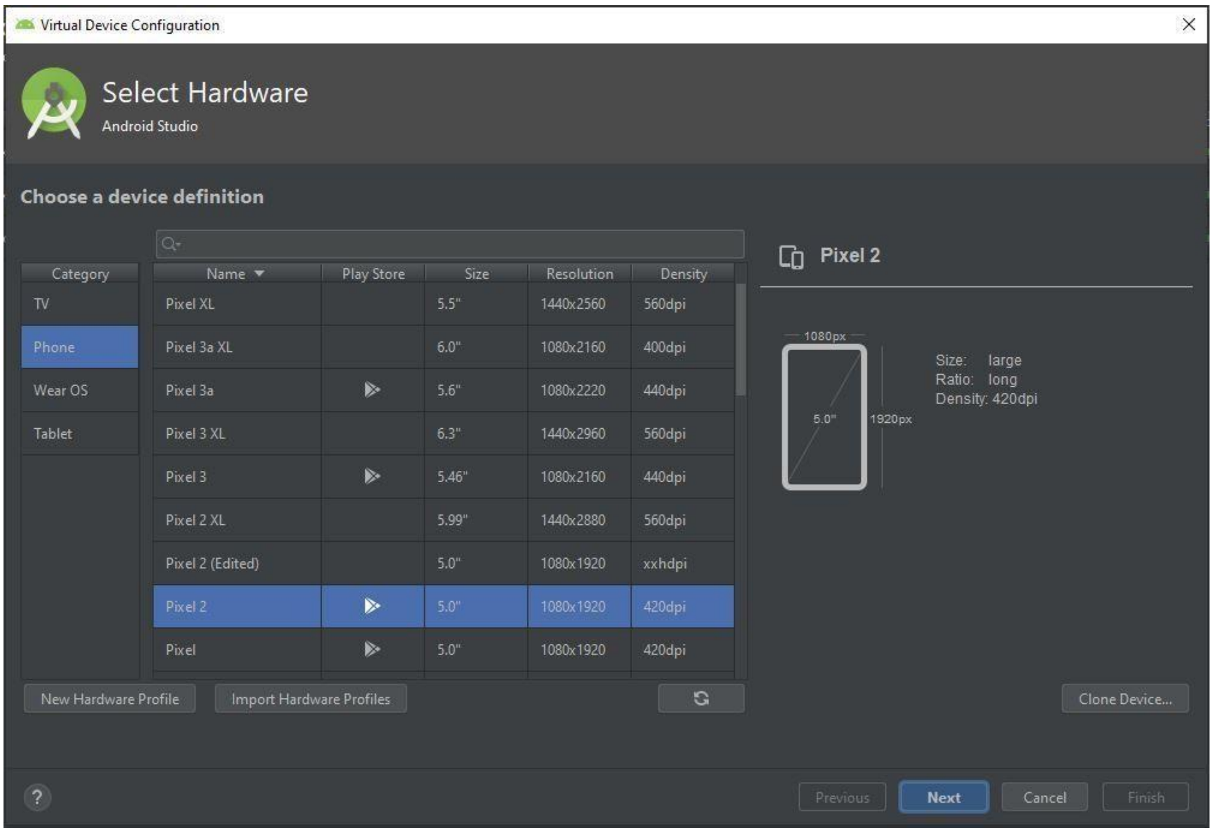


Android Virtual Device Manager

Your Virtual Devices
Android Studio

Type	Name	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	Nexus 5X API 29 x86		1080 × 1920: 420dpi	29	Android 10.0 (Google APIs)	x86	801 MB	
	Nexus S API 29		480 × 800: hdpi	29	Android 10.0 (Google APIs)	x86	3.6 GB	
	Pixel 2 (Edited) API 29		1080 × 1920: xxhdpi	29	Android 10.0 (Google APIs)	x86	3.7 GB	

Step 3: A pop-up window will be there and here we select the category Phone because we are creating android app for mobile and select the model of mobile phone we want to install.



Choose a device definition

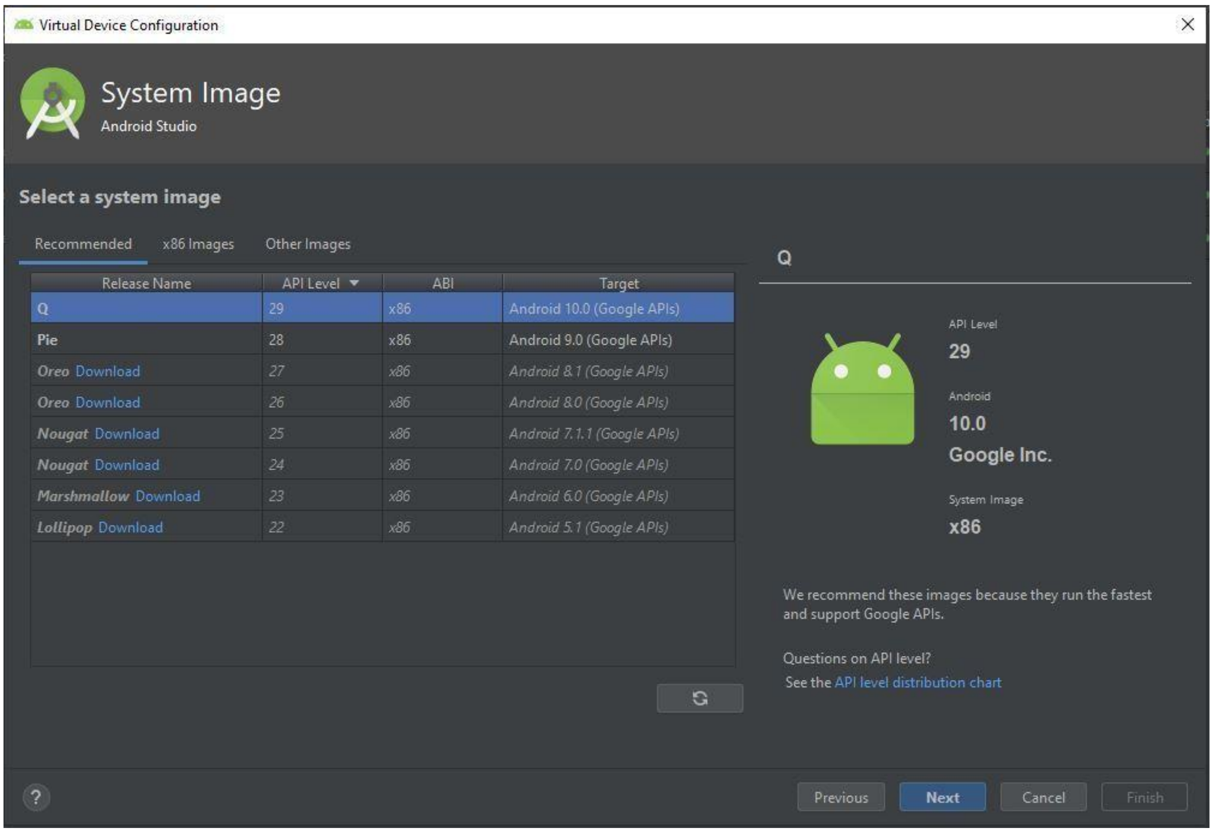
Category
TV
Phone
Wear OS
Tablet

Name	Play Store	Size	Resolution	Density
Pixel XL		5.5"	1440x2560	560dpi
Pixel 3a XL		6.0"	1080x2160	400dpi
Pixel 3a	▶	5.6"	1080x2220	440dpi
Pixel 3 XL		6.3"	1440x2960	560dpi
Pixel 3	▶	5.46"	1080x2160	440dpi
Pixel 2 XL		5.99"	1440x2880	560dpi
Pixel 2 (Edited)		5.0"	1080x1920	xxhdpi
Pixel 2	▶	5.0"	1080x1920	420dpi
Pixel	▶	5.0"	1080x1920	420dpi

Pixel 2

Size: large
Ratio: long
Density: 420dpi

Step 4: Here we select the android version to download like **Q**, **Pie**, **Oreo** etc and click **Next** button.


Virtual Device Configuration

System Image
Android Studio

Select a system image

Recommended | x86 Images | Other Images

Release Name	API Level	ABI	Target
Q	29	x86	Android 10.0 (Google APIs)
Pie	28	x86	Android 9.0 (Google APIs)
Oreo Download	*27*	*x86*	*Android 8.1 (Google APIs)*
Oreo Download	*26*	*x86*	*Android 8.0 (Google APIs)*
Nougat Download	*25*	*x86*	*Android 7.1.1 (Google APIs)*
Nougat Download	*24*	*x86*	*Android 7.0 (Google APIs)*
Marshmallow Download	*23*	*x86*	*Android 6.0 (Google APIs)*
Lollipop Download	*22*	*x86*	*Android 5.1 (Google APIs)*

Q

API Level
29

Android
10.0
Google Inc.

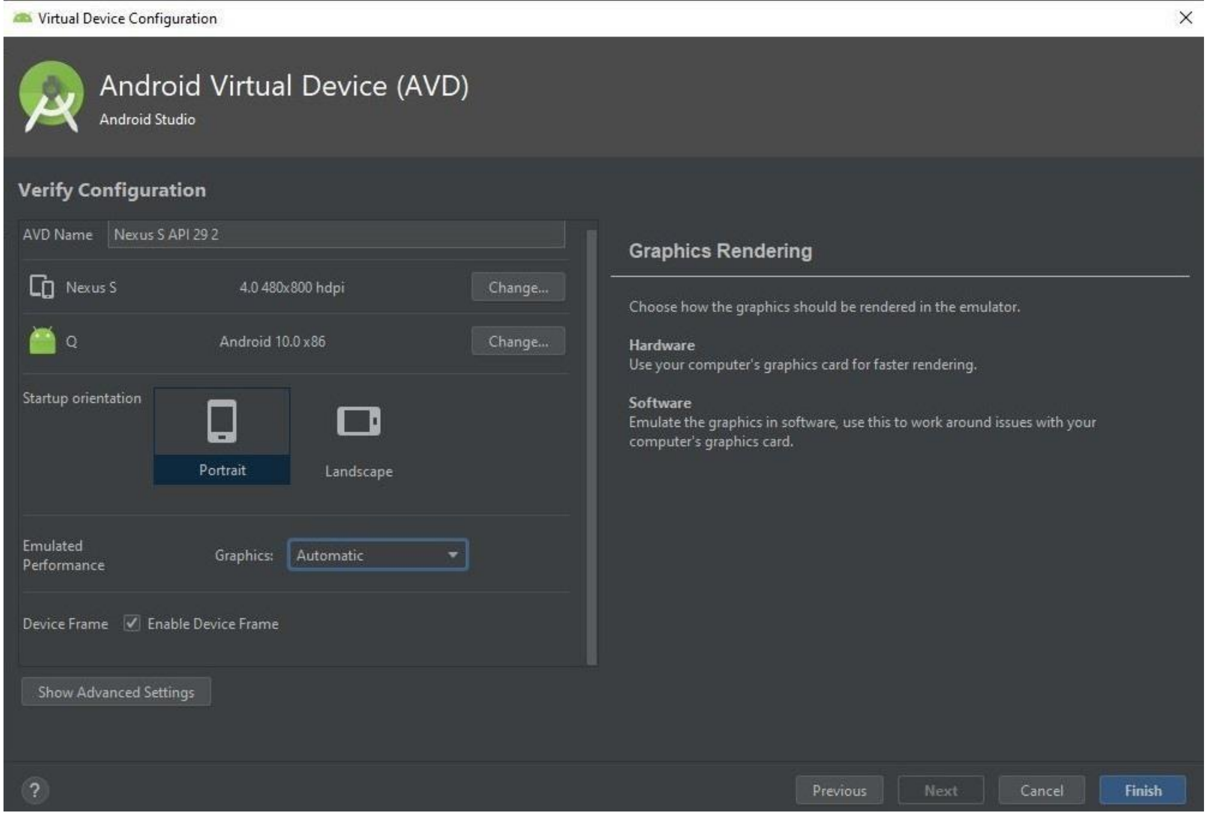
System Image
x86

We recommend these images because they run the fastest and support Google APIs.

Questions on API level?
See the API level distribution chart

Previous | Next | Cancel | Finish

Step 5: Click the **finish** button to complete the installation.


Virtual Device Configuration

Android Virtual Device (AVD)
Android Studio

Verify Configuration

AVD Name: Nexus S API 29 2

Nexus S 4.0 480x800 hdpi Change...

Q Android 10.0 x86 Change...

Startup orientation: Portrait | Landscape

Emulated Performance — Graphics: Automatic

Device Frame ✓ Enable Device Frame

Show Advanced Settings

Graphics Rendering

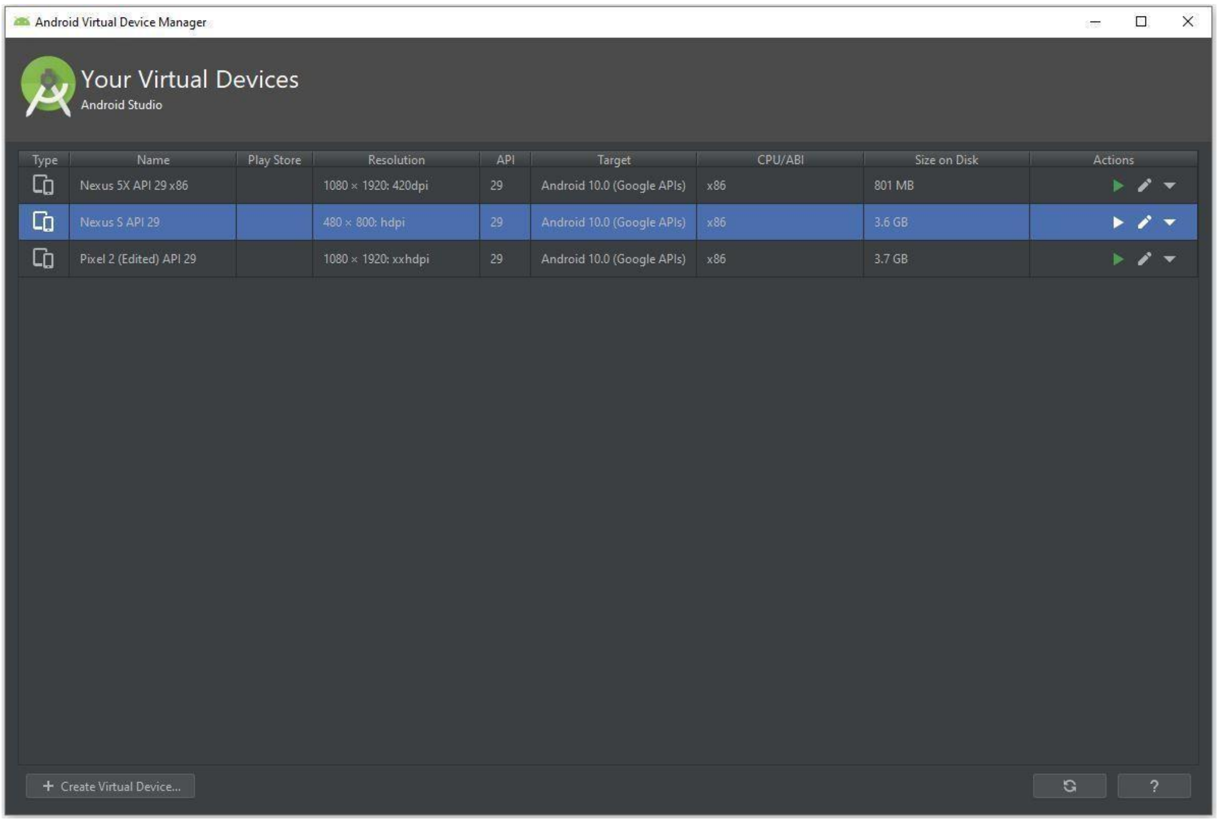
Choose how the graphics should be rendered in the emulator.

Hardware
Use your computer's graphics card for faster rendering.

Software
Emulate the graphics in software, use this to work around issues with your computer's graphics card.

Previous | Next | Cancel | Finish

Step 6: Now we can select the virtual device we want to run as emulator can click on the **run** icon.



Type	Name	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	Nexus 5X API 29 x86		1080 × 1920: 420dpi	29	Android 10.0 (Google APIs)	x86	801 MB	
	Nexus S API 29		480 × 800: hdpi	29	Android 10.0 (Google APIs)	x86	3.6 GB	
	Pixel 2 (Edited) API 29		1080 × 1920: xxhdpi	29	Android 10.0 (Google APIs)	x86	3.7 GB	

Step 7: Finally our virtual device is ready to run our android app.

the various types of android applications:

- Native apps: These apps use the device's native UI to provide a stylish user experience and high performance.
- Examples include WhatsApp, Spotify, and Pokemon GO.
- Hybrid apps: These apps are a mix between native and web apps and consist

of backend code and a native viewer that displays the backend in a web view.

- Web apps
- Progressive web apps
- Augmented reality apps
- Virtual reality apps
- Gaming apps
- Social media apps
- Educational apps
- Productivity apps

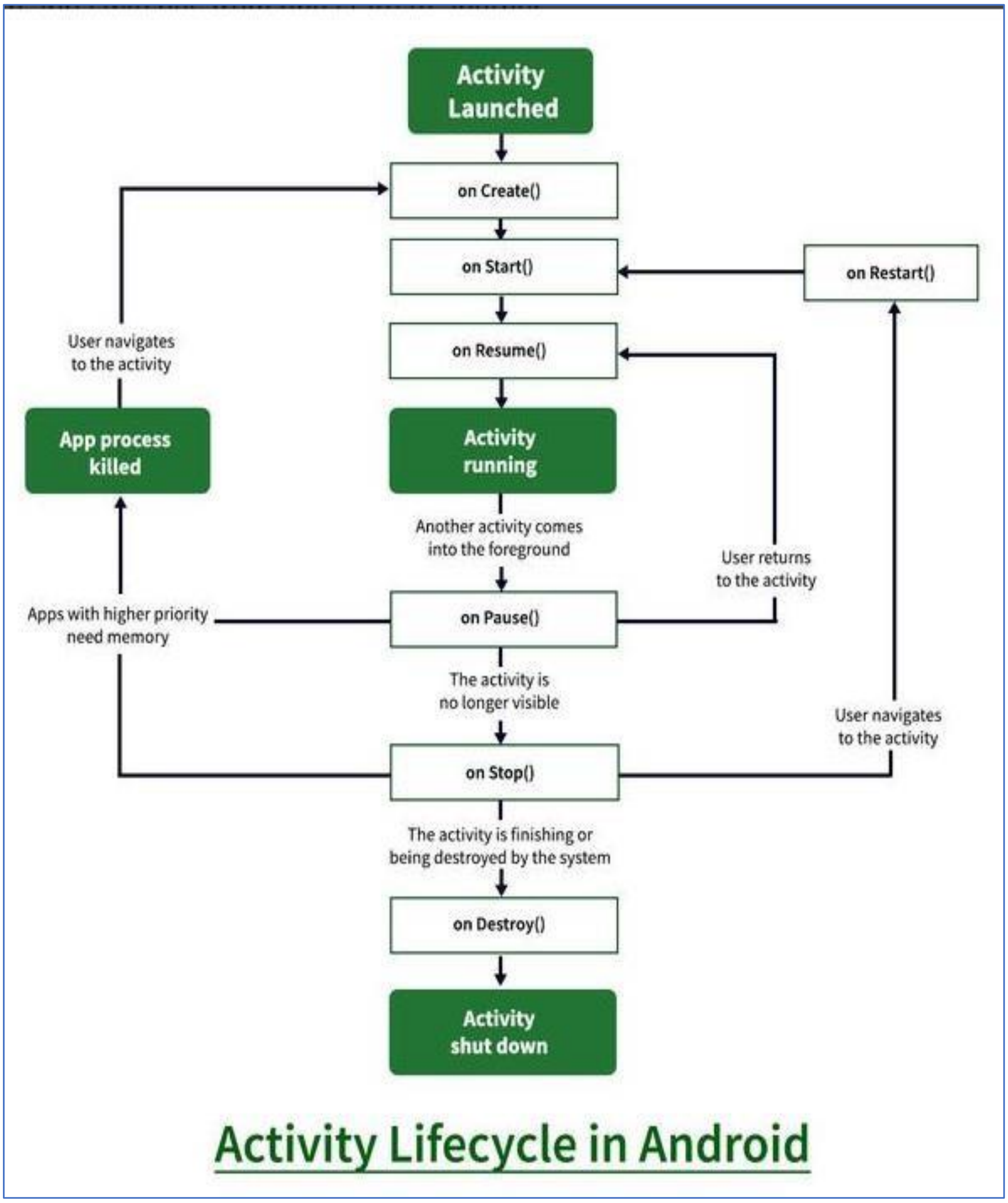
7.Define Activity Life Cycle? Explain with neat diagram Activity Life Cycle?

Ans:

The **Activity Life Cycle** describes the various states an Activity goes through from its creation to its destruction. Understanding these states is crucial for developing robust Android applications that handle user interactions and system events effectively.

diagram illustrating the Activity Life Cycle:

•


Activity Launched
on Create()
on Start()
on Restart()
User navigates to the activity
on Resume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
on Pause()
The activity is no longer visible
User navigates to the activity
on Stop()
The activity is finishing or being destroyed by the system
on Destroy()
Activity shut down
Activity Lifecycle in Android

1. onCreate( )

This callback is fired when the system first creates the activity. In onCreate method, all such operations are performed which should be done only once for the entire life of the activity.

This method has a parameter *savedInstanceState* which is a bundle object. It contains activity’s previous saved state. If the activity has never existed before, the value of Bundle object is null.

2. onStart( )

When the activity enters the Started state, the system invokes this callback. The *onStart( )* call makes the activity visible to the user, as the app prepares for the activity to enter the foreground and become interactive. This method can be used to initialize the code that maintains the user interface. After *onStop( )* , if activity restarts then *onStart( )* is called.

onCreate ( ) function is called only once but onStart( ) can be called multiple times , when activity enters the started state.

3. onResume( )

The *onStart( )* method completes very quickly and, as with the Created state, the activity does not stay resident in the Started state. Once this callback finishes, the activity enters the *Resumed* state, and the system invokes the

onResume( ) method.

This is the state in which the app interacts with the user. The app stays in this state until something happens to take focus away from the app. Such an event might be, for instance,

receiving a phone call,

the user's navigating to another activity, or

the device screen's turning off.

When such an event occurs, the activity enters the *Paused* state, and the system invokes the *onPause( )* callback.

4. onPause( )

The system calls this method as the first indication that the user is leaving the activity (though it does not always mean the activity is being destroyed); it indicates that the activity is no longer in the foreground (though it may still be visible if the user is in multi- window mode).

In most case onPause() method called by Android OS when user press Home button (Center Button on Device) to make hide. Activity is not visible to user and goes in background when onPause() method is executed

When the activity moves to the paused state, any lifecycle-aware component tied to the activity’s lifecycle will receive the *onPause( )* event. This is where the lifecycle components can stop any functionality that does not need to run while the component is not in the foreground

onPause( ) method can be used to release system resources, handles to sensors (like GPS), or any resources that affect battery life while your activity is paused and the user does not need them.

5. onStop( )

When your activity is no longer visible to the user, it has entered the *Stopped* state, and the system invokes the onStop( ) callback. Any activity gets stopped in case some other activity takes place of it. For example, if a user was on screen 1 and click on some button and moves to screen 2. In this case Activity displaying content for screen 1 will be stopped.

The system may also call onStop( ) when the activity has finished running, and is about to be terminated.

In the *onStop( )* method, the app should release or adjust resources that are not needed while the app is not visible to the user. It should be used to perform relatively CPU-intensive shutdown operations. So Activity will be in stopped state when hidden or replaced by other activities that have been launched or switched by user

6. onRestart( )

onStart( ) will always be called whenever you enter your Activity just after *onCreate( )* but the *onRestart( )* will only be called before *onStart( )* when your Activity comes from being stopped (passing from *onStop( )*) back to the vision.

7. onDestroy( )

onDestroy( ) is called before the activity is destroyed. The system invokes this callback either because:

1. if user pressed the back navigation button then activity will be destroyed after completing the lifecycle of pause and stop.

2.In case if user press the home button and app moves to background. User is not using it no more and it’s being shown in recent apps list. So in this case if system required resources need to use somewhere else then OS can destroy the Activity.

After the Activity is destroyed if user again click the app icon, in this case activity will be recreated and follow the same lifecycle again.

8.List the methods present in Android lifecycle. Discuss each method using code snippets.

In Android, the Activity lifecycle consists of several callback methods. These methods help manage the state of an Activity, ensuring that resources are properly initialized and released as the Activity transitions through different states. Here's a detailed discussion of each lifecycle method along with code snippets:

1. **onCreate(Bundle savedInstanceState)**
   - Called when the activity is first created.
   - Used to initialize the activity, set up the user interface, and initialize variables.

```
java
Copy code
@Override
protected void
onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstance
State);

setContentView(R.layout.acti
vity_main);  // Set the user
interface layout for this
Activity
```

```
    // Initialize variables,
set up listeners, etc.
    Log.d("Lifecycle",
"onCreate called");
}
```

2. **onStart()**
   - Called when the activity is becoming visible to the user.
   - This is followed by `onResume()` if the activity comes to the foreground, or `onStop()` if it becomes hidden.

```
java
Copy code
@Override
protected void onStart() {
    super.onStart();
    // The activity is about
to become visible.
    Log.d("Lifecycle",
"onStart called");
}
```

3. **onResume()**

- Called when the activity will start interacting with the user.
- This is a good place to start animations, open exclusive-access devices (such as the camera), etc.

```
java
Copy code
@Override
protected void onResume() {
    super.onResume();
    // The activity has
become visible (it is now
"resumed").
    Log.d("Lifecycle",
"onResume called");
}
```

4. **onPause()**
   - Called when the system is about to put the activity into the background.
   - Use this method to pause animations, save unsaved data, release resources that are not needed while the activity is not in the foreground, etc.

```
java
Copy code
@Override
protected void onPause() {
    super.onPause();
    // Another activity is
taking focus (this activity
is about to be "paused").
    Log.d("Lifecycle",
"onPause called");
    // Pause animations,
save data, etc.
}
```

5. **onStop()**
   - Called when the activity is no longer visible to the user.
   - This may happen because the activity is being destroyed, a new activity is starting, or an existing activity is being brought to the front.

```
java
Copy code
@Override
protected void onStop() {
    super.onStop();
    // The activity is no
longer visible (it is now
"stopped").
    Log.d("Lifecycle",
"onStop called");
    // Release resources,
save data, etc.
}
```

6. **onRestart()**
   - Called after the activity has been stopped, just prior to it being started again.
   - Use this method to re-initialize components that were released during `onStop()`.

```
java
Copy code
@Override
protected void onRestart() {
    super.onRestart();
    // The activity is about
to be restarted.
```

```
    Log.d("Lifecycle",
"onRestart called");
    // Re-initialize
resources that were released
in onStop().
}
```

7. **onDestroy()**
   - Called before the activity is destroyed.
   - This is the final call that the activity will receive. Use this method to clean up any resources, such as threads, that need to be explicitly terminated.

```
java
Copy code
@Override
protected void onDestroy() {
    super.onDestroy();
    // The activity is about
to be destroyed.
    Log.d("Lifecycle",
"onDestroy called");
    // Clean up resources.
}
```

Here is a complete example incorporating all the lifecycle methods:

```
public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
      Log.d("Lifecycle", "onCreate called");
   @Override
   protected void onStart() {
      super.onStart();
      Log.d("Lifecycle", "onStart called");
   }
   @Override
   protected void onResume() {
        super.onResume();
        Log.d("Lifecycle",                 "onResume
    called");
    }
    @Override
    protected void onPause() { super.onPause();
        Log.d("Lifecycle", "onPause called");

    @Override
    protected void onStop() {
        super.onStop();
        Log.d("Lifecycle", "onStop called");
    }
    @Override
    protected void onRestart() {
        super.onRestart();
        Log.d("Lifecycle",                    "onRestart
called");


    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.d("Lifecycle",                   "onDestroy
called");     }}
```

9.Design a screen with edittext, button and textview component. Display the message in textview whenever user clicks on submit button.

activity_main.xml :

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout
xmlns:android="http://schemas.android.com/
apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <EditText
        android:id="@+id/editTextMessage"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Enter your message" />

    <Button
        android:id="@+id/buttonSubmit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Submit"

android:layout_gravity="center_horizontal"
        android:layout_marginTop="16dp" />
    <TextView
        android:id="@+id/textViewDisplay"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text=""
        android:layout_marginTop="16dp"
        android:textSize="18sp" />
</LinearLayout>
```

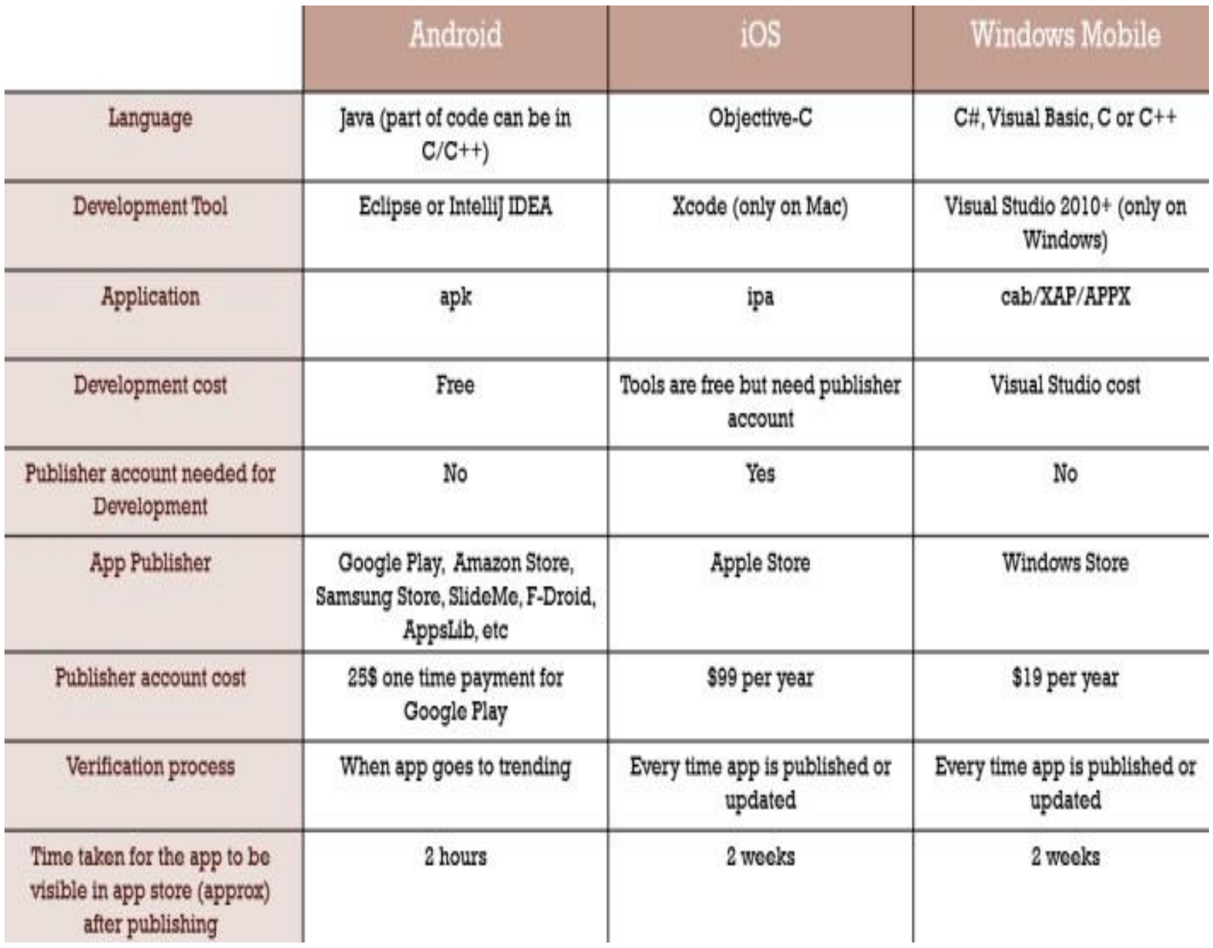
MainActivity.java

```
package com.example.yourpackage;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import
androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends
AppCompatActivity {
    private EditText editTextMessage;
    private Button buttonSubmit;
    private TextView textViewDisplay;

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

        // Initialize the UI components
        editTextMessage                              =
findViewById(R.id.editTextMessage);
        buttonSubmit                                 =
findViewById(R.id.buttonSubmit);
        textViewDisplay                              =
findViewById(R.id.textViewDisplay);

        // Set an OnClickListener on the button
        buttonSubmit.setOnClickListener(new
View.OnClickListener() {
            @Override
        public void onClick(View v) {
            // Get the text from EditText
            String message = editTextMessage.getText().toString();

            // Display the message in the TextView
            textViewDisplay.setText(message);
        }
    });
  }
}
```

QUESTION-10:

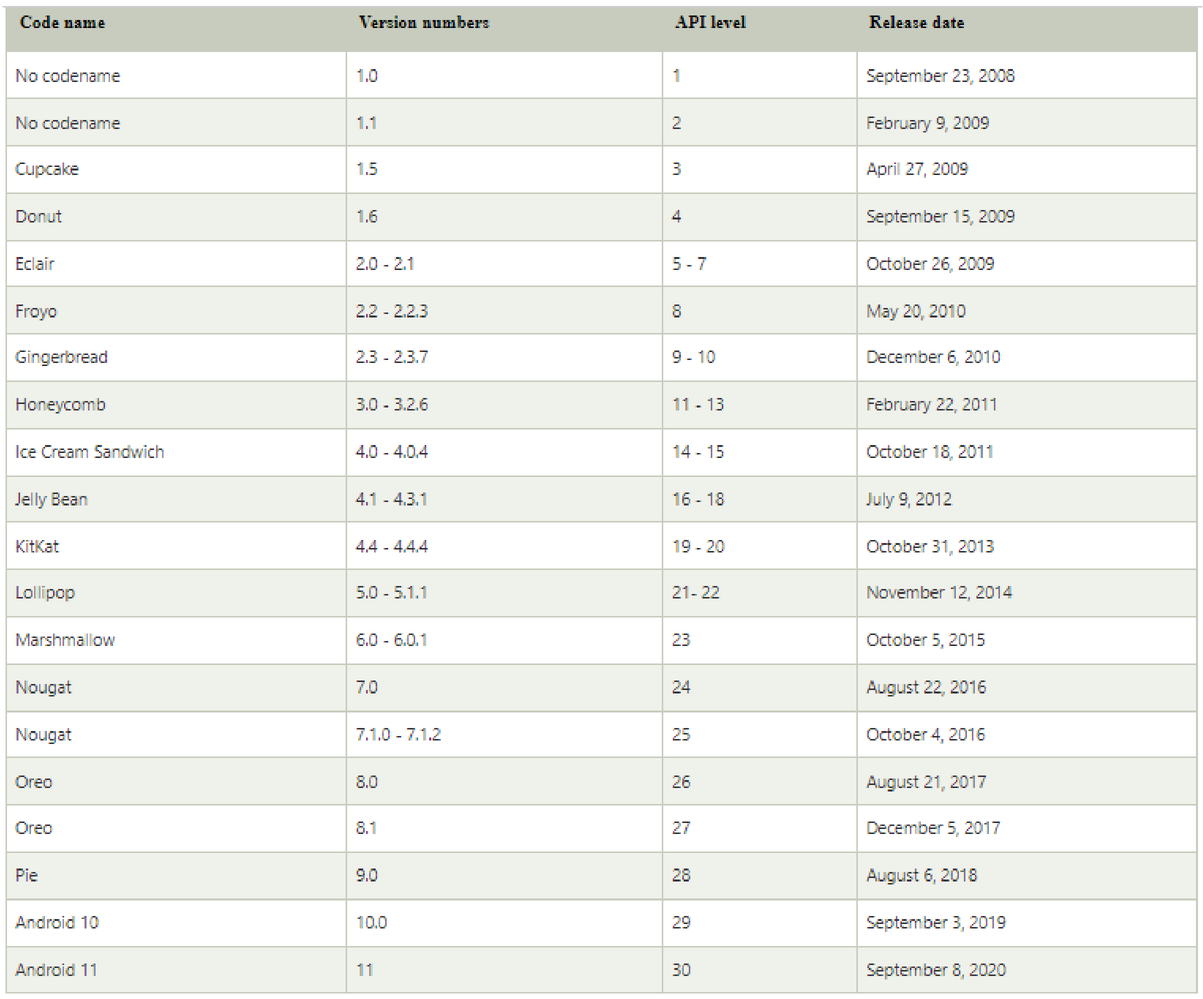
a) Compare Android OS with any other OS.(4M)
b) Enlist various Android versions.(4M)
c) Explain need of Android.(4M)

ANSWER FOR a:



	Android	iOS	Windows Mobile
Language	Java (part of code can be in C/C++)	Objective-C	C#, Visual Basic, C or C++
Development Tool	Eclipse or IntelliJ IDEA	Xcode (only on Mac)	Visual Studio 2010+ (only on Windows)
Application	apk	ipa	cab/XAP/APPX
Development cost	Free	Tools are free but need publisher account	Visual Studio cost
Publisher account needed for Development	No	Yes	No
App Publisher	Google Play, Amazon Store, Samsung Store, SlideMe, F-Droid, AppsLib, etc	Apple Store	Windows Store
Publisher account cost	25$ one time payment for Google Play	$99 per year	$19 per year
Verification process	When app goes to trending	Every time app is published or updated	Every time app is published or updated
Time taken for the app to be visible in app store (approx) after publishing	2 hours	2 weeks	2 weeks

Answer for b:



Code name	Version numbers	API level	Release date
No codename	1.0	1	September 23, 2008
No codename	1.1	2	February 9, 2009
Cupcake	1.5	3	April 27, 2009
Donut	1.6	4	September 15, 2009
Eclair	2.0 - 2.1	5 - 7	October 26, 2009
Froyo	2.2 - 2.2.3	8	May 20, 2010
Gingerbread	2.3 - 2.3.7	9 - 10	December 6, 2010
Honeycomb	3.0 - 3.2.6	11 - 13	February 22, 2011
Ice Cream Sandwich	4.0 - 4.0.4	14 - 15	October 18, 2011
Jelly Bean	4.1 - 4.3.1	16 - 18	July 9, 2012
KitKat	4.4 - 4.4.4	19 - 20	October 31, 2013
Lollipop	5.0 - 5.1.1	21- 22	November 12, 2014
Marshmallow	6.0 - 6.0.1	23	October 5, 2015
Nougat	7.0	24	August 22, 2016
Nougat	7.1.0 - 7.1.2	25	October 4, 2016
Oreo	8.0	26	August 21, 2017
Oreo	8.1	27	December 5, 2017
Pie	9.0	28	August 6, 2018
Android 10	10.0	29	September 3, 2019
Android 11	11	30	September 8, 2020

Answer for c:

Need of Android | Mobile Application Development

Need of Android:

1. Android Market:

The android market is the tech-based and fastest-growing market all over the world. We can download almost any software we think for in the android market. The ready or available software to run on our android phone means that our phone will have more functionality and thus making it convenient to use.

2. Customization:

Android offers the pretty best in terms of end-user experience mainly because of the customization. Android is open-source software that means it is freely available to everyone. This creates unbounded customization in terms of color, design, functionality, and general appearance of our devices

3. Android Community:

Community is one of the major reasons which define the need for an android device is the best decision we will ever make. Almost 70 to 80% of devices are running Android and this reason enough for the need of Android devices. The hudge and large android community provide us help and support to help you get the best out of your device. The limitless forums, blogs, websites, and guides are written by experts will go a long way in boosting your experience as a user.

4. Rooting:

Rooting Android devices makes it possible for us to modify and change system files in the device and thus enabling us to further, customize it for a better experience.

5. Powerful Development Framework:

Android provides almost everything we need to build the best app experiences. Android provides a single application model that lets you deploy your apps broadly to hundreds of millions of users across a wide range of devices - from phones to tablets.

Android also gives us tools for creating apps that look great and take advantage of the hardware capabilities available on each device. It automatically adapts your UI to look its best on each device, while giving you as much control as you want over your UI on different device types.

6. Global Partnerships and Large Installed Base:

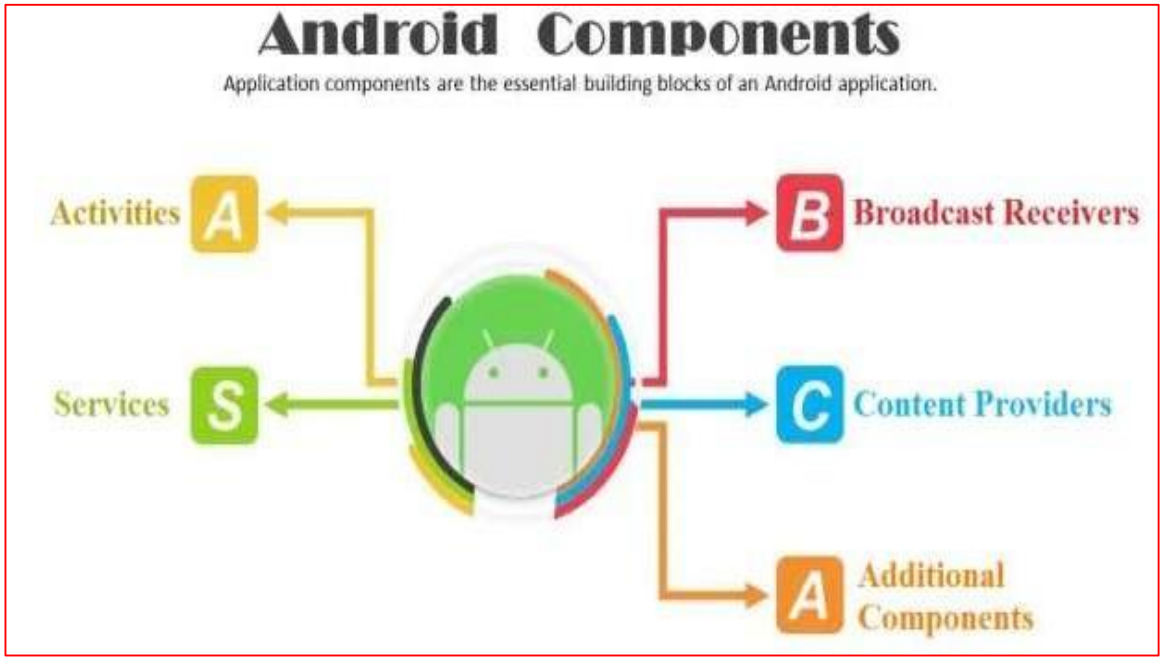
Building on the contributions of the open-source Linux community and more than 300 hardware, software, and carrier partners, Android has rapidly become the fastest-growing mobile OS(Operating System). Android's openness has made it a favorite for consumers and developers alike, driving strong growth in app consumption. Android users download more than 1.5 billion apps and games from Google Play each month. With its partners, Android is continuously pushing the boundaries of hardware and software forward to bring new capabilities to users and developers.

UNIT-2

11.Name different Android Application Components? Explain them in detail?

Android application components :

Application components are the essential building blocks of an Android application. These components are loosely coupled by the application manifest file **AndroidManifest.xml** that describes each component of the application and how they interact.



There are following four main components that can be used within an Android application:

Components	**Description**
Activities	They dictate the UI and handle the user interaction to the smartphone screen
Services	They handle background processing associated with an application.
Broadcast Receivers	They handle communication between Android OS and applications.
Content Providers	They handle data and database management issues.

1. Activities

An ***activity*** is the entry point for interacting with the user. It represents a single screen with a user interface.

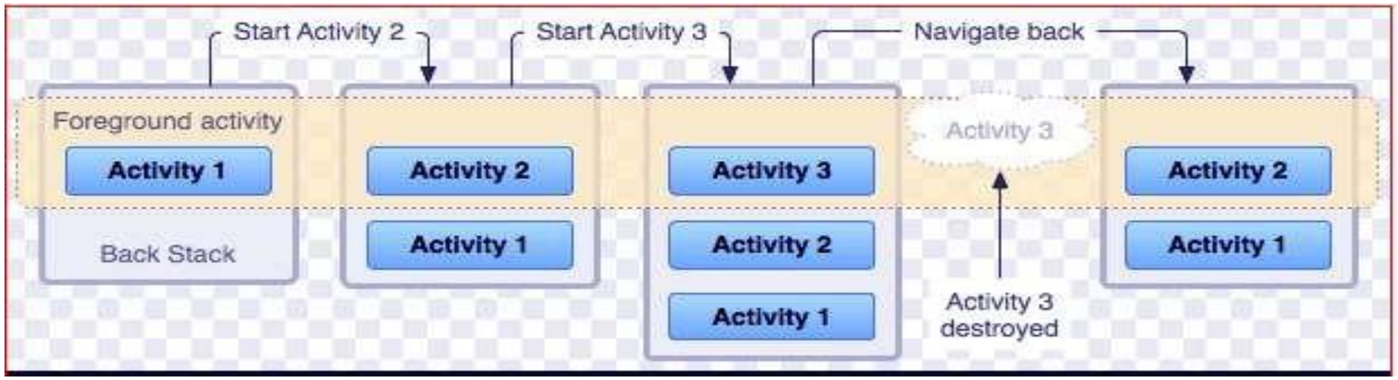
For example, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails.

An activity facilitates the following key interactions between system and app:

- **Keeping track** of what the user currently cares about (what is on screen)
- Knowing that previously used processes contain things the **user may return to** (stopped activities)
- Providing a way for apps to **implement user flows between each other.**

Each activity is independent of one another.

For example – Camera application can be started in an email application to compose an email that shares an image. The picture below depicts how each new activity adds an item to back stack and how the current activity is destroyed and previous activity is resumed.



To implement an activity, **extend the Activity class** in your subclass:

```
public class MainActivity extends Activity
{
            //code
 }
```

2. Services

- A ***service*** is a component that **runs in the background to perform long-term operations** or to perform work for remote processes. it acts as **an invisible worker** of our application.
- It performs tasks **when applications are not active.**
- A service **does not provide a user interface**.
- An **example** of service is we can surf the internet or use any other application while listening to music.

To execute services, **extend the Services class in your sub-class:**

```
public class MyService extends Services
 {
//code
}
```

3. Broadcast Receivers

A *broadcast receiver* is an Android application component which allows you to register for system generated events or application events.

Whenever those events occur the system notifies all the registered broadcast receivers and then the desired action is being done. Broadcast originates from the system as well as applications. The Broadcast Receiver's job is to pass a notification to the user when a specific event occurs.

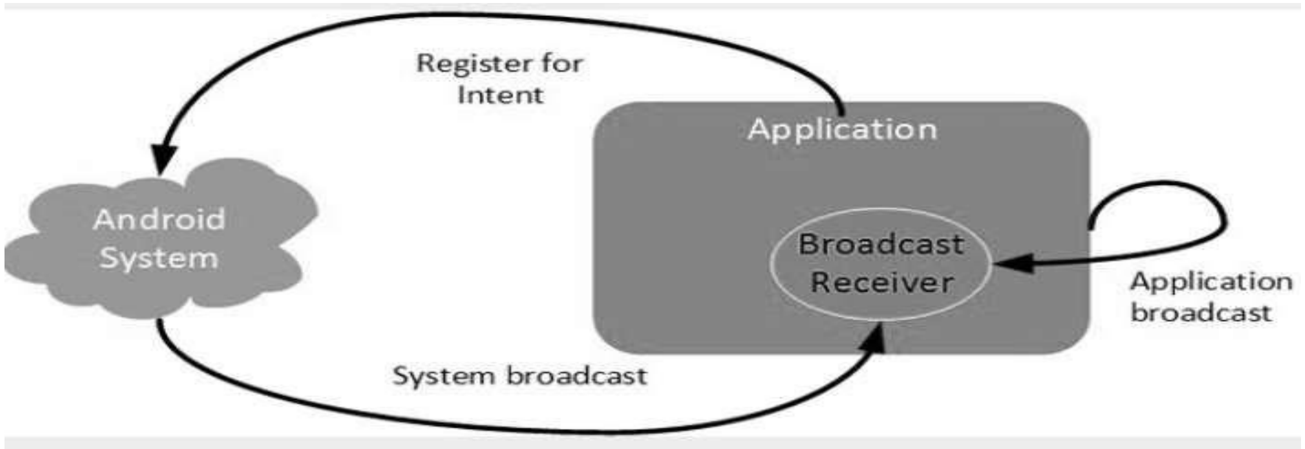
Using a Broadcast Receiver, applications can register for a particular type of event. Once the event occurs, the system will notify all the registered applications in the system.

Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs.

Example

The alarm notification, low battery notification etc. are the example of broadcast originating from the system.

While getting the push notifications for desired application describes the example for broadcast originating from the application.



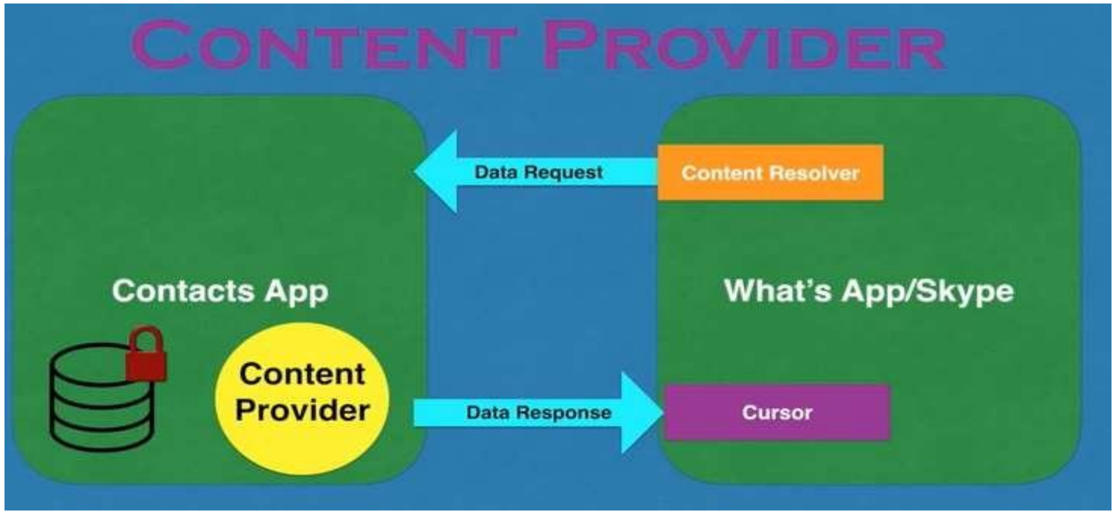
4.Content Providers

Content Provider is a component that **allows applications to share data among multiple applications**.

It **hides the details of the database** and can be used to read and write private data of the application which is not shared. It would be a mess to access data from other applications without ***content providers***.

The prime purpose of a content provider is to **serve as a central repository** of data where users can store and can fetch the data. The access of this repository is given to other applications also but in a safe manner in order to serve the different requirements of the user.

For example, the **contacts data is used by multiple applications** and must be stored in a content provider.



12 Discuss the importance of Android Manifest File and what are the essential tags available in Manifest File?

Android Manifest file

The Android Manifest is **an XML file which contains important metadata about the Android app.**

Every android project that we make, **need to have an Android Manifest file within**. This file is present in the **root of the project source set**.

It describes all the **important information of the application to the Android build tools**.

This includes the **package name, activity names, main activity (the entry point to the app), Android version support, hardware features support, permissions, and other configurations**

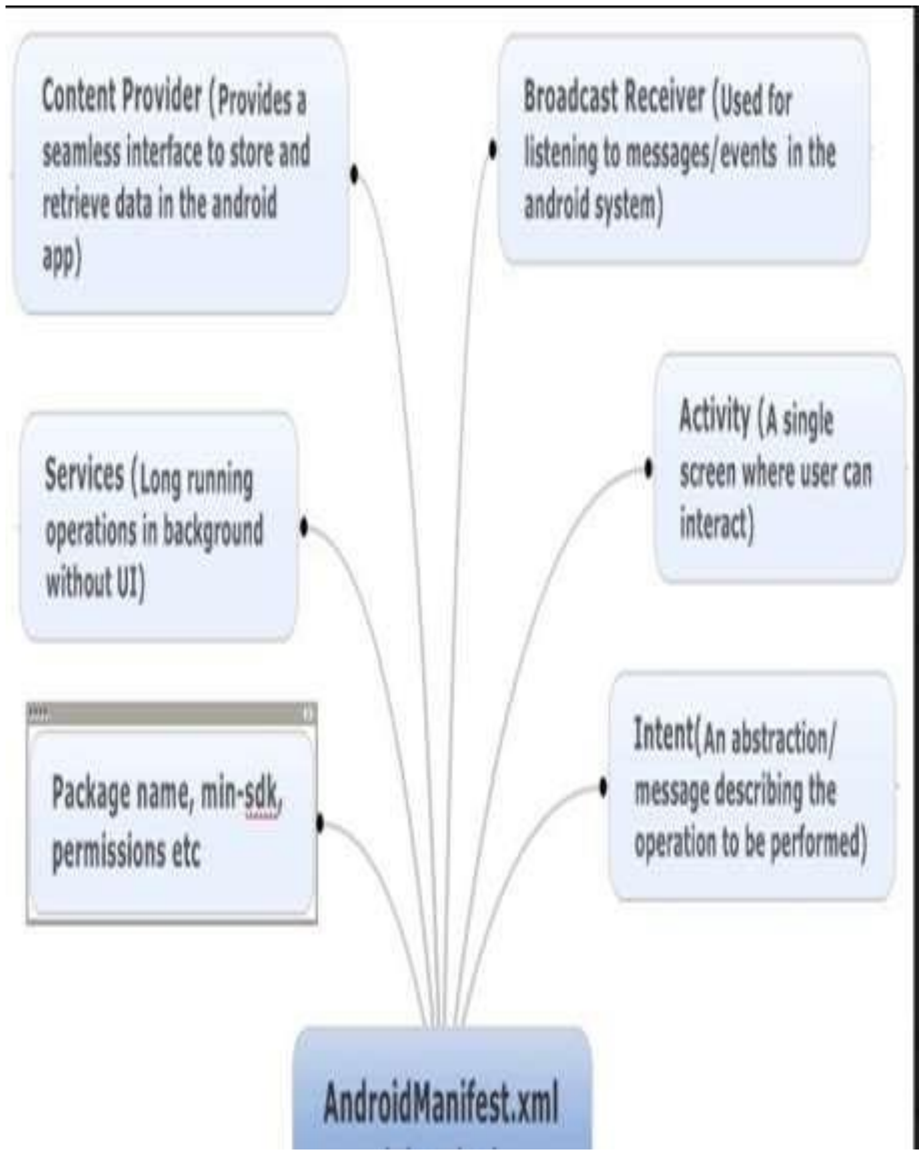
The **information that is stored in the Manifest file** is as follows:

- ➢ The **name of the application's package**, it is generally the **code's namespace**. This information is used to determine the location of the code while building the project.
- ➢ Another component is the one, that includes all the **activities, services, receivers, and content providers**.
- ➢ The **permissions that are required by the application** to access the protected parts of the system and other apps.

- The features required by the app, that affect **which devices can install the app from Google Play**. These features include both hardware and software features.
- It also specifies the application metadata, which includes the **icon, version number, themes, etc**.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <application android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Week3A"
        tools:targetApi="31">
```

```
 <activity  android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action
android:name="android.intent.action.MAIN" />
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```


Content Provider (Provides a seamless interface to store and retrieve data in the android app)
Broadcast Receiver (Used for listening to messages/events in the android system)
Services (Long running operations in background without UI)
Activity (A single screen where user can interact)
Package name, min-sdk, permissions etc
Intent(An abstraction/ message describing the operation to be performed)
AndroidManifest.xml

Element Tags of Manifest File

Following are the essential element tags of Manifest.xml files:

1. <manifest>

It is the **root element of this element**. It consists of a package attribute package that tells the activity's package name.

2. <application>

It is the **sub-element of the manifest file** that includes the declaration of the namespace. It contains certain attributes. These attributes declare the application components, and these attributes include:

- **android:icon** represents the icon for all the android application components.
- **allowBackup**
- **android:label** works as the default label for all the application components.
- **android:theme** represents a common theme for all the android activities.

3. <activity>

Activity is a **sub-element of application**. It has the declaration of the activity that must be there in the manifest file. It also has certain attributes like name, label, theme, etc.

- **android:label** represents a label i.e. displayed on the screen.
- **android:name** represents a name for the activity class. It is required attribute.

4. <intent-filter>

It is an **element in the activity**, it describes the type of intent in which the Android components can respond.

5. <action>

This element provides an **action for the intent filter**. Each intent filter **must have at least one action element** in it.

6. <category>

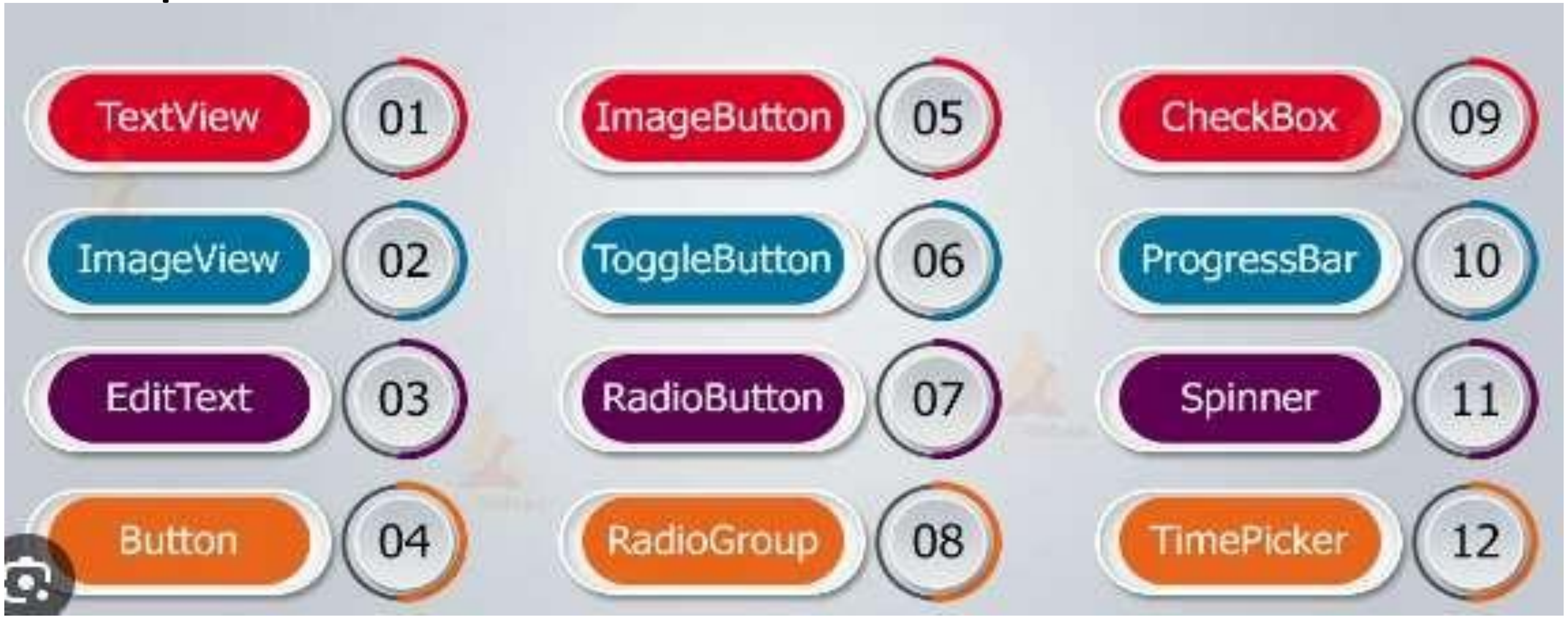
This element **adds the category name** in an intent-filter.

7. <service>

This element contains the operations that are provided by libraries or APIs.

13 a) List out the fundamental components used to build user Interface in android.(4M)

UI Components



Sr. No.	UI Control & Description
1	TextView This control is used to display text to the user.
2	EditText EditText is a predefined subclass of TextView that includes rich editing capabilities.
3	Button A push-button that can be pressed, or clicked, by the user to perform an action.
4	ImageButton An ImageButton is an AbsoluteLayout which enables you to specify the exact location of its children. This shows a button with an image (instead of text) that can be pressed or clicked by the user.
5	CheckBox An on/off switch that can be toggled by the user. You should use check box when presenting users with a group of selectable options that are not mutually exclusive.
6	ToggleButton An on/off button with a light indicator.

7	RadioButton The RadioButton has two states: either checked or unchecked.
8	RadioGroup A RadioGroup is used to group together one or more RadioButtons.
9	ProgressBar The ProgressBar view provides visual feedback about some ongoing tasks, such as when you are performing a task in the background.
10	Spinner A drop-down list that allows users to select one value from a set.
11	TimePicker The TimePicker view enables users to select a time of the day, in either 24-hour mode or AM/PM mode.
12	DatePicker The DatePicker view enables users to select a date of the day.

13b) Write a program to implement Radio Button component in android.(8M)

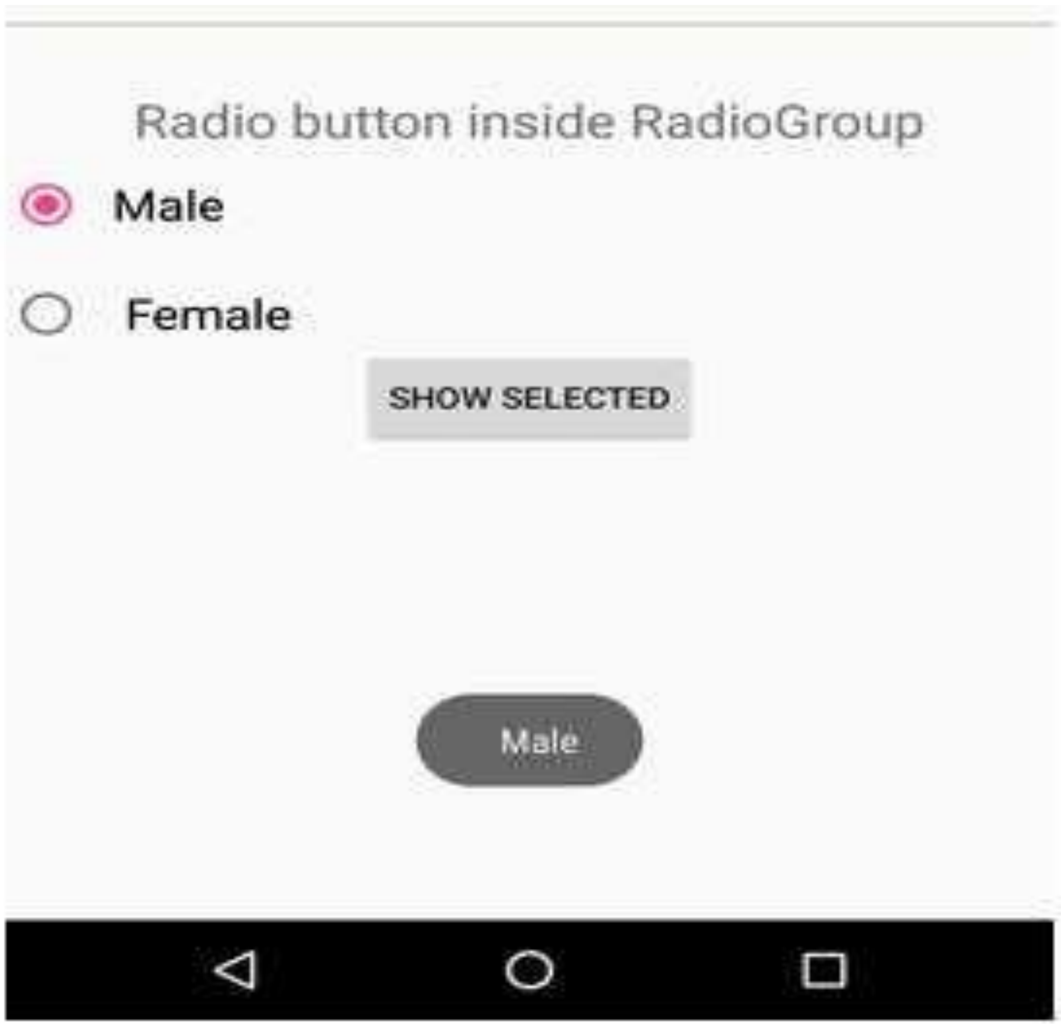
- **RadioButton** is a two states button which is either checked or unchecked. If a single radio button is unchecked, we can click it to make checked radio button. Once a radio button is checked, it cannot be marked as unchecked by user.

- RadioButton is generally used with *RadioGroup*.

- RadioGroup contains several radio buttons, marking one radio button as checked makes all other radio buttons as unchecked.

```
<RadioGroup
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/radioGroup">

    <RadioButton
      android:id="@+id/radioMale"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="  Male"
      android:layout_marginTop="10dp"
      android:checked="false"
      android:textSize="20dp" />

    <RadioButton
      android:id="@+id/radioFemale"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="    Female"
      android:layout_marginTop="20dp"
      android:checked="false"

      android:textSize="20dp" />
  </RadioGroup>
```

Output


Radio button inside RadioGroup
Male
Female
SHOW SELECTED
Male

14.DefineLayout in android and explain various layout views available for android development?

Layouts

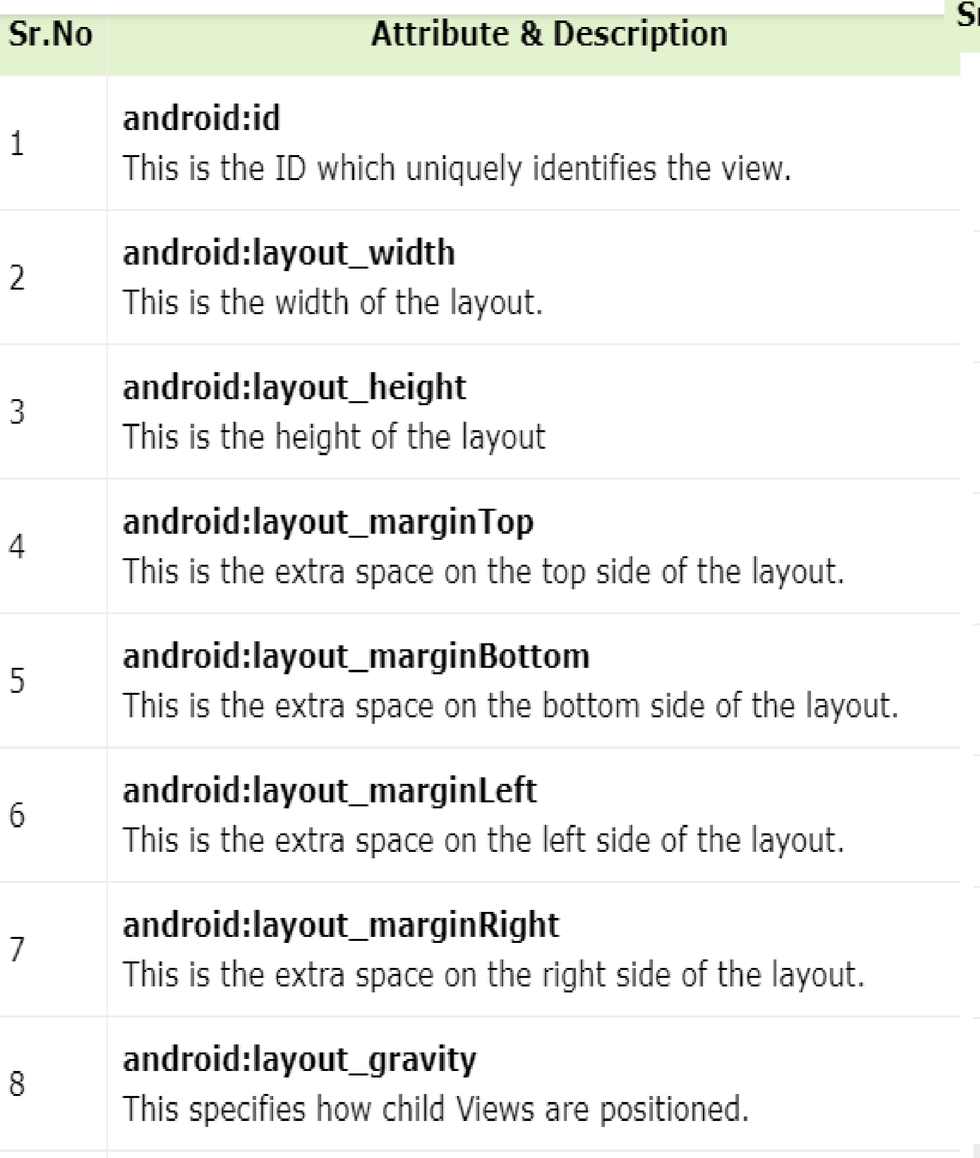
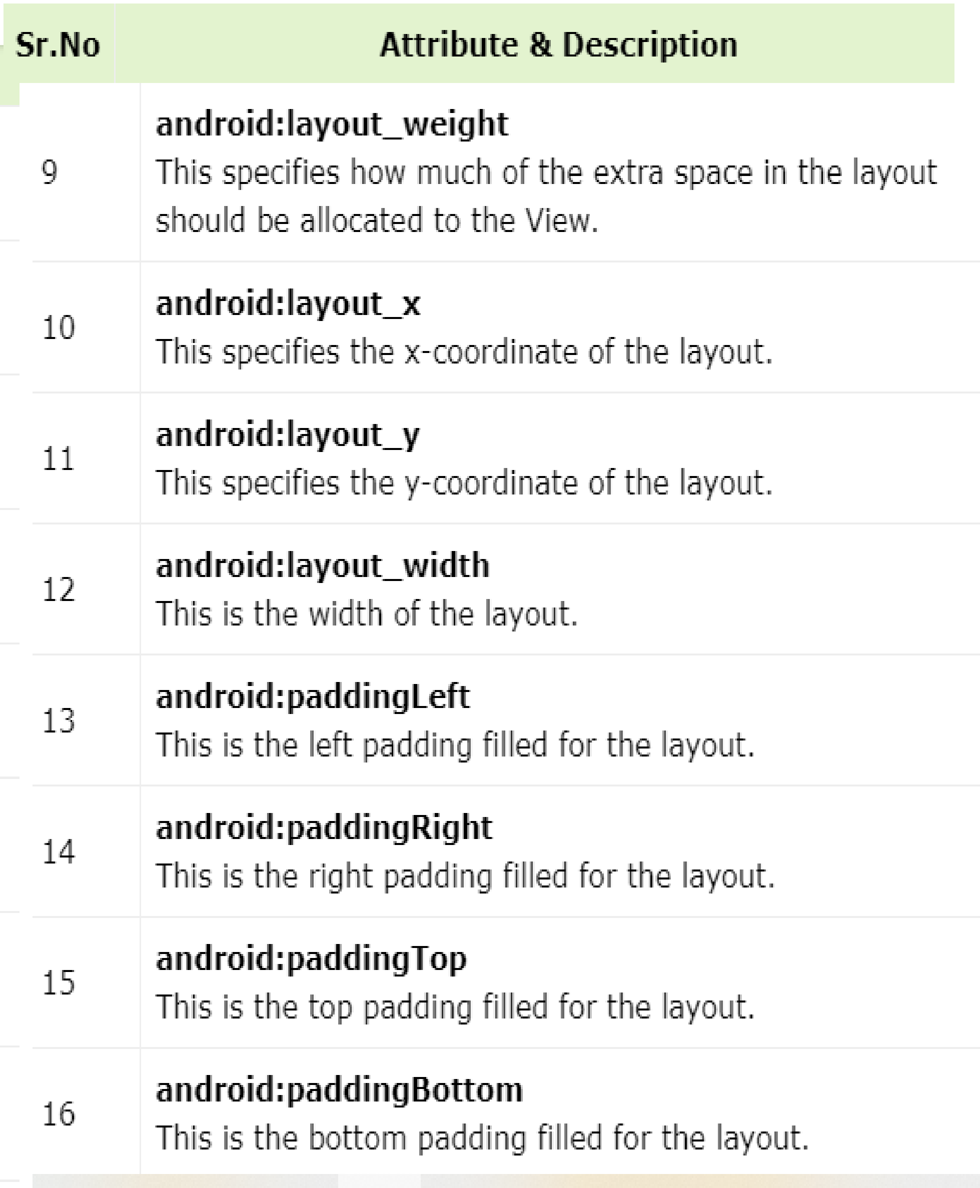
Layout basically refers to the **arrangement of elements on a page** these elements are likely to be images, texts or styles. These are a part of **Android Jetpack**. They define the structure of android user interface in the app, like in an activity.

All elements in the layout are built with the help of Views and ViewGroups. These layouts can have various widgets (components) like buttons, labels, textboxes, and many others. **Layouts are container views (and, therefore, subclassed from ViewGroup) designed to control how child views are positioned on the screen.**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:id="@+id/layout2“
            android:layout_width="match_parent“
            android:layout_height="match_parent"
            android:layout_weight="1“
            android:background="#8ED3EB“
            android:gravity="center“
            android:orientation="vertical" >
<TextView android:id="@+id/textView4“
                        android:layout_width="match_parent“
                  android:layout_height="wrap_content“
                        android:layout_marginLeft="10dp“
                        android:layout_marginTop="-40dp“
                        android:fontFamily="@font/almendra_bold"
                        android:text="This is a TextView" />
</LinearLayout>
```

Layout Attributes

Each layout has a set of attributes which define the visual properties of that layout. There are few common attributes

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the view.
2	**android:layout_width** This is the width of the layout.
3	**android:layout_height** This is the height of the layout
4	**android:layout_marginTop** This is the extra space on the top side of the layout.
5	**android:layout_marginBottom** This is the extra space on the bottom side of the layout.
6	**android:layout_marginLeft** This is the extra space on the left side of the layout.
7	**android:layout_marginRight** This is the extra space on the right side of the layout.
8	**android:layout_gravity** This specifies how child Views are positioned.
9	**android:layout_weight** This specifies how much of the extra space in the layout should be allocated to the View.
10	**android:layout_x** This specifies the x-coordinate of the layout.
11	**android:layout_y** This specifies the y-coordinate of the layout.
12	**android:layout_width** This is the width of the layout.
13	**android:paddingLeft** This is the left padding filled for the layout.
14	**android:paddingRight** This is the right padding filled for the layout.
15	**android:paddingTop** This is the top padding filled for the layout.
16	**android:paddingBottom** This is the bottom padding filled for the layout.

Android Layout Types

The **Android SDK includes the following layout views** that may be used within an Android user interface design:

1. **ConstraintLayout** – Introduced in **Android 7**, this layout manager is recommended for most layout requirements. Constraint Layout allows the positioning and behavior of the views in a layout to be **defined by simple constraint settings assigned to each child view.** The flexibility of this layout allows complex layouts to be **quickly and easily created without the necessity to nest other layout types inside each other**, resulting in improved layout performance. Constraint Layout is also tightly integrated into the Android Studio Layout Editor tool.

2. LinearLayout – Positions child views in a **single row or column depending on the orientation selected**. A *weight* value can be set on each child to specify how much of the layout space that child should occupy relative to other children.

3. TableLayout – Arranges child views into a **grid format of rows and columns**. Each row within a table is represented by a *TableRow* object child, which, in turn, contains a view object for each cell.

4. FrameLayout – The purpose of the FrameLayout is to **allocate an area of the screen, typically to display a single view**. If multiple child views are added, they will, by **default, appear on top of each other and be positioned in the top left-hand corner** of the layout area. Alternate positioning of individual child views can be achieved by **setting gravity values on each child.**

For example, setting a **center_vertical gravity value** on a child will cause it to be positioned in the vertical center of the containing FrameLayout view.

5. RelativeLayout – The RelativeLayout allows child views to **be positioned relative to each other and the containing layout view through the specification of alignments and margins on child views**.

For example, child *View A* may be configured to be positioned in the vertical and horizontal center of the containing RelativeLayout view. *View B*, on the other hand, might also be configured to be centered horizontally within the layout view but positioned 30 pixels above the top edge of *View A*, thereby making the vertical position *relative* to that of *View A*. **The RelativeLayout manager can be helpful when designing a user interface that must work on various screen sizes and orientations.**

6. AbsoluteLayout – Allows child views to be **positioned at specific X and Y coordinates** within the containing layout view. Using this layout is discouraged since it lacks the flexibility to respond to screen size and orientation changes.

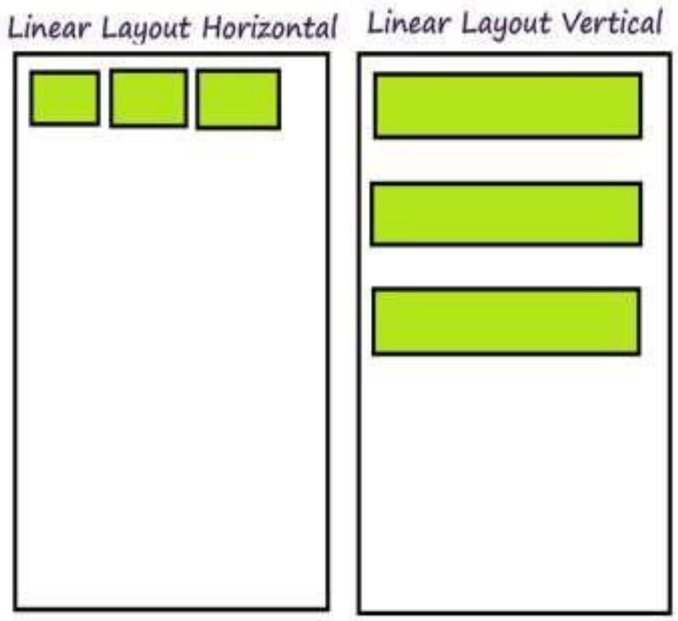
7. GridLayout – A GridLayout instance is **divided by invisible lines that form a grid containing rows and columns of cells.** Child views are then placed in cells and may be configured to cover multiple cells horizontally and vertically, allowing a wide range of layout options to be quickly and easily implemented. Gaps between components in a GridLayout may be implemented by placing a special type of view called a *Space* view into adjacent cells or setting margin parameters.

15 Discuss Linear Layout with the help of example program in detail.

1. Linear Layout

We use this layout to **place the elements in a linear manner**. A Linear manner means **one element per line**.
This layout creates various kinds of forms on Android. In this, arrangement of the elements is in a **top to bottom manner**.

Linear layout is simple and easy to use, it creates a scroll bar if the length of the window exceeds the length of the screen.
Vertically linear layout has only one item per row. **Linear layout has many different attributes** which can be used to customize linear layout according to needs.
This can have two orientations:
a. Vertical Orientation – It is shown above where the widgets such as TextViews, and all in a vertical manner.
b. Horizontal Orientation – It is shown above where the widgets such as TextViews, and all in a horizontal manner.

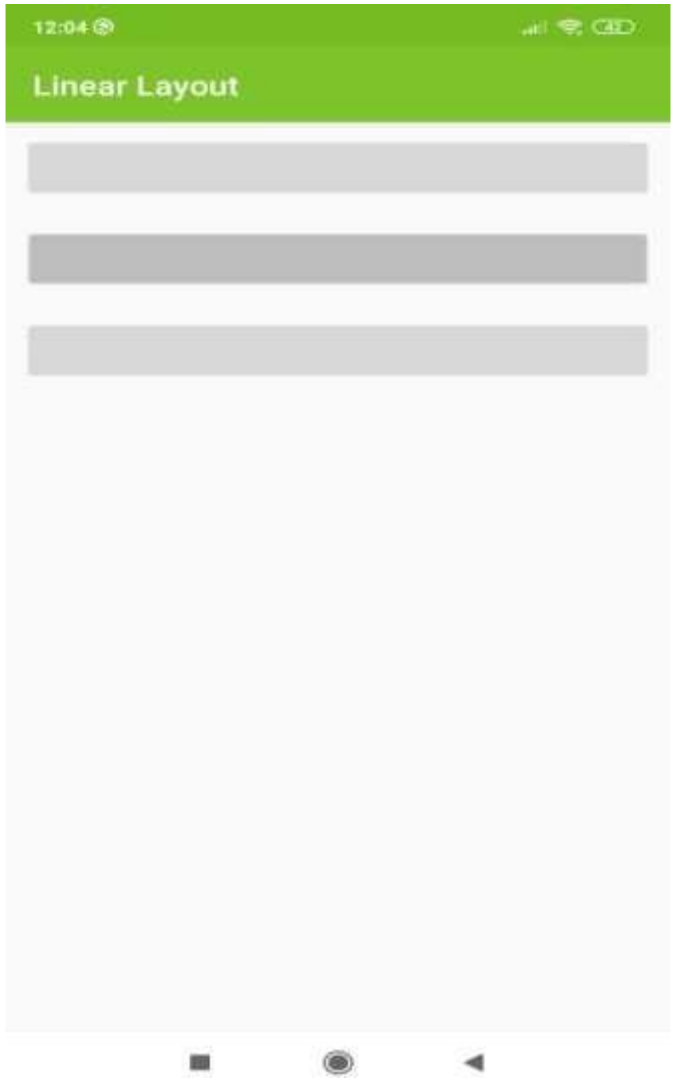


Attributes of Linear Layout

android:layout_weight	It is defined individually to the child's views to specify how Linear Layout **divides the remaining space amongst the views it contains**
android:weightSum	Defines the total weight sum
android:orientation	How the elements should be arranged in the layout. It can be horizontal or vertical.
android:gravity	It specifies how an object should position its content on its X and Y axes. Possible values are – center_vertical, fill, center, bottom, end, etc.
android:layout_gravity	Sets the gravity of the View or Layout relative to its parent. Possible values are – center_vertical, fill, center, bottom, end, etc.
android:baselineAligned	This must be a Boolean value, either "true" or "false" and prevents the layout from aligning its children's baselines.
android:id	This gives a unique id to the layout.

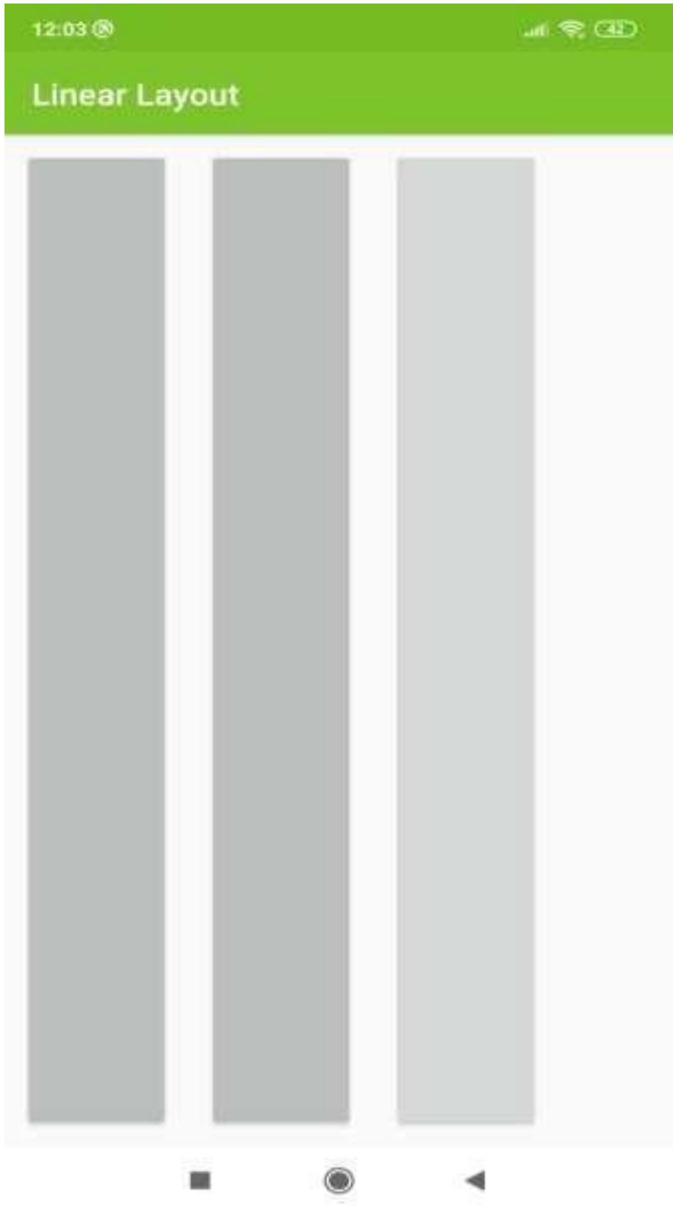
Arrange children views in a vertical manner

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">
    <!-- Add vertical in the android:orientation-->
    <!-- Add Button-->
    <Button
        android:layout_width="match_parent"
        android:layout_margin="10dp"
        android:layout_height="wrap_content"/>
    <!-- Add Button-->
    <Button
        android:layout_width="match_parent"
        android:layout_margin="10dp"
        android:layout_height="wrap_content"/>
    <!-- Add Button-->
    <Button
        android:layout_width="match_parent"
        android:layout_margin="10dp"
        android:layout_height="wrap_content"/>
  </LinearLayout>
```



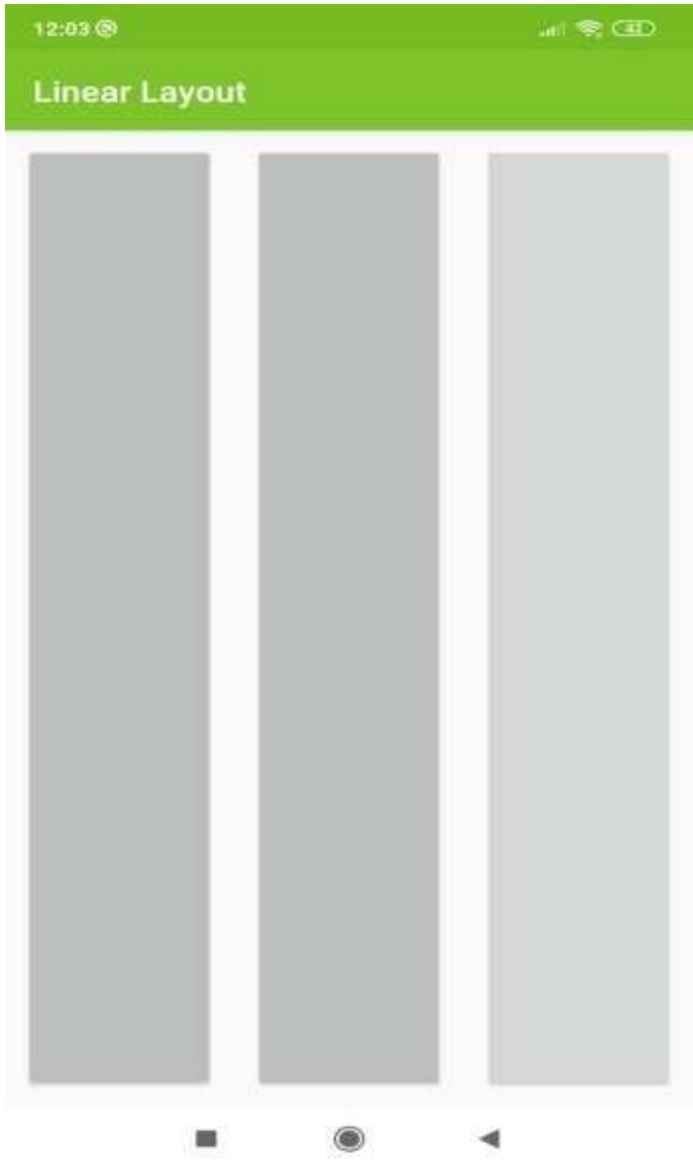
Arrange children views in a horizontal manner

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    tools:context=".MainActivity">
    <!-- Add horizontal in the android:orientation-->
    <!-- Add Button-->
    <Button
        android:layout_width="wrap_content"
        android:layout_margin="10dp"
        android:layout_height="match_parent"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_margin="10dp"
        android:layout_height="match_parent"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_margin="10dp"
        android:layout_height="match_parent"/>
  </LinearLayout>
```



How to use layout_weight and weightSum?

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal"
        android:weightSum="3"
        tools:context=".MainActivity">

        <!-- Add value in the android:weightSum-->
        <!-- Add horizontal in the android:orientation-->
        <!-- Add Button-->
        <!-- Add value in the android:layout_weight-->
        <Button
                android:layout_width="wrap_content"
                android:layout_height="match_parent"
                android:layout_margin="10dp"
                android:layout_weight="1" />
        <!-- Add Button-->
        <!-- Add value in the android:layout_weight-->
        <Button
                android:layout_width="wrap_content"
                android:layout_height="match_parent"
                android:layout_margin="10dp"
                android:layout_weight="1" />
        <!-- Add Button-->
        <!-- Add value in the android:layout_weight-->
        <Button
                android:layout_width="wrap_content"
                android:layout_height="match_parent"
                android:layout_margin="10dp"
                android:layout_weight="1" />
</LinearLayout>
```



16 Write a program that illustrates Relative Layout and discuss about different attributes used in Relative layout.

Relative Layout

This layout is for **specifying the position of the elements in relation to the other elements** that are present there.

In Relative Layout, you **can use attributes "above, below, left and right"** to arrange the child view's position in relation to other child view components.

The position of each view can be **specified as relative to Parent elements or relative to the Child elements**. RelativeLayout is the most used layout after LinearLayout in android because RelativeLayout can be used to position our views based on the relative or sibling component's position.

To define alignment of the position of the **elements to the parent container**

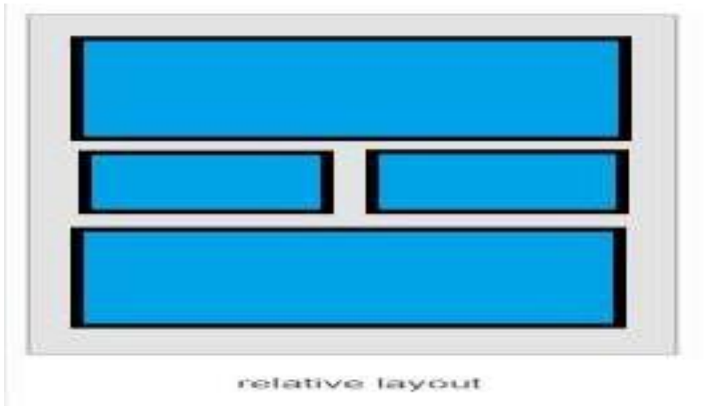
- **android:layout_alignParentTop= "true"**
- **android:layout_alignParentLeft= "true"**

The above code, the element will get aligned on
the **top left of the parent container.**

If we want to align it **with some other element in the same container**

- **android:layout_alignLeft= "@+id/element_name"**
- **android:layout_below= "@+id/element_name"**

This will **align the element below the other element to its left.**



There are so many properties that are supported by relative layout.
Some of the most used properties are listed below.

layout_alignParentTop
layout_alignParentBottom
layout_alignParentRight
layout_alignParentLeft
layout_centerHorozontal
layout_centerVertical
layout_above
layout_below

Example on Relative Layout

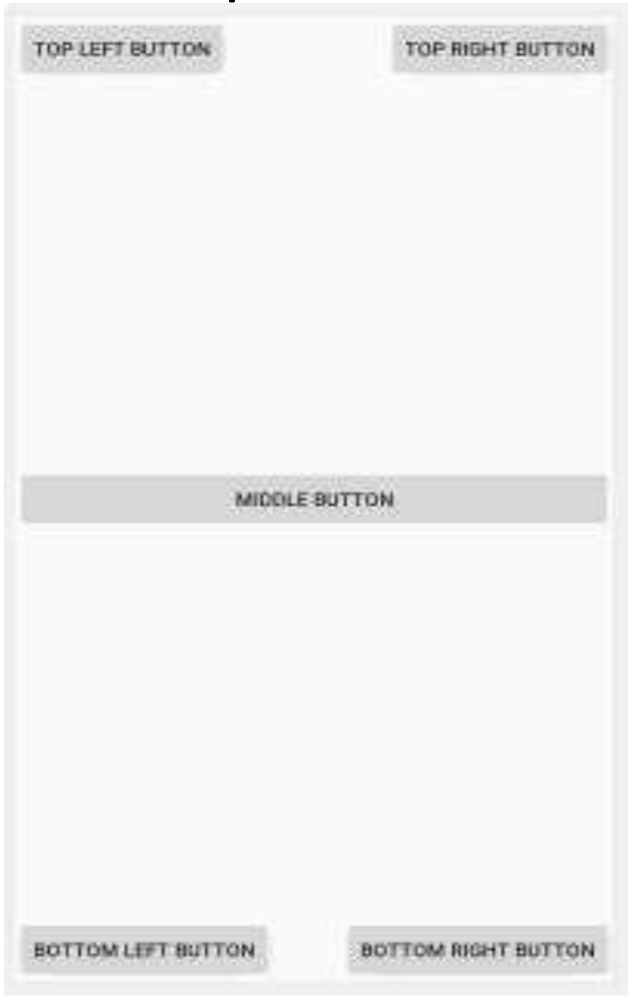
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        xmlns:android="http://schemas.android.com/apk/res/android">
<Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="Top
        Left Button"  android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"/>
<Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Right Button"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true"/>
<Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"  android:text="Bottom
        Left Button"  android:layout_alignParentLeft="true"
        android:layout_alignParentBottom="true"/>
```

```
<Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom Right Button"
        android:layout_alignParentRight="true"
        android:layout_alignParentBottom="true"/>
<Button
        android:id="@+id/button5"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Middle Button"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true"/>

</RelativeLayout>
```

MainActivity.java

```
package com.example.example1;
import androidx.appcompat.app.AppCompatActivity; import
android.os.Bundle;

public class MainActivity extends AppCompatActivity {


        @Override

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

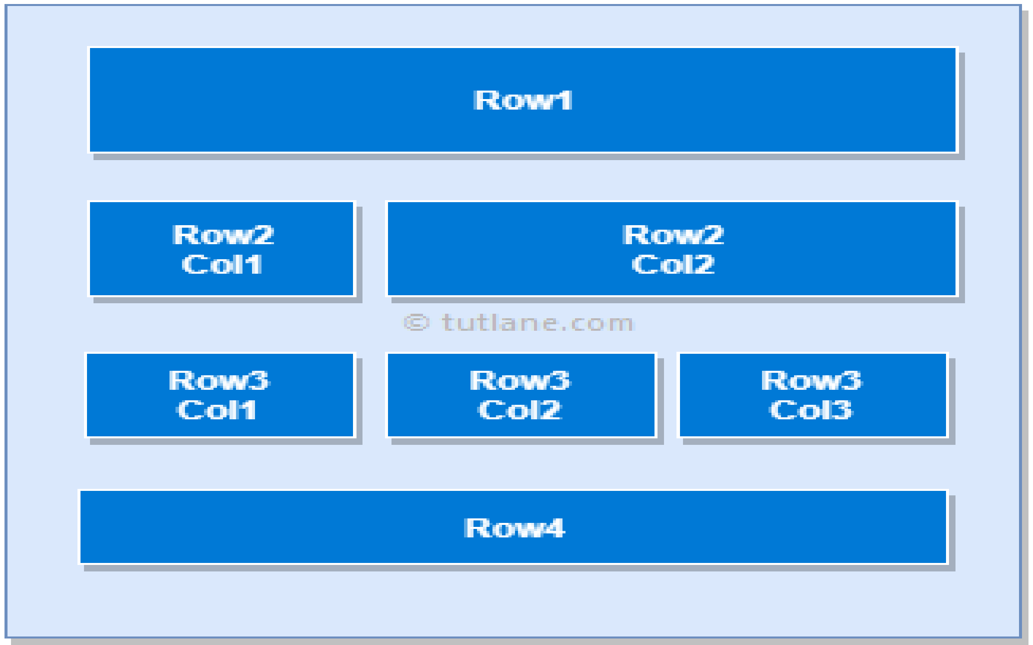
Output:



17 Explain Table Layout with example program in detail. Discuss about attributes used in Table Layout.

Table Layout

- In android, Table Layout is a ViewGroup subclass that is used to display the child View elements in rows and columns.
- In android, Table Layout will position its children elements into rows and columns and it won't display any border lines for rows, columns or cells.
- The Table Layout in android will work same as the HTML table and the table will have as many columns as the row with the most cells.
- The Table Layout can be explained as <TableLayout> and TableRow is like <tr> element.
- Following is the pictorial representation of table layout in android applications.



Some of the common attributes in TableLayout are:

Attributes	Where it is used	Why it is used
android:stretchColumns	TableLayout	When a column width is less and you need to expand(or stretch) it, you use this attribute.
android:shrinkColumns	TableLayout	When you do not need the extra space in a column, you can use this attribute to shrink and remove the space.
android:collapseColumns	TableLayout	It hides the column of the given index in the TableLayout.
android:layout_span	Any View inside the Table Row	If a view takes only one column width but when you want your view to take more than one column space, then you can use this attribute.
android:layout_column	Any view inside the TableRow	When you want your view present in the first TableRow to appear below the other TableRow's view, you can use this attribute.

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"  android:layout_marginTop="100dp"
   android:paddingLeft="10dp"
   android:paddingRight="10dp" >

   <TableRow android:background="#0079D6" android:padding="5dp">
      <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_weight="1"
         android:text="UserId" />
      <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_weight="1"
         android:text="User Name" />
      <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_weight="1"
         android:text="Location" />
   </TableRow>
```

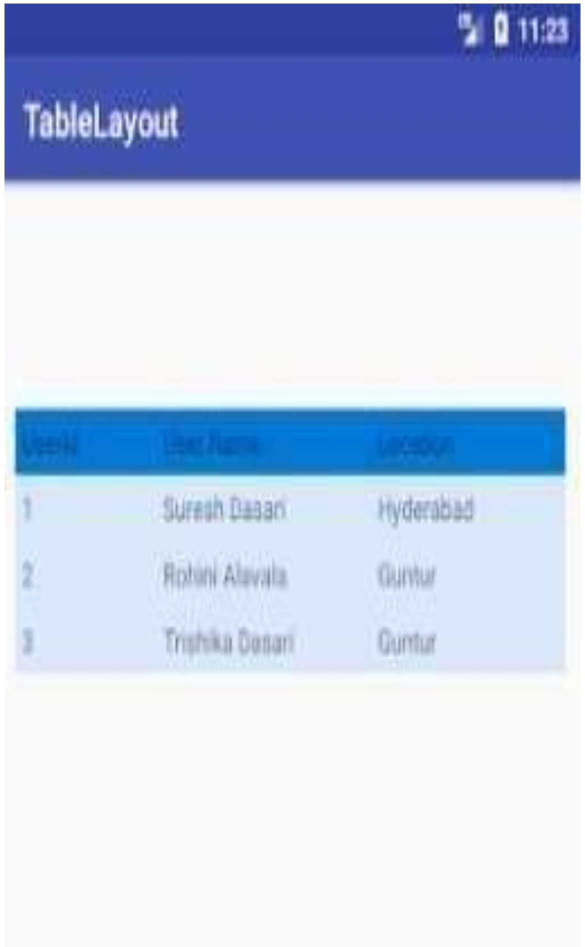
```
<TableRow android:background="#DAE8FC" android:padding= "5dp">
        <TextView  android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1" android:text="1" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1" android:text="Suresh
            Dasari" />
        <TextView  android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"  android:text="Hyderabad"
            />
    </TableRow>
<TableRow android:background="#DAE8FC" android:padding="5dp">

       <TextView  android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:layout_weight="1"  android:text="2" />
       <TextView  android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:layout_weight="1"  android:text="Rohini
           Alavala" />
       <TextView  android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:layout_weight="1"
           android:text="Guntur" />
</TableRow>
```

```
<TableRow android:background="#DAE8FC" android:padding="5dp">

    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1" android:text="3" />
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Trishika Dasari" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Guntur" />


    </TableRow>
  </TableLayout>
```

MainActivity.java

```
package com.example.linearlayout;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

    }
}
```

Output:



UserId	User Name	Location
1	Suresh Dasari	Hyderabad
2	Rohini Alavala	Guntur
3	Trishika Dasari	Guntur

18 What is Grid Layout? Explain Grid Layout with example program in detail.

Grid Layout

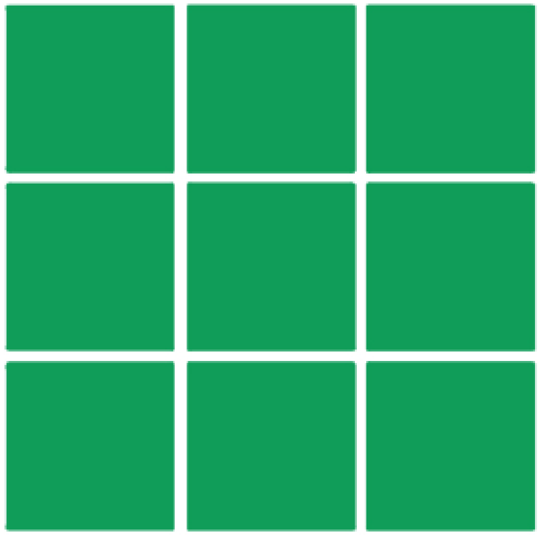
- Android GridLayout is used to display elements and views in the form of a rectangular grid.

- GridLayout and GridView are two completely different terms and are used for other purposes.

- GridView is a view, whereas GridLayout is a layout that can hold various views in it. Using GridView, you can display the items that are coming from the Adapter.

- GridLayout allows you to place many views and elements inside the layout and then style them. It allows you to set the number of rows and columns for your grid.

- The number of rows and columns within the grid can be declared using the android:rowCount and android:columnCount properties.



Attributes of GridView

•**android:numColumns**: This attribute of GridView will be used to decide the number of columns that are to be displayed in Grid.

android:numColumns="2"

•**android:horizontalSpacing**: This attribute is used to define the spacing between two columns of GridView.

android:horizontalSpacing="6dp"

•**android:verticalSpacing**: This attribute is used to specify the spacing between two rows of GridView.

android:verticalSpacing="6dp"

Attributes of Android GridLayout:

android:alignmentMode – It sets the alignment of the margins and the boundaries.

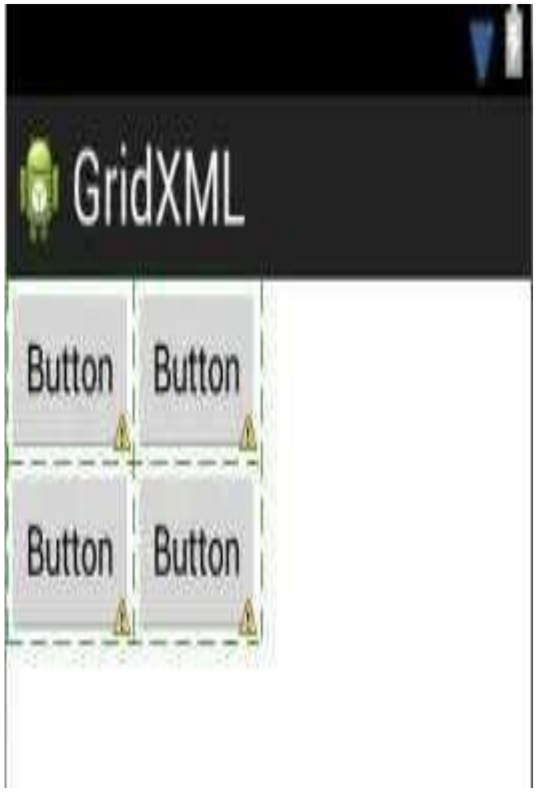
android:columnCount – It sets themaximum number of columns to create when positioning children automatically. **android:rowCount –** It set the maximum no. of rows to create when positioning children automatically.

android:useDefaultMargins – It tells the GridLayout to use margins if no margin is specified in the parameter. For this, it needs to be set on True.

```
<GridLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/GridLayout1" android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:columnCount="2"
android:rowCount="2"
android:orientation="horizontal“
tools:context=".GridXMLActivity" >

 <Button
 android:id="@+id/button1“
 android:layout_gravity="left|top“
 android:text="Button" />

 <Button
android:id="@+id/button2"
android:layout_gravity="left|top"
android:text="Button" />
```

```
<Button android:id="@+id/button3“
 android:layout_gravity="left|top“
 android:text="Button" />

 <Button android:id="@+id/button4“
android:layout_gravity="left|top"
android:text="Button" />


 </GridLayout>
```

Output:

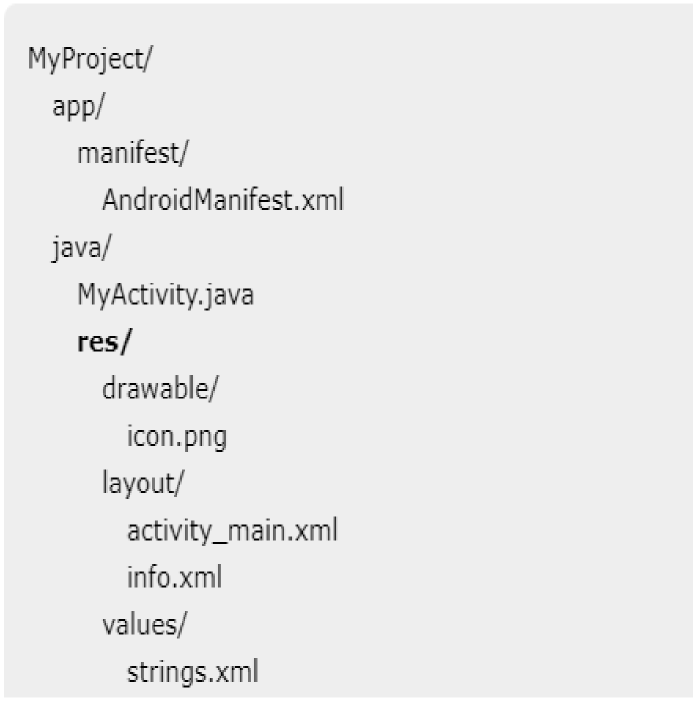


19 What is the use of Externalization Resources in Android? Discuss on color.xml and String.xml in detail with example for each.

External Resources in Android

- It's always good practice to keep **non-code resources like images and string constants** external to your code.
- Android supports the **externalization of resources ranging from simple values such as strings and colors to more complex resources like images (Drawables), animations, and themes.**
- These resources are always maintained separately in various sub-directories under **res/ directory** of the project.
- You should also provide alternative resources for specific device configurations, by grouping them in specially-named resource directories.
- At runtime, Android uses the appropriate resource based on the current configuration. For example, you might want to provide a different UI layout depending on the screen size or different strings depending on the language setting.

Organize resource in Android Studio



```
MyProject/
  app/
    manifest/
      AndroidManifest.xml
  java/
    MyActivity.java
    res/
      drawable/
        icon.png
      layout/
        activity_main.xml
        info.xml
      values/
        strings.xml
```

colors.xml

The colors.xml file is responsible to **hold all the types of colors required for the application**. It may be primary brand color and its variants and secondary brand color and its variants. The colors help uphold the brand of the applications.

The colors need to be **defined in the hex code format**.

We can define some colors in a resource file, then apply those colors elsewhere in our app. By convention, colors are defined in a colors.xml file. Colors are considered “value

Add the below lines inside the **colors.xml** file.

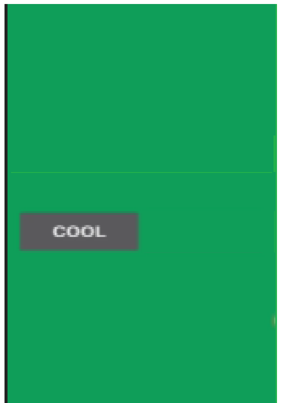
```
<color name="colorPrimary">#6200EE</color>

<color name="colorPrimaryDark">#3700B3</color>

<color name="colorAccent">#03DAC5</color>

<color name="green">#0F9D58</color>

<color name="cool">#188FCF</color>

<color name="warm">#F1D416</color>
```

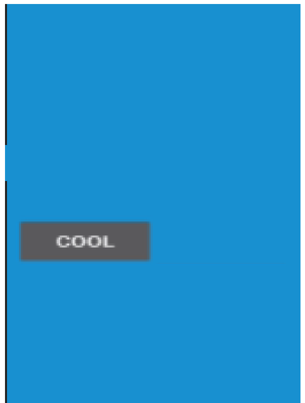
activity_main.xml file

```
<RelativeLayout
   xmlns:android="http://schemas.android.
com/apk/res/android"
   xmlns:tools="http://schemas.android.co
m/tools"
   android:id="@+id/rlVar1"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:background="@color/g
reen"
   tools:context=".MainActivity">
  <Button
         android:id="@+id/btVar1"
         android:layout_width="150dp"
         android:layout_height="wrap_cont

ent"
         android:padding="20dp"
         android:text="Cool"
         android:textSize="25dp"
         android:onClick=change()/>
RelativeLayout>
```

MainActivity.java file

```
public class MainActivity extends
AppCompatActivity {
    @Override
 protected void onCreate(Bundle
savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main
);
 RELayout = findViewById(R.id.rlVar1);
public void change(View view)
{
              // set the color to relative layout
RELayout.setBackgroundResource(
R.color.cool);
  }
```





String.xml

String.xml file contains all the strings which will be **used frequently in Android project**.
String.xml file present in the values folder which is **sub folder of res folder** in project structure.
In Android Studio, we have many Views such as TextView,Button, EditText, CheckBox, RadioButton etc.
These views have a common attribute named text. **we can assign value to this attribute directly or via string.xml.**
In large applications, we use **some text many times in our projec**t so rather than define that particular text many times. it **is best to store in string.xml file and call it by its string name in project.**
For example: Suppose we have a text:**-** "Hey How are You"
and print them many times
Best way is to store this text in string.xml once and call string wherever required. This will save a lot of efforts and less line of code.

How can we use String Array ?

Step1
Go to String.xml and add the below xml code
inside the <resource></resource> tag.

String.xml

```
<string-array
name="country">
   <item>India</item>
   <item>USA</item>
   <item>Africa</item>
   <item>Norway</item>
</string-array>
```

Activity_main.xml

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android
"
  xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical"
    android:padding="20dp">
     <Spinner
        android:id="@+id/spinner"
        android:layout_width="227dp"
        android:layout_height="97dp"
         android:entries="@array/country"  />

<TextView
         android:id="@+id/tv"
         android:layout_width="285dp"
         android:layout_height="198dp"
         android:text="TextView"
         android:padding="20dp"
         android:layout_marginTop="50dp"/>
</LinearLayout>
```

mainActivity.java

```
public class MainActivity extends AppCompatActivity implements
AdapterView.OnItemSelectedListener
{
        @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Spinner sp = findViewById(R.id.spinner);
         sp.setOnItemSelectedListener(this);
    }
    @Override
 public void onItemSelected(AdapterView<?> adapterView, View view, int i, long
l)
{
     TextView tvobj =findViewById(R.id.tv);
    String txt = adapterView.getItemAtPosition(i).toString();
        tvobj.setText(txt);
    }
    @Override
    public void onNothingSelected(AdapterView<?> adapterView) {
    }
```

20 a) Explain how to add Images in Android application discuss with example?(4M)

Drawable Folder

- When you need to display static images in your app, you can use the Drawable class and its subclasses to draw shapes and images.

- You can add graphics to your app by referencing an image file from your project resources. Supported file types are PNG (preferred), JPG (acceptable), and GIF (discouraged). App icons, logos, and other graphics, such as those used in games, are well suited for this technique.

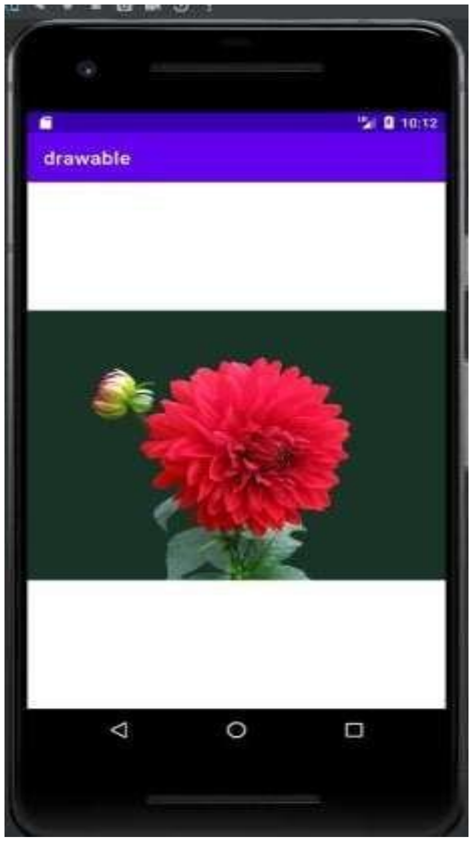
- To use an image resource, add your file to the res/drawable/ directory of your project. Once in your project, you can reference the image resource from your code or your XML layout.

- **How to add image in an application?**

Right click on the drawable folder and select paste option.

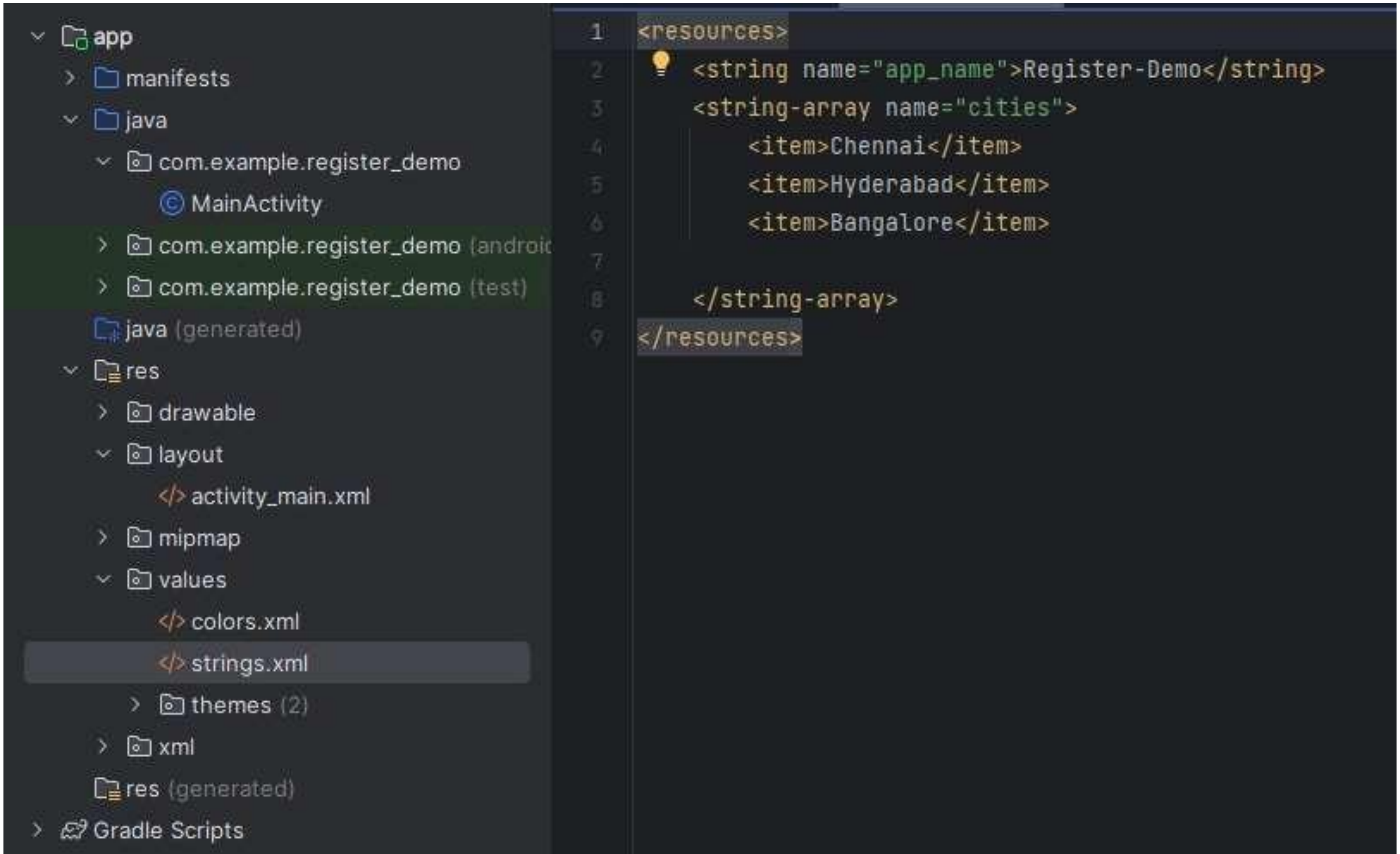
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

<ImageView
   android:src="@drawable/flower"
   android:layout_centerVertical="true"
   android:layout_centerHorizontal="true"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content">
</ImageView>

</RelativeLayout>
```

Output:



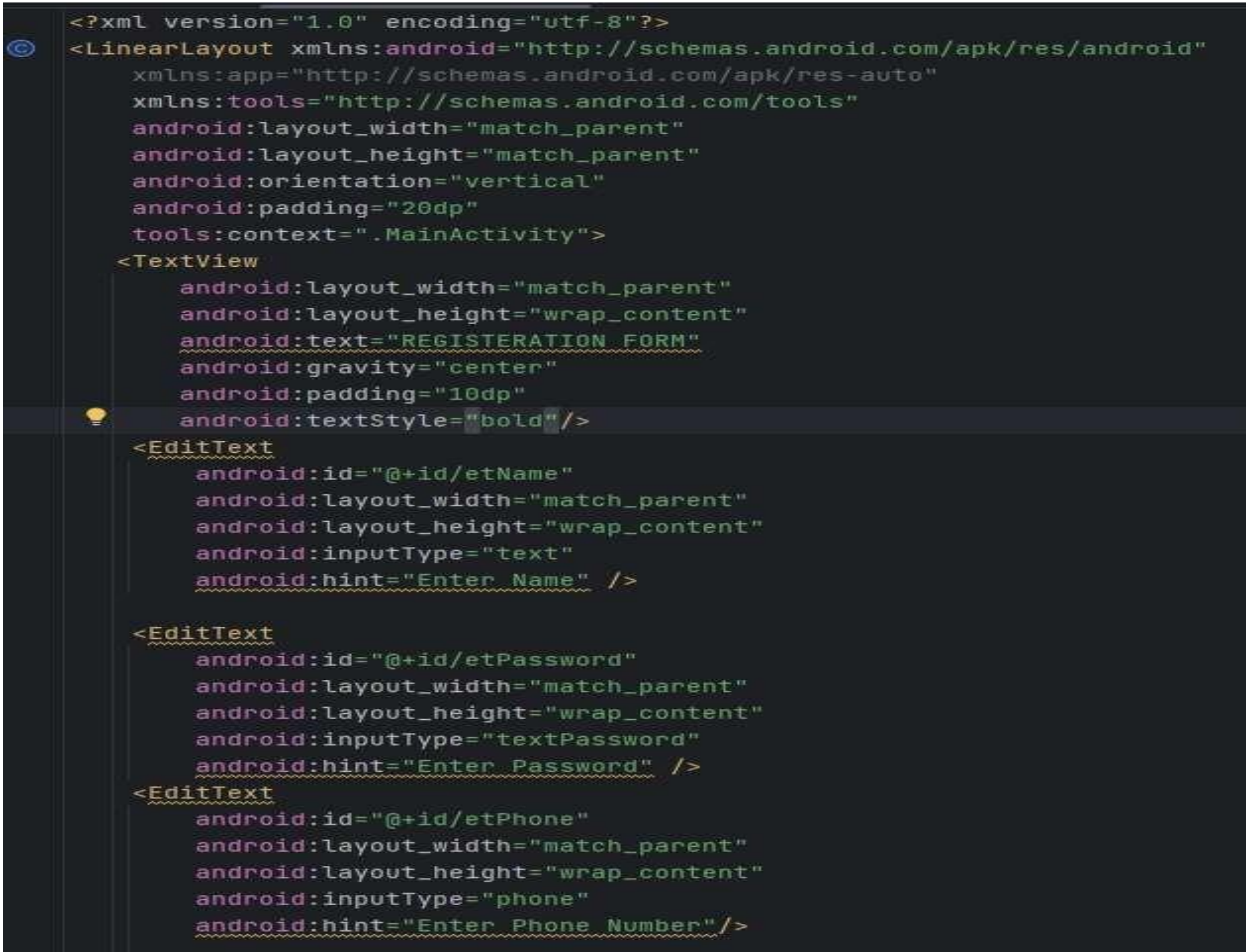
20b)Design student Registration form with fields username,password,age,address and button,display the details of student when user clicks Submit button.(8M)

Strings.xml

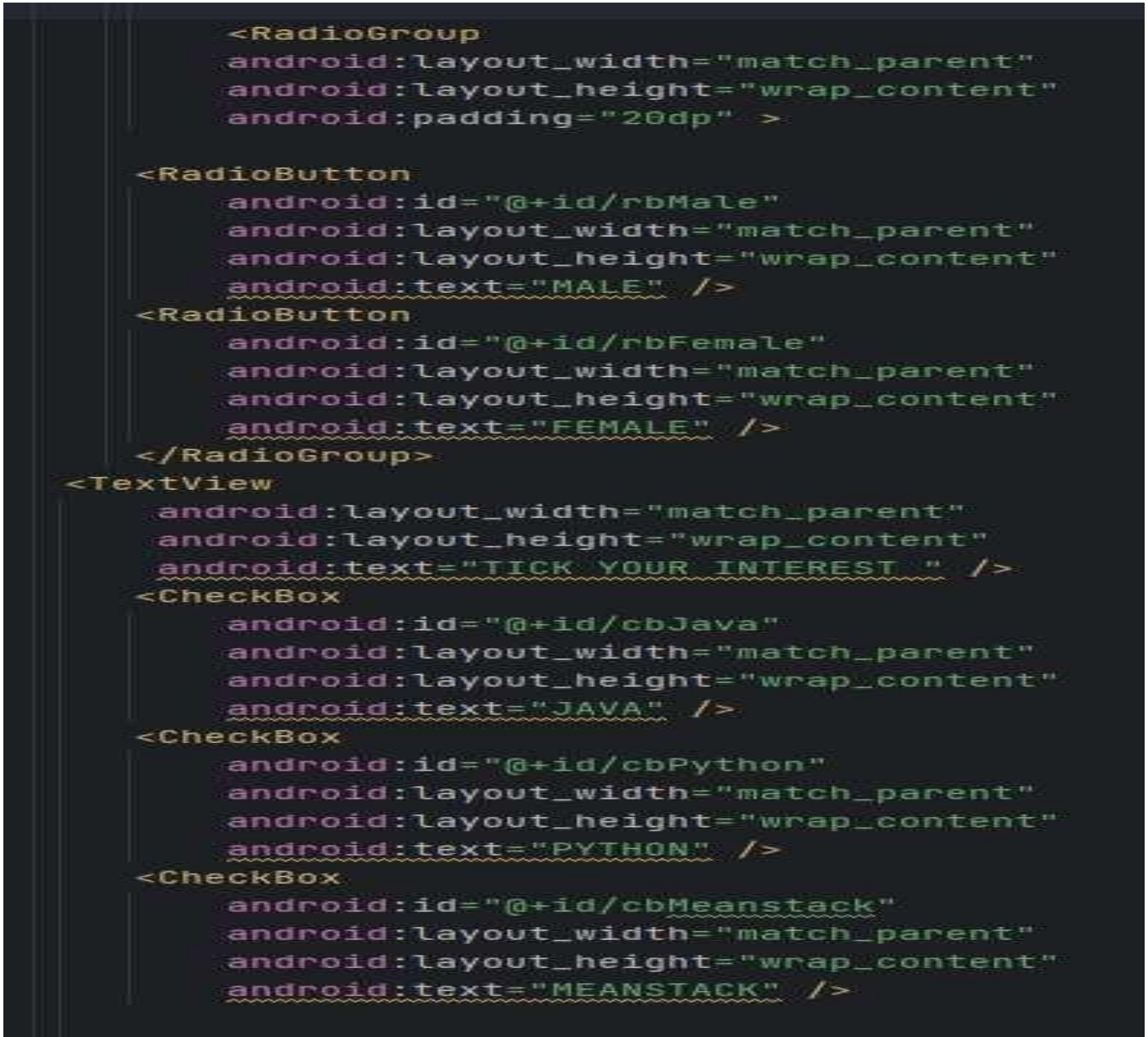


```xml
<resources>
    <string name="app_name">Register-Demo</string>
    <string-array name="cities">
        <item>Chennai</item>
        <item>Hyderabad</item>
        <item>Bangalore</item>

    </string-array>
</resources>
```

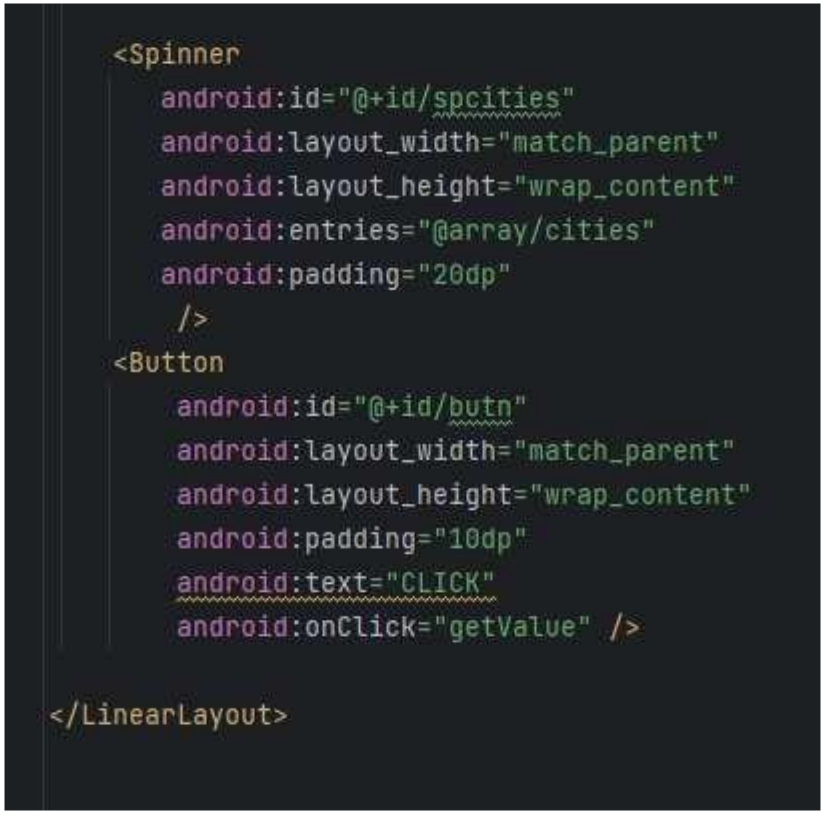
activity_main.xml



```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="20dp"
    tools:context=".MainActivity">
   <TextView
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:text="REGISTERATION FORM"
       android:gravity="center"
       android:padding="10dp"
       android:textStyle="bold"/>
    <EditText
        android:id="@+id/etName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="text"
        android:hint="Enter Name" />

    <EditText
        android:id="@+id/etPassword"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textPassword"
        android:hint="Enter Password" />
    <EditText
        android:id="@+id/etPhone"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="phone"
        android:hint="Enter Phone Number"/>
```

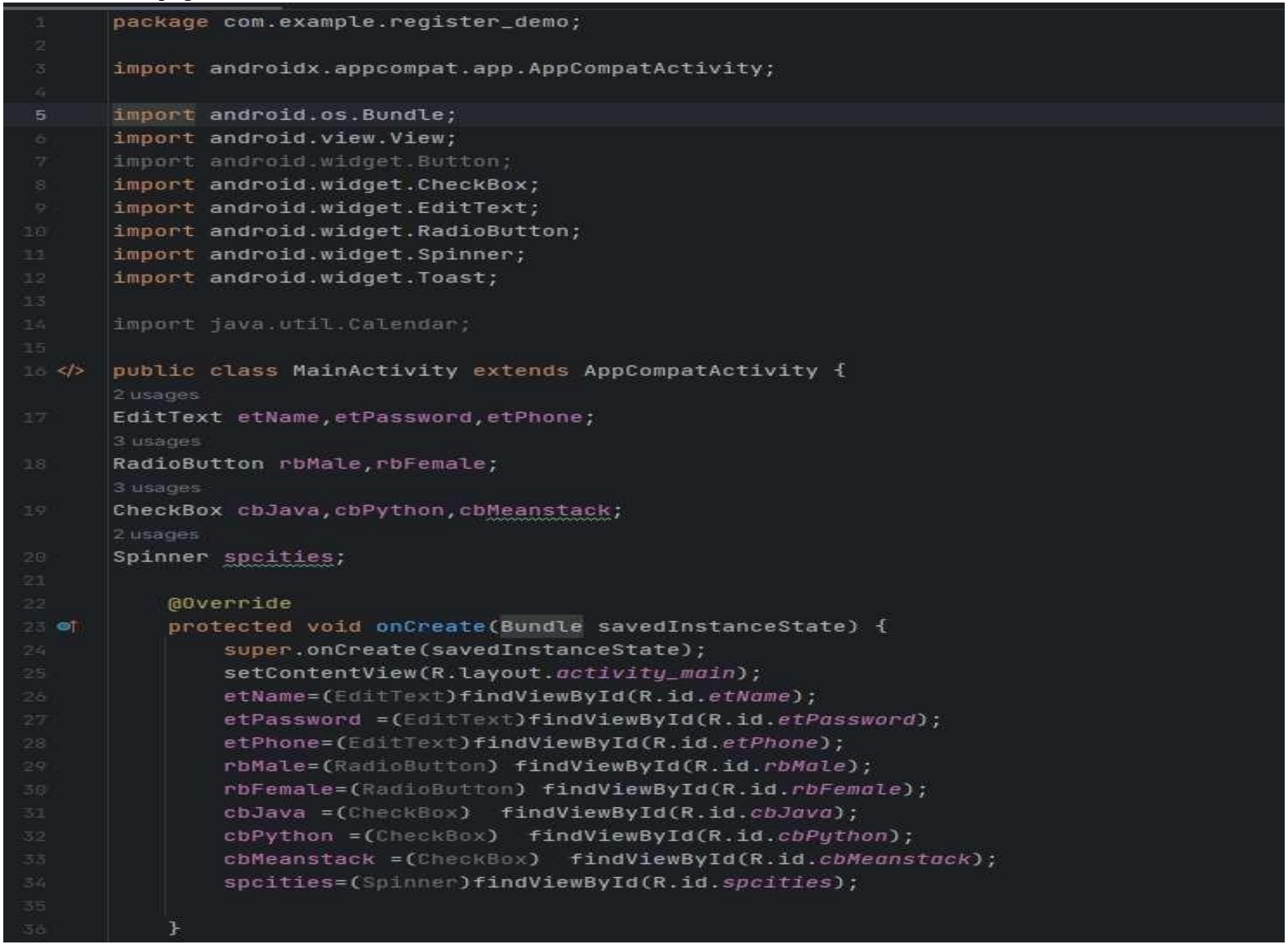


```
        <RadioGroup
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="20dp" >

    <RadioButton
        android:id="@+id/rbMale"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="MALE" />
    <RadioButton
        android:id="@+id/rbFemale"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="FEMALE" />
    </RadioGroup>
<TextView
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:text="TICK YOUR INTEREST " />
    <CheckBox
        android:id="@+id/cbJava"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="JAVA" />
    <CheckBox
        android:id="@+id/cbPython"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="PYTHON" />
    <CheckBox
        android:id="@+id/cbMeanstack"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="MEANSTACK" />
```

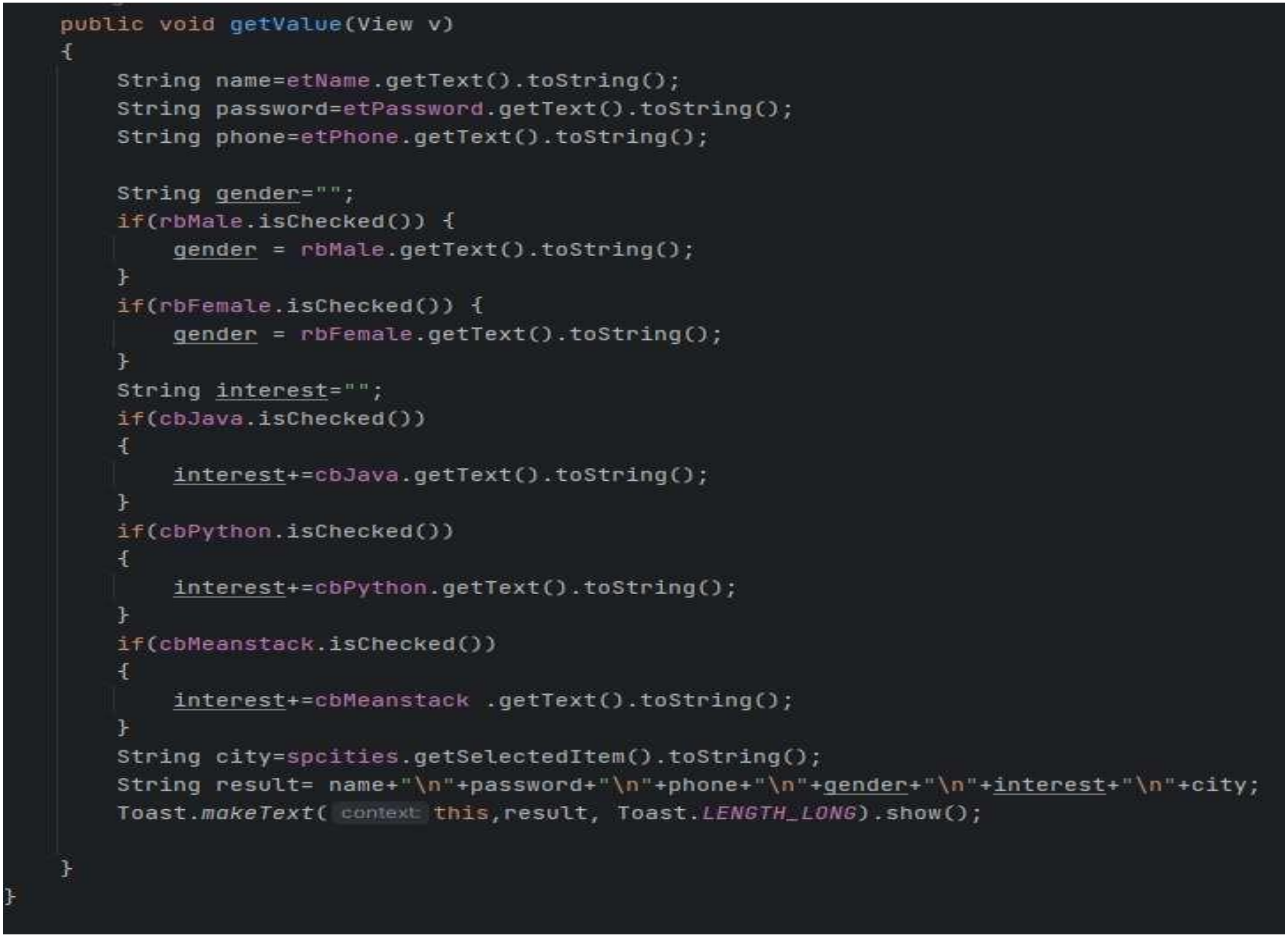


```
    <Spinner
        android:id="@+id/spcities"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:entries="@array/cities"
        android:padding="20dp"
         />
    <Button
         android:id="@+id/butn"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:padding="10dp"
         android:text="CLICK"
         android:onClick="getValue" />

</LinearLayout>
```

MainActivity.java

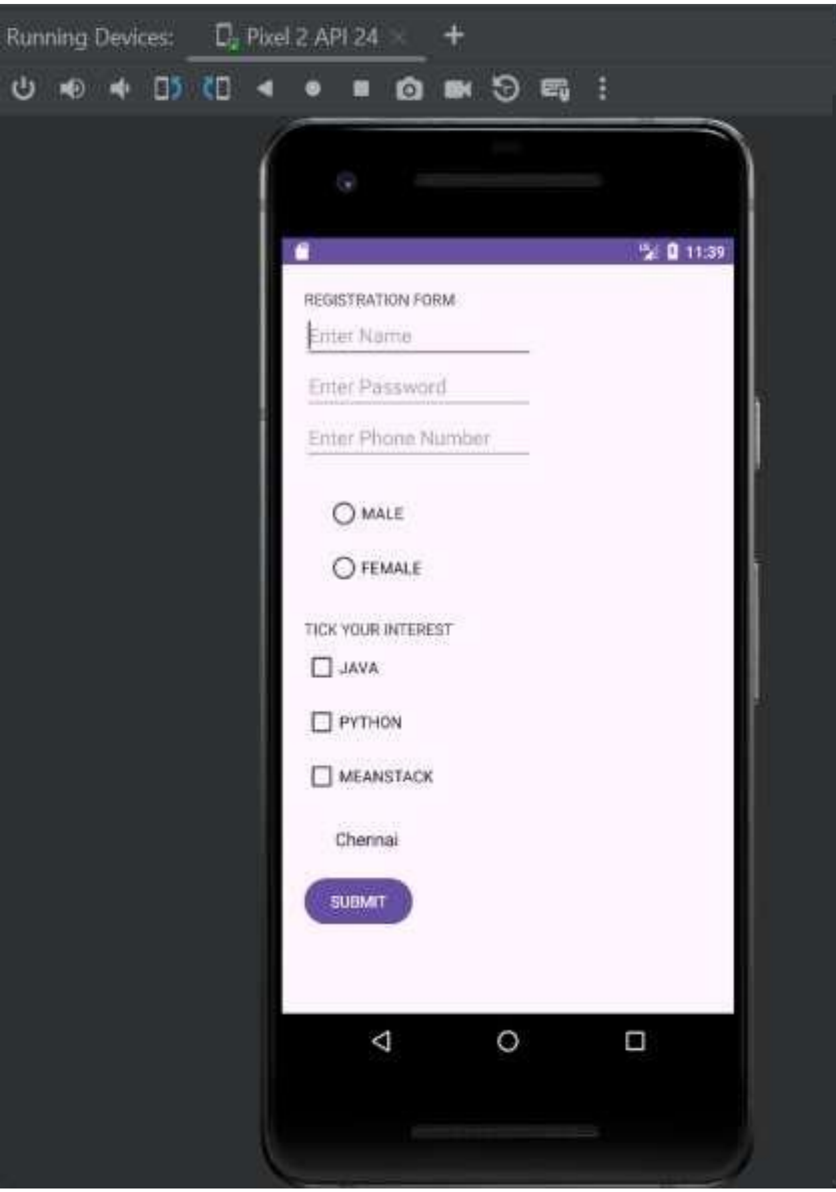


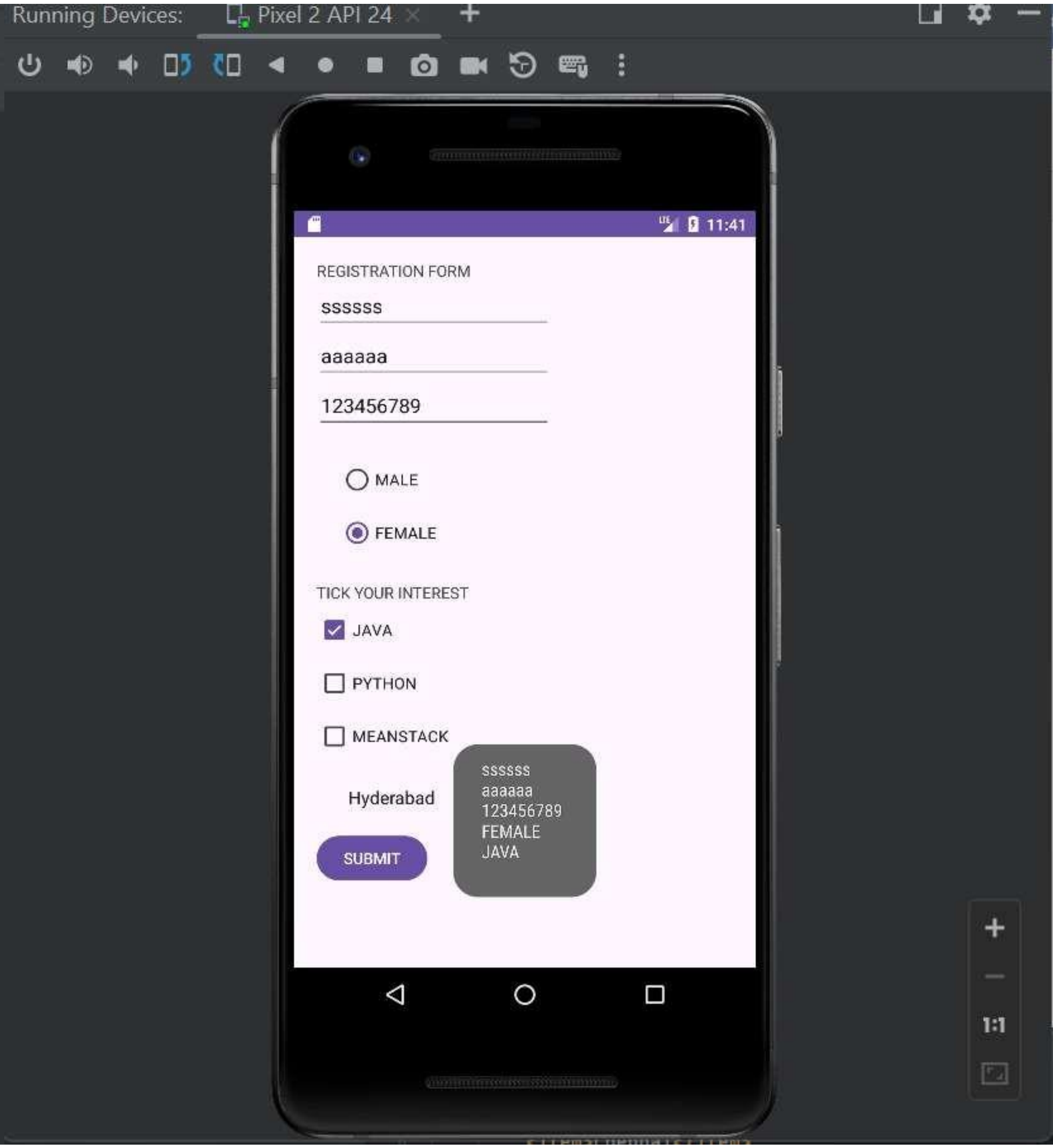
```java
package com.example.register_demo;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.EditText;
import android.widget.RadioButton;
import android.widget.Spinner;
import android.widget.Toast;

import java.util.Calendar;

public class MainActivity extends AppCompatActivity {
2 usages
EditText etName,etPassword,etPhone;
3 usages
RadioButton rbMale,rbFemale;
3 usages
CheckBox cbJava,cbPython,cbMeanstack;
2 usages
Spinner spcities;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        etName=(EditText)findViewById(R.id.etName);
        etPassword =(EditText)findViewById(R.id.etPassword);
        etPhone=(EditText)findViewById(R.id.etPhone);
        rbMale=(RadioButton) findViewById(R.id.rbMale);
        rbFemale=(RadioButton) findViewById(R.id.rbFemale);
        cbJava =(CheckBox)  findViewById(R.id.cbJava);
        cbPython =(CheckBox)  findViewById(R.id.cbPython);
        cbMeanstack =(CheckBox)  findViewById(R.id.cbMeanstack);
        spcities=(Spinner)findViewById(R.id.spcities);

    }
```



```
    public void getValue(View v)
    {
        String name=etName.getText().toString();
        String password=etPassword.getText().toString();
        String phone=etPhone.getText().toString();

        String gender="";
        if(rbMale.isChecked()) {
            gender = rbMale.getText().toString();
        }
        if(rbFemale.isChecked()) {
            gender = rbFemale.getText().toString();
        }
        String interest="";
        if(cbJava.isChecked())
        {
            interest+=cbJava.getText().toString();
        }
        if(cbPython.isChecked())
        {
            interest+=cbPython.getText().toString();
        }
        if(cbMeanstack.isChecked())
        {
            interest+=cbMeanstack .getText().toString();
        }
        String city=spcities.getSelectedItem().toString();
        String result= name+"\n"+password+"\n"+phone+"\n"+gender+"\n"+interest+"\n"+city;
        Toast.makeText( context: this,result, Toast.LENGTH_LONG).show();

    }
}
```

OUTPUT





REGISTRATION FORM

ssssss

aaaaaa

123456789

( ) MALE

(●) FEMALE

TICK YOUR INTEREST

[✓] JAVA

[ ] PYTHON

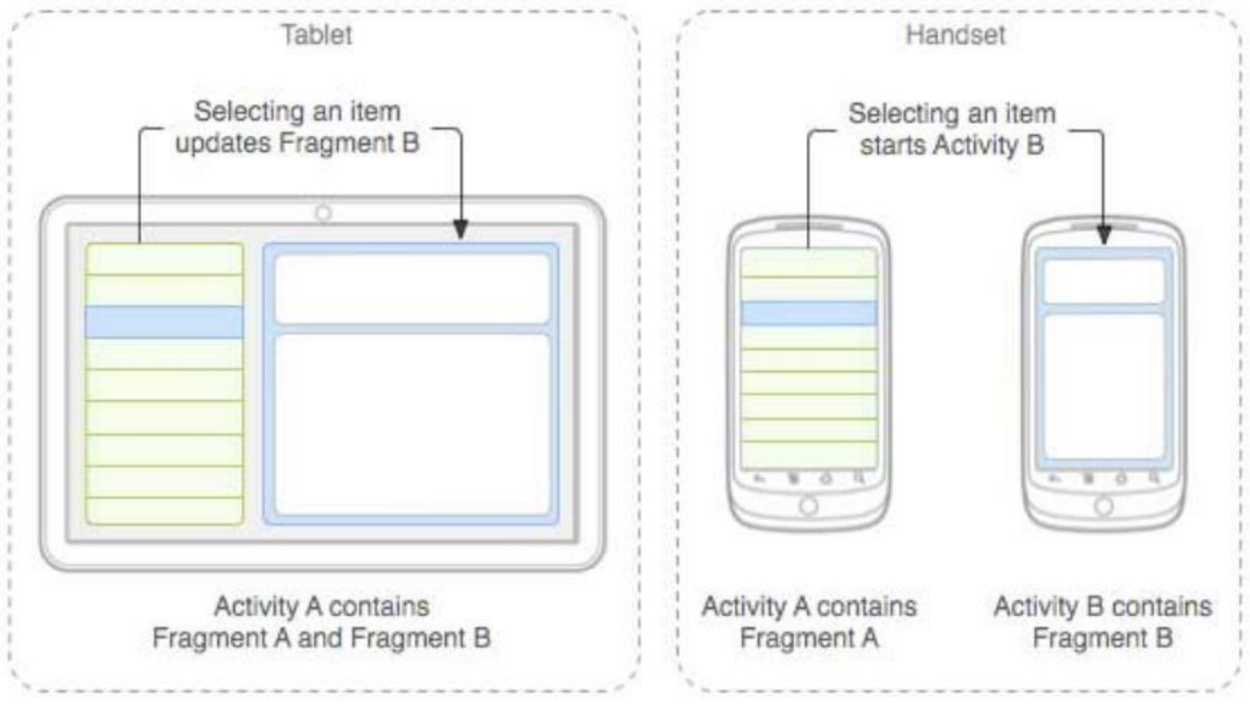
[ ] MEANSTACK

```
ssssss
aaaaaa
123456789
FEMALE
JAVA
```

Unit-3

21a). What is Fragment? Explain the procedure to create Fragment.(6M)

Fragments are used to place two activities on a single activity when creating the layout of the user interface. Fragments can't exist on their own; they require activity or other fragments. The fragment's layout, lifecycle, and input events are managed and defined by themselves. The interaction between fragment objects is managed by the fragment manager class.


Tablet
Selecting an item updates Fragment B
Activity A contains Fragment A and Fragment B
Handset
Selecting an item starts Activity B
Activity A contains Fragment A
Activity B contains Fragment B

There are four steps for making fragments.

1. Extend fragment class
2. Provide appearance in XML /Java
3. Override onCreateView to link the appearance
4. Use the fragment in XML / Java

Following are important points about fragment −

- A fragment has its own layout and its own behaviour with its own life cycle callbacks.
- You can add or remove fragments in an activity

while the activity is running.

- You can combine multiple fragments in a single activity to build a multi-pane UI.
- A fragment can be used in multiple activities.
- Fragment life cycle is closely related to the life cycle of its host activity which means when the activity is paused, all the fragments available in the activity will also be stopped.
- A fragment can implement a behaviour that has no user interface component.
- Fragments were added to the Android API in Honeycomb version of Android which API version 11.

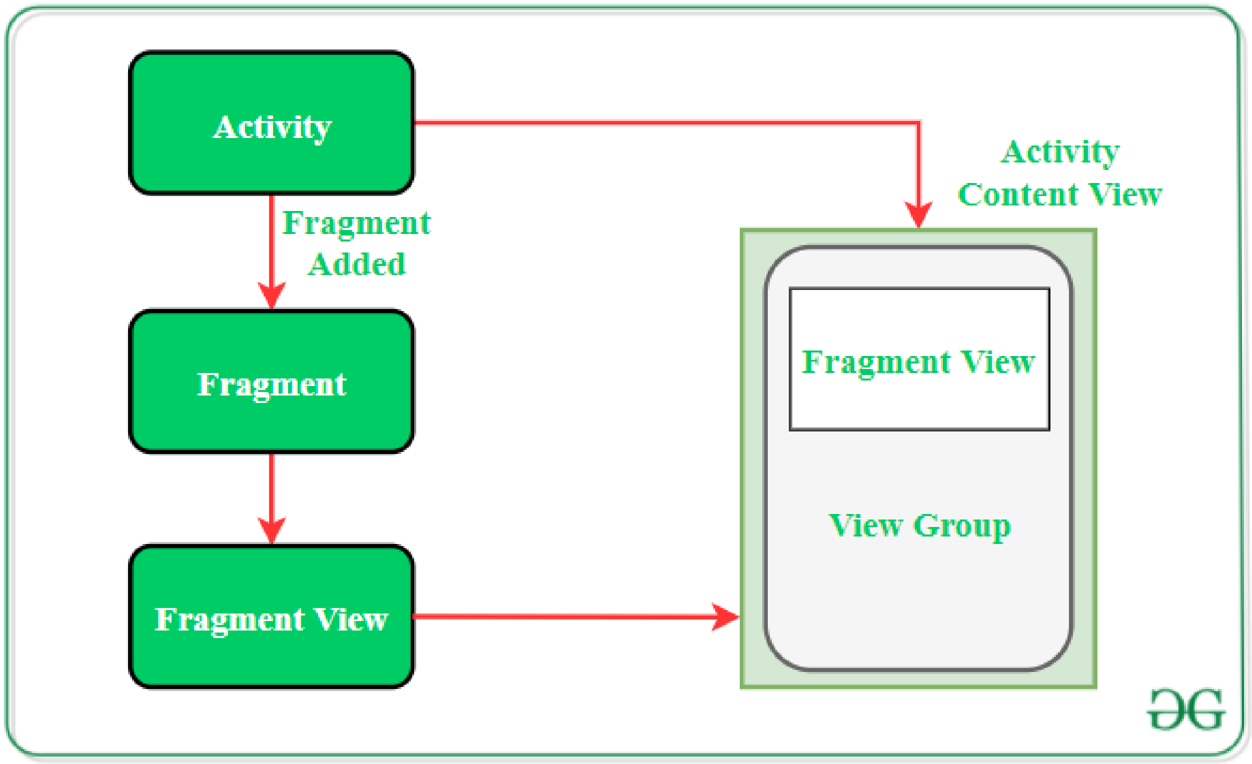
You create fragments by extending Fragment class and You can insert a fragment into your activity layout by declaring the fragment in the activity's layout file, as a <fragment> element.

b) Distinguish between Activity and Fragment. (6M)

Activity	Fragment
Activity is an application component that gives a user interface where the user can interact.	The fragment is only part of an activity, it basically contributes its UI to that activity.
Activity is not dependent on fragment	Fragment is dependent on activity. It can't exist independently.
we need to mention all activity it in the manifest.xml file	Fragment is not required to mention in the manifest file
We can't create multi-screen UI without using fragment in an activity,	After using multiple fragments in a single activity, we can create a multi-screen UI.
Activity can exist without a Fragment	Fragment cannot be used without an Activity.

Creating a project using only Activity then it's difficult to manage	While Using fragments in the project, the project structure will be good and we can handle it easily.
Lifecycle methods are hosted by OS. The activity has its own life cycle.	Lifecycle methods in fragments are hosted by hosting the activity.
Activity is not lite weight.	The fragment is the lite weight.

22.Explain lifecycle of Fragment with the help of flowchart

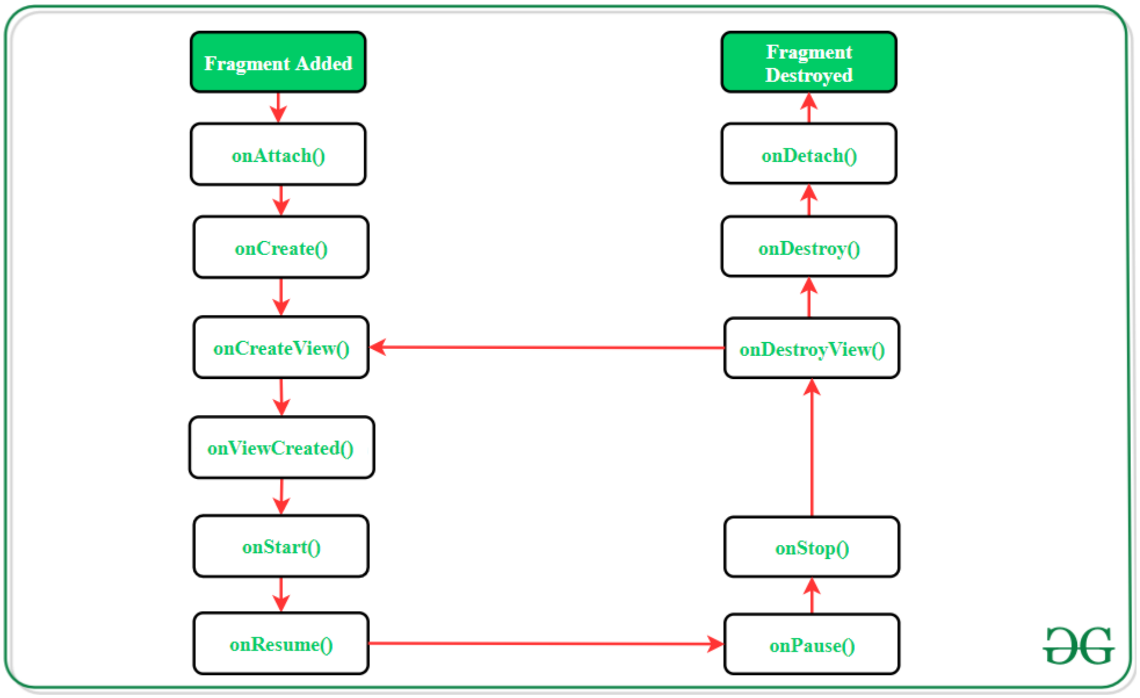


Types of Android Fragments

1. **Single Fragment:** Display only one single view on the device screen. This type of fragment is mostly used for mobile phones.

2. **List Fragment:** This Fragment is used to display a list-view from which the user can select the desired sub-activity. The menu drawer of apps like Gmail is the best example of this kind of fragment.
3. **Fragment Transaction:** This kind of fragments supports the transition from one fragment to another at run time. Users can switch between multiple fragments like switching tabs.

Fragment Lifecycle



Note: *Each fragment has it's own lifecycle but due to the connection with the Activity it belongs to, the fragment lifecycle is influenced by the activity's lifecycle.*

Methods of the Android Fragment

Methods	**Description**
onAttach()	The very first method to be called when the fragment has been associated with the activity. This method executes only once during the lifetime of a fragment. When we attach fragment(child) to Main(parent) activity then it call first and then not call this method any time(like you run an app and close and reopen) simple means that this method call only one time.
onCreate()	This method initializes the fragment by adding all the required attributes and components.
onCreateView()	System calls this method to create the user interface of the fragment. The root of the fragment's layout is returned as the View component by this method to draw the UI. You should inflate your layout in onCreateView but shouldn't initialize other views using findViewById in onCreateView.
onViewCreated()	It indicates that the activity has been created in which the fragment exists. View hierarchy of the fragment also instantiated before this function call.
onStart()	The system invokes this method to make the fragment visible on the user's device.
onResume()	This method is called to make the visible fragment interactive.
onPause()	It indicates that the user is leaving the fragment. System call this method to commit the changes made to the fragment.

onStop()	Method to terminate the functioning and visibility of fragment from the user's screen.
onDestroyView()	System calls this method to clean up all kinds of resources as well as view hierarchy associated with the fragment. It will call when you can attach new fragment and destroy existing fragment Resoruce
onDestroy()	It is called to perform the final clean up of fragment's state and its lifecycle.
onDetach()	The system executes this method to disassociate the fragment from its host activity. It will call when your fragment Destroy(app crash or attach new fragment with existing fragment)

23.a) Write the differences between Frame Layout and Relative Layout.(6M)

`RelativeLayout` and `FrameLayout` are both layout classes in Android used to arrange the views and widgets in a user interface.
`RelativeLayout` arranges the child views relative to each other or to the parent view. It allows you to position views in relation to other views or the parent layout.
`FrameLayout`, on the other hand, is designed to block out an area on the screen to display a single item. It is typically used to hold a single child view, such as a fragment, and is useful when you want to display one view at a time.
In summary, `RelativeLayout` is used for arranging views relative to each other, while `FrameLayout` is used to block out an area for a single view.

FrameLayout is designed to block out an area on the screen to display a single item. Generally, FrameLayout should be used to hold a single child view, because it can be difficult to organize child views in a way that's scalable to different screen sizes without the children overlapping each other. You can, however, add multiple children to a FrameLayout and control their position within the FrameLayout by assigning gravity to each child, using the`android:layout_gravity` attribute.
RelativeLayout is a view group that displays child views in relative positions. The position of each view can be specified as relative to sibling elements (such as to the left-of or below another view) or in positions relative to the parentRelativeLayout area (such as aligned to the bottom, left or center).

(b) Discuss the attributes related to Frame Layout.(6M)

FrameLayout Attributes

Following are the important attributes specific to FrameLayout −

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:foreground** This defines the drawable to draw over the content and possible values may be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
3	**android:foregroundGravity** Defines the gravity to apply to the foreground drawable. The gravity defaults to fill. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
4	**android:measureAllChildren** Determines whether to measure all children or just those in the VISIBLE or INVISIBLE state when measuring. Defaults to false.

24.Discuss about the following classes

a) Fragment Manager (6M)

FragmentManager is the class responsible for performing actions on your app's fragments, such as adding, removing, or replacing them and adding them to the back stack.

Access the FragmentManager

You can access the FragmentManager from an activity or from a fragment.

FragmentActivity and its subclasses, such as AppCompatActivity, have access to the FragmentManager through the getSupportFragmentManager() method.

Fragments can host one or more child fragments. Inside a fragment, you can get a reference to the FragmentManager that manages the fragment's children through getChildFragmentManager(). If you need to access its host FragmentManager, you can use getParentFragmentManager().

Here are a couple of examples to see the relationships between fragments, their hosts, and the FragmentManager instances associated with each.

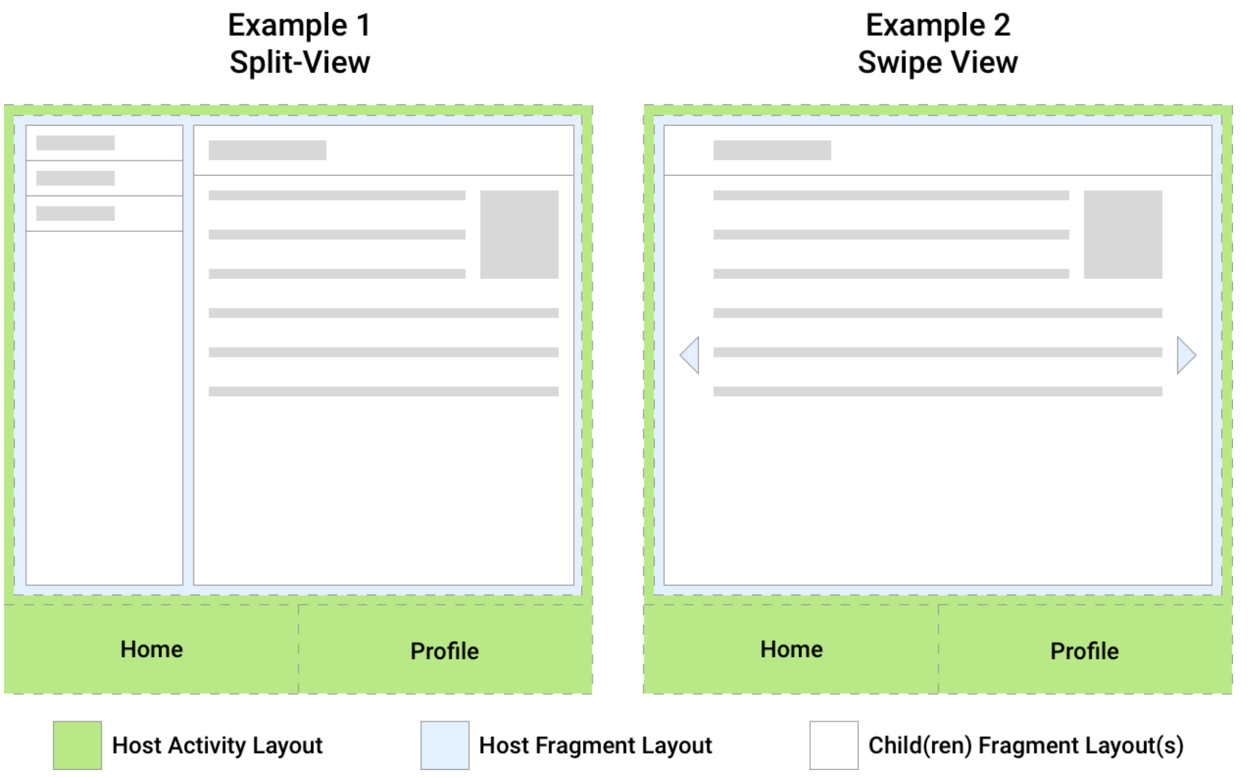


Figure 1. Two UI layout examples showing the relationships between fragments and their host activities.

Figure 1 shows two examples, each of which has a single activity host. The host activity in both of these examples displays top-level navigation to the user as a BottomNavigationView that is responsible for swapping out the host fragment with different screens in the app. Each screen is implemented as a separate fragment.

The host fragment in Example 1 hosts two child fragments that make up a split-view screen. The host fragment in Example 2 hosts a single child fragment that makes up the display fragment of a swipe view.

Given this setup, you can think about each host as having a FragmentManager associated with it that manages its child fragments. This is illustrated in figure 2 along with property mappings between supportFragmentManager, parentFragmentManager, and childFragmentManager.

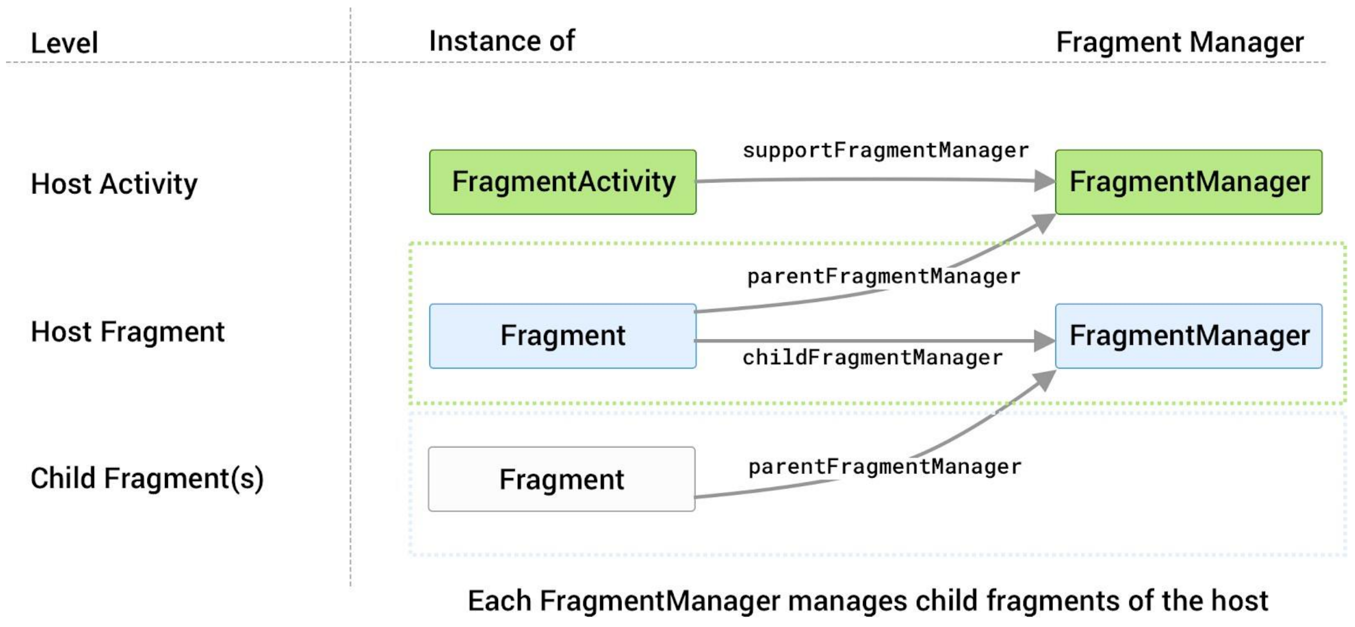


Figure 2. Each host has its own FragmentManager associated with it that manages its child fragments.

The appropriate FragmentManager property to reference depends on where the callsite is in the fragment hierarchy along with which fragment manager you are trying to access.

Once you have a reference to the FragmentManager, you can use it to manipulate the fragments being displayed to the user.

b) Fragment Transaction(6M)

Fragment Transaction: While for an dynamic activity we are set buttons for an interactive UI. If we are set after clicking the button the fragment should appear then we have to get help from Fragment Manager. It handle all the fragment in an activity. We need to set fragment transaction with the help of fragment manager and and begin transaction, and then simply replace the layout of the fragment with desired place.

At runtime, a FragmentManager can add, remove, replace, and perform other actions with fragments in response to user interaction. Each set of fragment changes that you commit is called a *transaction*, and you can specify what to do inside the transaction using the

APIs provided by the FragmentTransaction class. You can group multiple actions into a single transaction—for example, a transaction can add or replace multiple fragments. This grouping can be useful for when you have multiple sibling fragments displayed on the same screen, such as with split views.

You can save each transaction to a back stack managed by the FragmentManager, allowing the user to navigate backward through the fragment changes—similar to navigating backward through activities.

You can get an instance of FragmentTransaction from the FragmentManager by calling beginTransaction(), as shown in the following example:

```
FragmentManager fragmentManager = ...
FragmentTransaction fragmentTransaction =
fragmentManager.beginTransaction();
```

The final call on each `FragmentTransaction` must commit the transaction. The `commit()` call signals to the `FragmentManager` that all operations have been added to the transaction.

```
FragmentManager fragmentManager = ...
FragmentTransaction fragmentTransaction =
fragmentManager.beginTransaction();

// Add operations here

fragmentTransaction.commit();
```

25.Explain the process of interfacing between Fragments and activities.

In most use cases you often find your self trying to communicate between activity and fragment whether to send data back to activity or send an event happened in the hosted fragment, so how can we achieve this?

There are three ways to do this or at least three ways i have tried during my carrier as an android developer and they are all have a similar purpose which to communicate between the two component.

the three ways are :

- Interface
- Instance Method
- Events bus

i will write a simple example for the three ways demonstrating how to achieve this

Interface

1- First define an interface describing the data you want to move,

it could be receive anything

```
interface FragmentCallback {
 fun onNameAdded(name: String)
}
```

or nothing

```
interface FragmentCallback {
    fun onNameAdded()
}
```

it all depends on your use-case

2- Implement the defined interface in your activity

```
class MainActivity : AppCompatActivity(), FragmentCallback {
    override fun onNameAdded(name: String) {
    // here were you recevie the data or event
    }

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

3- Acquire instance of the Interface From Fragment

We use polymorphism to get instace of the implemented interface in MainActivity, read more https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html

```
private lateinit var fragmentCallback: FragmentCallback
override fun onAttach(context: Context?) {
super.onAttach(context)
fragmentCallback = context as FragmentCallback
}
```

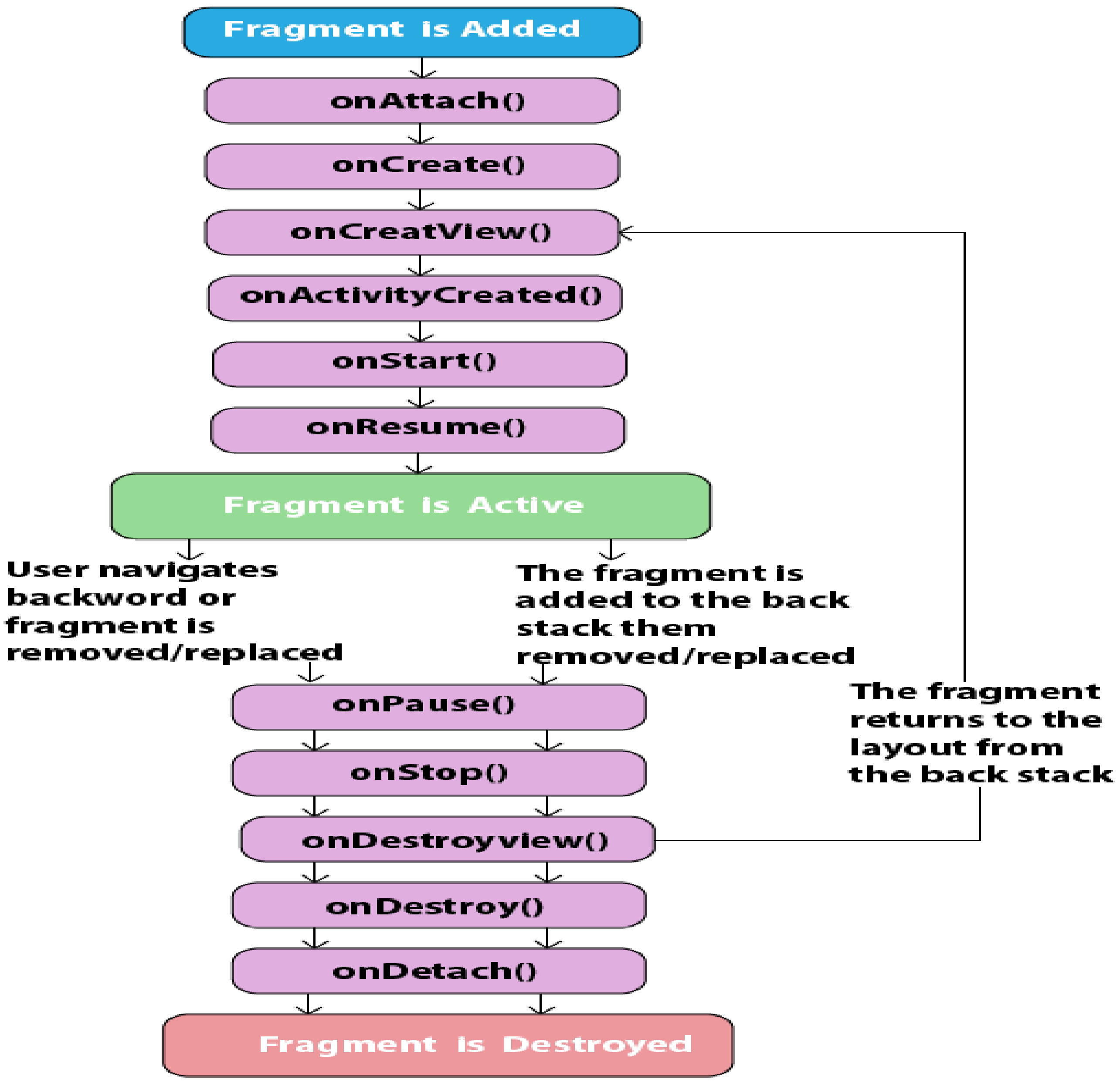
4- Invoke the instance of interface when needed

```
saveButton.setOnClickListener {
 if (nameEditText.text.isEmpty())
 return@setOnClickListener
 fragmentCallback.onNameAdded("Mega")//here
 }
```

26.List out the methods involved in Fragment life cycle and explain all the methods using code snippets.

Android Fragment Lifecycle

The lifecycle of android fragment is like the activity lifecycle. There are 12 lifecycle methods for fragment.

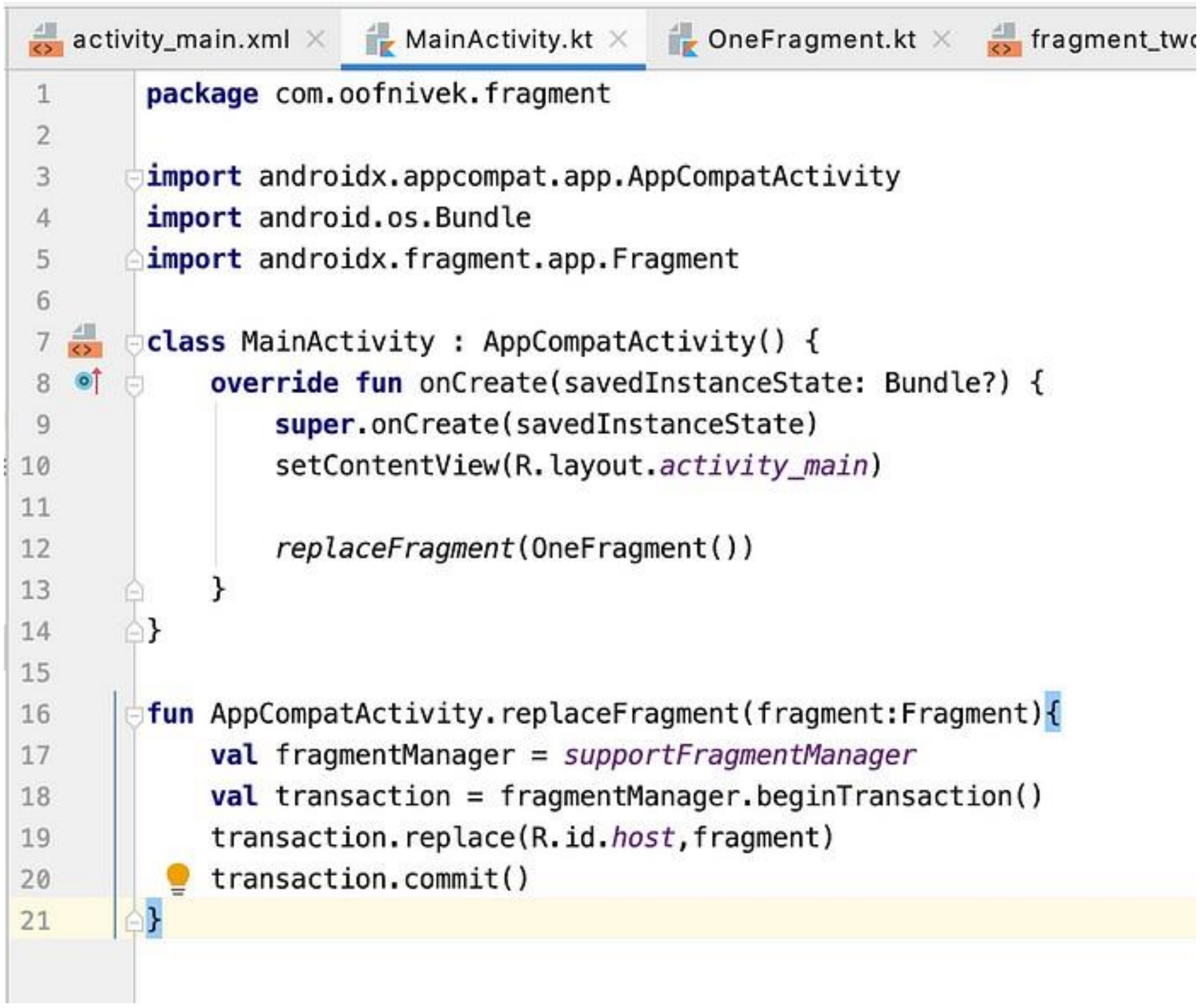


Android Fragment Lifecycle Methods

No.	Method	Description
1)	onAttach(Activity)	it is called only once when it is attached with activity.
2)	onCreate(Bundle)	It is used to initialize the fragment.

3)	onCreateView(LayoutInflater, ViewGroup, Bundle)	creates and returns view hierarchy.
4)	onActivityCreated(Bundle)	It is invoked after the completion of onCreate() method.
5)	onViewStateRestored(Bundle)	It provides information to the fragment that all the saved state of fragment view hierarchy has been restored.
6)	onStart()	makes the fragment visible.
7)	onResume()	makes the fragment interactive.
8)	onPause()	is called when fragment is no longer interactive.
9)	onStop()	is called when fragment is no longer visible.
10)	onDestroyView()	allows the fragment to clean up resources.
11)	onDestroy()	allows the fragment to do final clean up of fragment state.
12)	onDetach()	It is called immediately prior to the fragment no longer being associated with its activity.

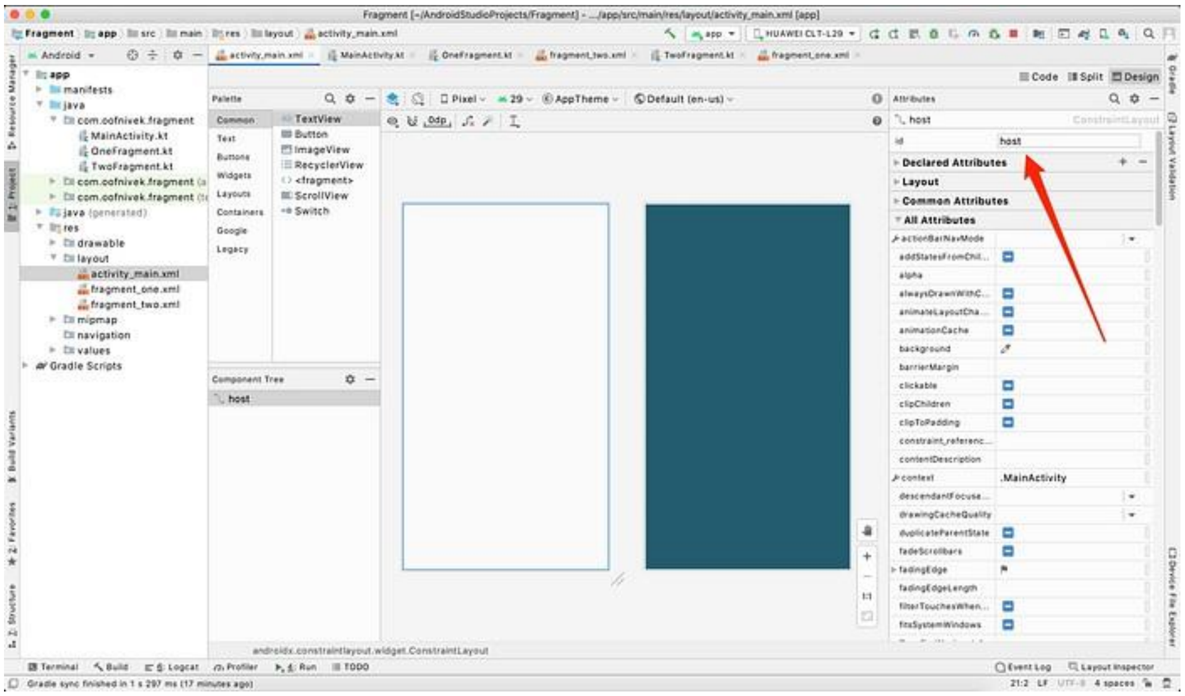
27.Design an activity for replacing Fragments with Fragment transactions.


activity_main.xml × MainActivity.kt × OneFragment.kt × fragment_two

```kotlin
package com.oofnivek.fragment

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import androidx.fragment.app.Fragment

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        replaceFragment(OneFragment())
    }
}

fun AppCompatActivity.replaceFragment(fragment:Fragment){
    val fragmentManager = supportFragmentManager
    val transaction = fragmentManager.beginTransaction()
    transaction.replace(R.id.host,fragment)
    transaction.commit()
}
```

At MainActivity.kt , add the function “replaceFragment”.

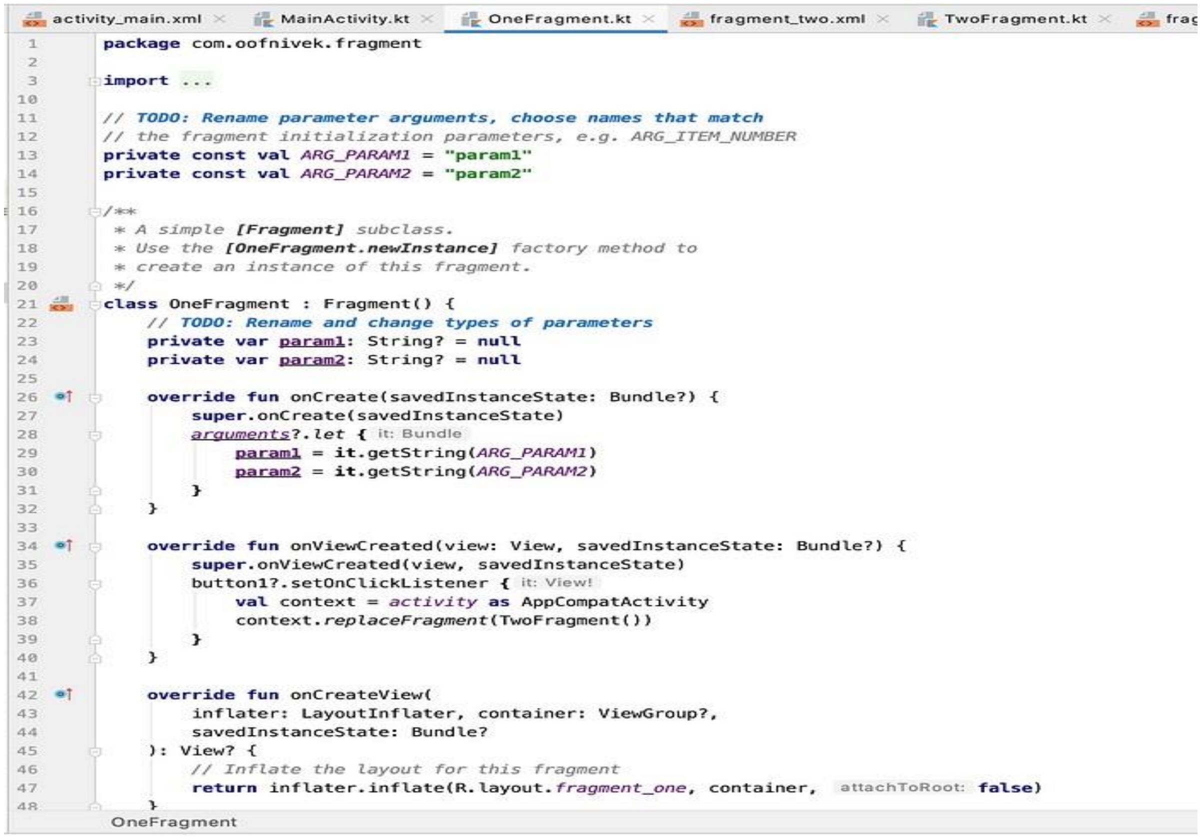
```kotlin
fun                    AppCompatActivity.replaceFragment(fragment:Fragment){
val            fragmentManager             =           supportFragmentManager
val          transaction          =          fragmentManager.beginTransaction()
transaction.replace(R.id.host,fragment)
//transaction.addToBackStack(null)
transaction.commit()
}
```



id: host

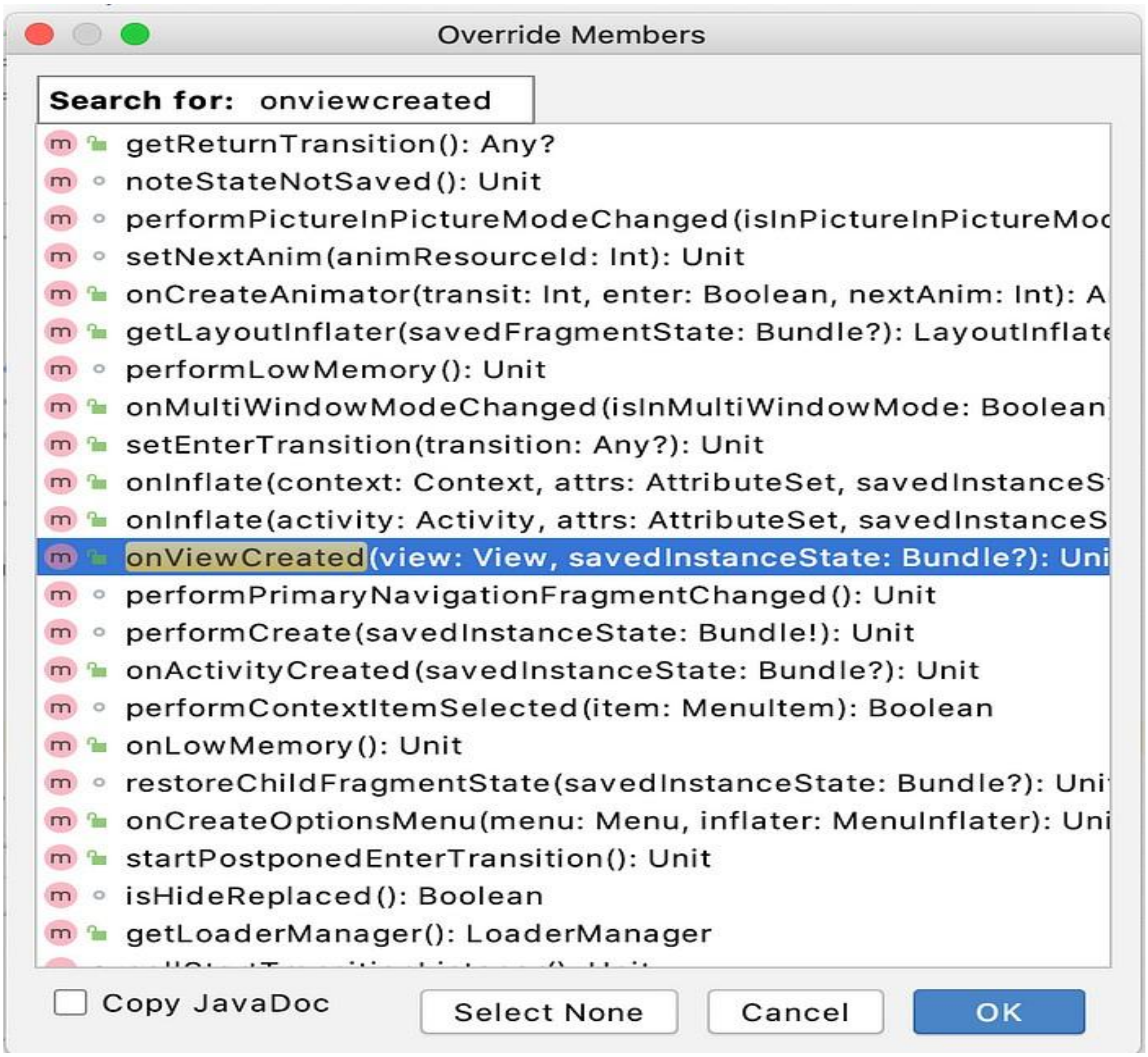
context: .MainActivity

The id of my activity_main.xml constraint layout is “host”. By default it is empty when you create a new project.



```kotlin
package com.oofnivek.fragment

import ...

// TODO: Rename parameter arguments, choose names that match
// the fragment initialization parameters, e.g. ARG_ITEM_NUMBER
private const val ARG_PARAM1 = "param1"
private const val ARG_PARAM2 = "param2"

/**
 * A simple [Fragment] subclass.
 * Use the [OneFragment.newInstance] factory method to
 * create an instance of this fragment.
 */
class OneFragment : Fragment() {
    // TODO: Rename and change types of parameters
    private var param1: String? = null
    private var param2: String? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        arguments?.let { it: Bundle
            param1 = it.getString(ARG_PARAM1)
            param2 = it.getString(ARG_PARAM2)
        }
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        button1?.setOnClickListener { it: View!
            val context = activity as AppCompatActivity
            context.replaceFragment(TwoFragment())
        }
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_one, container,  attachToRoot: false)
    }
```

To generate onViewCreated. Right click "Generate..." > "Override Methods...". Then search for "onViewCreated".

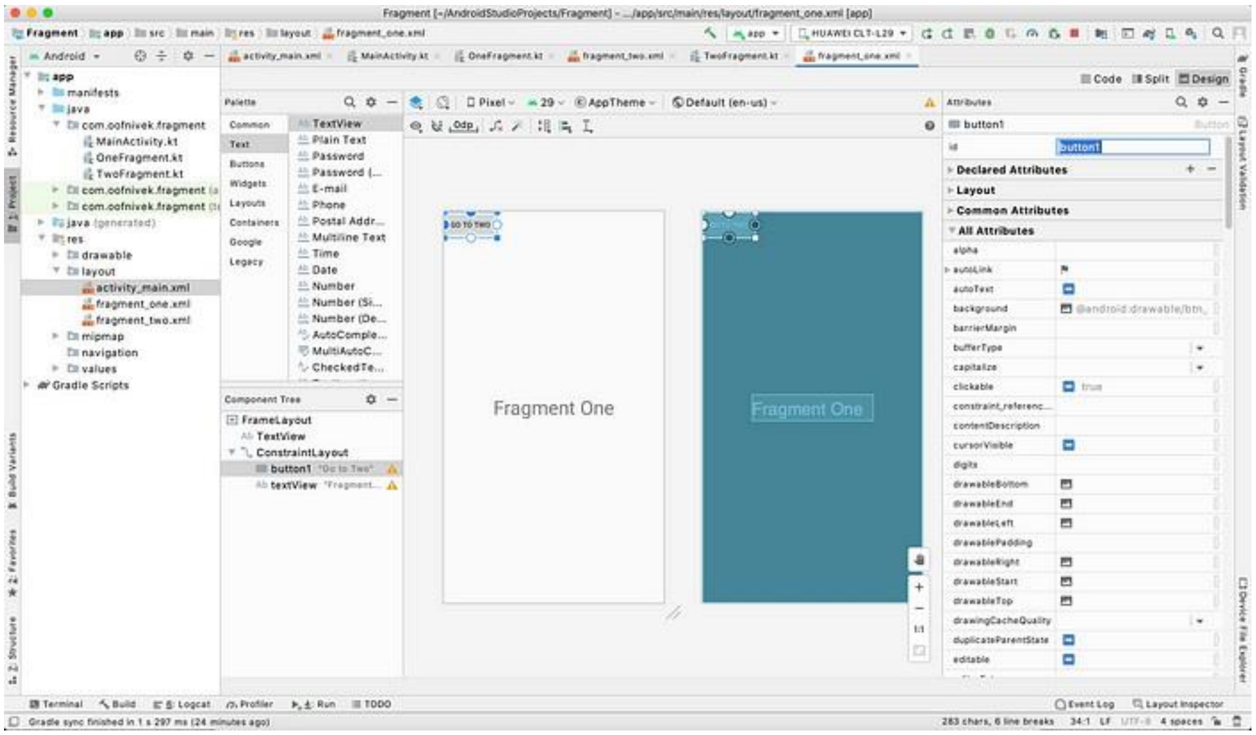


Override Members

Search for: onviewcreated

```
getReturnTransition(): Any?
noteStateNotSaved(): Unit
performPictureInPictureModeChanged(isInPictureInPictureMoc
setNextAnim(animResourceId: Int): Unit
onCreateAnimator(transit: Int, enter: Boolean, nextAnim: Int): A
getLayoutInflater(savedFragmentState: Bundle?): LayoutInflate
performLowMemory(): Unit
onMultiWindowModeChanged(isInMultiWindowMode: Boolean
setEnterTransition(transition: Any?): Unit
onInflate(context: Context, attrs: AttributeSet, savedInstanceS
onInflate(activity: Activity, attrs: AttributeSet, savedInstanceS
onViewCreated(view: View, savedInstanceState: Bundle?): Uni
performPrimaryNavigationFragmentChanged(): Unit
performCreate(savedInstanceState: Bundle!): Unit
onActivityCreated(savedInstanceState: Bundle?): Unit
performContextItemSelected(item: MenuItem): Boolean
onLowMemory(): Unit
restoreChildFragmentState(savedInstanceState: Bundle?): Uni
onCreateOptionsMenu(menu: Menu, inflater: MenuInflater): Uni
startPostponedEnterTransition(): Unit
isHideReplaced(): Boolean
getLoaderManager(): LoaderManager
```

```
button1?.setOnClickListener                                          {
val      context      =      activity      as      AppCompatActivity
context.replaceFragment(TwoFragment())
}
```



In fragment_one.xml , setOnClickListener for button1 so that it will navigate to fragment_two.xml

28.Create an application with three buttons in main activity named Red, Green and Blue. The background of the activity should be changed with appropriate colour when user clicks on any one of these buttons.

A ColorStateList is an object you can define in XML and apply as a color that actually changes colors depending on the state of the View object it is applied to. For example, a Button widget can exist in one of several states: pressed, focused, or neither. Using a color state list, you can provide a different color for each state.

You describe the state list in an XML file. Each color is defined in an <item> element inside a single <selector> element. Each <item> uses various attributes to describe the state in which it is used.

During each state change, the state list is traversed top to bottom, and the first item that matches the current state is used. The selection is *isn't* based on the "best" match, but rather the first item that meets the minimum criteria of the state.

Note: If you want to provide a static color resource, use a simple color value.

file location:

res/color/*filename*.xml
The filename is used as the resource ID.

compiled resource datatype:

Resource pointer to a ColorStateList

resource reference:

In Java: R.color.*filename*
In XML: @[*package*:]color/*filename*

syntax:

```
<?xml version="1.0" encoding="utf-8"?>
<selector  xmlns:android="http://schemas.android.com/apk/res/android"  >
   <item
       android:color="hex_color"
       android:lStar="floating_point_value"
       android:state_pressed=["true" | "false"]
       android:state_focused=["true" | "false"]
       android:state_selected=["true" | "false"]
       android:state_checkable=["true" | "false"]
       android:state_checked=["true" | "false"]
       android:state_enabled=["true" | "false"]
       android:state_window_focused=["true" | "false"] />
</selector>
```

elements:

<selector>

Required. This is the root element. Contains one or more <item> elements.

Attributes:

xmlns:android

String. **Required.** Defines the XML namespace, which is "http://schemas.android.com/apk/res/android".

<item>

Defines a color to use during certain states, as described by its attributes. It is a child of a <selector> element.

Attributes:

android:color

Hexadeximal color. **Required**. The color is specified with an RGB value and optional alpha channel.

The value always begins with a pound (#) character, followed by the Alpha-Red-Green-Blue information in one of the following formats:

- *#RGB*
- *#ARGB*
- *#RRGGBB*
- *#AARRGGBB*

android:lStar

Floating point. **Optional**. This attribute modifies the base color's perceptual luminance. It takes either a floating-point value between 0 and 100 or a theme attribute that resolves as such. The item's overall color is calculated by converting the base color to an accessibility friendly color space and setting its L* to the value specified on the lStar attribute.

Example: android:lStar="50"

android:state_pressed

Boolean. "true" if this item is used when the object is tapped, such as when a button is touched or clicked. It's "false" if this item is used in the default, non-tapped state.

android:state_focused

Boolean. "true" if this item is used when the object is focused, such as when a button is highlighted using the trackball or D-pad. It's "false" if this item is used in the default, non-focused state.

android:state_selected

Boolean. "true" if this item is used when the object is selected, such as when a tab is opened. It's "false" if this item it used when the object isn't selected.

android:state_checkable

Boolean. "true" if this item is used when the object is checkable. It's "false" if this item is used when the object isn't checkable. Only useful if the object can transition between a checkable and non-checkable widget.

android:state_checked

Boolean. "true" if this item is used when the object is checked. It's "false" if it is used when the object is deselected.

android:state_enabled

Boolean. "true" if this item is used when the object is enabled, capable of receiving touch or click events. It's "false" if it is used when the object is disabled.

android:state_window_focused

Boolean. "true" if this item is used when the application window has focus, meaning the application is in the foreground. It's "false" if this item is used when the application window doesn't have focus, such as if the notification shade is pulled down or a dialog appears.

Note: The first item in the state list that matches the current state of the object is applied. So, if the first item in the list contains none of the preceding state attributes, then it applies every time. For this reason, place your default value last, as shown in the following example.

example:

XML file saved at res/color/button_text.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
          android:color="#ffff0000"/> <!-- pressed -->
    <item android:state_focused="true"
          android:color="#ff0000ff"/> <!-- focused -->
    <item android:color="#ff000000"/> <!-- default -->
</selector>
```

The following layout XML applies the color list to a View:

```
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    android:textColor="@color/button_text" />
```

29.Explain how to add Fragments to activity with the help of example program.

create xml file for fragment view fragment_abc.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
android:layout_width="match_parent" android:layout_height="match_parent"
android:orientation="vertical" >

<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="TextView" />

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button" />

</LinearLayout>
```

create fragment ABCFragment.java

```
import androidx.fragment.app.Fragment;

public class FooFragment extends Fragment {
// The onCreateView method is called when Fragment should create its View object hierarchy,
// either dynamically or via XML layout inflation.

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent, Bundle
savedInstanceState) {
    // Defines the xml file for the fragment
    return inflater.inflate(R.layout.fragment_abc, parent, false);
}

// This event is triggered soon after onCreateView().
// Any view setup should occur here.  E.g., view lookups and attaching view listeners.
@Override
public void onViewCreated(View view, Bundle savedInstanceState) {
    // Setup any handles to view objects here
    // EditText etFoo = (EditText) view.findViewById(R.id.etFoo);
}
}
```

Add frameLayout in your activity

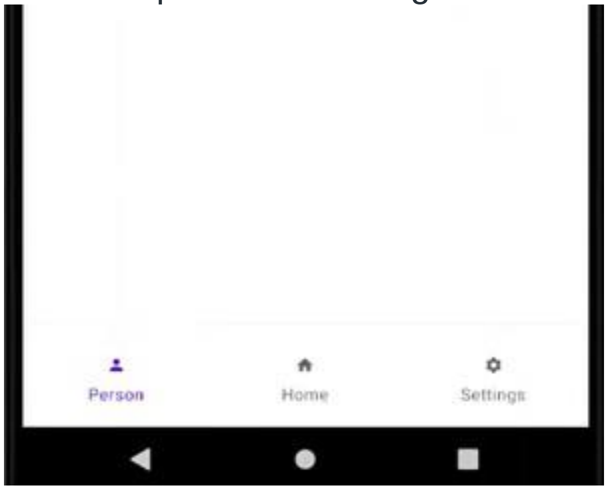
```
<FrameLayout
   android:id="@+id/your_placeholder"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
```

now in activity, add following method

```
protected void setFragment() {
    // Begin the transaction
    FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
    // Replace the contents of the container with the new fragment
    ft.replace(R.id.fragment_container, new ABCFragment());
    // or ft.add(R.id.your_placeholder, new ABCFragment());
    // Complete the changes added above
    ft.commit();
}
```

30.Design bottom navigation menu (Home, Profile, Settings) using the concept of Fragments.

We all have come across apps that have a Bottom Navigation Bar. Some popular examples include Instagram, WhatsApp, etc. In this article, let's learn how to implement such a functional Bottom Navigation Bar in the Android app. Below is the preview of a sample Bottom Navigation Bar:



Why do we need a Bottom Navigation Bar?

- It allows the user to switch to different activities/fragments easily.
- It makes the user aware of the different screens available in the app.
- The user is able to check which screen are they on at the moment.

The following is an anatomy diagram for the Bottom Navigation Bar:

Steps for Creating Bottom Navigation Bar

Step 1: Create a new Android Studio project

To create a new project in Android Studio please refer to **How to Create/Start a New Project in Android Studio**.

Step 2: Adding the dependency to the build.gradle(:app) file

We will be using Android's Material Design Library so we need to import it in the **build.gradle(:app)** file**.** Here's the dependency we need to add:

implementation 'com.google.android.material:material:1.2.0'

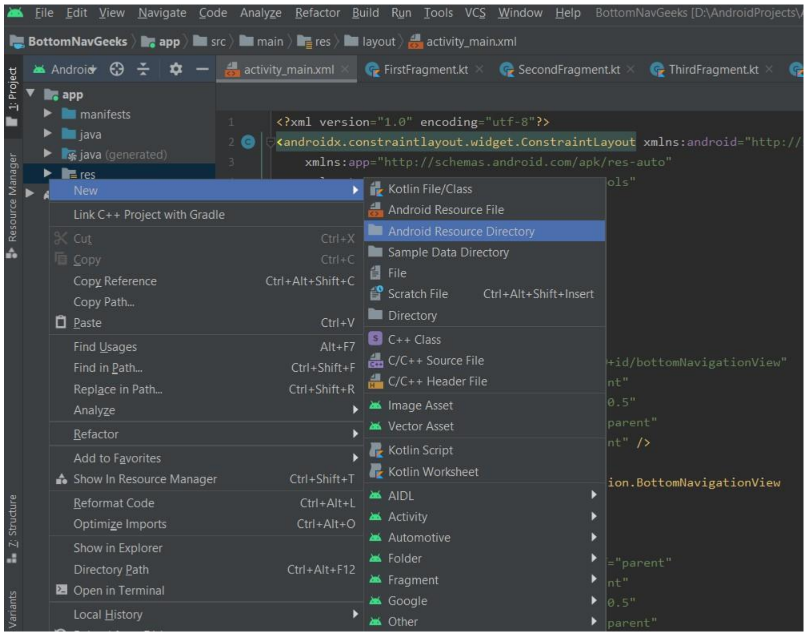
Step 3: Working with activity_main.xml file

For this example, create a basic app with a **FrameLayout** and a **Bottom Navigation Bar**. The **FrameLayout** will contain **Fragments** which will change as the user click on the items in the **Bottom Navigation Bar**. This is how the **activity_main.xml** looks like:

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

    <FrameLayout
        android:id="@+id/flFragment"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        app:layout_constraintBottom_toTopOf="@+id/bottomNavigationView"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <com.google.android.material.bottomnavigation.BottomNavigationView
        android:id="@+id/bottomNavigationView"
        android:layout_width="match_parent"
        android:layout_height="75dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:menu="@menu/bottom_nav_menu"/>

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

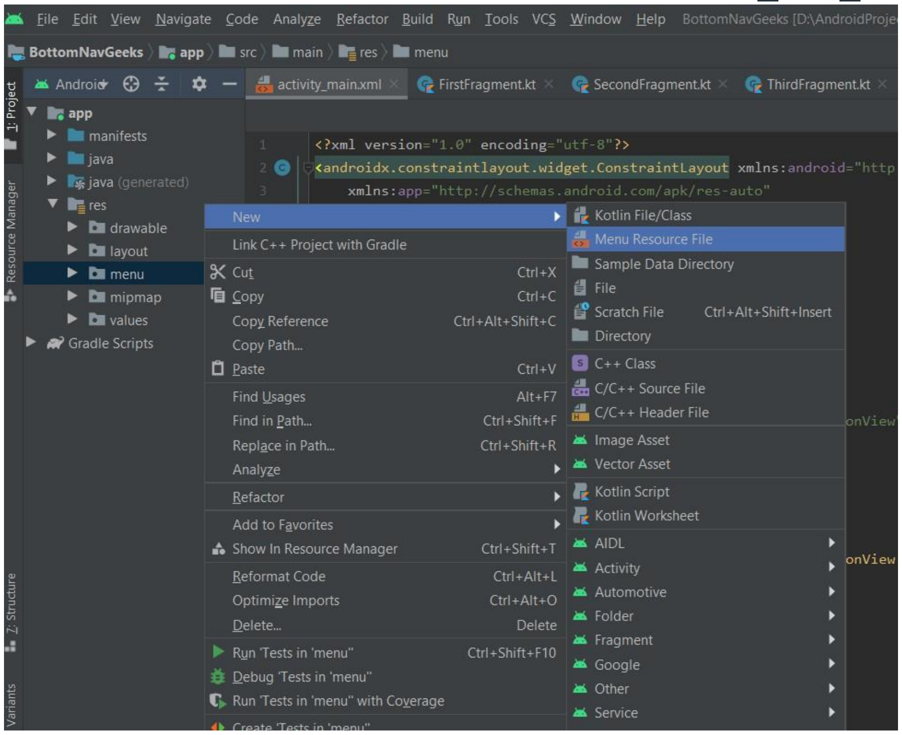
Step 4: Creating a menu for the Bottom Navigation Bar

The Navigation Bar needs to have some items which will create using **Menu**. To create a **Menu**, first, create a **Menu Directory** by clicking on the **app -> res**(right-click) -> **New** -> **Android Resource Directory** and select **Menu** in the **Resource Type**.



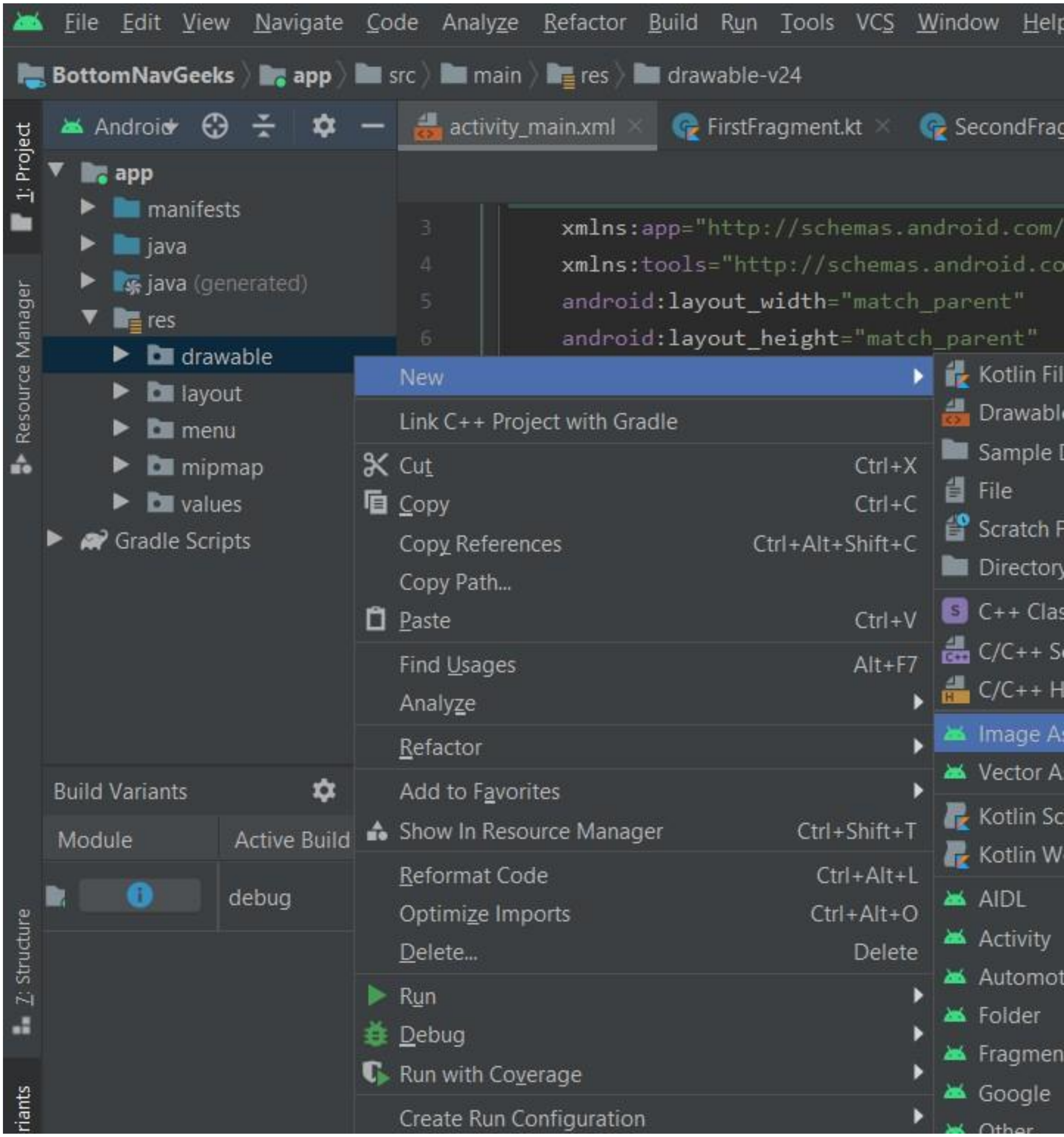
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://
    xmlns:app="http://schemas.android.com/apk/res-auto"
```

To create a **Menu Resource File** , click on the **app** -> **res** -> **menu**(right-click) -> **New** -> **Menu Resource File** and name it **bottom_nav_menu**.



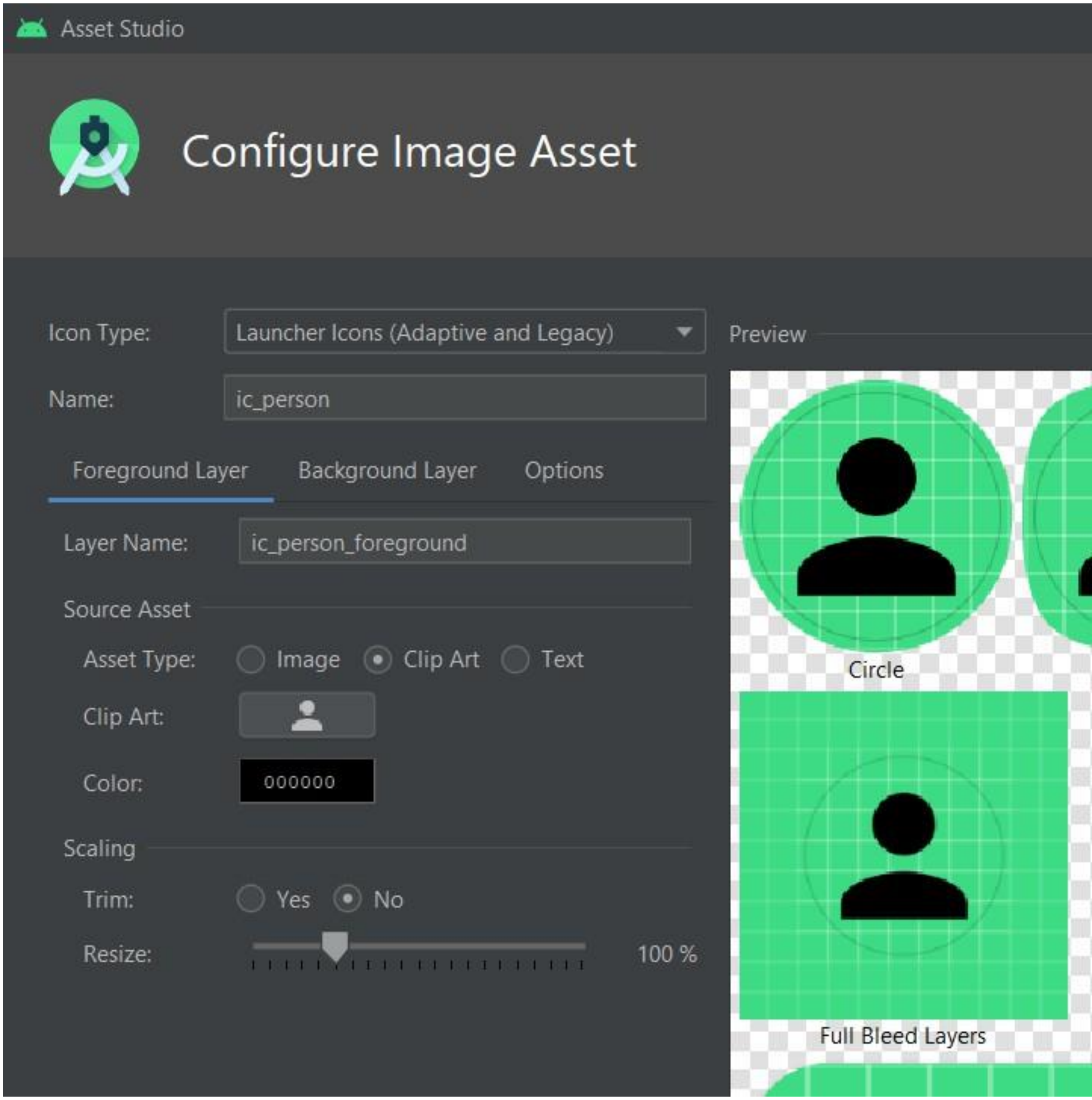
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http
    xmlns:app="http://schemas.android.com/apk/res-auto"
```

Now the user can create as many items as he wants in the **bottom_nav_menu.xml** file. The user also needs to create an icon for each of these items. To create an icon, click on the **app** -> **res** -> **drawable**(right-click) -> **New** -> **Image Asset.**



```
xmlns:app="http://schemas.android.com/
xmlns:tools="http://schemas.android.co
android:layout_width="match_parent"
android:layout_height="match_parent"
```

In the window that opens, the user can name the icon whatever he wants but **it should not comprise any uppercase letter**. The user can select the icon he wants by searching it and when the user is done, click **Next**-> **Finish.**



Configure Image Asset

Icon Type: Launcher Icons (Adaptive and Legacy)

Name: ic_person

Layer Name: ic_person_foreground

Source Asset

Asset Type: Image [✓ Clip Art] Text

Clip Art:

Color: 000000

Scaling

Trim: Yes [✓ No]

Resize: 100 %

Now add these items in the **bottom_nav_menu.xml**. This is how the **bottom_nav_menu.xml** file looks like after adding the items:

- XML

```
<?xml version="1.0" encoding="utf-8"?>

<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item

        android:id="@+id/person"

        android:title="Person"

        android:icon="@drawable/ic_person_foreground"/>

    <item

        android:id="@+id/home"

        android:title="Home"

        android:icon="@drawable/ic_home_foreground"/>

    <item

        android:id="@+id/settings"

        android:title="Settings"

        android:icon="@drawable/ic_settings_foreground"/>

</menu>
```

Step 5: Changing the Action Bar style

Since we are using **Google's Material Design Library**, we need to change the action bar's style to use the same library otherwise the Bottom Navigation Bar will be black and its items will be invisible. To change it, navigate to **styles.xml** by clicking on the **app** -> **res** -> **values** -> **styles.xml** and change the **style** opening tag as:

<style name="AppTheme" parent="Theme.MaterialComponents.Light.DarkActionBar">

This is what the **styles.xml** file looks like:

- XML

```
<resources>

    <!-- Base application theme. -->

    <style name="AppTheme" parent="Theme.MaterialComponents.Light.DarkActionBar">

        <!-- Customize your theme here. -->

        <item name="colorPrimary">@color/colorPrimary</item>

        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>

        <item name="colorAccent">@color/colorAccent</item>

    </style>



</resources>
```

Step 6: Creating Fragments to display
Now that we have our Bottom Navigation Bar, we would want it to be functional by taking us to a different fragment/activity when an item is clicked. In this example, create a fragment for each item and navigate to them whenever a corresponding item is clicked. Since we created three items in the Bottom Navigation Bar, we will be creating three Fragments. To create a Fragment, click on the **app**(right-click) -> **New** -> **Fragment** -> **Fragment (Blank)**. Name the fragment as **FirstFragment** and the corresponding XML file as **fragment_first**. To keep things simple, all three of the fragments will just contain a **TextView**. However, we can tweak this as we want it to be in the app. This is how the **fragment_first.xml** looks like after adding a **TextView**:

- XML

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout

    xmlns:android="http://schemas.android.com/apk/res/android"

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

    xmlns:tools="http://schemas.android.com/tools"

    android:layout_width="match_parent"

    android:layout_height="match_parent"

    tools:context=".MainActivity">



    <TextView

        android:id="@+id/firstFragment"

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:text="Geeks for Geeks"

        android:textColor="#43a047"

        android:textSize="40sp"

        android:textStyle="italic|bold"

        app:layout_constraintBottom_toBottomOf="parent"

        app:layout_constraintEnd_toEndOf="parent"

        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"

        app:layout_constraintTop_toTopOf="parent" />



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

Next, code the **FirstFragment** to display the **fragment_first.xml**. For this, delete all the previously written code in **FirstFragment** and replace it with the below code. The below code just takes the layout we created for our fragment and inflates it.
Note: *If we want our fragment to have any logic or perform any task, we will add that code in our **FirstFragment.***

- Kotlin
- Java

```
import java.io.*;

import androidx.fragment.app.Fragment



public class FirstFragment extends Fragment {



    public FirstFragment(){

        // require a empty public constructor

    }



    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedIn

        // Inflate the layout for this fragment

        return inflater.inflate(R.layout.fragment_first, container, false);
```

Unit-4

---

31) Define Intent. Mention different types of intents. Briefly discuss about uses of intents and methods supported by Intent class.Section-IV-12 M

Ans)

Intent:

- Intents are primarily used to facilitate interaction between Android applications. For example, a user may need another application to send an email from their email application. In this case, the email application can create an Intent and add the email data they want to send to this Intent.

Types of Intent:

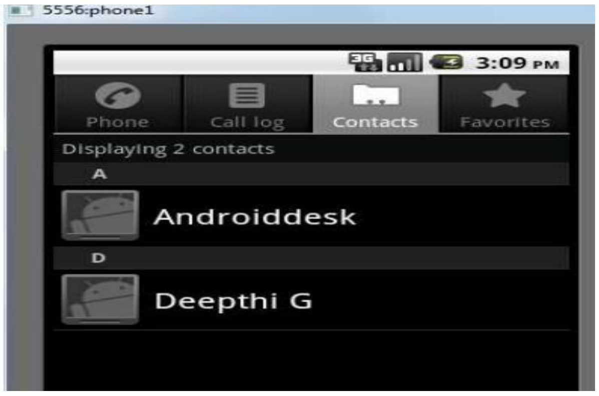
- There are following two types of intents supported by Android
  i) Implicit Intents
  ii)Explicit Intents

  i) Implicit Intent:

- These intents do not name a target and the field for the component name is left blank. Implicit intents are often used to activate components in other applications.
  For example :

```
Intent read1=new Intent();
read1.setAction(android.content.Intent.ACTION_VIEW);
read1.setData(ContactsContract.Contacts.CONTENT_URI);
startActivity(read1);
```

Above code will give result as shown below



The target component which receives the intent can use the getExtras() method to get the extra data sent by the source component.

ii) **ExplicitIntent:**

- Explicit intent going to be connected internal world of application,suppose if you wants to connect one activity to another activity, we can do this quote by explicit intent, below image is connecting first activity to second activity by clicking button.
- These intents designate the target component by its name and they are typically used for application- internal messages - such as an activity starting a subordinate service or launching a sister activity.

For example :

```
Intent i = new Intent(getApplicationContext(), SecondActivity.class);
startActivity(i);
```

Intent Uses:

- Android uses Intents for facilitating communication between its components like Activities, Services and Broadcast Receivers.

Intent for an Activity:

- Every screen in Android application represents an activity. To start a new activity you need to pass an Intent object to startActivity() method. This Intent object helps to start a new activity and passing data to the second activity.

Intent for Services:

- Services work in background of an Android application and it does not require any user Interface. Intents could be used to start a Service that performs one-time task(for example: Downloading some file) or for starting a Service you need to pass Intent to startService() method.

Intent for Broadcast Receivers:

- There are various message that an app receives, these messages are called as Broadcast Receivers. (For example, a broadcast message could be initiated to intimate that the file downloading is completed and ready to use). Android system initiates some broadcast message on several events, such as System Reboot, Low Battery warning message etc.

Methods Supported By intent class:

i) Context.startActivity() :
This is to launch a new activity or get an existing activity to be action.

ii) Context.startService():
This is to start a new service or deliver instructions for an existing service.

iii) Context.sendBroadcast():
This is to deliver the message to broadcast receivers.

32) What is Implicit intent? Explain implicit intent with the help of example program. Section-IV-12 M

Ans:

Implicit Intent:

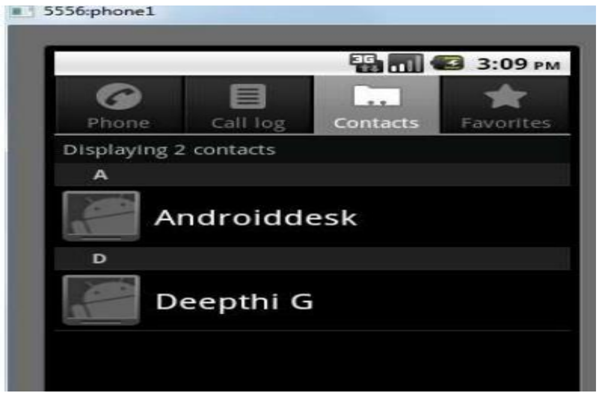
i)Implicit Intent:

- These intents do not name a target and the field for the component name is left blank. Implicit intents are often used to activate components in other applications.

  For example :

```
Intent read1=new Intent();
read1.setAction(android.content.Intent.ACTION_VIEW);
read1.setData(ContactsContract.Contacts.CONTENT_URI);
startActivity(read1);
```

Above code will give result as shown below



Displaying 2 contacts
A
Androiddesk
D
Deepthi G

Example Program For Implicit Intent:

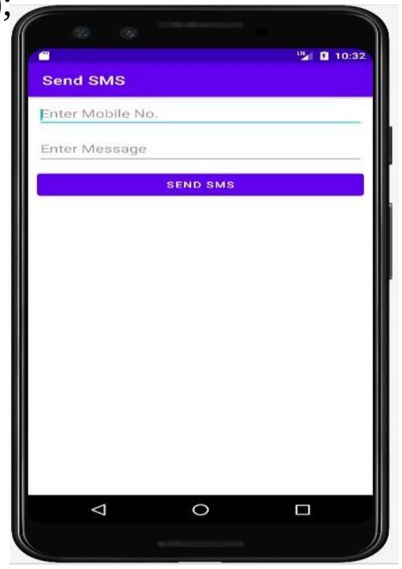
<u>MainActivity.xml</u>

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:layout_margin="10dp"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/textMobileNo"
        android:hint="Enter Mobile No."
        android:layout_width="match_parent"
        android:inputType="phone"
        android:layout_height="40dp"></EditText>

    <EditText
        android:id="@+id/textMessage"
        android:hint="Enter Message"
        android:layout_width="match_parent"
        android:inputType="textMultiLine"
        android:layout_marginTop="10dp"
        android:layout_height="50dp"></EditText>

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="SendSMS"
        android:text="SEND SMS"
        android:layout_marginTop="10dp"
        ></Button>

</LinearLayout>
```

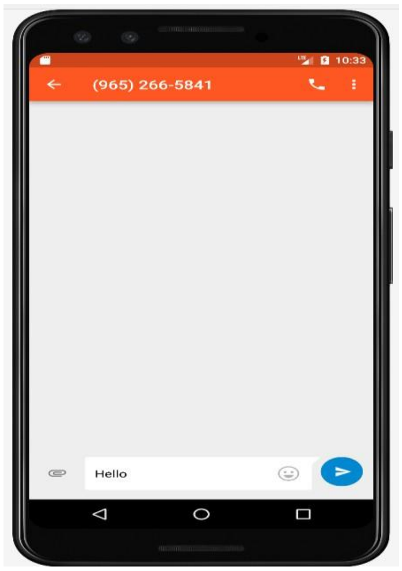
MainActivity.java:

```
package com.example.sendsmsdemo;
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void SendSMS(View view){
        EditText textMobileNo = findViewById(R.id.textMobileNo);
        EditText textMessage = findViewById(R.id.textMessage);

        String mobileNo = textMobileNo.getText().toString();
        String message = textMessage.getText().toString();

        Intent intent=new Intent(Intent.ACTION_VIEW,
Uri.fromParts("sms",mobileNo,null));
        intent.putExtra("sms_body",message);
        startActivity(intent);
    }
}
```

Output:



MainActivity



Native Send SMS App

33) What is Explicit intent? Discuss how explicit intent is helpful to navigate from one activity to another activity with the help of example. Section-IV-12 M

Ans:

ExplicitIntent:

- Explicit intent going to be connected internal world of application,suppose if you wants to connect one activity to another activity, we can do this quote by explicit intent, below image is connecting first activity to second activity by clicking button.
- These intents designate the target component by its name and they are typically used for application- internal messages - such as an activity starting a subordinate service or launching a sister activity.

For example :

```
Intent i = new Intent(getApplicationContext(), SecondActivity.class);
startActivity(i);
```

Example:

MainActivity.xml

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

    <EditText
        android:id="@+id/editTextName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Enter Name"
        android:inputType="text"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/btnSubmit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="SayHello"
        android:text="Submit"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.60" />

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

MainActivity.java

```
package com.example.demo2;

import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

import android.widget.EditText;

public class MainActivity extends AppCompatActivity {

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

    public void SayHello(View view){
        EditText editTextName;
        editTextName = findViewById(R.id.editTextName);

        String name = editTextName.getText().toString();

        Intent intent = new Intent(MainActivity.this, SecondActivity.class);
        intent.putExtra("name",name);
        startActivity(intent);

    }
}
```

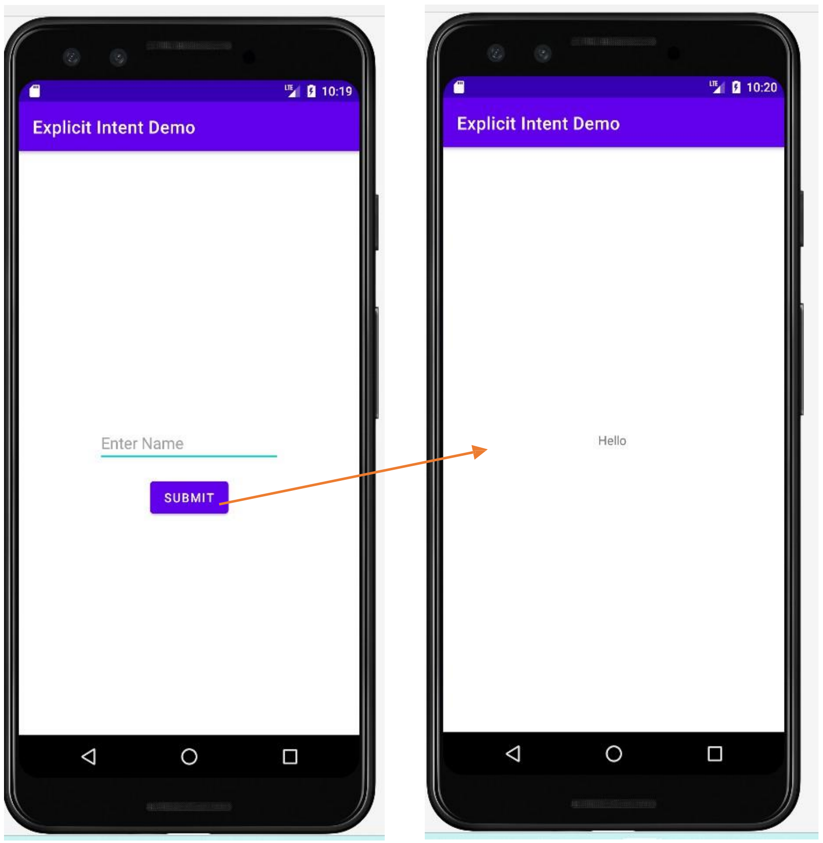
SecondActivity.xml

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

            <TextView
                android:id="@+id/textReceived"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="TextView"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toTopOf="parent" />
        </androidx.constraintlayout.widget.ConstraintLayout>
```

SecondActivity.java

```
package com.example.demo2;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.TextView;

public class SecondActivity extends AppCompatActivity {

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

        String name = getIntent().getExtras().getString("name");
        TextView textReceived = findViewById(R.id.textReceived);
        textReceived.setText(name);

    }
}
```

Output:



MainActivity SecondActivity

34) Create “Send SMS” application using the concept of implicit intent. Section-IV-12 M

Ans:

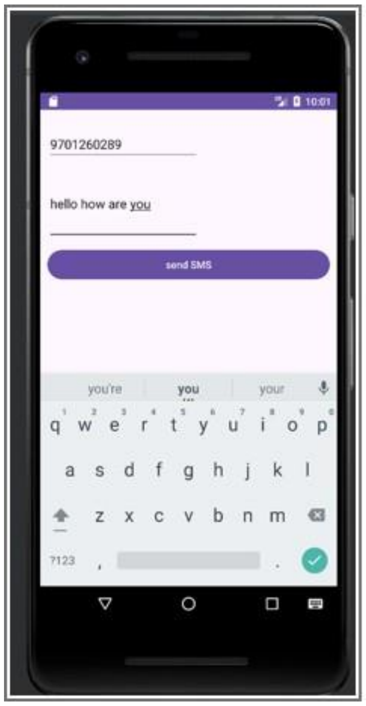
MainActivity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="10dp"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/etMobile"
        android:layout_width="wrap_content"
        android:layout_height="40dp"
        android:layout_marginTop="40dp"
        android:ems="10"
        android:hint="enter mobile number"
        android:inputType="text" />

    <EditText
        android:id="@+id/etText"
        android:layout_width="325dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="40dp"
        android:ems="10"
        android:hint="enter text message"
        android:inputType="text" />

    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="40dp"
        android:baselineAligned="false"
        android:onClick="SendSMS"
        android:text="send SMS" />
</LinearLayout>
```

<u>MainActivity.java</u>

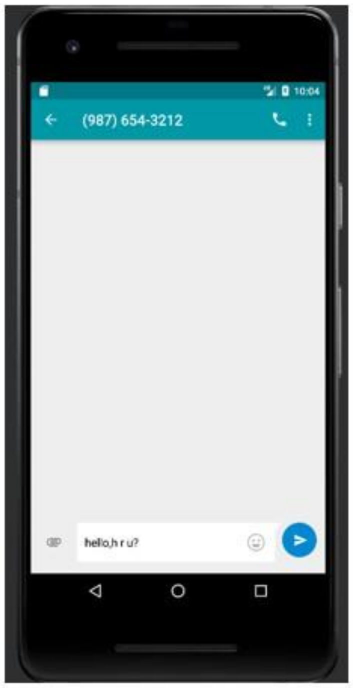
```
package com.example.madweek5;
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
public class MainActivity extends AppCompatActivity {
    EditText Mobile, Message;

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

    public void SendSMS(View view) {
        Mobile = findViewById(R.id.etMobile);
        Message = findViewById(R.id.etText);
        String MobNum = Mobile.getText().toString();
        String TxtMes = Message.getText().toString();
        Intent intent = new Intent(Intent.ACTION_VIEW, Uri.fromParts("sms",
MobNum,null));
        intent.putExtra("sms_body", TxtMes);
        startActivity(intent);
    }

}
```

Output:



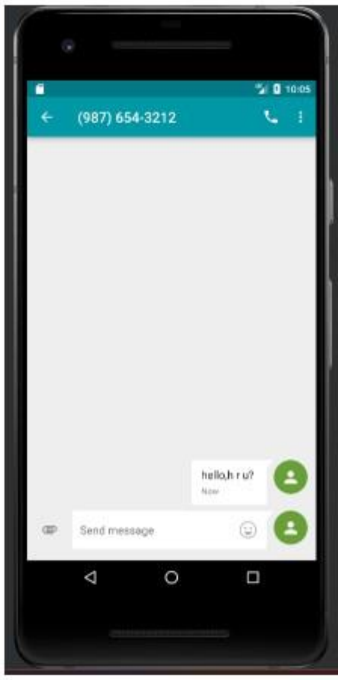
9701260289

hello how are you



(987) 654-3212

hello,h r u?



(987) 654-3212

hello,h r u?

Now

35) Discuss how we can pass data to intents with the help of example. Section-IV-12 M

Ans:

- "Send the data from one activity to second activity using Intent". In this example, we have two activities, activity_first which are the source activity, and activity_second which is the destination activity. We can send the data using the putExtra() method from one activity and get the data from the second activity using the getStringExtra() method.

- In this example, one EditText is used to input the text. This text is sent to the second activity when the "Send" button is clicked. For this, Intent will start and the following methods will run:

- putExtra() method is used for sending the data, data in key-value pair key is variable name and value can be Int, String, Float, etc.
- getStringExtra() method is for getting the data(key) that is sent by the above method. According to the data type of value, there are other methods like getIntExtra(), getFloatExtra().

- How to Create an Android App to Send and Receive the Data between Two Activity

Step by Step Implementation

- Step 1: Create a New Project in Android Studio
- To create a new project in Android Studio please refer to How to Create/Start a New Project in Android Studio. The code for that has been given in both Java and Kotlin Programming Language for Android. This will create an XML file and a Java File. Please refer to the prerequisites to learn more about this step.
- Step 2: Working with the XML Files
  Next, go to the activity_main.xml file, which represents the UI of the project. Below is the code for the activity_main.xml file. Comments are added inside the code to understand the code in more detail. Open the "activity_first_activity.xml" file and add the following widgets in a Relative Layout.

- An EditText to Input the message
- A Button to send the data

- Also, Assign the ID to each component along with other attributes as shown in the image and the code below. The assigned ID on a component helps that component to be easily found and used in the Java/Kotlin files.

Example:

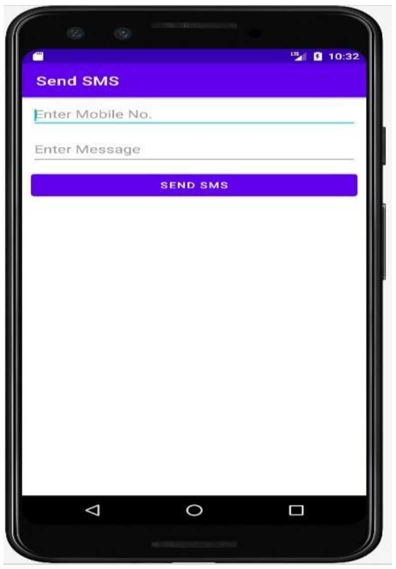
MainActivity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   android:layout_margin="10dp"
   tools:context=".MainActivity">

   <EditText
      android:id="@+id/textMobileNo"
      android:hint="Enter Mobile No."
      android:layout_width="match_parent"
      android:inputType="phone"
      android:layout_height="40dp"></EditText>

   <EditText
      android:id="@+id/textMessage"
      android:hint="Enter Message"
      android:layout_width="match_parent"
      android:inputType="textMultiLine"
      android:layout_marginTop="10dp"
      android:layout_height="50dp"></EditText>

   <Button
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:onClick="SendSMS"
      android:text="SEND SMS"
      android:layout_marginTop="10dp"></Button>

</LinearLayout>
```

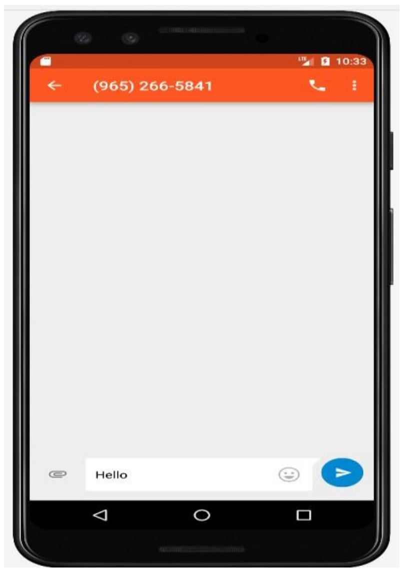
MainActivity.java:

```
package com.example.sendsmsdemo;
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void SendSMS(View view){
        EditText textMobileNo = findViewById(R.id.textMobileNo);
        EditText textMessage = findViewById(R.id.textMessage);

        String mobileNo = textMobileNo.getText().toString();
        String message = textMessage.getText().toString();

        Intent intent=new Intent(Intent.ACTION_VIEW,
Uri.fromParts("sms",mobileNo,null));
        intent.putExtra("sms_body",message);
        startActivity(intent);
    }
}
```

Output:



MainActivity



Native Send SMS App

36) Explain how we can get the results from second activity to first activity with the help of example. Section-IV-12 M

Ans:

Intent-based Communication (Android):

- In Android, activities can communicate with each other using Intents.
- To pass data back from the second activity to the first, you can use an Intent in the second activity.
- Create an Intent in the second activity to hold the data you want to send back.
- Set the result of the second activity using the setResult() method, passing in a result code and the intent containing the data.
- In the first activity, listen for the result by overriding the onActivityResult() method. This method is called when the second activity finishes and returns a result. You can retrieve the data from the intent passed to the method.

Event-based Communication:

- You can use an event-based communication model (such as an event bus) to pass data between the first and second activity.
- The second activity can post an event containing the data, which the first activity can listen for and handle.

Shared ViewModel (MVVM Pattern):

- ➢ If you are using a Model-View-ViewModel (MVVM) architecture, you can share a ViewModel between the two activities.
- ➢ The second activity can update the ViewModel with the data, and the first activity can observe changes in the ViewModel to receive the data.

Shared Preferences or Local Storage:

- ➢ For simple data, you can use shared preferences or local storage to save data in the second activity and read it in the first activity.

Callback Interface:

- ➢ You can create a callback interface in the first activity.
- ➢ The first activity passes the callback interface to the second activity.
- ➢ When the second activity has the results, it calls the methods in the callback interface to pass the data back to the first activity.

Example:

MainActivity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:content=".MainActivity">
    <TextView
        android:id="@+id/activityTV"
        android:layout_width="match_parent"
        android:layout_height="348dp"
        android:layout_weight="1"
        android:gravity="center"
        android:hint="TextView"
     android:textAppearance="@style/TextAppearance.AppCompat.Display1"/>
    <FrameLayout
        android:id="@+id/frag_container"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"/>
</LinearLayout>
```

MainActivity.java

```
package com.example.week7;
import android.os.Bundle;
import android.widget.TextView;

import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.FragmentManager;
import androidx.fragment.app.FragmentTransaction;


public class MainActivity extends AppCompatActivity implements
FirstFragment.MessageListener{
   TextView msgTV;
   @Override
protected void onCreate(Bundle savedInstanceState){
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      FirstFragment mfObj=new FirstFragment();
      FragmentManager FM =getSupportFragmentManager();
      FragmentTransaction FT= FM.beginTransaction();
      FT.replace(R.id.frag_container,mfObj);
      FT.commit();
      }
      public void MessageRead(String data){
      msgTV =findViewById(R.id.activityTV);
      msgTV.setText(data);
      }
      }
```

firstfragment.xml

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

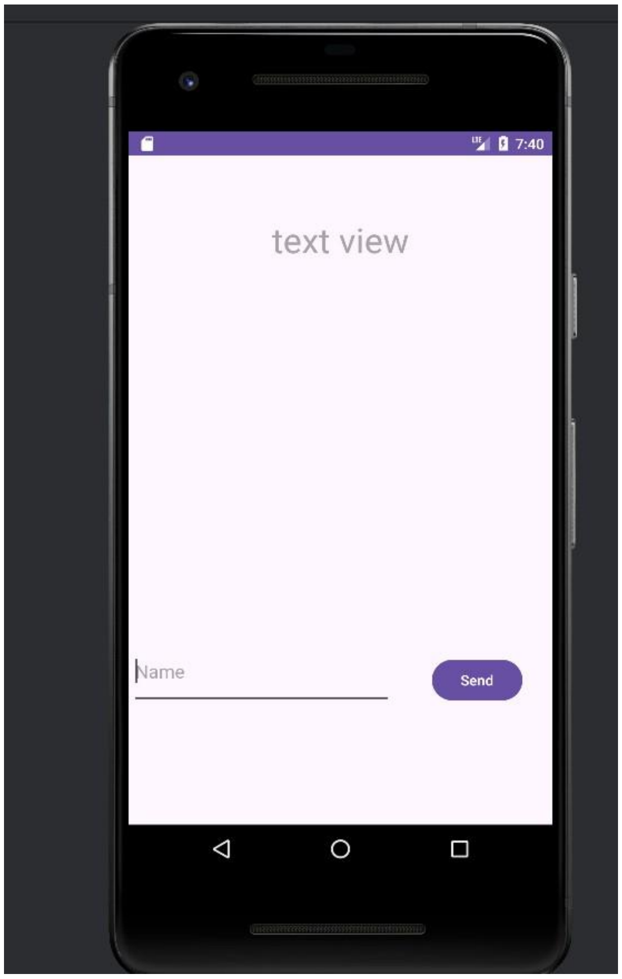
```
    <!-- TODO: Update blank fragment layout -->
    <EditText
        android:id="@+id/fET"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Name"
        android:inputType="text"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.079"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.962">
    </EditText>
    <Button
        android:id="@+id/fBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Send"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.881"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.963">
    </Button>
</androidx.constraintlayout.widget.ConstraintLayout>
```

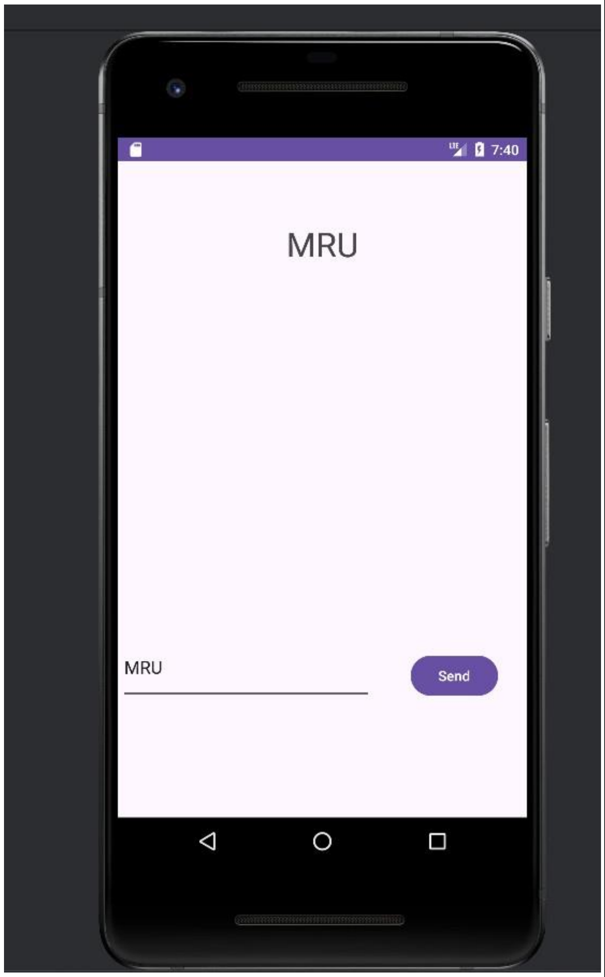
Firstfragment.java

```
package com.example.week7;

import android.os.Bundle;
import android.widget.Button;
import androidx.fragment.app.Fragment;
import android.widget.EditText;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
```

```
/**
 * A simple {@link Fragment} subclass.
 * Use the {@link FirstFragment#newInstance} factory method to
 * create an instance of this fragment.
 */
public class FirstFragment extends Fragment {

    // TODO: Rename parameter arguments, choose names that match
    // the fragment initialization parameters, e.g. ARG_ITEM_NUMBER
    private static final String ARG_PARAM1 = "param1";
    private static final String ARG_PARAM2 = "param2";

    // TODO: Rename and change types of parameters
    private String mParam1;
    private String mParam2;
    private EditText fragmentET;
    private Button fragmentBtn;
    MessageListener mlObj;
    public interface MessageListener {
        public void MessageRead(String msg);
    }

    public FirstFragment() {
        // Required empty public constructor
    }

    /**
     * Use this factory method to create a new instance of
     * this fragment using the provided parameters.
     *
     * @param param1 Parameter 1.
     * @param param2 Parameter 2.
     * @return A new instance of fragment FirstFragment.
     */
    // TODO: Rename and change types and number of parameters
    public static FirstFragment newInstance(String param1, String param2) {
        FirstFragment fragment = new FirstFragment();
        Bundle args = new Bundle();
        args.putString(ARG_PARAM1, param1);
        args.putString(ARG_PARAM2, param2);
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (getArguments() != null) {
            mParam1 = getArguments().getString(ARG_PARAM1);
            mParam2 = getArguments().getString(ARG_PARAM2);
        }
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_first, container, false);
        fragmentET = v.findViewById(R.id.fET);
        fragmentBtn = v.findViewById(R.id.fBtn);
        fragmentBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                String data = fragmentET.getText().toString();
                ((MainActivity) requireActivity()).MessageRead(data);
            }
        });
        return v;
    }
}
```

Output:





37) Define Service in Android. List and explain different types of services.- Section-IV-12 M

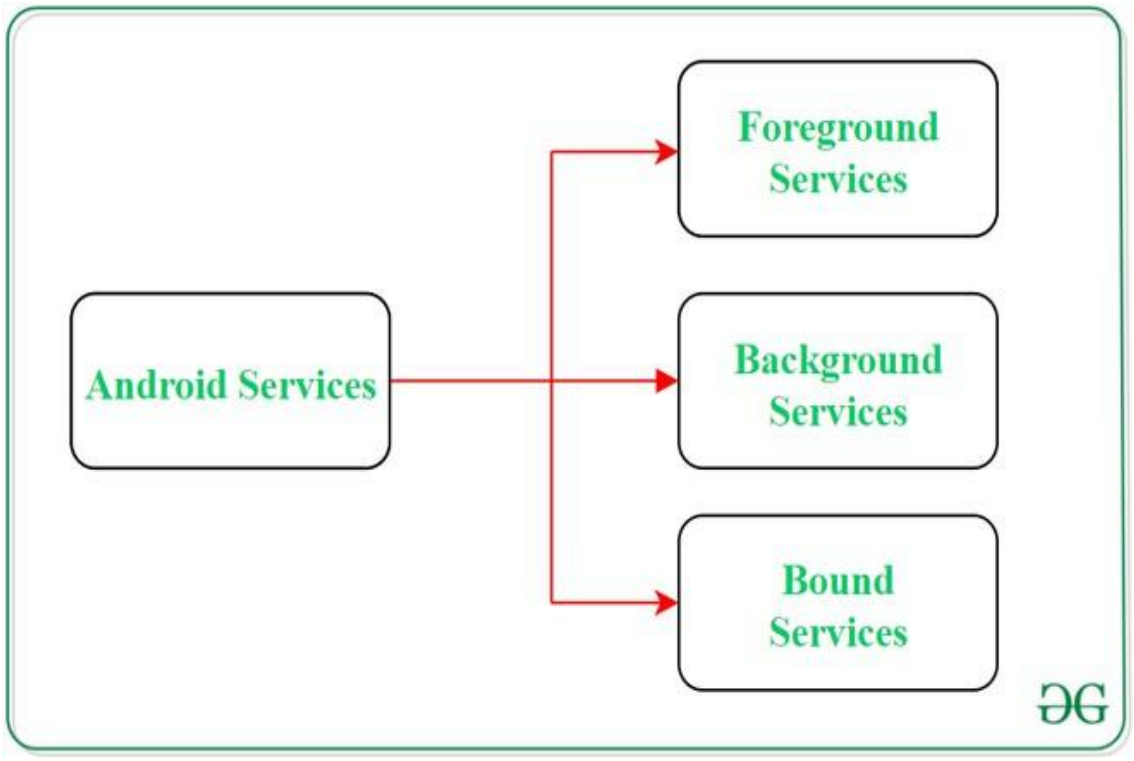
Ans:

Services:

- Services in Android are a special component that facilitates an application to run in the background in order to perform long-running operation tasks.
- The prime aim of a service is to ensure that the application remains active in the background so that the user can operate multiple applications at the same time.

- A user-interface is not desirable for android services as it is designed to operate long-running processes without any user intervention. A service can run continuously in the background even if the application is closed or the user switches to another application.

<u>Types Of Services:</u>



1. Foreground Services:

- Services that notify the user about its on-going operations are termed as Foreground Services. Users can interact with the service by the notifications provided about the on-going task. Such as in downloading a file, the user can keep track of the progress in downloading and can also pause and resume the process.

2. Background Services:

- ➢ Background services do not require any user intervention. These services do not notify the user about on-going background tasks and users also cannot access them. The process like schedule syncing of data or storing of data fall under this service.

3. Bound Services:

- ➢ This type of android service allows the components of the application like activity to bound themselves with it. Bound services perform their task as long as any application component is bound to it. More than one component is allowed to bind themselves with a service at a time. In order to bind an application component with a service bindService() method is used.

38 a) Define intent filter. Write the syntax for Intent-Filter tag.Section -IV-6 M

Ans:

<u>Intent Filter:</u>

- ➢ An intent filter is an instance of the IntentFilter class. Intent filters are helpful while using implicit intents, It is not going to handle in java code, we have to set it up in AndroidManifest.xml. Android must know what kind of intent it is launching so intent filters give the information to android about intent and actions.

Most of the intent filter are describe by its
1.<action>
2.<category>
3.<data>.

1. <action>:
Syntax:
XML

```
<action android:name="string" />
```

Adds an action to an intent filter. An <intent-filter> element must contain one or more <action> elements. If there are no <action> elements in an intent filter, the filter doesn’t accept any Intent objects.
Examples of common action:
ACTION_VIEW: Use this action in intent with startActivity() when you have some information that activity can show to the user like showing an image in a gallery app or an address to view in a map app
ACTION_SEND: You should use this in intent with startActivity() when you have some data that the user can share through another app, such as an email app or social sharing app.

2. <category>:

Syntax:
XML

```
<category android:name="string" />
```

Adds a category name to an intent filter. A string containing additional information about the kind of component that should handle the intent.

Example of common categories:

CATEGORY_BROWSABLE: The target activity allows itself to be started by a web browser to display data referenced by a link.

3. <data>:

Syntax:
XML

```
<data android:scheme="string"
      android:host="string"
      android:port="string"
      android:path="string"
      android:pathPattern="string"
      android:pathPrefix="string"
      android:mimeType="string" />
```

Adds a data specification to an intent filter. The specification can be just a data type, just a URI, or both a data type and a URI.

38 b) Explain how intents can be used to launch activities.
Section-IV-6 M
Ans:
Intent:
- ➢ Intents are primarily used to facilitate interaction between Android applications. For example, a user may need another application to send an email from their email application. In this case, the email application can create an Intent and add the email data they want to send to this Intent.

<u>how intents can be used to launch activities:</u>

- In Android development, you can use Intents to launch activities. An Intent is a message that can be used to communicate between different components of an application, such as activities, services, and broadcast receivers. Here's how you can use intents to launch activities:

1. Create an Intent:

- To launch an activity, you need to create an intent. An intent specifies the activity you want to start and any additional data you want to pass to it.

- For example, to launch SecondActivity from FirstActivity, create an intent like this:

```
Intent intent = new Intent(FirstActivity.this, SecondActivity.class);
```

2.Start the Activity:

- To start the second activity, call the startActivity() method with the intent as a parameter:

```
        startActivity(intent);
```

3. Handle the Result:

- If you used startActivityForResult(), you need to handle the result in the first activity using the onActivityResult() method.
- This method is called when the second activity finishes and returns a result:

Example Code

```
@Override
protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == 1 && resultCode == RESULT_OK) {
        String value = data.getStringExtra("key");
    }
}
```

39) Define Broadcast Receiver. List and explain the system generated events supported by Broadcast Receivers.Section-IV-6 M

Ans:

<u>Broadcast Receiver:</u>

- Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself. These messages are sometime called events or intents. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.
- There are following two important steps to make BroadcastReceiver works for the system broadcasted intents:
  i)Creating the Broadcast Receiver
  ii)Registering Broadcast Receiver

i)Creating the Broadcast Receiver:

- A broadcast receiver is implemented as a subclass of BroadcastReceiver class and overriding the onReceive() method where each message is received as a Intent object parameter.

Example code:

```
public class MyReceiver extends BroadcastReceiver {
@Override
public void onReceive(Context context, Intent intent) {
Toast.makeText(context, "Intent Detected.",
Toast.LENGTH_LONG).show();
}
```

ii) Registering Broadcast Receiver:

- An application listens for specific broadcast intents by registering a broadcast receiverin AndroidManifest.xml file. Consider we are going to register MyReceiver for system generated event ACTION_BOOT_COMPLETED which is fired by the system once the Android system has completed the boot process.

ExampleCode:

```
<application android:icon="@drawable/ic_launcher"
android:label="@string/app_name"
android:theme="@style/AppTheme" >
<receiver android:name="MyReceiver">
<intent-filter>
<action
android:name="android.intent.action.BOOT_COMPLETED">
</action>
</intent-filter>
</receiver>
</application>
```

- There are several system generated events defined as final static fields in the Intent class. The following few important system events:

1 .android.intent.action.BATTERY_CHANGED :
Sticky broadcast containing the charging state, level, and other information about the battery.

2 .android.intent.action.BATTERY_LOW :
Indicates low battery condition on the device.

3 .android.intent.action.BATTERY_OKAY :
Indicates the battery is now okay after being low.

4 .android.intent.action.BOOT_COMPLETED :
This is broadcast once, after the system has finished booting.

5 .android.intent.action.BUG_REPORT :
Show activity for reporting a bug.

6 .android.intent.action.CALL :
Perform a call to someone specified by the data.

7.android.intent.action.CALL_BUTTON :
The user pressed the "call" button to go to the dialer or other appropriate UI for placing a call.

8 .android.intent.action.DATE_CHANGED :
The date has changed.

9 .android.intent.action.REBOOT :
Have the device reboot.

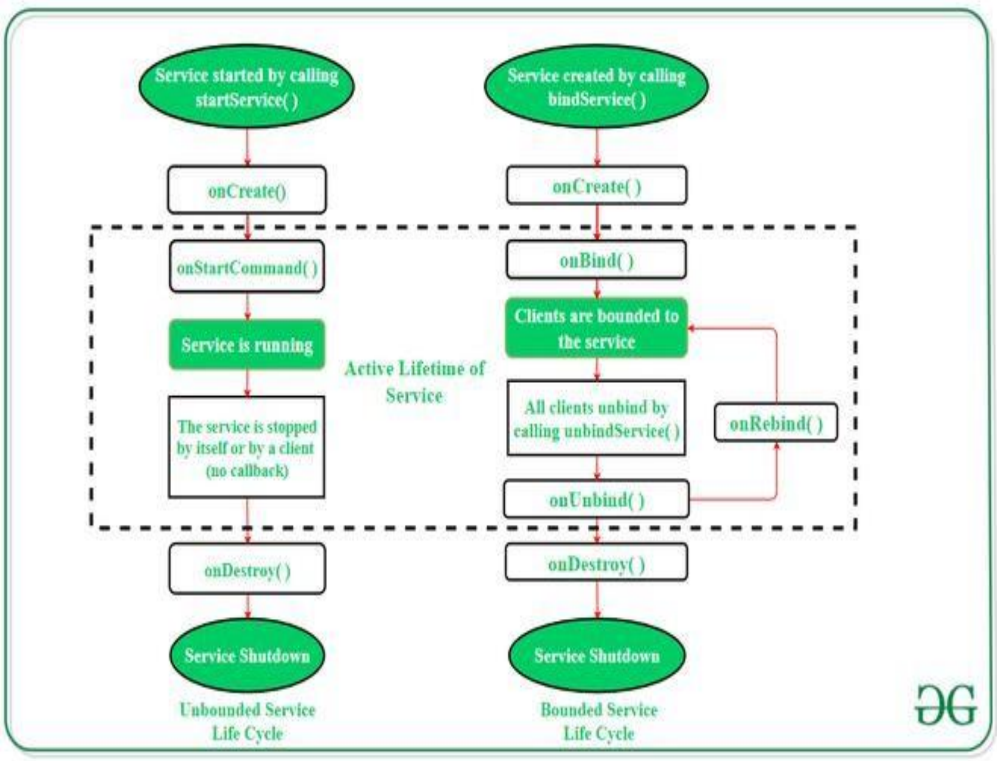
40) Explain the lifecycle of Android services with neat sketch.
Section-IV-12 M

Ans:

Android Services:

- **Services** in Android are a special component that facilitates an application to run in the background in order to perform long-running operation tasks.
- The prime aim of a service is to ensure that the application remains active in the background so that the user can operate multiple applications at the same time.
- A user-interface is not desirable for android services as it is designed to operate long-running processes without any user intervention. A service can run continuously in the background even if the application is closed or the user switches to another application.

Life Cycle Of Android Services:



Service started by calling startService( )
Service created by calling bindService( )
onCreate()
onCreate( )
onStartCommand( )
onBind( )
Service is running
Clients are bounded to the service
Active Lifetime of Service
The service is stopped by itself or by a client (no callback)
All clients unbind by calling unbindService( )
onRebind( )
onUnbind( )
onDestroy( )
onDestroy( )
Service Shutdown
Service Shutdown
Unbounded Service Life Cycle
Bounded Service Life Cycle

- In android, services have 2 possible paths to complete its life cycle namely Started and Bounded.

1. Started Service (Unbounded Service):

- By following this path, a service will initiate when an application component calls the startService() method. Once initiated, the service can run continuously in the background even if the component is destroyed which was responsible for the start of the service. Two option are available to stop the execution of service:
- By calling stopService() method,
- The service can stop itself by using stopSelf() method.

2. Bounded Service:

It can be treated as a server in a client-server interface. By following this path, android application components can send requests to the service and can fetch results. A service is termed as bounded when an application component binds itself with a service by calling bindService() method. To stop the execution of this service, all the components must unbind themselves from the service by using unbindService() method.

Unit-5

41a)Differentiate SQL & SQLite databases? What are the advantages and limitations of SQLite.-Section-V-6 M

Ans)

SQL	**SQLite**
SQL is Structured Query Language used to query Relational Database System. It is written in C language.	SQLite is an Relational Database Management System which is written in ANSI-C.
SQL is standard which specifies how relational schema is created, data is inserted or updated in relations, transactions are started and stopped, etc.	SQLite is file-based. It is different from other SQL databases because unlike most other SQL databases, SQLite does not have separate server process.
Main components of SQL are Data Definition Language(DDL), Data Manipulation Language(DML), Data Control Language(DCL).	SQLite supports many features of SQL and has high performance but does not support stored procedures.
SQL is Structured Query Language which is used with databases like MySQL, Oracle, Microsoft SQL Server, IBM DB2, etc.	SQLite is portable database resource. It could get an extension in whatever programming language used to access that database.

A conventional SQL database needs to be running as service like OracleDB to connect to and provide lot of functionalities.	SQLite database system does not provide such functionalities.
SQL is query language which is used by other SQL databases. It is not database itself.	SQLite is relational database management system itself which uses SQL.

Limitations of SQLite:

- SQLite only supports LEFT OUTER JOIN. It neither supports RIGHT nor FULL OUTER JOIN.
- SQLite only allows normal file access permissions. It does not support GRANT and REVOKE commands as SQLite reads and writes to the disk files.
- In SQLite, using the ALTER table statement, you can only add a column or rename a table.
- SQLite doesn't support FOR EACH STATEMENT triggers. It only supports FOR EACH ROW triggers.
- In SQLite, VIEWs are read-only, and you can't write INSERT, DELETE, or UPDATE statements into the view.

Advantages Of SQLite:

- **Easy to Use:** No installation or configuration needed, just use it right out of the box.
- **Lightweight:** Small in size and stores everything in a single file.
- **Fast:** Performs well for small to medium-sized databases, especially with read-heavy operations.
- **Reliable:** ACID-compliant, ensuring data integrity and reliability even in the event of crashes.

- **Cross-Platform:** Works on almost any operating system and the database files can be moved between systems without issues.

41b) List and explain Data Types used in SQLite database. Section-V-6 M

Ans:

SQLite uses a more general dynamic type system. In SQLite, the datatype of a value is associated with the value itself, not with its container. The dynamic type system of SQLite is compatible with the static type system of other database engines in the sense that SQL statements that work on statically typed databases should work the same way in SQLite. However, the dynamic typing in SQLite allows it to do things which are not possible in traditional rigidly typed databases.

SQLite supports the following data types:

- **INTEGER:** The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.
- **REAL:** The value is a floating point value, stored as an 8-byte IEEE floating point number.
- **TEXT:** The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE).

SQLite does not have a separate Boolean storage class. Instead, Boolean values are stored as integers 0 (false) and 1 (true). SQLite does not have a separate Date/Time storage class. Instead, dates and times can be stored as TEXT, REAL, or INTEGER values.

42a)Explain about Constructors and Methods of SQLiteOpenHelper class. -Section-V-6 M

Ans:

The SQLiteOpenHelper class is an abstract helper class for managing database creation and version management. It provides the functionality to use and maintain a SQLite database in your Android applications.

Constructors:

- **SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version):** This constructor takes four parameters: the application context, the name of the database, a cursor factory, and the database version number. The cursor factory parameter is typically set to null, allowing the default factory to be used.

- **SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version, DatabaseErrorHandler errorHandler):** This constructor is similar to the previous one but includes an additional DatabaseErrorHandler parameter that can be used to handle database corruption errors.

Methods:

- **onCreate(SQLiteDatabase db):** This method is called when the database is created for the first time. This is where the creation of tables and the initial population of the tables should happen.
- **onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion):** This method is called when the database needs to be upgraded. It is used to drop tables, add tables, or do anything else it needs to upgrade to the new schema version.
- **getReadableDatabase():** This method creates or opens a database that will be used for reading and writing.
- **getWritableDatabase():** This method creates or opens a database for reading and writing. If the database cannot be opened for writing, this method will open it in read-only mode.
- **onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion):** This method is called when the database needs to be downgraded. This is not common and is used less frequently than onUpgrade().
- **close():** This method closes the database object effectively freeing all resources.
- SQLiteOpenHelper is an abstract class, so you’ll need to create your own subclass where you implement onCreate() and onUpgrade() (at a minimum).

42b) Discuss about the methods present in SQLiteDatabase class.-Section-V-6 M

Ans:

Data is stored in the SQLite database in the form of tables. When we stored this data in our SQLite database it is arranged in the form of tables that are similar to that of an excel sheet. Below is the representation of our SQLite database which we are storing in our SQLite database.

Methods in SQLite Database :

- getColumnNames() :This method is used to get the Array of column names of our SQLite table.
- getCount() :This method will return the number of rows in the cursor.
- isClosed() :This method returns a Boolean value when our cursor is closed.
- getColumnCount() :This method returns the total number of columns present in our table. getColumnName(int columnIndex) :This method will return the name of the column when we passed the index of our column in it.
- getColumnIndex(String columnName) :This method will return the index of our column from the name of the column.
- getPosition() :This method will return the current position of our cursor in our table.

43) Explain how do you create and use SQLite database.-Section-V-12 M

Ans:

Creating and using an SQLite database is straightforward and involves a few key steps. Here's a step-by-step guide to get you started:

1. Install SQLite:

First, ensure that SQLite is installed on your system. SQLite comes pre-installed on many systems, but you can download it from the official SQLite website if needed.

2. Creating an SQLite Database:
You can create a new SQLite database by simply opening a terminal (or command prompt) and using the SQLite command-line tool. Here's how:

Open Terminal/Command Prompt:

- On Windows: Press Win + R, type cmd, and press Enter.
- On macOS/Linux: Open Terminal.

Start SQLite:

- Type sqlite3 followed by the name of the database file you want to create. For example:
    - sqlite3 mydatabase.db
- If the file doesn't exist, SQLite will create it.

3. Creating Tables:
Once inside the SQLite prompt, you can create tables using the CREATE TABLE statement. For example:

```
CREATE TABLE users (
   id INTEGER PRIMARY KEY,
   name TEXT NOT NULL,
   age INTEGER,
   email TEXT UNIQUE
);
```

4. Inserting Data:
Insert data into the table using the INSERT INTO statement. For example:

```
INSERT INTO users (name, age, email) VALUES ('Alice', 30,
'alice@example.com');
INSERT INTO users (name, age, email) VALUES ('Bob', 25,
'bob@example.com');
```

5. Querying Data:
Retrieve data from the table using the SELECT statement. For example:

```
SELECT * FROM users;
```
This command will display all records in the users table.

6. Updating Data:

Update existing records using the UPDATE statement. For example:

```
UPDATE users SET age = 31 WHERE name = 'Alice';
```

7. Deleting Data:
Remove records from the table using the DELETE statement. For example:

```
DELETE FROM users WHERE name = 'Bob';
```

8. Closing the Database:
To exit the SQLite prompt and close the database, type:

```
.quit
```

Using SQLite in a Program
You can also interact with an SQLite database programmatically using various programming languages. Here’s a quick example using Python:

Python Example
Install SQLite Module:
Make sure you have the SQLite module installed (usually comes with Python). If not, you can install it via pip:

```
pip install sqlite3
```

Connect to the Database:

```
import sqlite3

# Connect to the database (or create it if it doesn't exist)
conn = sqlite3.connect('mydatabase.db')
cursor = conn.cursor()
```

Create a Table:

```
cursor.execute('''
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    age INTEGER,
    email TEXT UNIQUE
)
''')
conn.commit()
```

Insert Data:

```
cursor.execute('''
INSERT INTO users (name, age, email) VALUES (?, ?, ?)
''', ('Alice', 30, 'alice@example.com'))
conn.commit()
```

Query Data:

```
cursor.execute('SELECT * FROM users')
rows = cursor.fetchall()
for row in rows:
    print(row)
```

Close the Connection:

```
conn.close()
```

44)Explain insert operation with the help of example in SQLite database.-Section-V-12 M

Ans:

Insert: To perform insert operation using parameterized query we have to call insert function available in SQLiteDatabase class. insert() function has three parameters like public long insert(String tableName,String nullColumnHack,ContentValues values) where tableName is name of table in which data to be inserted.

Example:

In SQLite, the INSERT INTO statement is used to add new rows of data into a table in the database.

Here’s an example of how you can insert data into the stocks table we created earlier:

```
import sqlite3
```

```
# Create a connection
connection = sqlite3.connect('my_database.db')

# Create a cursor object
cursor = connection.cursor()

# Insert a row of data
cursor.execute("INSERT INTO stocks VALUES ('2022-05-24', 'BUY', 'AAPL', 100, 35.14)")

# Save (commit) the changes
connection.commit()

# Close the connection
connection.close()
```

In the INSERT INTO statement, you first specify the name of the table. Then in parentheses, you provide the data that you want to insert. The data must be in the same order as the columns in the table.
In this case, we're inserting a row into the stocks table. The row contains the following data:

- date: '2022-05-24'
- trans: 'BUY'
- symbol: 'AAPL'
- qty: 100
- price: 35.14

Remember to always commit your changes with connection.commit() and close the connection with connection.close() when you're done. If you don't commit your changes before closing the connection, your changes will be lost.

45)Discuss the procedure to Browse SQLite Database using DB Browser Tool. -Section-V-12 M

Ans:

DB Browser for SQLite is a high quality, visual, open-source tool to create, design, and edit database files compatible with SQLite.

Step 1: Download and Install DB Browser for SQLite DB Browser for SQLite is a free and open-source tool, so you can download it directly from the official website. The website provides versions for different operating systems including Windows, macOS, and Linux. Follow the instructions provided on the website to install the software on your system.

Step 2: Open DB Browser for SQLite Once installed, you can find DB Browser for SQLite in your system's list of applications. Open the application to start interacting with SQLite databases.

Step 3: Open the SQLite Database File In the DB Browser for SQLite application, go to the "File" menu and select "Open Database". This will open a dialog box that allows you to navigate to the SQLite database file you want to open. If the file is not in the default .sqlite, .db, or .db3 formats, you can manually enter the file name and extension.

Step 4: Browse Data After opening the database, you can see a list of tables in the database on the left side of the application window. Click on a table name to view its data in the main panel. You can scroll through the data, sort it by clicking on column headers, and search for specific values using the search box.

Step 5: Edit Data (Optional) DB Browser for SQLite allows you to edit the data directly in the "Browse Data" tab. To edit a value, double-click on the cell, make your changes, and then press Enter or click outside the cell to save the changes. Remember, any changes you make are immediately saved to the database, so be careful when editing data.

Step 6: Execute SQL (Optional) The "Execute SQL" tab allows you to manually enter and execute SQL commands. This is useful for performing operations that aren't available through the graphical interface, such as complex queries or batch updates. After typing your SQL command in the text box, click the "Play" button to execute it. The results will be displayed in the panel below.

Step 7: Close the Database When you're finished with your database, it's important to close it to ensure all changes are saved and resources are freed. Go to the "File" menu and select "Close Database". If there are unsaved changes, DB Browser for SQLite will ask you if you want to save them before closing the database.

DB Browser for SQLite also includes additional features like the ability to import and export data, compact (vacuum) databases, and view a log of all SQL commands issued during a session. You can access these features through the menus at the top of the application window.

46a) List and explain Data Types used in SQLite database. -Section-V-6 M

Ans:

SQLite uses a more general dynamic type system. In SQLite, the datatype of a value is associated with the value itself, not with its container. The dynamic type system of SQLite is compatible with the static type system of other database engines in the sense that SQL statements that work on statically typed databases should work the same way in SQLite. However, the dynamic typing in SQLite allows it to do things which are not possible in traditional rigidly typed databases.

SQLite supports the following data types:

- **INTEGER:** The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.
- **REAL:** The value is a floating point value, stored as an 8-byte IEEE floating point number.
- **TEXT:** The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE).

SQLite does not have a separate Boolean storage class. Instead, Boolean values are stored as integers 0 (false) and 1 (true). SQLite does not have a separate Date/Time storage class. Instead, dates and times can be stored as TEXT, REAL, or INTEGER values.

46b) Mention applications of SQLite Database. -Section-V-6 M

Ans:

SQLite is a highly versatile database management system and it has a wide range of applications. Here are some of the key areas where SQLite is commonly used:

1. **Embedded Applications**: SQLite is an embedded database which means it is an integral part of the application that uses it. This makes it a great choice for devices like mobile phones, televisions, cameras, home electronic devices, and other embedded systems.
2. **Web Application Development**: SQLite is often used in web applications, especially for prototyping, development, and testing, due to its simplicity and ease of use.
3. **Local/Client-side Storage**: In desktop applications, SQLite can be used to store application data such as user preferences, game scores, or other application-specific data. It’s also used as a local database in web browsers to store offline data for web applications.

4. **Server-side Database**: While not as common as other use cases, SQLite can be used as a database for low to medium traffic websites (less than 100k hits/day).
5. **Data Analysis and Scientific Research**: SQLite is used in data analysis and scientific research to manage and query datasets efficiently.
6. **Cache for Enterprise Data**: SQLite is used as a cache for enterprise data. It serves as a fast, local cache for data that doesn't change often.
7. **Internet of Things (IoT) Devices**: SQLite is a popular choice for IoT devices due to its light footprint and the fact that it can run on a wide variety of hardware.
8. **Education and Training**: Due to its simplicity, SQLite is often used in education and training environments to teach SQL and database concepts.

47) Discuss the process of creating tables in SQLite database and write the steps to view the table in database.-Section-V-12 M

Ans:

Creating Tables in SQLite Database

In SQLite, tables are created using the CREATE TABLE SQL command. Here's a step-by-step guide:

1. **Connect to the SQLite Database**: Before you can create a table, you need to connect to the SQLite database. You can do this using the sqlite3.connect() function in Python.

```
import sqlite3
connection = sqlite3.connect('my_database.db')
```

2. **Create a Cursor Object**: A cursor is a database object used to manipulate rows in a table. You can create it using the cursor() method of the connection object.

```
cursor = connection.cursor()
```

3. **Execute the** CREATE TABLE **SQL Command**: You can execute SQL commands using the execute() method of the cursor object. The CREATE TABLE command requires you to specify the name of the table and the names, data types, and constraints of each column.

```
cursor.execute('''
    CREATE TABLE employees (
        employee_id INTEGER PRIMARY KEY,
        first_name TEXT NOT NULL,
        last_name TEXT NOT NULL,
        hire_date TEXT
    )
''')
```

4. **Commit Your Changes**: After executing your commands, you should commit your changes. This saves your changes to the database.

```
connection.commit()
```

5. **Close the Connection**: After you’re done with the database, you should close the connection.

```
connection.close()
```

Viewing Tables in SQLite Database

To view the tables in an SQLite database, you can use the .tables command in the SQLite command-line shell. If you’re using a programming language like Python, you can query the sqlite_master table to get a list of all tables:

```
import sqlite3
connection = sqlite3.connect('my_database.db')
cursor = connection.cursor()

# Execute the command to get a list of all tables
cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")

# Fetch all the results
tables = cursor.fetchall()
```

```
# Print the tables
for table in tables:
    print(table)

# Close the connection
connection.close()
```

This script will print the names of all tables in the database. If you want to view the contents of a specific table, you can use the SELECT * FROM table_name command.

48) Explain update operation with the help of example in SQLite database. -Section-V-12 M

Ans:

Update: Update function is quite similar to insert but it requires two additional parameters, it doesn't required nullColumnHack. It has total four parameters two are similar to insert function that is tableName and contentValues. Another two are whereClause(String) and whereArgs(String[]).

The UPDATE statement is used to modify existing records in a table. It has the following syntax:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Here's a breakdown of the syntax:

- table_name: The name of the table where the records are to be updated.
- SET column1 = value1, column2 = value2, ...: The column names and the corresponding values that they should be updated to. You can update one or more columns at once.
- WHERE condition: The condition that determines which records should be updated. If you omit the WHERE clause, all records will be updated!

Example:

SQLite is a software library that provides a relational database management system. The update operation in SQLite is used to modify existing records in a table.

Here’s a more detailed breakdown of the example I provided earlier:

```
// Get a writable database
SQLiteDatabase db = this.getWritableDatabase();
```

In this line, we’re getting a writable instance of the SQLite database. This is necessary because we’re performing a write operation (updating rows).

```
// New value for one column
ContentValues values = new ContentValues();
values.put("EMAIL", "john_updated@example.com");
```

Here, we’re creating a ContentValues object and adding a new value for the EMAIL column. ContentValues is used to define the data for each column. In this case, we’re setting the EMAIL column to john_updated@example.com.

```
// Which row to update, based on the NAME
String selection = "NAME" + " LIKE ?";
String[] selectionArgs = { "John" };
```

In these lines, we’re defining the selection and selectionArgs for the update operation. The selection is a condition that determines which rows should be updated. In this case, we’re updating rows where the NAME column is John. The ? is a placeholder that will be replaced by the values in selectionArgs.

```
int count = db.update(
    "Users",   // the table to update
    values,    // the columns to update
    selection, // the column to check the condition
    selectionArgs); // the arguments for the WHERE clause
```

This line executes the update operation. The update method takes four parameters: the table name, the new column values, the selection, and the selectionArgs. It updates the rows that satisfy the selection and returns the number of rows affected.

```
// Close the database connection
db.close();
```

Finally, we close the database connection. It’s important to always close the database when you’re done with it to free up system resources and prevent memory leaks.

49) Explain delete operation with the help of example in SQLite database. -Section-V-12 M

Ans:

Delete: Similar to insert and update, delete function is available in SQLiteDatabase class, So delete is very similar to update function apart from ContentValues object as it’s not required in delete. public int delete(String tableName,String whereClause,String [] whereArgs) function has three parameters these are totally similar to update function’s parameters and are used in same way as in update function.

Example:

SQLite is a software library that provides a relational database management system. The delete operation in SQLite is used to remove one or more rows from a table in the database.

Here’s a more detailed breakdown of the example I provided earlier:

```
// Get a writable database
SQLiteDatabase db = this.getWritableDatabase();
```

In this line, we’re getting a writable instance of the SQLite database. This is necessary because we’re performing a write operation (deleting rows).

```
// Define the where clause and where arguments for the query
String whereClause = "NAME=?";
String[] whereArgs = new String[] { "John" };
```

Here, we're defining the whereClause and whereArgs for the delete operation. The whereClause is a condition that determines which rows should be deleted. In this case, we're deleting rows where the NAME column is John. The ? is a placeholder that will be replaced by the values in whereArgs.

```
// Execute the delete operation
int deletedRows = db.delete("Users", whereClause, whereArgs);
```

This line executes the delete operation. The delete method takes three parameters: the table name, the whereClause, and the whereArgs. It deletes the rows that satisfy the whereClause and returns the number of rows deleted.

```
// Close the database connection
db.close();
```

Finally, we close the database connection. It's important to always close the database when you're done with it to free up system resources and prevent memory leaks.

50) Write the syntax of the following operations in SQLite Database -Section-V

a) Create, read-6M

b) Update, delete-6 M

a) Create, read

Create:

In SQLite, the CREATE TABLE statement is used to create a new table. Here's the syntax:

SQL

```
CREATE TABLE table_name (
    column1 datatype PRIMARY KEY(one or more columns),
    column2 datatype,
    column3 datatype,
   ....
);
```

For example, to create a table named Users with ID, NAME, and EMAIL as columns, you would write:

```
CREATE TABLE Users (
    ID INT PRIMARY KEY,
    NAME TEXT,
    EMAIL TEXT
);
```

Read:
The SELECT statement is used to read data from a table. Here's the syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

For example, to select all users from the Users table where the NAME is John, you would write:

```
SELECT *
FROM Users
WHERE NAME = 'John';
```

In this example, * is used to select all columns. You can replace * with specific column names to select those columns.

b) Update, delete

Update:

The UPDATE statement is used to modify the existing records in a table. Here's the syntax:

```
UPDATE table_name
```

```
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

For example, to update the EMAIL of a user named John in the Users table, you would write:

```
UPDATE Users
SET EMAIL = 'john_updated@example.com'
WHERE NAME = 'John';
```

Delete:

The DELETE statement is used to delete existing records in a table. Here's the syntax:

```
DELETE FROM table_name
WHERE condition;
```

For example, to delete a user named John from the Users table, you would write:

```
DELETE FROM Users
WHERE NAME = 'John';
```

In both the UPDATE and DELETE examples, the WHERE clause specifies which record or records that should be updated or deleted. If you omit the WHERE clause, all records will be updated or deleted.