1. **What is Android? Explain its features.**

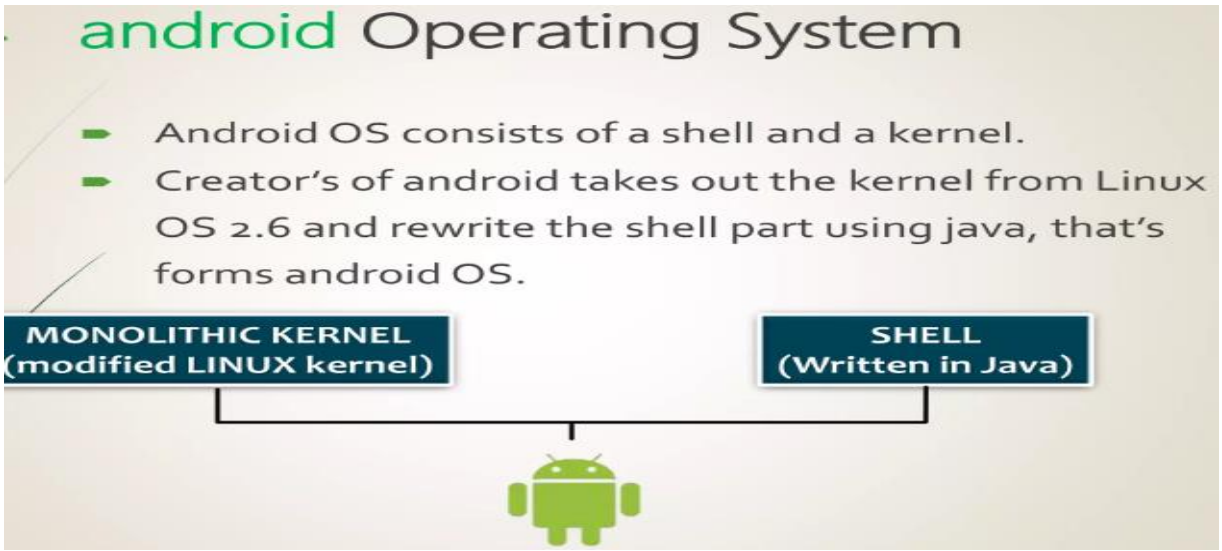
A: Android

Android is an open-source and Linux-based Operating System for mobile devices such as smartphones and tablet computers.

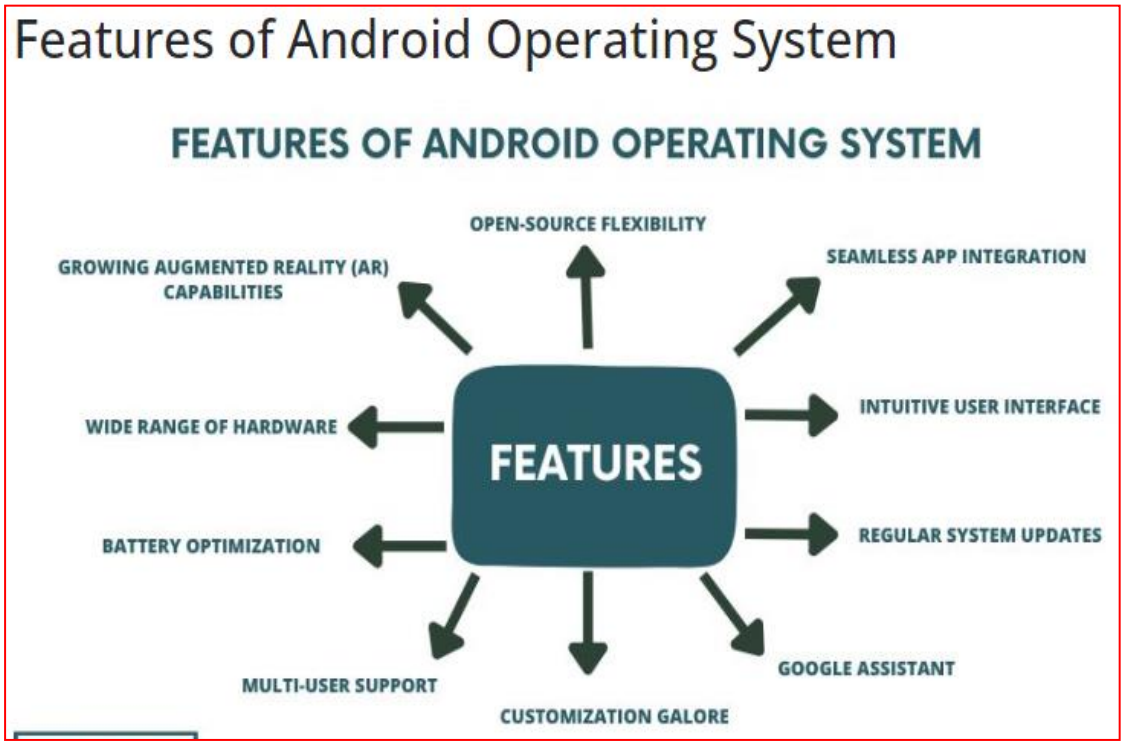
Android is a stripped version of the Linux Operating System i.e. it is compiled to run on smartphones and tablets with fewer resources in processing power and memory than a personal computer. However, it is still capable of doing daily productive jobs like word processing, sending emails, syncing, sharing, calling, and other needs of communication. It had also become a main entertainment device.

We can stream videos, listen to music, and play so many games by downloading apps from Google Play Market. Many of them are free.

This operating system is Open Source and many phone manufacturers like Samsung, HTC, and LG have launched new Android devices both phones and tablets based on Android.



Features of Android OS:



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 text box, when the user clicks the OK button.

A:

Objective: Create a Mobile App and Implement Button Click Event

PROCEDURE:

Step 1: Open Android Studio
Step 2: In the main Welcome to Android Studio window, click New Project
Step 3: choose Empty Activity
Step 4: Enter Project Details
Step 5: Write the Code to implement the Button Click Event
Step 6: Use Android Virtual Device (Emulator)
Step 7: Run the App on the Virtual Device

Code:

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/editTextName"

      android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:hint="Enter your name"
        android:inputType="textPersonName"
        android:layout_marginTop="16dp" />
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:layout_marginTop="16dp">
        <Button
            android:id="@+id/buttonSayHello"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Say Hello" />
    </LinearLayout>
    <!-- TextView to display greeting -->
    <TextView
        android:id="@+id/textViewGreeting"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:text="Hello!"
        android:textSize="18sp"
        android:textStyle="bold" />
</LinearLayout>
```

MainActivity.java:

```
package com.example.sayhelloapp;

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 {

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

        // Find views by their IDs
        EditText editTextName = findViewById(R.id.editTextName);
        Button buttonSayHello = findViewById(R.id.buttonSayHello);
        TextView textViewGreeting = findViewById(R.id.textViewGreeting);

        // Set a click listener for the "Say Hello" button
        buttonSayHello.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String name = editTextName.getText().toString();
                if (!name.isEmpty()) {
                    textViewGreeting.setText("Hello, " + name + "!");
                } else {
                    textViewGreeting.setText("Hello!");
                }
            }
        });
    }
}
```

Output:

3. Define Activity? Explain Activity Life Cycle with a neat sketch.

A:

Activity:

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.

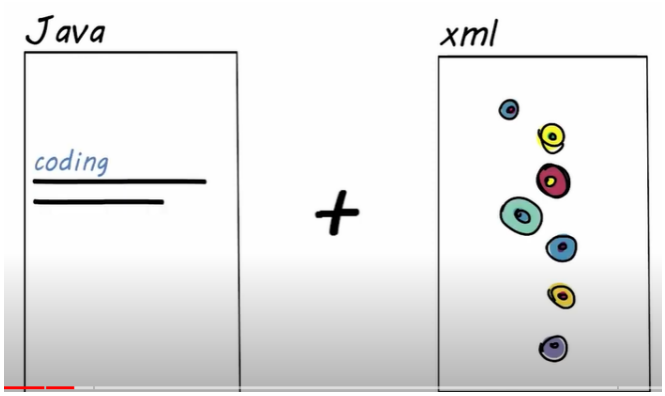
Activity is nothing but a java class in Android which has some pre-defined functions which are triggered at different App states

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.

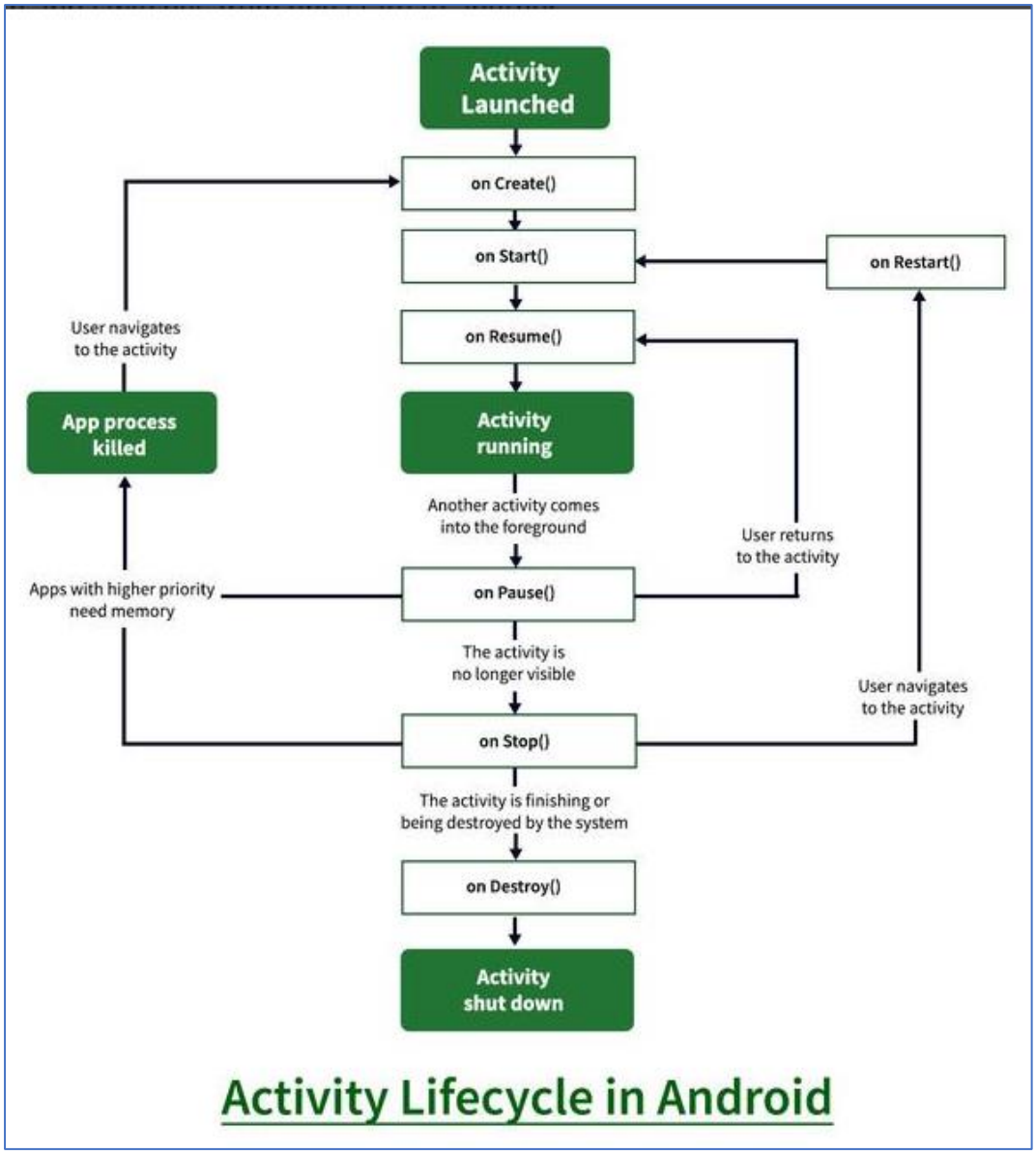
When you create a new project, android studio creates an activity by name "MainActivity.java" and its XML file my name "activity_main.xml". It is the activity that opens up when an app is opened.

Working of an activity:



- An activity in Android is a specific combination of XML files and JAVA files.
- It is basically a container that contains the design as well as coding stuff.
- XML files provide the design of the screen and JAVA files deal with all coding stuff like handles, what is happening, design files, etc
- JAVA files and XML files make your activity complete.

Life cycle of an Activity:



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.

```
public class MainActivity extends AppCompatActivity {

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

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.

When you open the app it will go through below states:

onCreate() –> onStart() –> onResume()

When you press the back button and exit the app

onPaused() — > onStop() –> onDestory()

When you press the home button

onPaused() –> onStop()

After pressing the home button, again when you open the app from a recent task list

onRestart() –> onStart() –> onResume()

After dismissing the dialog or back button from the dialog

onResume()

If a phone is ringing and user is using the app

onPause() –> onResume()

After the call ends

onResume()

When your phone screen is off

onPaused() –> onStop()

When your phone screen is turned back on

onRestart() –> onStart() –> onResume()

---

4. Explain Android Development Framework with block diagram.

A: Android development framework

In general, the basic method of creating an Android app is to use Android Studio (As it is the official IDE for Android).

However, there are many other Android Development Frameworks that are quite popular. There are several types of mobile application development frameworks, each catering to different needs and platforms. Here are some common categories

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:

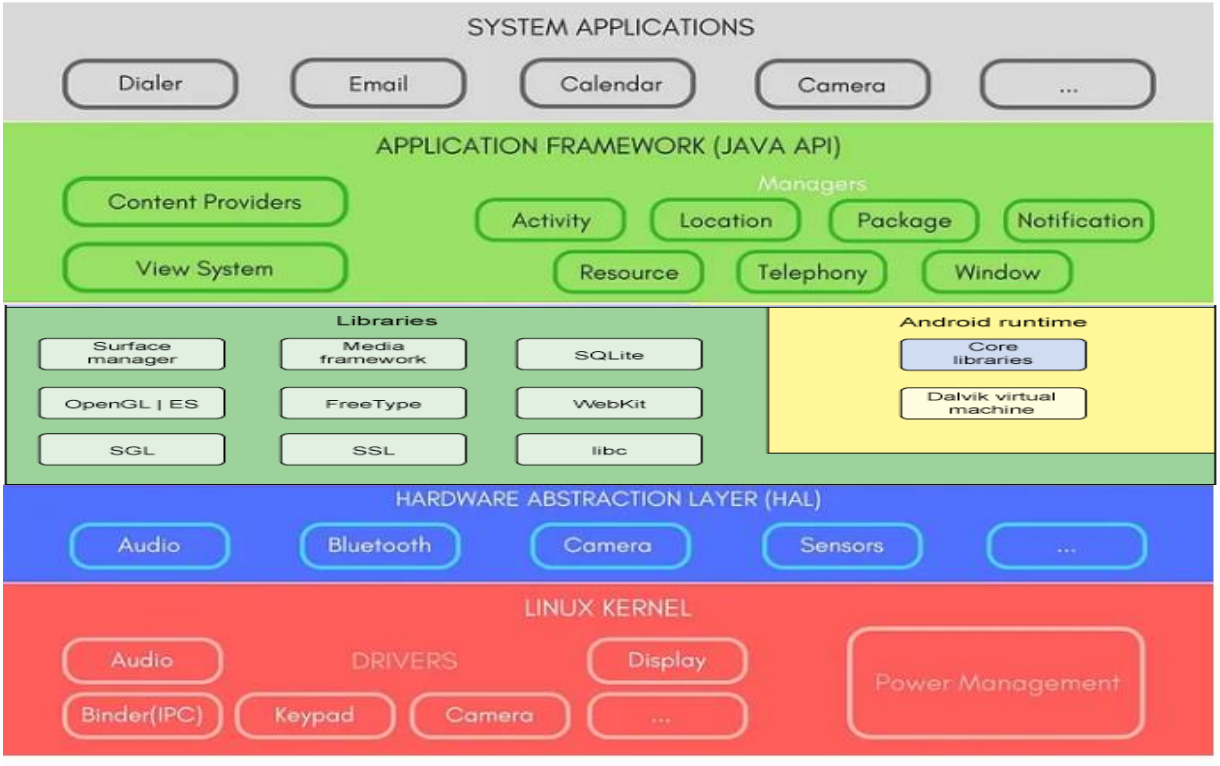
Hybrid mobile app development frameworks combine native and web technologies, usually relying on web views to render content within a native app. Examples include:
Apache Cordova (PhoneGap): It allows developers to use HTML, CSS, and JavaScript to build cross-platform apps. Adobe PhoneGap is a popular 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.

5. Explain Android Architecture with neat diagram?

A: 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. Linux kernel

The bottom layer of the Android operating system is the Linux kernel. Android is built on top of Linux 2.6 Kernel and a few architectural changes were 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 process. It converts the .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. Write the steps to create AVD's. Explain the various types of Android applications

A:

Android Virtual Device (AVD):

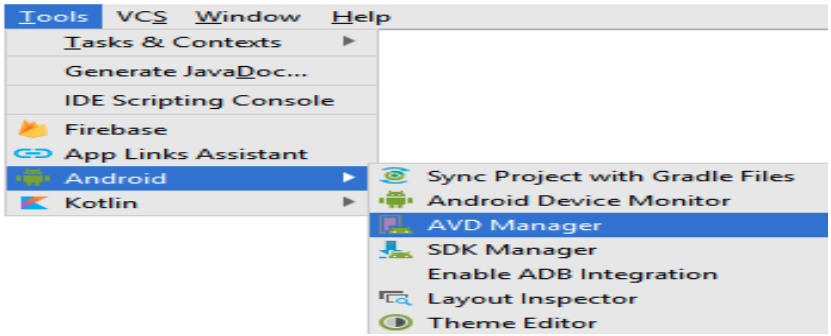
An Android Virtual Device is basically a device configuration. It is similar to Android Mobiles, Tablets, or any other Android device. The developers can test the applications that they have developed on AVDs. An AVD is a software that imitates a real hardware android device. It has a hardware image, a storage system, applications, and other things just like a real device.

We need it during the development of an Android Project or Application. Since it provides a virtual representation of an application, it helps us to run our application on itself. It is required to ensure that our app works correctly.It eliminates the task of connecting a mobile device and installing the application again and again. It helps us test the app and its functions correctly and resolve malfunctions

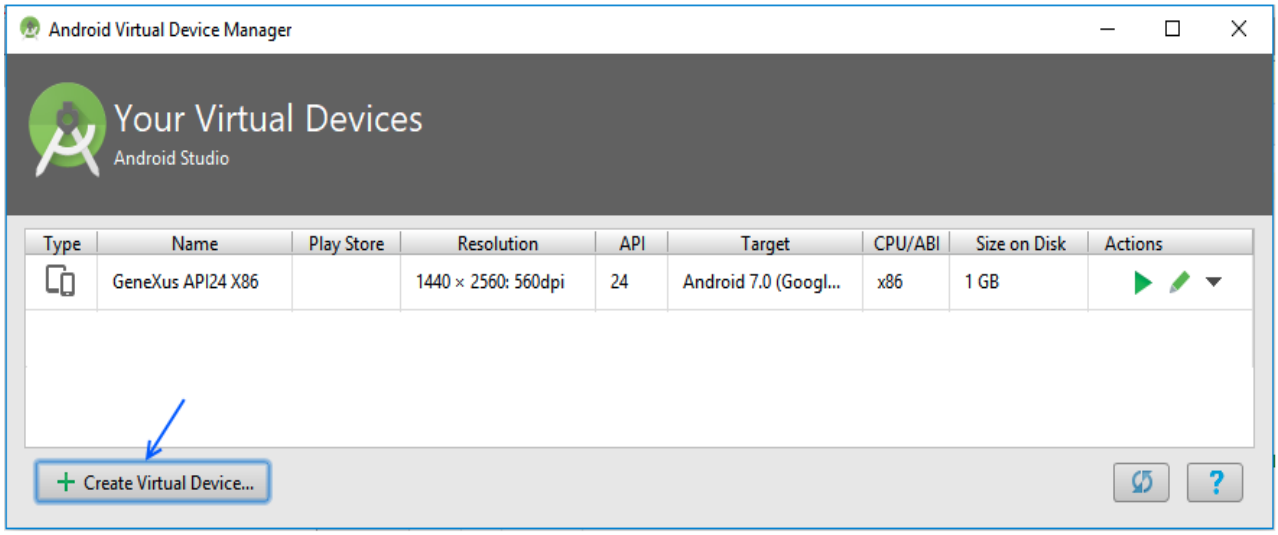
Create Android Virtual Device (AVD):

Step 1: Open Android Studio

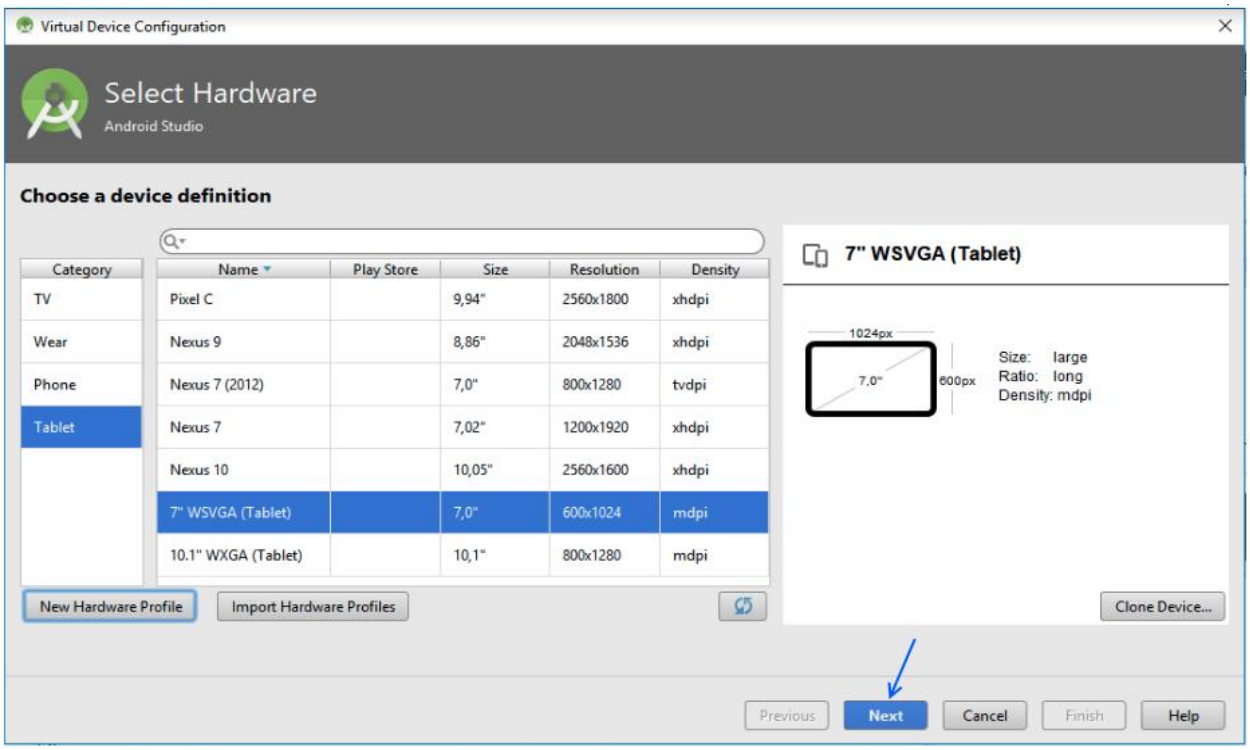
Step 2: Go to Tools > Android > AVD Manager.



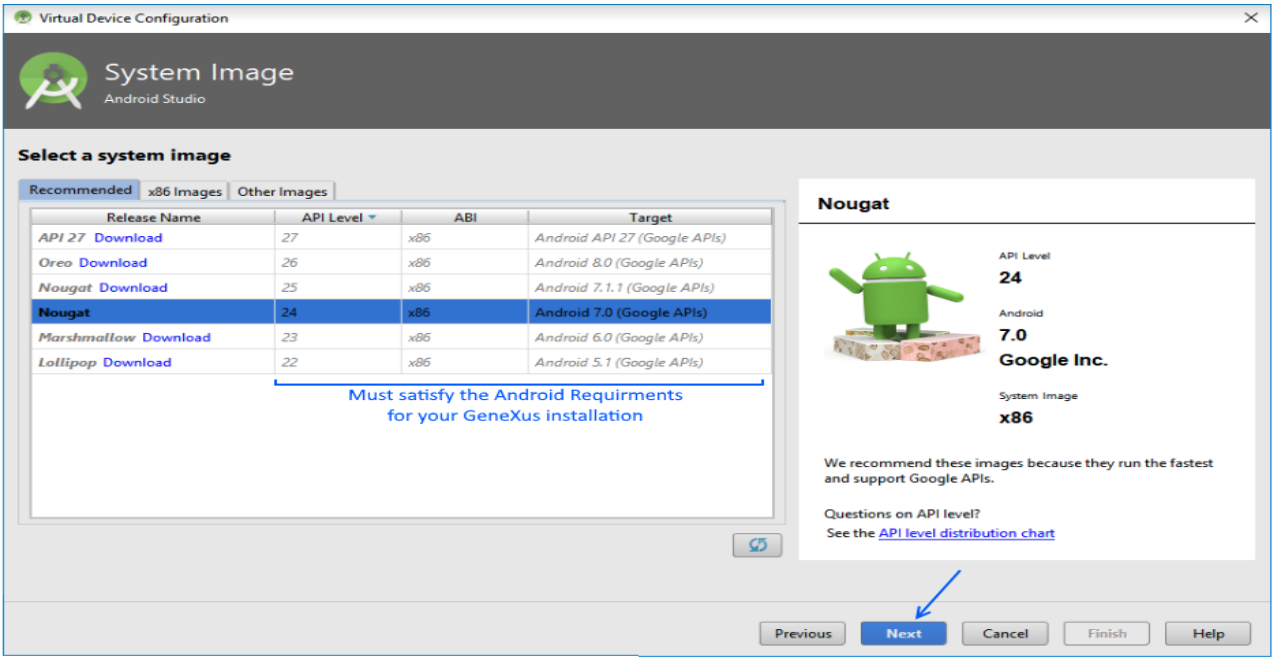
Step 3: Click on the "Create Android Device.." button**.**



Step 4: Select which device you want to create (you can also customize, create new, or import one previously created).

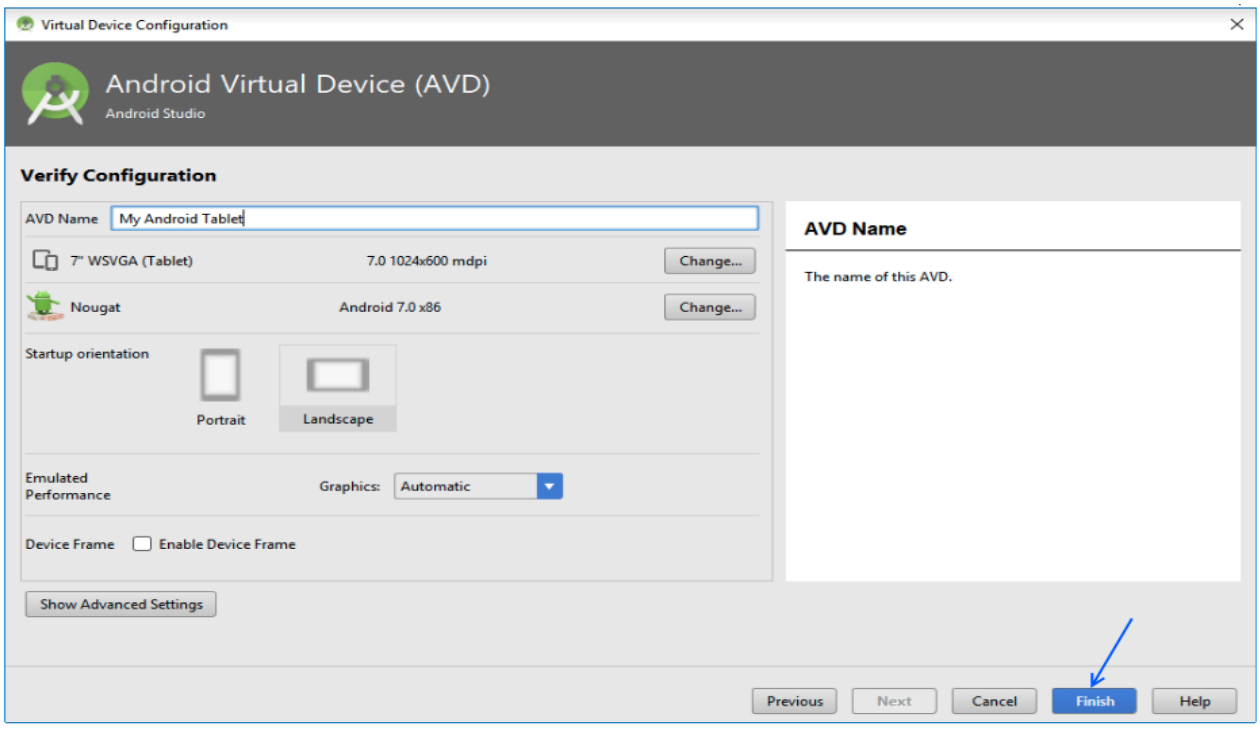


Step 5: Select the system image. Must satisfy Android Requirements for your GeneXus Installation.

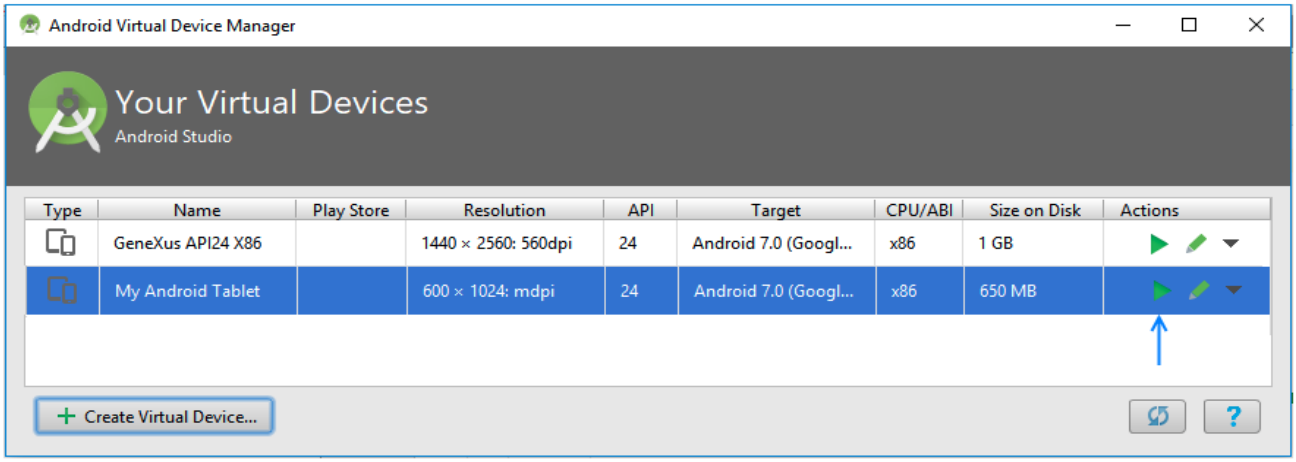


Then, click on the "Next" button.

Step 6: Select the name for your AVD and click on Finish.



Step 7: Finally, select the device previously created and click on the "Play" icon to launch it.



Types of Android application:

1. Native Apps:

Native Apps are written in a specific programming language to work on a particular Operating system.

A majority of the smartphones either run an Android OS or the iOS if it's an apple device. Native Apps are specifically built for specific OS to make the most of the functionalities of the devices that run the particular OS. Hence native apps cannot be used on different types of operating systems. Blackberry, Symbian, and Windows phones are functional only on their respective platform. Technology Used: Xcode and Objective-C are mainly used for iOS apps, and Eclipse, Kotlin, and Java are used to build Android apps.

Native apps are generally built to make the most of all the features and tools of the phones such as contacts, cameras, sensors, etc. Native apps ensure a high performance and elegant user experience as the developers use the native device UI to build apps.

Native apps are easily available on the OS-specific app stores. For example, you will find native Android apps on the Google Play Store, iOS apps on the App Store, and Windows apps on the Microsoft Store

Advantages of native apps:

1. Fast performance due to simple code specific to device and OS.
2. Better use of OS and device specific functionalities.
3. Interactive UI/UX.
4. Lesser compatibility issues and faster to configure.
5. Offline connection

Disadvantages of native apps:

1. Building OS specific apps can be time-consuming
2. OS specific programming languages like swift and java are hard to learn.
3. Longer release cycles to ensure stability.
4. Requires separate codebase to add new features.

Few Popular native mobile applications

2. Web Apps

Web apps behave similarly to native apps but are **accessed via a web browser** on your mobile device.

They're **not standalone apps** in the sense of having to **download and install code** into your device.

It **does not require any storage space or installation process to use the app**. Mobile web **apps adapt to various screen sizes and devices easily**.

One of the major differences between the native apps and mobile apps, is that native mobile apps can function both in the **offline mode and the online mode**, whereas the web apps require an active internet connection for them to work. Since these apps are not installed on your smartphone, there is **no need for updating the web app as they update themselves on the web-hosted servers.**

Technology Used: Web apps are designed using **HTML5, CSS, JavaScript, React, Ruby**, and similar programming languages used for web work.

Advantages of web apps:

1. Reduced business cost.
2. No installation needed.
3. Better reach as it can be accessed from anywhere.
4. Always up-to-date.

Disadvantages of web apps:

1. Web apps fail to work when you are offline.
2. Limited number of functionalities as compared to Native apps.
3. It takes a longer time to develop.
4. Security risk.

Few Mobile Web Apps

3. Progressive Web Apps (PWAs)

Progressive Web Apps (PWAs) are extensions of the website that you can save on your computer systems or devices and use like an app. When you come across the option to "install" a web app, it often simply bookmarks the website URL on your device.

One kind of web app is the progressive web app (PWA), which is basically a native app running inside a browser. PWAs use web browser APIs and functionalities to bring a native app-like experience across devices. It is a type of a webpage that can be added onto your devices or computer systems to mimic a web application. The PWAs run fast regardless of the Operating Systems and devices types.

Advantages of Progressive web apps:

1. They use very little data – An app which takes close to 10 MBs as a native app, can be reduced to about 500KB when made a PWA.
2. PWAs get updated like web-pages. They automatically get updated every time you use them.
3. There is no need for installation as PWAs are simple web-pages. Users choose to 'install' when they like it.
4. You can easily share PWAs by simply sending its URL.

Disadvantages of progressive apps:

1. There are limitations to using all the Hardware and Operating Systems features.
2. PWAs can pose a few hardware integration problems.
3. Full support is not available in default browsers of some of the manufacturer's.

4. Hybrid Apps

Hybrid apps combine the best of both native and web apps. The hybrid apps are written using HTML, Javascript, and CSS web technologies and work across devices running different OSs.

Development teams will not need to struggle with Objective-C or Swift to build native apps anymore, and use standard web technologies like Javascript, Angular, HTML and CSS.

The mobile development framework Cordova wraps the Javascript/HTML code and links the hardware and functions of the device.

Hybrid Apps are built on a single platform and distributed across various app stores such as Google Play store or Apple's app store similar to Native apps.

Hybrid apps are best used when you want to build apps that do not require high-performance, full device access apps.

Advantages of hybrid apps:

1. Easy to build
2. Shareable code makes it cheaper than a native app
3. Easy to push new features since it uses a single code base.
4. Can work offline.
5. Shorter time to market, as the app can be deployed for multiple Oss

Disadvantages of Hybrid apps:

1. Complex apps with many functions will slow down the app.
2. More expensive than web apps
3. Less interactive than native apps
4. Apps cannot perform OS specific tasks

Examples of Hybrid Apps:

---

7. **Define Activity Life Cycle? Explain with neat diagram Activity Life Cycle?**
   A: Refer 3rd question
8. **What are the methods available in Android life cycle. Discuss each method using code snippets.**
   A: Refer 3rd question

9. **Design a screen with edit text, button, and text view component. Display the message in text view whenever the user clicks on the submit button.**

A: Objective: Create a Mobile App and displaying message by clicking the submit button.

PROCEDURE:

Step 1: Open Android Studio
Step 2: In the main Welcome to Android Studio window, click New Project
Step 3: choose Empty Activity
Step 4: Enter Project Details
Step 5: Write the Code to implement the Button Click Event
Step 6: Use Android Virtual Device (Emulator)
Step 7: Run the App on the Virtual Device

Code:

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_marginTop="16dp" />

    <TextView
        android:id="@+id/textViewDisplay"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Your message will appear here"
        android:layout_marginTop="16dp"
        android:textSize="18sp" />
</LinearLayout>
```

MainActivity.java:

```
package com.example.mysampleapp;

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 {

    EditText editTextMessage;
    Button buttonSubmit;
    TextView textViewDisplay;
```

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

        editTextMessage = findViewById(R.id.editTextMessage);
        buttonSubmit = findViewById(R.id.buttonSubmit);
        textViewDisplay = findViewById(R.id.textViewDisplay);

        buttonSubmit.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String message = editTextMessage.getText().toString();
                textViewDisplay.setText(message);
            }
        });
    }
}
```

Output:

10.a) Compare Android OS with any other OS.(3M)

b) Enlist various Android versions. (3M)

c) Explain the need for Android. (4M)

A:

a) <u>Android os vs iOS</u>

Feature	Android OS	iOS (Apple)
Developer	Google	Apple
Devices	Used by many manufacturers (Samsung, Xiaomi, etc.)	Only available on Apple devices (iPhone, iPad)
App Store	Google Play Store	Apple App Store
Customization	Highly customizable (widgets, launchers)	Limited customization
Open Source	Yes (based on Linux and AOSP)	No (closed source)
Voice Assistant	Google Assistant	Siri
File Sharing	Easy via Bluetooth, USB, or apps	Limited to AirDrop, iCloud
Security	Good, but depends on manufacturer updates	Very secure, regular updates

App Flexibility	Allows third-party app stores and sideloading	Only from App Store
User Interface	Varies by manufacturer (Samsung One UI, etc.)	Consistent and uniform

b) Android versions:

August

Octobe

The story of Android dates back to 2003 when Andy Rubin, Rich Miner, Nick Sears, and Chris White co-founded a start-up *Android Inc.* in Palo Alto, California. However, the company was later faced with the insufficiency of funds which brought Google into the picture.

Google could sense the potential the product carried within and sealed a deal worth $50 Million to acquire Android in 2005.

All the four Co-founders soon moved to the Googleplex to continue to develop the OS further under their new owners. The first public Android Beta Version 1.0 was finally published on 5th November 2007.

Nougat	Android 7.0 – 7.1	24 – 25	August 22, 2016
Oreo	Android 8.0	26	August 21, 2017
Pie	Android 9.0	27	August 6, 2018
Android Q	Android 10.0	29	September 3, 2019
Android 11	Android 11.0	30	September 8, 2020
Snow Cone	Android 12.0 – 12.1	31-32	October 4, 2021

C) Need for Android:

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.

1. **Open-Source Platform**
   Android is free and open-source, allowing manufacturers and developers to customize it freely, encouraging innovation.
2. **Wide Device Compatibility**
   It runs on a variety of devices – smartphones, tablets, TVs, smartwatches, and even cars (Android Auto).
3. **Massive App Ecosystem**
   With millions of apps on the Google Play Store, Android supports diverse user needs like communication, entertainment, productivity, etc.
4. **Customizability**
   Users can personalize their device experience with launchers, widgets, and themes.
5. **Affordable Technology**
   Android powers a wide range of budget-friendly devices, making smartphones accessible to a larger population.
6. **Regular Updates and Features**
   Android provides continuous updates, new features, and security patches to improve the user experience.
7. **Global Reach**
   It's the **most widely used mobile OS** in the world, helping connect billions of users globally.

MAD UNIT-2

(1) Name different Android Application Components? Explain them in detail?

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.

1. Activities
2. Services
3. Broadcast Receivers
4. Content Providers

S. No	Components & Description
1	**Activities** They dictate the UI and handle the user interaction to the smart phone screen.
2	**Services** They handle background processing associated with an application.
3	**Broadcast Receivers** They handle communication between Android OS and applications.
4	**Content Providers** They handle data and database management issues.

Activities:

An activity represents a single screen with a user interface, in-short Activity performs actions on the screen.

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

If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched.

An activity is implemented as a subclass of Activity class or AppCompatActivity as follows:

```
public class MainActivity extends Activity {

}
```

2. Services:

A service is a component that runs in the background to perform long- running operations.

For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity.

A service is implemented as a subclass of Service class as follows: public class MyService extends Service {

}

3. Broadcast Receivers:

Broadcast Receivers simply respond to broadcast messages from other applications or from the system.

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.

A broadcast receiver is implemented as a subclass

of BroadcastReceiver class and each message is broadcaster as an Intent object.

public class MyReceiver extends BroadcastReceiver { public void onReceive(context, intent) {

}

4. Content Providers:

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the ContentResolver class.

The data may be stored in the file system, the database or somewhere else entirely.

A content provider is implemented as a subclass of ContentProvider class and must implement a standard set of APIs that enable other applications to perform transactions.

public class MyContentProvider extends ContentProvider { public void onCreate() {

}

Additional Components

There are additional components which will be used in the construction of above mentioned entities, their logic, and wiring between them.

S. No	Components & Description
1	**Fragments** Represents a portion of user interface in an Activity.
2	**Views** UI elements that are drawn on-screen including buttons, lists, forms etc.
3	**Layouts** View hierarchies that control screen format and appearance of the views.
4	**Intents** It is an inter-application message passing framework for communication between android components.
5	**Resources** External elements, such as strings, constants and drawable pictures.
6	**Manifest** Configuration file for the application.

(2) Discuss the importance of Android Manifest File and what are the essential tags available in Manifest File?

Android Manifest file

The AndroidManifest.xml file contains information of your package, including components of the application such as activities, services, broadcast receivers, content providers etc. It performs some other tasks also:

It is responsible to protect the application to access any protected parts by providing the permissions.

It also declares the android api that the application is going to use.

It lists the instrumentation classes. The instrumentation classes provides profiling and other informations. These informations are removed just before the application is published etc.

This is the required xml file for all the android application and located inside the root directory.

Elements of the AndroidManifest.xml file:

<manifest>

manifest is the root element of the AndroidManifest.xml file. It

has **package** attribute that describes the package name of the activity class.

<application>

- **application** is the sub-element of the manifest. It includes the namespace declaration. This element contains several sub-elements that declares the application component such as activity etc.

The commonly used attributes are of this element are **icon**, **label**, **theme** etc.

android:icon represents the icon for all the android application components.

android:label works as the default label for all the application components.

android:theme represents a common theme for all the android activities.

<activity>

activity is the sub-element of application and represents an activity that must be defined in the AndroidManifest.xml file. It has many attributes such as

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

<intent-filter>

intent-filter is the sub-element of activity that describes the type of intent to which activity, service or broadcast receiver can respond to.

<action>

It adds an action for the intent-filter. The intent-filter must have at least one action element.

<category>

It adds a category name to an intent-filter.

(13)

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

[b]Write a program to implement Radio Button component in android.(6M)

(a) In an Android application, the user interface is built using View and ViewGroup objects. There are many types of views and view groups, each of which is a descendant of the View class.

“View objects” are the basic units of user interface expression on the Android

platform. The View class serves as the base for subclasses called "widgets," which offer fully implemented UI objects, like text fields and buttons.

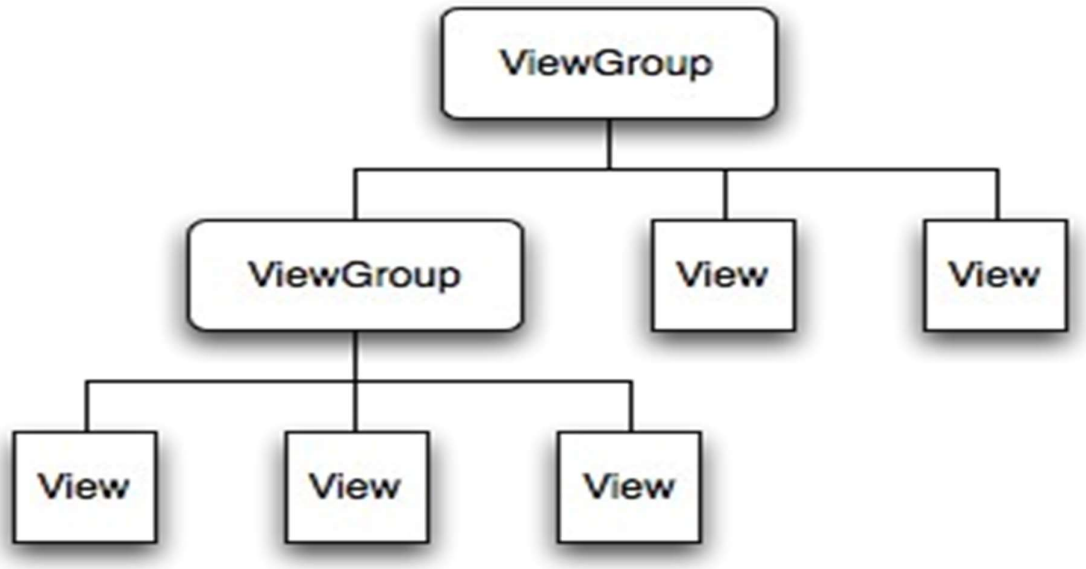
The “ViewGroup” class serves as the base for subclasses called "layouts," which offer different kinds of layout architecture, like linear, tabular and relative.

A View object is a data structure whose properties store the layout parameters and content for a specific rectangular area of the screen. A View object handles its own measurement,

layout, drawing, focus change, scrolling, and key/gesture interactions for the rectangular area of the screen in which it resides. As an object in the user interface, a View is also a point of interaction for the user and the receiver of the interaction events.

View Hierarchy:

On the Android platform, you define an Activity's UI using a hierarchy of View and ViewGroup nodes, as shown in the diagram below. This hierarchy tree can be as simple or complex as you need it to be, and you can build it up using Android's set of predefined widgets and layouts, or with custom Views that you create yourself.



Android Layout:

In android, Layout is used to define the user interface for an app or activity and it will hold the UI elements that will appear to the user.

In android, we can define a layouts in two ways, those are

Declare UI elements in XML

Instantiate layout elements at

The android framework will allow us to use either or both of these methods to define our application's UI.

Linear layout, Relative Layout, Table Layout, Absolute Layout

(b)

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

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

    <EditText
        android:id="@+id/editName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Enter Name" />

    <RadioButton
        android:id="@+id/radioMale"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Male" />

    <RadioButton
        android:id="@+id/radioFemale"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Female" />

    <Button
        android:id="@+id/btnSubmit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Submit" />
</LinearLayout>
```

Main Activity.java:

```
package com.example.simpleform;

import android.os.Bundle;
import android.widget.*;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
    EditText editName;
    RadioButton radioMale, radioFemale;
    Button btnSubmit;

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

        editName = findViewById(R.id.editName);
        radioMale = findViewById(R.id.radioMale);
        radioFemale = findViewById(R.id.radioFemale);
        btnSubmit = findViewById(R.id.btnSubmit);

        btnSubmit.setOnClickListener(v -> {
            String name = editName.getText().toString();
            String gender = radioMale.isChecked() ? "Male" : radioFemale.isChecked() ? "Female" :
"";
        });
    }
```

}

(14) Define Layout in android and explain various layout views available for android development?

Android Layout:

In android, Layout is used to define the user interface for an app or activity and it will hold the UI elements that will appear to the user.

In android, we can define a layouts in two ways, those are

Declare UI elements in XML

Instantiate layout elements at

The android framework will allow us to use either or both of these methods to define our application's UI.

S. No	Layout & Description
1	Linear Layout LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.
2	Relative Layout RelativeLayout is a view group that displays child views in relative positions.
3	Table Layout TableLayout is a view that groups views into rows and columns.
4	Absolute Layout AbsoluteLayout enables you to specify the exact location of its children.
5	Frame Layout The FrameLayout is a placeholder on screen that you can use to display a single view.
6	List View ListView is a view group that displays a list of scrollable items.

Note: Need to explain in detailed manner of the above types

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

Linear Layout is the most basic layout in android studio, that aligns all the children sequentially either in a horizontal manner or a vertical manner by specifying the android:orientation attribute.

If one applies android:orientation="vertical" then elements will be arranged one after another in a vertical manner and If you apply android:orientation="horizontal"

```
<?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">

   <Button
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="Button 1" />

   <Button
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="Button 2" />

   <Button
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="Button 3" />

</LinearLayout>
```

MainActivity.java:

```
package com.example.layoutdemo;

import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main); // linking to activity_main.xml
   }
}
```

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

Relative Layout

Relative Layout is a view group that displays child views in relative positions.

- The Relative Layout is very flexible layout used in android for custom layout designing.
- It gives us the flexibility to position our component/view based on the relative or sibling component's position.

  It allows us to position the component anywhere we want so it is considered as most flexible layout.
- For the same reason Relative layout is the most used layout after the Linear Layout in Android.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:padding="16dp">

  <!-- Top Left Button -->
  <Button
    android:id="@+id/buttonTopLeft"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Top Left Button"
    android:layout_alignParentStart="true"
    android:layout_alignParentTop="true" />

  <!-- Top Right Button -->
  <Button
    android:id="@+id/buttonTopRight"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Top Right Button"
    android:layout_alignParentEnd="true"
    android:layout_alignParentTop="true" />

  <!-- Middle Button -->
  <Button
    android:id="@+id/buttonMiddle"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Middle Button"
```

```
    android:layout_centerInParent="true" />

  <!-- Bottom Left Button -->
  <Button
    android:id="@+id/buttonBottomLeft"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Bottom Left Button"
    android:layout_alignParentBottom="true"
    android:layout_alignParentStart="true" />

  <!-- Bottom Right Button -->
  <Button
    android:id="@+id/buttonBottomRight"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Bottom Right Button"
    android:layout_alignParentBottom="true"
    android:layout_alignParentEnd="true" />

</RelativeLayout>
```

MainActivity.java:

```
package com.example.relativelayoutdemo;

import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;

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

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

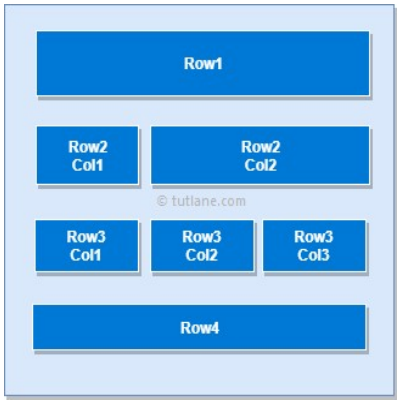
(17) Explain Table Layout with example program in detail. Discuss about attributes used in Table Layout.

Table Layout

Table Layout is a **ViewGroup** subclass that is used to display the child View elements in rows and columns.
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.

Important Points to Remember

There is no need to provide the layout_width and layout_height for TableRow because by default, its width is match_parent and height is wrap_content.
A table row which has maximum views inside it, that many number of columns are created.
The width of the column automatically adjusts based on the size



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.

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

GridLayout is a type of ViewGroup in Android that places its child views in a rectangular grid structure of rows and columns.

- It was introduced in Android 4.0 (API Level 14).
- It is useful when you want to arrange items in a structured grid without nesting multiple layouts.
- It is more flexible and performance-efficient compared to older layouts like TableLayout.

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:rowCount="2"
    android:columnCount="2"
    android:padding="16dp">

    <!-- First Button -->
```

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button 1"
    android:layout_row="0"
    android:layout_column="0"/>

<!-- Second Button -->
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button 2"
    android:layout_row="0"
    android:layout_column="1"/>

<!-- Third Button -->
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button 3"
    android:layout_row="1"
    android:layout_column="0"/>

<!-- Fourth Button -->
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button 4"
```

```
        android:layout_row="1"
        android:layout_column="1"/>

</GridLayout>
```

MainActivity.java

```
package com.example.gridlayoutdemo;

import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main); // Connects the layout XML
    }
}
```

When you run this program, you will see four buttons arranged in a 2x2 grid like this:

Button 1 Button 2

Button 3 Button 4

Each button will occupy one cell of the grid.

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

- Resources are the additional files and static content that your code uses, such as bitmaps,

- ➢ layout definitions,
- ➢ user interface strings,
- ➢ animation instructions, and more.

You should always externalize app resources such as **images** and **strings** from your code, so that you can maintain them independently.

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.

Group ing external Resources:

MyProject/ src/

MyActivity.java res/

drawable/ graphic.png

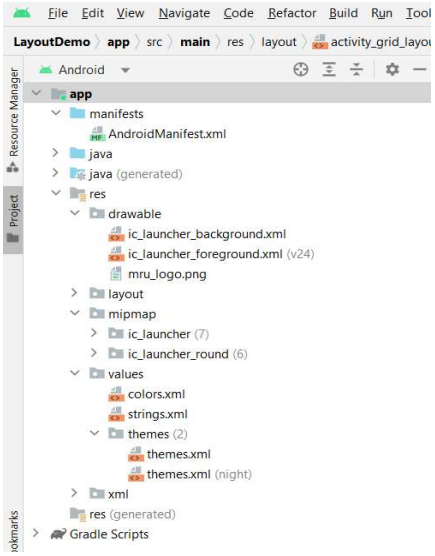
layout/

main.xml info.xml

mipmap/ icon.png

values/

strings.xml



Values folder:

The res/values folder is used to store the values for the resources that are used in many Android projects to include features of color, styles, dimensions etc.

colors.xml: The *colors.xml* is an XML file which is used to store the colors for the resources. An Android project contains 3 essential colours namely:

colorPrimary
colorPrimaryDark
colorAccent

Colors.xml:

name="purple_200">#FFBB86FC</**color**>

<**color name="purple_500"**>#FF6200EE</**color**>

<**color name="purple_700"**>#FF3700B3</**color**>

<**color name="teal_200"**>#FF03DAC5</**color**>

<**color name="teal_700"**>#FF018786</**color**>

<**color name="black"**>#FF000000</**color**>

<**color name="white"**>#FFFFFFFF</**color**>

</**resources**>

strings.xml

<**resources**>

<**string name="app_name"**>Layout Demo</**string**>

</**resources**>

(20)

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

b) Design student Registration form with fields username, password, age, address and button, display the details of student when user

clicks Submit button. (6M)

a)

Of course! Here's a **proper description with steps**, very clean and professional, without emojis:

Steps to Add Images in Android Application

Step 1: Choose and Prepare the Image

- Select the image you want to use (formats like .png, .jpg, .jpeg, .webp are supported).
- Make sure the image file name is in **lowercase** letters and uses **underscores** instead of spaces.
    - Example: my_image.png

Step 2: Add Image to Project

- Open your Android project in **Android Studio**.
- Copy the image file.
- Paste it into the **res/drawable** folder.
    - Path: app > src > main > res > drawable

Step 3: Use the Image in XML Layout

- Open your layout file (for example activity_main.xml).
- Add an <ImageView> element.
- Set the src attribute to point to your image inside the drawable folder.

Example:

```
<ImageView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:src="@drawable/my_image"
   android:contentDescription="Sample Image" />
```

Step 4: (Optional) Access or Change Image in Java/Kotlin

- You can also set the image programmatically in your Java file (MainActivity.java).

Example:

```
ImageView imageView = findViewById(R.id.imageView);
imageView.setImageResource(R.drawable.my_image);
```

Step 5: Run and Test

- Build and run your app.
- You will see the image displayed on your screen inside the ImageView.

Important Tips:

Tip	Description
File Naming	Always use lowercase letters and underscores. No spaces allowed.
Folder Location	Images must be placed inside res/drawable.
Content Description	Always add android:contentDescription for accessibility and better app standards.

Summary:

- Prepare your image properly.
- Place it inside the drawable folder.
- Use an ImageView in XML to display it.
- Optionally change it dynamically using Java/Kotlin.

(b)

1. XML Layout (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/etUsername"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Username" />

<EditText
    android:id="@+id/etPassword"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Password"
    android:inputType="textPassword" />

<EditText
    android:id="@+id/etAge"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Age"
    android:inputType="number" />

<EditText
    android:id="@+id/etAddress"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Address"
    android:inputType="textPostalAddress" />

<Button
```

```
        android:id="@+id/btnSubmit"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Submit" />

    <TextView
        android:id="@+id/tvResult"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text=""
        android:textSize="16sp"
        android:paddingTop="20dp"/>
</LinearLayout>
```

2. Java Code (MainActivity.java)

```
package com.example.registrationform;

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 {

    EditText etUsername, etPassword, etAge, etAddress;
    Button btnSubmit;
```

```
TextView tvResult;

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

    etUsername = findViewById(R.id.etUsername);
    etPassword = findViewById(R.id.etPassword);
    etAge = findViewById(R.id.etAge);
    etAddress = findViewById(R.id.etAddress);
    btnSubmit = findViewById(R.id.btnSubmit);
    tvResult = findViewById(R.id.tvResult);

    btnSubmit.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            String username = etUsername.getText().toString();
            String password = etPassword.getText().toString();
            String age = etAge.getText().toString();
            String address = etAddress.getText().toString();

            String result = "Username: " + username + "\n"
                    + "Password: " + password + "\n"
                    + "Age: " + age + "\n"
                    + "Address: " + address;

            tvResult.setText(result);
```

```
            }
        });
    }
}
```

UNIT-III

FRAGMENTS

21) a) What is Fragment? Explain the procedure to create Fragment.(5M)

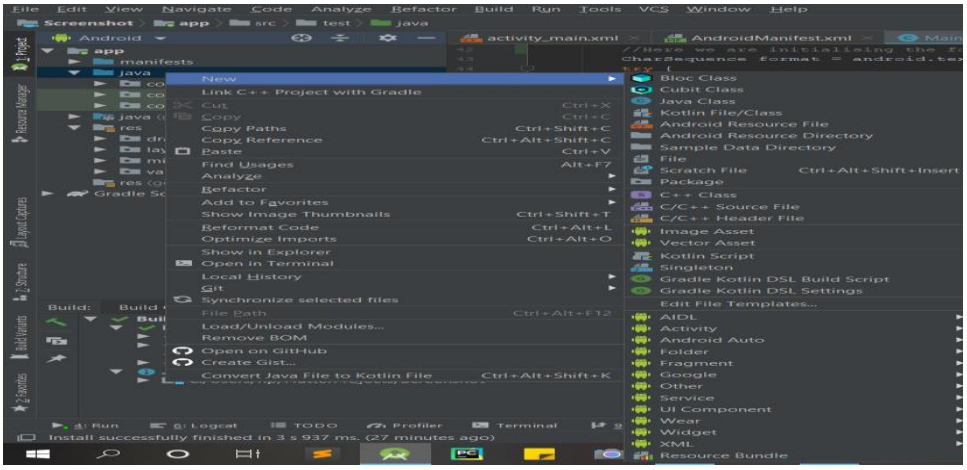
- Android Fragment is the part of activity, it is also known as sub-activity.
- In other words, A Fragment in Android is a component which can be used over an activity to define an independent modular UI component attached to the activity
- There can be more than one fragment in an activity.
- Fragments represent multiple screen inside one activity.
- It functions independently, but as it is linked to the Activity, when an activity is destroyed, the fragment also gets destroyed.
- Android fragment lifecycle is affected by activity lifecycle because fragments are included in activity.
- Also, a fragment is a re-usable component, hence, a single fragment can be included in multiple activities, if required.
- Generally, fragments are used to create multi-pane UI in Android apps.
- A Fragment has it's own Layout for the UI (user interface), but we can even define a fragment without any layout, to implement a behaviors which has no user interface, more like a background service**.**

Step by Step Implementation of creating new fragment.

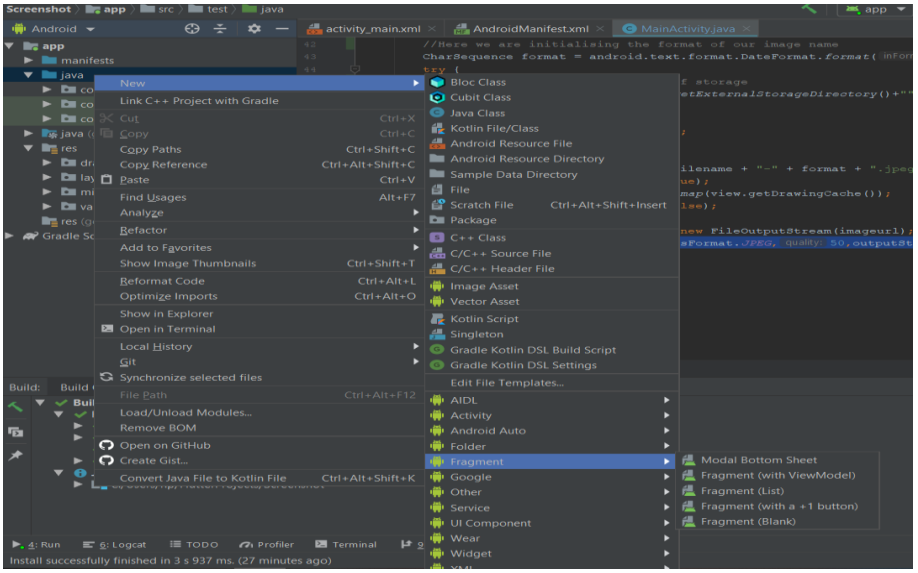
Step 1: Create a New Project in Android Studio

Step 2: Create New Fragment

- Right-Click on the First button inside Java, then click on New.

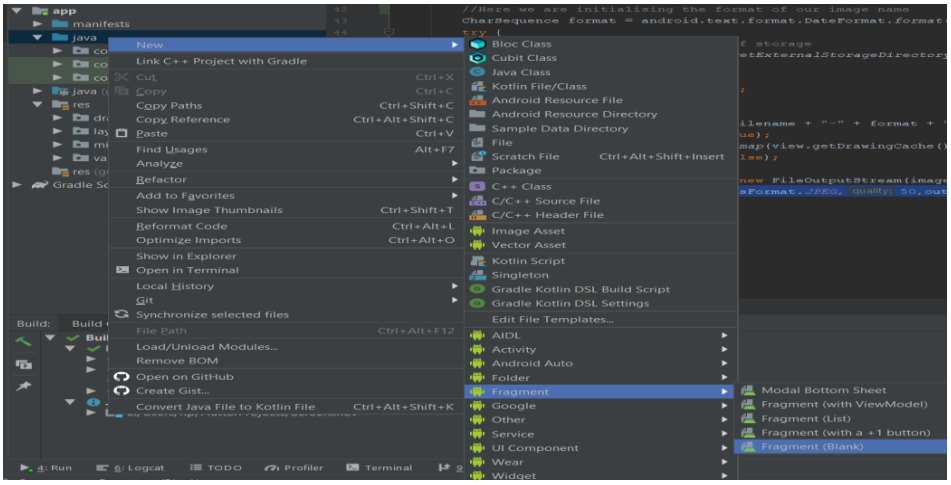


Then Go to Fragment.

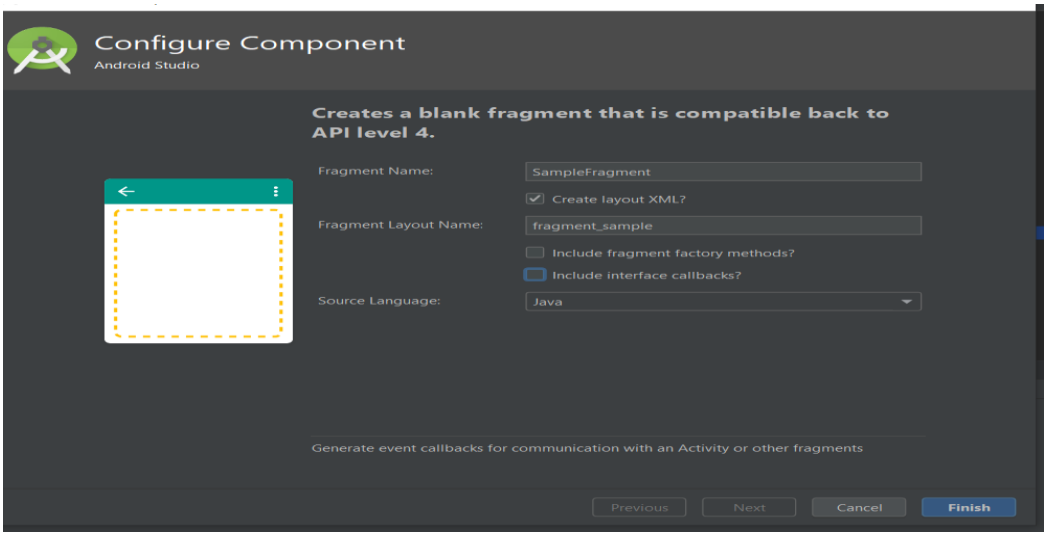


- The next step is to choose the Fragment type.

  Click on Fragment(Blank).



- Now, A new Dialog Box will appear.
- Then, fill in the Fragment Name text field. An XML file is used to provide functionalities of the user interface for our app. An XML file of the fragment is created using the first word in the fragment name.
- Click on Finish. And you have created your Fragment.

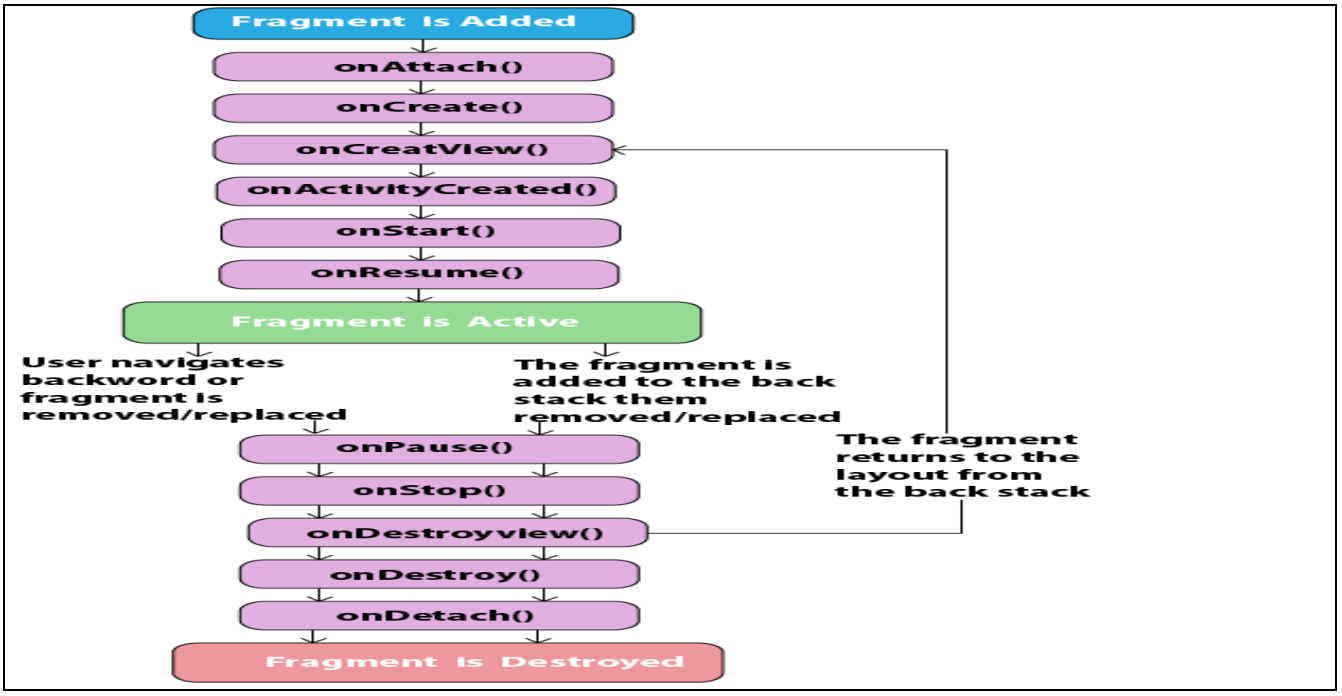


b)Distinguish between Activity and Fragment.(5M)

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?

- In Android, Fragment is a part of an activity which enable more modular activity design. Fragment is a kind of sub-activity.
- In Android, Fragments have their own life cycle very similar to an Activity but it has extra events that are particular to the Fragment's view hierarchy, state and attachment to its activity.



onAttach() The fragment instance is associated with an activity instance.The fragment and the

activity is not fully initialized. Typically you get in this method a reference to the activity which66 uses the fragment for further initialization work.

onCreate() The system calls this method when creating the fragment. You should initializeessential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.

onCreateView() The system calls this callback when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a View component from this method that is the root of your fragment's layout. You can return null if the fragmentdoes not provide a UI.

onActivityCreated() The onActivityCreated() is called after the onCreateView() method when the host activity is created. Activity and fragment instance have been created as well as the view hierarchy of the activity. At this point, view can be accessed with the findViewById() method. example. In this method you can instantiate objects which require a Context object

onStart() The onStart() method is called once the fragment gets visible.

onResume() Fragment becomes active.

onPause() The system calls this method as the first indication that the user is leaving the fragment. This is usually where you should commit any changes that should be persisted beyond the current user session.

onStop() Fragment going to be stopped by calling onStop()

onDestroyView() Fragment view will destroy after call this method

onDestroy() onDestroy() called to do final clean up of the fragment's state but Not guaranteed to be called by the Android platform.

23) a) Write the differences between Frame Layout and Relative Layout.(5M)

Feature	Frame Layout	Relative Layout
Basic Concept	Stacks all child views one over the other, like layers.	Positions views relative to each other or the parent container.
View Positioning	Each view starts from the top-left corner by default; newer views can cover older ones.	Views can be placed above, below, left, right of other views, or aligned to parent edges.
Layout Complexity	Simple and minimal. Used for basic view stacking.	More complex. Suitable for designing detailed and flexible UI layouts.
Overlapping	Common due to stacking behavior; needs	Overlapping is less likely due to relative positioning.

Feature	Frame Layout	Relative Layout
	margins/padding to avoid overlap.	
XML Attributes Used	Uses fewer layout attributes (e.g., layout_gravity).	Uses many attributes like layout_alignParentTop, layout_below, layout_toRightOf, etc.
Use Cases	Useful for displaying a single child, background images, or switching fragments.	Ideal for login forms, profile screens, or any interface requiring precise element placement.
Performance	Lightweight, hence better performance in simple UIs.	Slightly more resource-intensive for resolving position dependencies.
Flexibility	Less flexible in arranging UI components.	Highly flexible for dynamic and responsive UI designs.

b)Discuss the attributes related to Frame Layout.(5M)

Attributes Related to Frame Layout

android:layout_width and android:layout_height

- Define the width and height of the layout or view.
- Values can be match_parent, wrap_content, or a fixed size (e.g., 200dp).

android:layout_gravity

- Sets the position of the view inside the FrameLayout (e.g., center, bottom, right, etc.).
- Useful when placing views at specific areas within the FrameLayout.

android:foreground

- Used to display a drawable or image on top of all child views.
- Common for adding a visual overlay or a semi-transparent effect.

android:foregroundGravity

- Controls the position of the foreground drawable (e.g., center, top, end).
- Works with android:foreground.

android:padding / android:layout_margin

- Padding adds space inside the view boundaries.
- Margin adds space outside the view boundaries.
- Helps in preventing overlap or crowding of views.

24) Create an application which converts the Text to Speech.

Steps to Create Text-to-Speech App in Android (Java)

Design Layout: 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/editTextInput"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Enter text to speak" />

    <Button
        android:id="@+id/buttonSpeak"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Speak"
        android:layout_marginTop="16dp" />
</LinearLayout>
```

Java Code: MainActivity.java

```
package com.example.texttospeechapp;

import android.os.Bundle;
import android.speech.tts.TextToSpeech;
import android.widget.Button;
import android.widget.EditText;
import androidx.appcompat.app.AppCompatActivity;
import java.util.Locale;

public class MainActivity extends AppCompatActivity {

    EditText editTextInput;
    Button buttonSpeak;
    TextToSpeech textToSpeech;

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

        editTextInput = findViewById(R.id.editTextInput);
        buttonSpeak = findViewById(R.id.buttonSpeak);

        textToSpeech = new TextToSpeech(getApplicationContext(), status -> {
            if (status == TextToSpeech.SUCCESS) {
                textToSpeech.setLanguage(Locale.ENGLISH);
            }
        });

        buttonSpeak.setOnClickListener(v -> {
            String text = editTextInput.getText().toString();
            if (!text.isEmpty()) {
                textToSpeech.speak(text, TextToSpeech.QUEUE_FLUSH, null, null);
            }
```

```
        });
    }

    @Override
    protected void onDestroy() {
        if (textToSpeech != null) {
            textToSpeech.stop();
            textToSpeech.shutdown();
        }
        super.onDestroy();
    }
}
btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                FragmentManager fm=getSupportFragmentManager();
                FragmentTransaction ft=fm.beginTransaction();
                Bundle bundle=new Bundle();
                bundle.putString("message",e1.getText().toString());
                FirstFragment f1=new FirstFragment();
                f1.setArguments(bundle);
                ft.replace(R.id.frameLayout,f1).commit();
            }
        });
    }
}
```

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

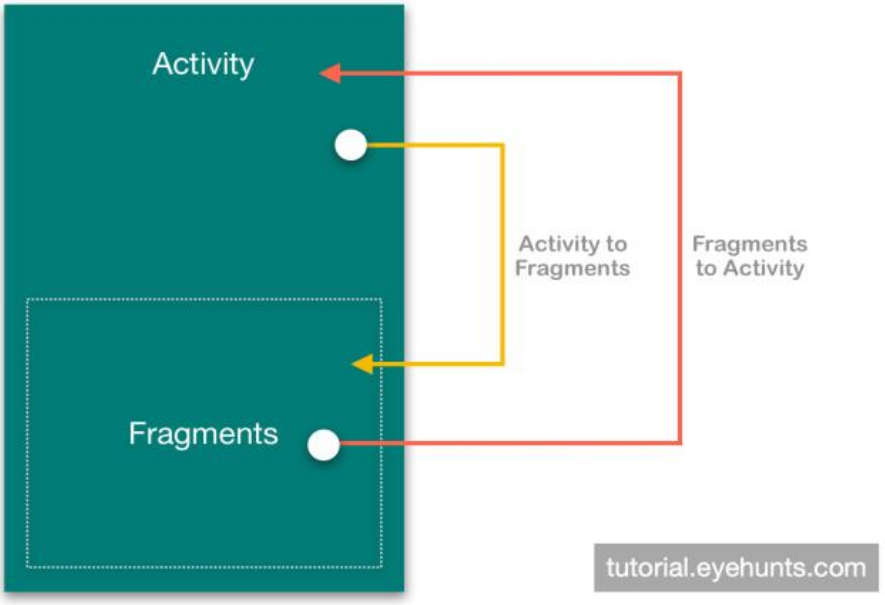
Interfacing Between Fragments and Activities Use the getActivity method within any Fragment to return a reference to the Activity within which it's embedded.

This is particularly useful for finding the current Context, accessing other Fragments using the Fragment Manager, and finding Views within the Activity's View hierarchy.

```
TextView textView = (TextView)getActivity().findViewById(R.id.textview);
```

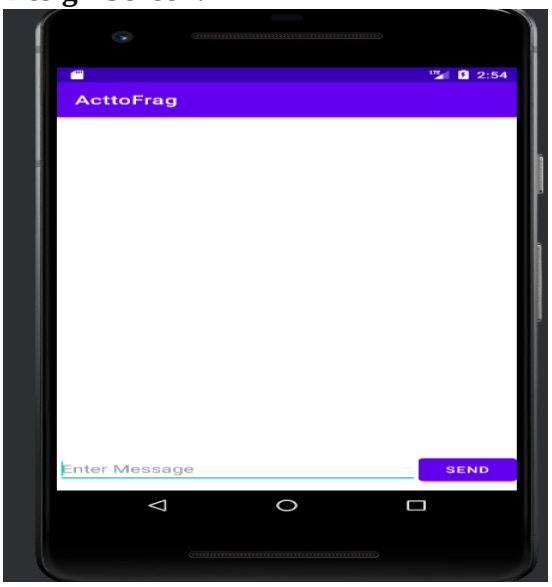
Although it's possible for Fragments to communicate directly using the host Activity's Fragment Manager, it's generally considered better practice to use the Activity as an intermediary.

Activity and fragment communication are important when you want to update data or action.



Example that helps you to understand how data can be passed from Activity to Fragment.

Design Screen:



activity_main.xml

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

    <FrameLayout
        android:layout_weight="1"
        android:id="@+id/frameLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

    </FrameLayout>
    <LinearLayout
        android:layout_marginBottom="10dp"
```

```
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap
<EditText
            android:id="@+id/e1"
            android:layout_weight="1"
            android:hint="Enter Message"
            android:layout_width="match_parent"
            android:layout_height="wrap_content">
        </EditText>
        <Button
            android:text="Send"
            android:id="@+id/btn"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">

        </Button>
    </LinearLayout>

</LinearLayout>
```

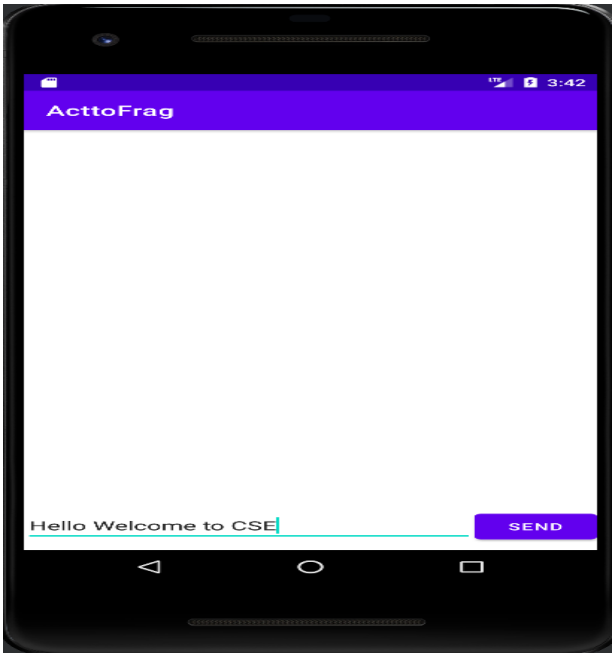
fragment_first.xml:

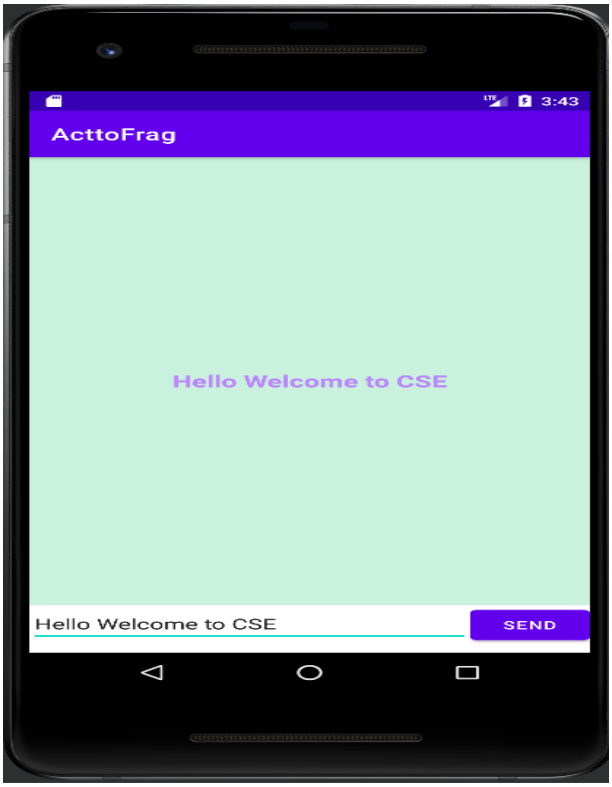
```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background=" #CAF1DE"
    tools:context=".FirstFragment">


    <TextView
        android:id="@+id/t1"
        android:textColor="@color/purple_200"
        android:textStyle="bold"
        android:textSize="20sp"
        android:layout_gravity="center"
        android:gravity="center"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
    />

</FrameLayout>
```

MainActivity.java

```
package com.example.acttofrag;

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

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

```
import android.widget.EditText;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {
Button btn;
EditText e1;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        e1=findViewById(R.id.e1);
        btn=findViewById(R.id.btn);
```





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

onAttach() The fragment instance is associated with an activity instance.The fragment and the activity is not fully initialized. Typically you get in this method a reference to the activity which66 uses the fragment for further initialization work.

onCreate() The system calls this method when creating the fragment. You should initializeessential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.

onCreateView() The system calls this callback when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a View component from this method that is the root of your fragment's layout. You can return null if the fragmentdoes not provide a UI.

onActivityCreated() The onActivityCreated() is called after the onCreateView() method when the host activity is created. Activity and fragment instance have been created as well as the view hierarchy of the activity. At this point, view can be accessed with the findViewById() method. example. In this method you can instantiate objects which require a Context object

onStart() The onStart() method is called once the fragment gets visible.

onResume() Fragment becomes active.

onPause() The system calls this method as the first indication that the user is leaving the fragment. This is usually where you should commit any changes that should be persisted beyond the current user session.

onStop() Fragment going to be stopped by calling onStop()

onDestroyView() Fragment view will destroy after call this method

onDestroy() onDestroy() called to do final clean up of the fragment's state but Not guaranteed to be called by the Android platform.

```
public class MyFragment extends Fragment {

   // 1. Called when fragment is first attached to the activity
   @Override
   public void onAttach(Context context) {
      super.onAttach(context);
      Log.d("Lifecycle", "onAttach called");
   }
```

```
// 2. Called to initialize the fragment
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   Log.d("Lifecycle", "onCreate called");
}

// 3. Creates and returns the UI layout of the fragment
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
   Log.d("Lifecycle", "onCreateView called");
   return inflater.inflate(R.layout.fragment_my, container, false);
}

// 4. Called immediately after onCreateView
@Override
public void onViewCreated(View view, Bundle savedInstanceState) {
   super.onViewCreated(view, savedInstanceState);
   Log.d("Lifecycle", "onViewCreated called");
}

// 5. Called when the activity and fragment's view hierarchy is created
@Override
public void onActivityCreated(Bundle savedInstanceState) {
   super.onActivityCreated(savedInstanceState);
   Log.d("Lifecycle", "onActivityCreated called");
}

// 6. Called when fragment becomes visible
@Override
public void onStart() {
   super.onStart();
   Log.d("Lifecycle", "onStart called");
}

// 7. Called when fragment is active (user can interact)
@Override
public void onResume() {
   super.onResume();
   Log.d("Lifecycle", "onResume called");
}

// 8. Called when fragment is paused (not in foreground)
@Override
public void onPause() {
   super.onPause();
   Log.d("Lifecycle", "onPause called");
}
```

```
    // 9. Called when fragment is no longer visible
    @Override
    public void onStop() {
        super.onStop();
        Log.d("Lifecycle", "onStop called");
    }

    // 10. Called when the fragment's UI is destroyed
    @Override
    public void onDestroyView() {
        super.onDestroyView();
        Log.d("Lifecycle", "onDestroyView called");
    }

    // 11. Called to clean up remaining fragment state
    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d("Lifecycle", "onDestroy called");
    }

    // 12. Called when the fragment is detached from the activity
    @Override
    public void onDetach() {
        super.onDetach();
        Log.d("Lifecycle", "onDetach called");
    }
}
```

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

- Fragment Transactions can be used to add, remove, and replace Fragments within an Activity at run time. Using Fragment Transactions, you can make your layouts dynamic — that is, they will adapt and change based on user interactions and application state.
- Each Fragment Transaction can include any combination of supported actions, including adding, removing, or replacing Fragments. They also support the specification of the transition animations to display and whether to include the Transaction on the back stack.
- A new Fragment Transaction is created using the beginTransaction method from the Activity's Fragment Manager. Modify the layout using the add, remove, and replace methods, as required, before setting the animations to display, and setting the appropriate back-stack behavior. When you are ready to execute the change, call commit to add the transaction to the UI queue.
- **Create Two Fragments:**
  Fragment A: FragmentA.java

```
public class FragmentA extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        return inflater.inflate(R.layout.fragment_a, container, false);
    }
}
```

Fragment B: FragmentB.java

```
public class FragmentB extends Fragment {
    @Override
```

```
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        return inflater.inflate(R.layout.fragment_b, container, false);
    }
}
```

2. Create Layouts for Fragments:

Fragment A Layout: fragment_a.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center">

    <TextView android:text="Fragment A" android:textSize="18sp"/>
</LinearLayout>
```

Fragment B Layout: fragment_b.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center">

    <TextView android:text="Fragment B" android:textSize="18sp"/>
</LinearLayout>
```

3. Main Activity with Fragment Replacement:

MainActivity.java

```
public class MainActivity extends AppCompatActivity {

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

        if (savedInstanceState == null) {
            // Initial fragment
            getSupportFragmentManager().beginTransaction()
                .replace(R.id.fragment_container, new FragmentA())
                .commit();
        }

        // Button to switch fragments
        findViewById(R.id.btnReplaceFragment).setOnClickListener(v -> {
            Fragment fragment =
(getSupportFragmentManager().findFragmentById(R.id.fragment_container) instanceof FragmentA)
                    ? new FragmentB() : new FragmentA();
            getSupportFragmentManager().beginTransaction()
                .replace(R.id.fragment_container, fragment)
                .commit();
        });
    }
}
```

4. Main Activity Layout:

activity_main.xml

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

    <FrameLayout
        android:id="@+id/fragment_container"
        android:layout_width="match_parent"
        android:layout_height="300dp"/>

    <Button
        android:id="@+id/btnReplaceFragment"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Replace Fragment"
        android:layout_marginTop="20dp"
        android:layout_gravity="center"/>
</LinearLayout>
```

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 color when user clicks on any one of these buttons.

Main Activity: MainActivity.java

```
package com.example.colorchanger;

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

public class MainActivity extends AppCompatActivity {

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

        // Get buttons by their IDs
        Button redButton = findViewById(R.id.redButton);
        Button greenButton = findViewById(R.id.greenButton);
        Button blueButton = findViewById(R.id.blueButton);

        // Set click listeners for each button
        redButton.setOnClickListener(v ->
getWindow().getDecorView().setBackgroundColor(getResources().getColor(R.color.red)));
 greenButton.setOnClickListener(v ->
getWindow().getDecorView().setBackgroundColor(getResources().getColor(R.color.green)));
blueButton.setOnClickListener(v ->
getWindow().getDecorView().setBackgroundColor(getResources().getColor(R.color.blue)));
    }
}
```

Main Activity Layout: 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:gravity="center"
    android:padding="16dp">

    <!-- Red Button -->
    <Button
        android:id="@+id/redButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Red" />

    <!-- Green Button -->
    <Button
        android:id="@+id/greenButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Green"
        android:layout_marginTop="20dp"/>

    <!-- Blue Button -->
    <Button
        android:id="@+id/blueButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Blue"
        android:layout_marginTop="20dp"/>

</LinearLayout>
```

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

To add fragments to an activity in Android, we need to use FragmentTransaction to dynamically add fragments to the activity's layout. Fragments can also be added statically by defining them directly in the layout XML, but in most cases, especially when fragments need to change during runtime, you will use FragmentTransaction.

Steps to Add Fragments to Activity

1. Define the Fragment: Create a class that extends Fragment and override the necessary lifecycle methods.
2. Activity Layout: Define a FrameLayout or any container where the fragment will be placed.
3. FragmentTransaction: Use FragmentTransaction in the activity to add the fragment dynamically.

1. Create Fragment: MyFragment.java

```
public class MyFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        return inflater.inflate(R.layout.fragment_my, container, false);
    }
}
```

2. Fragment Layout: fragment_my.xml

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

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

    <TextView
        android:id="@+id/fragmentText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello from MyFragment"
        android:textSize="18sp"/>
</LinearLayout>
```

3. Main Activity Layout: 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">

    <Button
        android:id="@+id/buttonAddFragment"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Add Fragment"
        android:layout_gravity="center"/>

    <FrameLayout
        android:id="@+id/fragment_container"
        android:layout_width="match_parent"
        android:layout_height="300dp"
        android:layout_marginTop="20dp"/>
</LinearLayout>
```

4. Main Activity: MainActivity.java

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

        Button buttonAddFragment = findViewById(R.id.buttonAddFragment);

        // Set a listener to add fragment when the button is clicked
        buttonAddFragment.setOnClickListener(v -> {
            // Create the fragment
            MyFragment fragment = new MyFragment();

            // Add the fragment to the container
            getSupportFragmentManager().beginTransaction()
                .add(R.id.fragment_container, fragment)
                .commit();
        });
    }
}
```

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

1. MainActivity Layout (activity_main.xml)

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

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

    <com.google.android.material.bottomnavigation.BottomNavigationView
        android:id="@+id/bottom_navigation"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:menu="@menu/bottom_navigation_menu"/>
</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

2. Bottom Navigation Menu (res/menu/bottom_navigation_menu.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/nav_home" android:title="Home"
android:icon="@android:drawable/ic_menu_home"/>
    <item android:id="@+id/nav_profile" android:title="Profile"
android:icon="@android:drawable/ic_menu_myplaces"/>
    <item android:id="@+id/nav_settings" android:title="Settings"
android:icon="@android:drawable/ic_menu_manage"/>
</menu>
```

3. Create Fragments (HomeFragment, ProfileFragment, SettingsFragment)

HomeFragment.java

```
public class HomeFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        return inflater.inflate(R.layout.fragment_home, container, false);
    }
}
```

ProfileFragment.java

```
public class ProfileFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        return inflater.inflate(R.layout.fragment_profile, container, false);
    }
}
```

SettingsFragment.java

```
public class SettingsFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        return inflater.inflate(R.layout.fragment_settings, container, false);
    }
```

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

        BottomNavigationView bottomNavigationView = findViewById(R.id.bottom_navigation);

        // Set default fragment
        getSupportFragmentManager().beginTransaction()
                .replace(R.id.fragment_container, new HomeFragment())
                .commit();

        bottomNavigationView.setOnNavigationItemSelectedListener(item -> {
            Fragment selectedFragment = null;

            switch (item.getItemId()) {
                case R.id.nav_home:
                    selectedFragment = new HomeFragment();
                    break;
                case R.id.nav_profile:
                    selectedFragment = new ProfileFragment();
                    break;
                case R.id.nav_settings:
                    selectedFragment = new SettingsFragment();
                    break;
            }

            getSupportFragmentManager().beginTransaction()
                    .replace(R.id.fragment_container, selectedFragment)
                    .commit();
            return true;
        });
    }
}
```
5. Fragment Layouts:

fragment_home.xml
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center">
    <TextView android:text="Home Fragment" android:textSize="20sp"/>
</LinearLayout>
```
fragment_profile.xml
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center">
    <TextView android:text="Profile Fragment" android:textSize="20sp"/>
</LinearLayout>
```
fragment_settings.xml
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

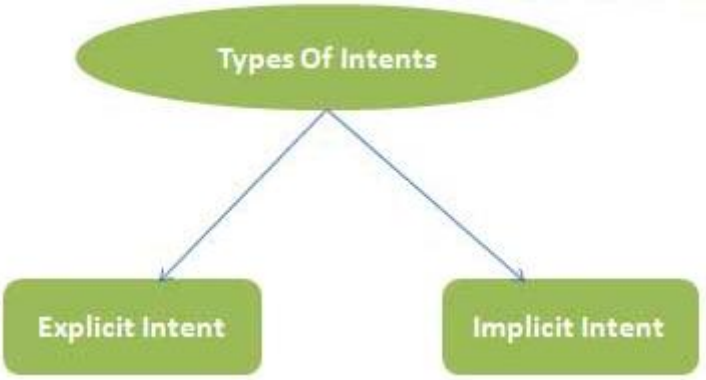
```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center">
    <TextView android:text="Settings Fragment" android:textSize="20sp"/>
</LinearLayout>
```

UNIT-IV

31) Define Intent. Mention different types of intents. Briefly discuss about uses of intents and methods supported by Intent class.

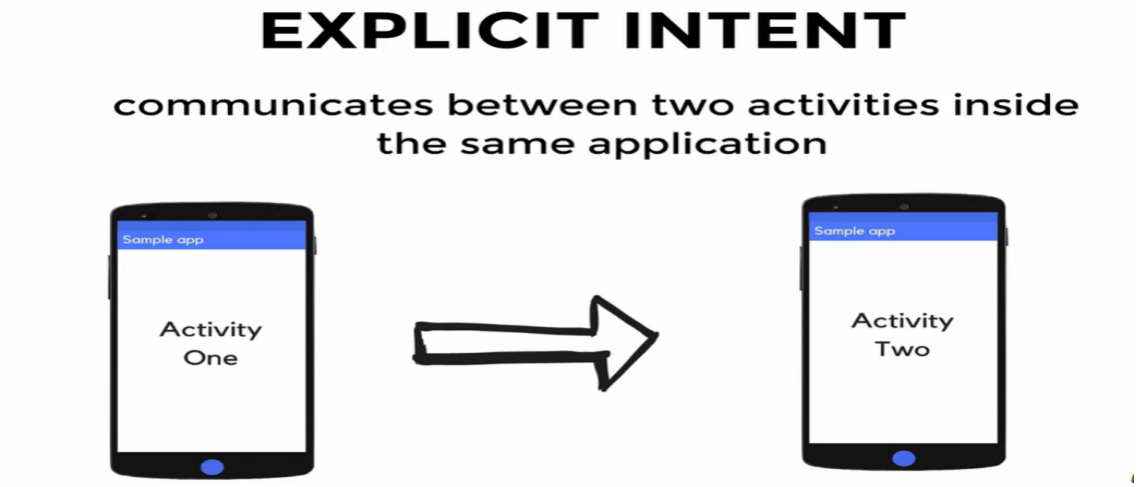
A)In **Android development**, an **Intent** is a messaging object used to request an action from another component of the Android system (such as activities, services, or broadcast receivers). It is a fundamental building block for facilitating communication between components.

- Using intents to launch Activities
- Types of Intents
- Passing data to Intents
- Getting results from Activities
- Using intents to launch Activities
- Types of Intents
- Passing data to Intents
- Getting results from Activities



1)Explicit Intent:

- Explicit Intent specifies the component. In such a case, intent provides the external class to be invoked.
- Specifies the **exact component** (e.g., a specific activity or service) to start.
- Used to **launch a specific class** within the same application.



- Example:
- **Syntax:**

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

2)**Implicit Intent:**

- Does **not specify a component**.
- Declares a **general action** to perform (like "view" or "send").
- The Android system then finds a suitable component

Example:
Syntax:
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("https://www.google.com")); startActivity(intent);

Uses of Intents:

- **Start another activity** (either within the app or external).
- **Start or bind a service** (background operation).
- **Deliver a broadcast** to multiple components.
- **Pass data** between activities or components using extras.
- **Interact with system apps** like camera, phone dialer, or browser.

Methods Supported by the Intent Class:

Method	Description
`setAction(String action)`	Sets the general action to be performed (e.g., `Intent.ACTION_VIEW`)
`setData(Uri data)`	Sets the data URI
`setType(String type)`	Sets the MIME type of the data
`putExtra(String name, Object value)`	Adds extended data to the intent
`getExtras()`	Retrieves the data from the intent
`getStringExtra(String name)`	Retrieves a string value from extras
`setComponent(ComponentName component)`	Specifies the exact class to be invoked

Method	Description
`setClass(Context packageContext, Class<?> cls)`	Specifies the class directly

32) What is Implicit intent? Explain implicit intent with the help of example program.

A) An **Implicit Intent** is used when you want to perform an action **without specifying the target component** (like a specific Activity or Service). Instead, you declare a general action you want to perform, and the **Android system finds the appropriate app or component** to handle that action.

Key Points:

- Used to **interact with system apps or other apps**.
- Common actions: ACTION_VIEW, ACTION_SEND, ACTION_DIAL, etc.
- Can be used to **open web pages, share data, send emails**, etc.

Example:
MainActivity.java

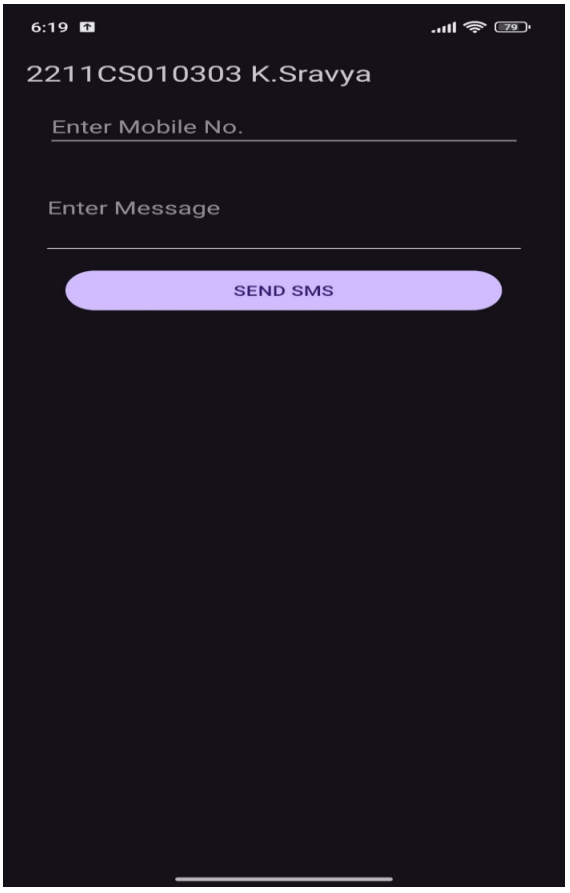
```
package com.example.implicitinetent;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

import androidx.activity.EdgeToEdge;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
   EditText Mobile, Message;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      EdgeToEdge.enable(this);
      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);
}
```

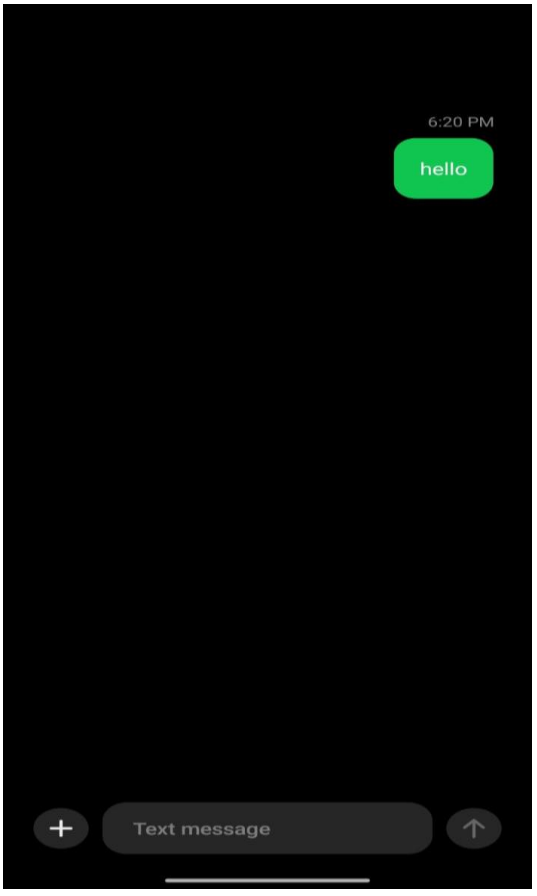
```
}
```

MainActivity.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   android:padding="16dp"
   tools:context=".MainActivity">

   <EditText
      android:id="@+id/etMobile"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="Enter Mobile Number"
      android:inputType="phone"
      android:layout_marginBottom="16dp" />

   <EditText
      android:id="@+id/etText"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="Enter Message"
      android:inputType="textMultiLine"
      android:minLines="3"
      android:gravity="top"
      android:layout_marginBottom="16dp" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Send SMS"
      android:onClick="SendSms" />
</LinearLayout>
```

OUTPUT:





33) What is Explicit intent? Discuss how explicit intent is helpful to navigate from one activity to another activity with the help of example program.

A)An **Explicit Intent** is used to **start a specific component** (such as another Activity, Service, or BroadcastReceiver) **within the same app** or a known external app by specifying the **exact class name**.

Key Points:

- Directly specifies the **target component** using class name.
- Commonly used to **navigate between activities** in the same application.
- Allows passing data from one activity to another using putExtra()

Use of Explicit Intent to Navigate Between Activities

Suppose we have two activities:

- MainActivity – the starting screen.
- SecondActivity – the target screen we want to navigate to.

Example:

MainActivity.java:

```
package com.example.explicit;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;

import androidx.activity.EdgeToEdge;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;

public class MainActivity extends AppCompatActivity {



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



    }
    public void SecondActivity(View view){



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

        startActivity(intent);

    }
}
```

main_activity.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:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/button"
        android:layout_width="155dp"
        android:layout_height="40dp"
        android:text="SecondActivity"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.499"
        android:onClick="SecondActivity"/>

    <TextView
        android:id="@+id/textView"
        android:layout_width="144dp"
        android:layout_height="29dp"
        android:text="First Activity"
        app:layout_constraintBottom_toTopOf="@+id/button"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

SecondActivity.java:

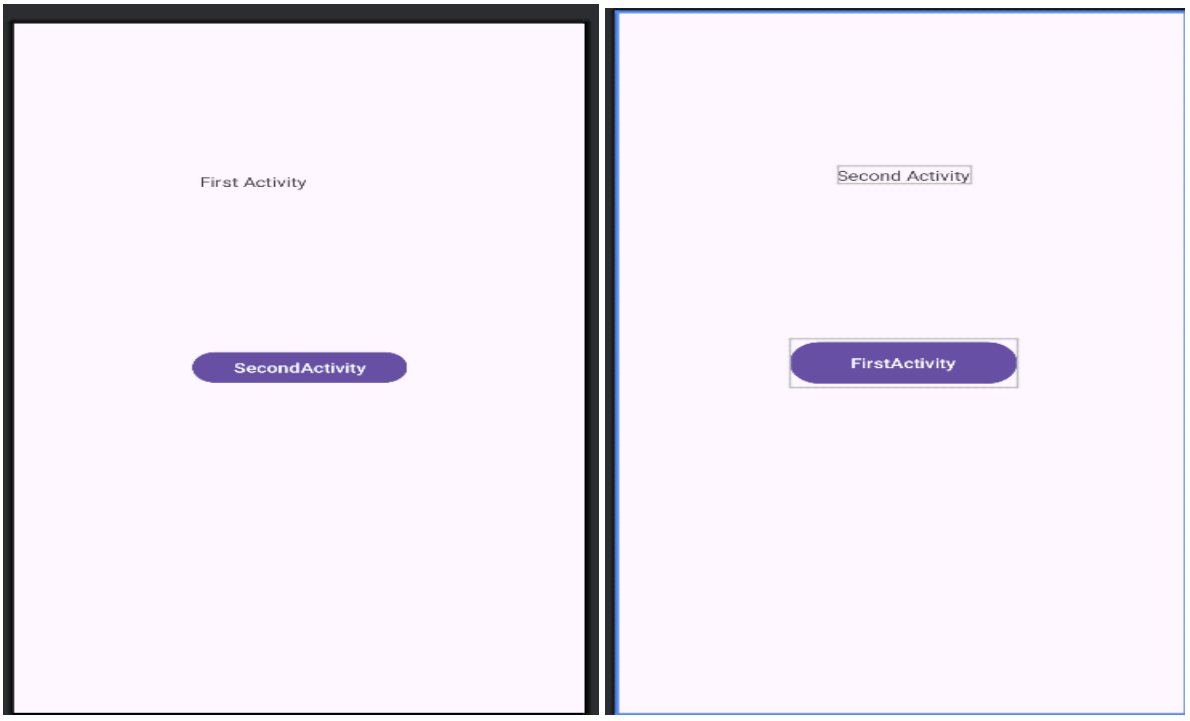
```
package com.example.explicit;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;

import androidx.activity.EdgeToEdge;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;

public class SecondActivity extends AppCompatActivity {

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

```
    }
    public void FirstActivity(View view){



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

        startActivity(intent);

    }
}
```

activity_second:

```
<?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:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SecondActivity">

    <Button
        android:id="@+id/button2"
        android:layout_width="164dp"
        android:layout_height="51dp"
        android:text="FirstActivity"
        android:onClick="FirstActivity"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Second Activity"
        app:layout_constraintBottom_toTopOf="@+id/button2"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

OUTPUT:



34) Create “Send SMS” application using the concept of implicit intent.

A) Sure! Here's a simple **“Send SMS” application** using **Implicit Intent** in Android. This app allows the user to enter a mobile number and a message, then opens the default SMS app with the message pre-filled.

Project Structure Overview

- **Activity:** MainActivity.java
- **Layout:** activity_main.xml
- **Functionality:** Uses Intent.ACTION_VIEW with "sms" scheme.

Example:
MainActivity.java

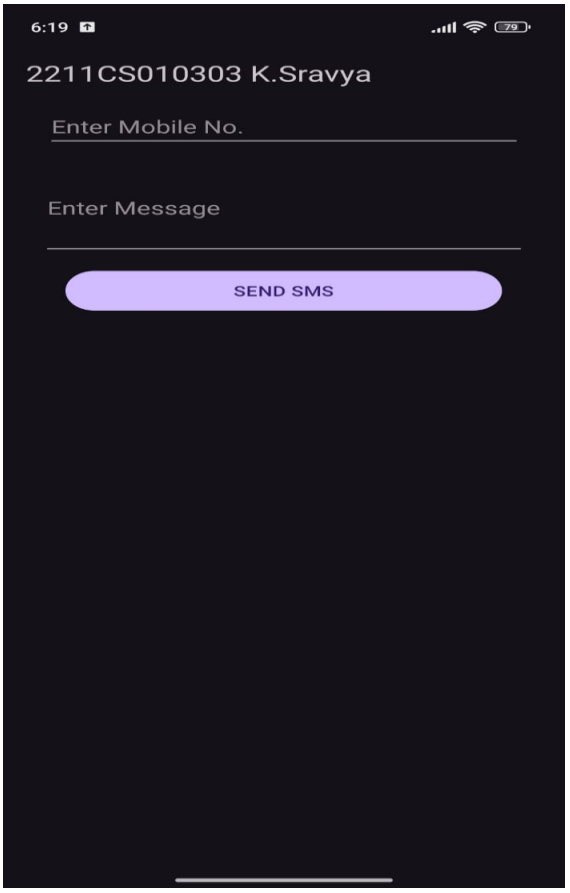
```
package com.example.implicitinetent;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

import androidx.activity.EdgeToEdge;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
   EditText Mobile, Message;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        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);
}

}
```

MainActivity.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/etMobile"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Enter Mobile Number"
        android:inputType="phone"
        android:layout_marginBottom="16dp" />

    <EditText
        android:id="@+id/etText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Enter Message"
        android:inputType="textMultiLine"
        android:minLines="3"
        android:gravity="top"
        android:layout_marginBottom="16dp" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Send SMS"
        android:onClick="SendSms" />
</LinearLayout>
```

OUTPUT:



35)Discuss how we can pass data to intents with the help of an example.

A)Passing Data with Intents in Android

You can pass data to another activity using **Intent extras**. This is done using the putExtra() method to **attach key-value pairs** to the intent, and getIntent().get<Type>Extra() to retrieve them in the target activity.

Why pass data?
When navigating between screens (activities), you often want to send:

- A string (e.g., username)
- An integer (e.g., score)
- An object (using Serializable or Parcelable)

Example:

MainActivity.java
package com.example.passdata;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;

```
import android.widget.Button;
import android.widget.EditText;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
  EditText inputName;

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

    inputName = findViewById(R.id.etName);
    Button sendButton = findViewById(R.id.btnSend);

    sendButton.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        String name = inputName.getText().toString();

        // Create intent and attach data
        Intent intent = new Intent(MainActivity.this, SecondActivity.class);
        intent.putExtra("username", name);
        startActivity(intent);
      }
    });
  }
}
```

SecondActivity.java:

```
package com.example.passdata;

import android.os.Bundle;
import android.widget.TextView;
import androidx.appcompat.app.AppCompatActivity;

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

    TextView welcomeText = findViewById(R.id.tvWelcome);

    // Retrieve data from intent
    String name = getIntent().getStringExtra("username");
    welcomeText.setText("Welcome, " + name + "!");
  }
}
```

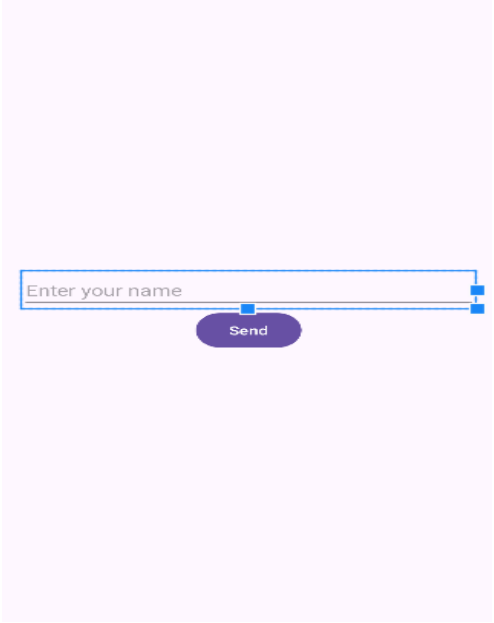
Activity_main.xml:

```
<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"
  android:gravity="center">

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

  <Button
    android:id="@+id/btnSend"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Send" />
</LinearLayout>
```

Activity_second.xml:

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

  <TextView
    android:id="@+id/tvWelcome"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Welcome!"
    android:textSize="20sp" />
</LinearLayout>
```

OUTPUT:



Method	Purpose
putExtra(String key, value)	Add data to intent
getIntent().getStringExtra("key")	Retrieve data from intent

36) Explain how we can get the results from second activity to first activity with the help of example.

A)Introduced in AndroidX to replace startActivityForResult & onActivityResult.

SecondActivity Returns a Name to MainActivity

Goal:

- MainActivity opens SecondActivity.
- User enters a message and taps "Send Back".
- The message is **returned and displayed** in MainActivity.

MainActivity.java:

```
package com.example.resultdemo;

import android.content.Intent;
import android.os.Bundle;
import android.widget.Button;
import android.widget.TextView;
import androidx.activity.result.ActivityResultLauncher;
import androidx.activity.result.contract.ActivityResultContracts;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
```

```
    TextView resultText;

    // Register launcher for result
    ActivityResultLauncher<Intent> resultLauncher = registerForActivityResult(
        new ActivityResultContracts.StartActivityForResult(),
        result -> {
            if (result.getResultCode() == RESULT_OK) {
                Intent data = result.getData();
                if (data != null) {
                    String returnedMessage = data.getStringExtra("returnData");
                    resultText.setText("Received: " + returnedMessage);
                }
            }
        });

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

        resultText = findViewById(R.id.tvResult);
        Button btnOpen = findViewById(R.id.btnOpen);

        btnOpen.setOnClickListener(v -> {
            Intent intent = new Intent(MainActivity.this, SecondActivity.class);
            resultLauncher.launch(intent);
        });
    }
}
```

SecondActivity.java:

```
package com.example.resultdemo;

import android.content.Intent;
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
import androidx.appcompat.app.AppCompatActivity;

public class SecondActivity extends AppCompatActivity {
    EditText editText;

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

```
        editText = findViewById(R.id.etReturnData);
        Button btnReturn = findViewById(R.id.btnReturn);

        btnReturn.setOnClickListener(v -> {
            String msg = editText.getText().toString();

            Intent resultIntent = new Intent();
            resultIntent.putExtra("returnData", msg);

            setResult(RESULT_OK, resultIntent);
            finish(); // End activity and return to MainActivity
        });
    }
}
```

Activity_main.xml:

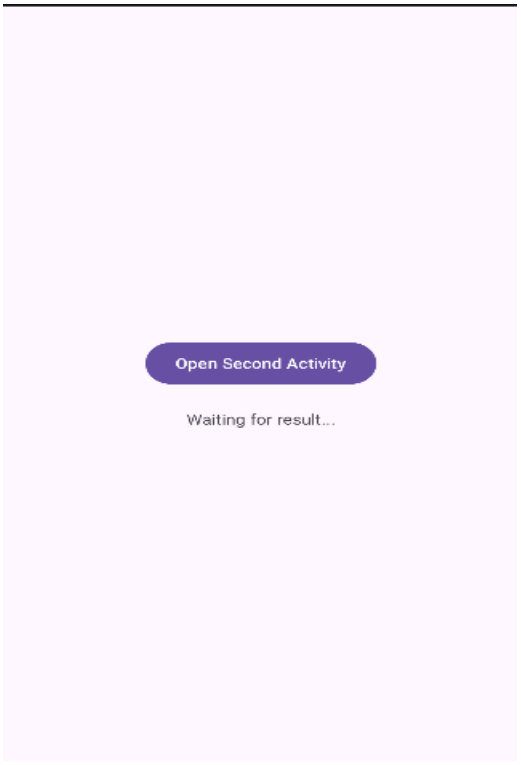
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="20dp"
    android:gravity="center">

    <Button
        android:id="@+id/btnOpen"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Open Second Activity" />

    <TextView
        android:id="@+id/tvResult"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Waiting for result..."
        android:layout_marginTop="20dp" />
</LinearLayout>
```

Activity_second.xml:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="20dp"
    android:gravity="center">

    <EditText
        android:id="@+id/etReturnData"
        android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
    android:hint="Enter data to send back" />

  <Button+
    android:id="@+id/btnReturn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Send Back"
    android:layout_marginTop="16dp" />
</LinearLayout>
```

OUTPUT:



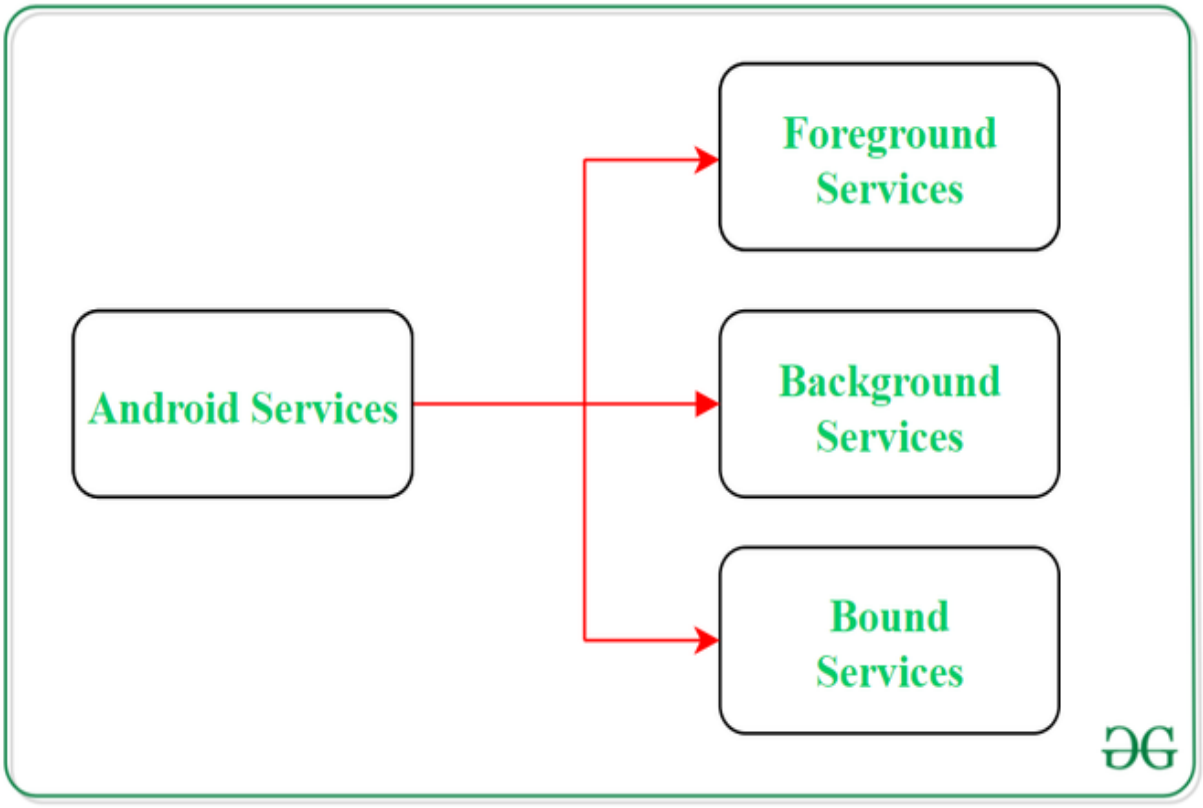
Method	Purpose
ActivityResultLauncher	Modern way to start activities for results
setResult() + finish()	Send data back to parent activity
getIntent().getStringExtra("key")	Retrieve the returned data

37) Define Service in Android. List and explain different types of services.

A) A **Service** in Android is a **background component** that performs long-running operations without providing a user interface. It is useful when an app needs to **perform tasks while the user is not interacting with it**, such as playing music, downloading files, or handling network transactions.

Key Characteristics of a Service:

- Runs **in the background**.
- Does **not have a UI**.
- Can run **indefinitely or until explicitly stopped**.
- Can run **in the main thread or in a separate thread** (for heavy tasks, use a thread or IntentService).



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.

Use Case: Music player, fitness app, file uploads.
Syntax:
startForeground(NOTIFICATION_ID, notification);

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.

Use Case: Playing music in the background.

```
• Intent intent = new Intent(this, MyService.class); startService(intent);
```

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.

Use Case: Interacting with a service that returns data (like a download progress).

```
Intent intent = new Intent(this, MyService.class);
bindService(intent, connection, Context.BIND_AUTO_CREATE);
```

38) Define intent filter. Write the syntax for Intent-Filter tag.

A) An **Intent Filter** is a declaration in the Android manifest file that specifies the types of **intents an activity, service, or broadcast receiver can respond to**.
It tells the Android system which component should handle a specific intent based on:

- **Action** (what to do)
- **Category** (additional information)
- **Data** (what type of data is used)

Purpose of Intent Filters:

- Enable **implicit intents** (e.g., open a web page, take a photo).
- Allow your app's components to be launched by **external apps**.
- Register components for specific events or actions.

Syntax of <intent-filter> tag:

```
<intent-filter>
   <action android:name="android.intent.action.VIEW" />
   <category android:name="android.intent.category.DEFAULT" />
   <data android:mimeType="text/plain" />
</intent-filter>
```

Common Tags Inside <intent-filter>**:**

Tag	Description
<action>	Describes the action that the component can handle (e.g., android.intent.action.SEND)

Tag	Description
<category>	Provides additional info (e.g., android.intent.category.DEFAULT)
<data>	Specifies the type of data (e.g., URI or MIME type) the component can handle

Example: Activity That Opens a Web Page

<activity android:name=".WebActivity"> <intent-filter> <action android:name="android.intent.action.VIEW" /> <category android:name="android.intent.category.DEFAULT" /> <category android:name="android.intent.category.BROWSABLE" /> <data android:scheme="http" /> </intent-filter>
</activity>

This filter allows the WebActivity to open URLs like http://example.com.

b) Explain how intents can be used to launch activities.

A)In Android, **Intents** are used to **start activities**, whether it's launching a new screen in the same app or starting an activity from another app. The **Intent** is a messaging object that allows components to request an action from another Android component.

Types of Intents to Launch Activities:

1. Explicit Intent – Used to start a specific activity within your app.
2. Implicit Intent – Used to start an activity that can handle a particular action from any app.

Using Explicit Intent
An explicit intent specifies the **exact class** to launch.
Example:
Intent intent = new Intent(CurrentActivity.this, TargetActivity.class); startActivity(intent);
This launches TargetActivity from CurrentActivity.

2. Using Implicit Intent
An implicit intent does **not name the class** to start, but declares a general action.
Example:
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("https://www.google.com")); startActivity(intent);
This opens a web browser to display the specified URL. The system checks for activities that can handle ACTION_VIEW with a URL.

Steps to Launch an Activity with an Intent:

1. Create an Intent object.
2. Specify the destination (explicit) or action/data (implicit).
3. Call startActivity(intent) to launch the new activity.

Why It's Useful:

- Navigating between app screens.
- Opening other apps (browser, camera, etc.).
- Sharing data across components and apps.

Intent Type	Usage Example	Launches
Explicit Intent	new Intent(this, SecondActivity.class)	Your activity
Implicit Intent	Intent.ACTION_VIEW with URL	System/browser

39) Define Broadcast Receiver. List and explain the system generated events supported by Broadcast Receivers.

A) A **Broadcast Receiver** in Android is a component that listens for **broadcast messages** from other applications or from the system. These broadcasts are usually sent when something happens that other components of the system or app might need to respond to, like a change in network status or the battery being low.
Broadcast receivers don't have a user interface and **don't interact directly** with the UI but **respond to events** by executing code based on those events.

Key Characteristics of Broadcast Receivers:

- **Listen for events**: Can listen for system-wide events or app-specific messages.
- **Handle global events**: Responds to broad notifications like connectivity changes, battery status, etc.
- **Runs in the background**: Acts on messages but doesn't show any UI to the user.
- **Can be static or dynamic**:
  - **Static Broadcast Receivers**: Registered in the manifest, receive broadcasts even if the app isn't running.
  - **Dynamic Broadcast Receivers**: Registered at runtime (programmatically) and only respond while the app is running.

Broadcast Action	Description
`android.intent.action.BOOT_COMPLETED`	Fired after the system finishes booting.
`android.intent.action.BATTERY_LOW`	Fired when the battery is low.
`android.intent.action.BATTERY_OKAY`	Fired when the battery has recovered from low state.
`android.intent.action.ACTION_POWER_CONNECTED`	Fired when device is connected to power.
`android.intent.action.ACTION_POWER_DISCONNECTED`	Fired when device is disconnected from power.

Broadcast Action	Description
`android.net.conn.CONNECTIVITY_CHANGE`	Fired when there is a change in network connectivity.
`android.intent.action.AIRPLANE_MODE`	Fired when airplane mode is changed.
`android.intent.action.PACKAGE_ADDED`	Fired when a new app is installed.
`android.intent.action.PACKAGE_REMOVED`	Fired when an app is uninstalled.
`android.intent.action.SCREEN_ON`	Fired when the screen is turned on.
`android.intent.action.SCREEN_OFF`	Fired when the screen is turned off.

Example:

Listening to BOOT_COMPLETED

```
public class BootReceiver extends BroadcastReceiver {

 @Override public void onReceive(Context context, Intent intent) {

 if (Intent.ACTION_BOOT_COMPLETED.equals(intent.getAction())) {

Toast.makeText(context, "Device Booted!", Toast.LENGTH_LONG).show(); } } }
```

Register in AndroidManifest.xml**:**

```
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<receiver android:name=".BootReceiver" android:enabled="true" android:exported="true">
<intent-filter> <action android:name="android.intent.action.BOOT_COMPLETED"/>
</intent-filter> </receiver>
```

40) Explain the lifecycle of Android services with neat sketch.

A) **The Life Cycle of Android Services**

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

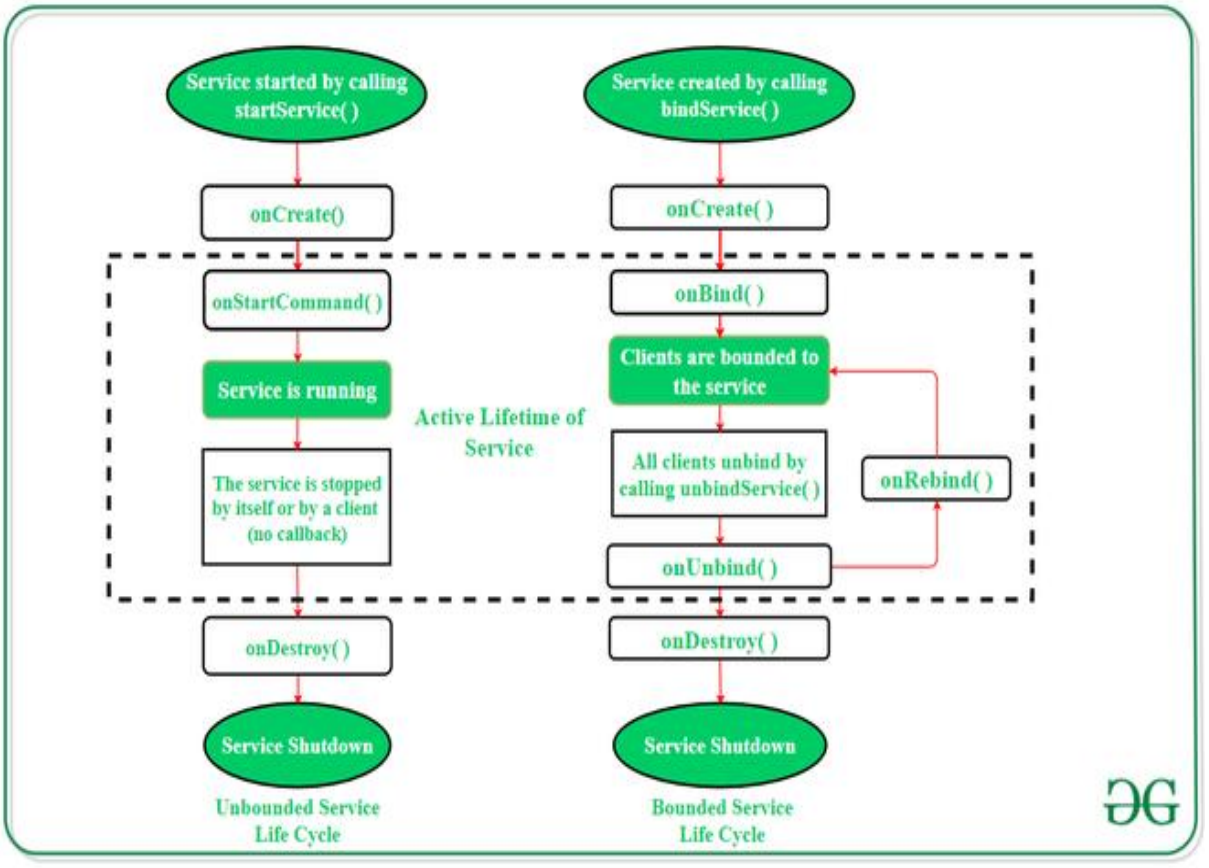
1. Started Service (Unbounded Service):

- 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.

Life cycle:



Fundamentals of Android Services

A user-defined service can be created through a normal class which is extending the **class Service** . Further, to carry out the operations of service on applications, there are certain callback methods which are needed to be **overridden** . The following are some of the important methods of Android Services:

Methods	Description
onStartCommand()	The Android service calls this method when a component(eg: activity) requests to start a service using startService(). Once the service is started, it can be stopped explicitly using stopService() or stopSelf()

Methods	Description
	methods.
onBind()	This method is mandatory to implement in android service and is invoked whenever an application component calls the bindService() method in order to bind itself with a service. User-interface is also provided to communicate with the service effectively by returning an IBinder object. If the binding of service is not required then the method must return null.
onUnbind()	The Android system invokes this method when all the clients get disconnected from a particular service interface.
onRebind()	Once all clients are disconnected from the particular interface of service and there is a need to connect the service with new clients, the system calls this method.
onCreate()	Whenever a service is created either using onStartCommand() or onBind(), the android system calls this method. This method is necessary to perform a one-time-set-up.
onDestroy()	When a service is no longer in use, the system invokes this method just before the service destroys as a final clean up call. Services must implement this method in order to clean up resources like registered listeners, threads, receivers, etc.

MAD QUESTION BANK UNIT:5

41a)Differentiate SQL & SQLite databases? What are the advantages and limitations of SQLite.

Ans:

Feature	SQL (General - MySQL, PostgreSQL, Oracle)	SQLite
Nature	SQL is a language for managing databases	SQLite is a database engine that uses SQL
Installation	Requires installation & server setup	No installation or server needed
Storage	Data stored on a remote server	Data stored in a local file (single .db)
Concurrency	Supports multiple users & concurrent access	Limited concurrency, uses file-locking
Speed	Slower for small applications due to server overhead	Faster for local/simple apps
Security	Provides user authentication, roles, encryption	Lacks built-in user authentication
Scalability	Highly scalable, suitable for large apps	Not scalable, for small/medium apps
Networking	Can be accessed over a network	No networking, local access only
Transaction Support	Full support (ACID-compliant)	Fully ACID-compliant, but simpler
Use Case	Web apps, enterprise systems, distributed systems	Mobile apps, testing, IoT, embedded apps
Data Size Handling	Can manage terabytes of data	Good for small to medium-sized datasets
Customization	Supports triggers, views, stored procedures	Limited support (e.g., no stored procedures)

Advantages of SQLite:

1. Lightweight – No installation, runs as a single .db file.
2. Serverless – Doesn't require a database server.

3. Zero Configuration – No need to configure username/password or ports.

4. Integrated with Android – Perfect for mobile apps and local data storage.

5. Fast and reliable – Good performance for small to medium apps.

Limitations of SQLite :

1. Not ideal for large-scale applications – Doesn't handle heavy traffic well.

2. No multi-user support – Lacks advanced concurrency controls.

3. Limited data types – Not as strict or flexible as full DBMS like MySQL.

4. Security – No built-in user authentication.

5. No stored procedures or advanced features – La Data Types in SQLite Database

b) List and explain Data Types used in SQLite database.

Ans:

SQLite is a lightweight database engine that supports dynamic typing, which means the data type of

a column is more flexible compared to traditional relational databases like MySQL or PostgreSQL.

While SQLite does allow flexibility in how data is stored, it uses type affinity to guide how data

should be treated in specific columns. Understanding the available data types in SQLite is crucial for

efficiently managing your database.

SQLite primarily supports 5 fundamental storage classes for storing data:

1. INTEGER

2. REAL

3. TEXT

4. BLOB

5. NULL

These data types are tied to the storage class that SQLite uses to store data. The actual storage used

depends on the affinity of the column and the type of data stored in it.

SQLite Data Types Explained

1. INTEGER

• Description: The INTEGER type is used to store whole numbers (positive or negative).

• Size: SQLite will use the most appropriate size (1, 2, 4, 8 bytes) depending on the value

stored.

• Examples: 1, 100, -5, 0

• Use Cases:

  o Storing IDs (primary keys).

  o Storing counters or amounts.

• Type Affinity: If you insert a number as a TEXT (like "5"), SQLite will automatically convert it to an INTEGER if needed.

2. REAL

• Description: The REAL data type is used for floating-point numbers or decimal values. It is

commonly used to store values that require precision, such as prices or scientific

measurements.

• Size: Always stored as 8 bytes (64-bit).

• Examples: 3.14, -0.99, 100.25

• Use Cases:

  o Storing scientific data, percentages, or prices.

  o Storing data that requires decimal places (e.g., 12.99 for a product price).

• Type Affinity: If you insert a string representing a number like "12.99", SQLite will treat it as a REAL if necessary.

3. TEXT

• Description: The TEXT data type stores strings (i.e., sequences of characters). It's used for

text data such as names, descriptions, or any field that holds alphanumeric data.

• Size: Variable-length, stored as UTF-8 or UTF-16 encoding.

• Examples: "John Doe", "Hello World", "Address"

• Use Cases:

  o Storing names, email addresses, or any textual content.

  o Storing long text fields (e.g., DESCRIPTION of a product).

• Type Affinity: If a number is stored as text (e.g., "100"), SQLite will still allow it, though it will treat the value as TEXT and not INTEGER.

4. BLOB

• Description: BLOB stands for Binary Large Object. It is used for storing binary data such as

images, files, or any kind of non-textual data (e.g., encoded files, PDFs, images, etc.).

• Size: Variable-sized, depending on the data.

• Examples: An image file, a video file, a PDF document.

• Use Cases:

  o Storing images or audio files directly in the database.

  o Storing encrypted data or serialized objects.

• Type Affinity: Unlike other data types, BLOB has no specific format (it stores data as is).

5. NULL

• Description: The NULL data type is used to represent missing or undefined values.

• Size: No storage is used for a NULL value.

• Examples: NULL (represents no value or unknown).

• Use Cases:

  o Storing data that is optional or unknown.

  o Used when a field has no value (e.g., a user's middle name which may not be provided).

• Type Affinity: A NULL value can be inserted into any column, regardless of the column's type affinity.

Example Table Using Data Types

```
CREATE TABLE students (id INTEGER PRIMARY KEY, name TEXT, marks REAL,
profile_pic BLOB, remarks TEXT);
```

42a) Explain about Constructors and Methods of SQLiteOpenHelper class.

Ans:

The `SQLiteOpenHelper` class in Android is a helper class designed to manage database creation and version management. It helps simplify working with SQLite databases by handling tasks like creating, opening, and upgrading the database.

Constructors of `SQLiteOpenHelper:

There are two main constructors:

1. SQLiteOpenHelper(Context context, String name, CursorFactory factory, int version)

- context: Application context.

- name: Name of the database file (e.g., "myDatabase.db"). Pass null to create an in-memory database.

- factory: Optional custom CursorFactory, or null for default.

- version: The version number of the database (starting from 1).

2. SQLiteOpenHelper(Context context, String name, CursorFactory factory, int version, DatabaseErrorHandler errorHandler)

- Same as above, but with a custom DatabaseErrorHandler to handle corruption errors.

Important Methods of `SQLiteOpenHelper:

1. onCreate(SQLiteDatabase db)

- Called only once when the database is created for the first time.

- Used to create tables and initialize data.

2. onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)

- Called when the database version is increased.

- Used to update schema, like adding columns or tables.

3. onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion)

- Called when the database version is decreased.

- Optional to override, but can be used to handle version rollbacks.

4. getReadableDatabase()

- Opens the database for reading.

- Returns an instance of SQLiteDatabase. Falls back to read-only mode if write access fails.

5. getWritableDatabase()

- Opens the database for writing (and reading).

- Automatically creates or upgrades the database as needed.

6. close()

- Closes the database and releases resources.

Ex:

.java

```
public class MyDatabaseHelper extends SQLiteOpenHelper {
   public MyDatabaseHelper(Context context) {
      super(context, "myDB", null, 1);
   }
   @Override
   public void onCreate(SQLiteDatabase db) {
      db.execSQL("CREATE TABLE users (id INTEGER PRIMARY KEY, name TEXT)");
   }
   @Override
   public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
      db.execSQL("DROP TABLE IF EXISTS users");
      onCreate(db);
   }}
```

b)Discuss about the methods present in SQLite Database class.

The `SQLiteDatabase` class in Android provides methods to manage and manipulate the SQLite database

— such as executing SQL statements,

___ performing insert/update/delete operations, and managing transactions.

1. Database Access Methods

Method	Description
`isOpen()`	Returns `true` if the database is open.
`close()`	Closes the database and releases its resources.
`isReadOnly()`	Returns `true` if the database was opened in read-only mode.

2. Data Manipulation Methods (DML):

- insert(String table, String nullColumnHack, ContentValues values)`
    - Inserts a new row into the database.

Java Synatx:

```
ContentValues values = new ContentValues();
values.put("name", "John");
db.insert("users", null, values);
```

- update(String table, ContentValues values, String whereClause, String[] whereArgs)

- Updates rows in the table based on conditions.

Java Syntax:

```
ContentValues values = new ContentValues();
values.put("name", "Jane");
db.update("users", values, "id=?", new String[]{"1"});
```

- delete(String table, String whereClause, String[] whereArgs)

    - Deletes rows from a table.

Java Synatx:

db.delete("users", "id=?", new String[]{"1"});

- execSQL(String sql)

- Executes a single SQL statement (e.g., `CREATE`, `DROP`, `INSERT`, etc.).

Java Synatx:

db.execSQL("CREATE TABLE users (id INTEGER PRIMARY KEY, name TEXT)");

3. Query Methods

- query(...)

- High-level method to perform `SELECT` queries.

Java Synatx:

Cursor cursor = db.query("users", null, null, null, null, null, null);

- rawQuery(String sql, String[] selectionArgs)

- Executes a raw SQL `SELECT` statement.

Java Syntax:

Cursor cursor = db.rawQuery("SELECT * FROM users WHERE id=?", new String[]{"1"});

4. Transaction Methods

Used for better performance and data consistency.

Method	Description
`beginTransaction()`	Starts a new transaction.
`setTransactionSuccessful()`	Marks the transaction as successful.
`endTransaction()`	Ends the transaction.

Java Syntax:

```
db.beginTransaction();
try {
   // operations
   db.setTransactionSuccessful();
} finally {
   db.endTransaction();
}
```

---

43) Explain how do you create and use SQLite database.
Ans:

Creating and Using SQLite Database in Android

Step 1: Create a Helper Class Extending SQLiteOpenHelper

This class will handle database creation and version management.

Java Syntax:

```
public class MyDatabaseHelper extends SQLiteOpenHelper {
   public MyDatabaseHelper(Context context) {
      super(context, "MyDatabase.db", null, 1);
   }
   @Override
   public void onCreate(SQLiteDatabase db) {
      // Raw query to create table
      String createTableQuery = "CREATE TABLE users (" +  "id INTEGER
PRIMARY KEY AUTOINCREMENT, " + "name TEXT)";
      db.execSQL(createTableQuery);
   }
   @Override
   public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
```

```
        // Drop and recreate if upgraded
        db.execSQL("DROP TABLE IF EXISTS users");
        onCreate(db);
    }
}
```

Step 2: Use the Helper Class in Your Activity

Use the `SQLiteDatabase` object returned by the helper to insert, query, update, and delete data.

Java Syntax:

```
public void insertUser(String name) {
    SQLiteDatabase db = dbHelper.getWritableDatabase();
    String insertQuery = "INSERT INTO users (name) VALUES ('" + name + "')";
    db.execSQL(insertQuery);
}
public void updateUser(int id, String newName) {
    SQLiteDatabase db = dbHelper.getWritableDatabase();
    String updateQuery = "UPDATE users SET name = '" + newName + "' WHERE
id = " + id;
    db.execSQL(updateQuery);
}
public void deleteUser(int id) {
    SQLiteDatabase db = dbHelper.getWritableDatabase();
    String deleteQuery = "DELETE FROM users WHERE id = " + id;
    db.execSQL(deleteQuery);
}
public void readUsers() {
    SQLiteDatabase db = dbHelper.getReadableDatabase();
    String selectQuery = "SELECT * FROM users";
```

```
    Cursor cursor = db.rawQuery(selectQuery, null);
    while (cursor.moveToNext()) {
        int id = cursor.getInt(cursor.getColumnIndex("id"));
        String name = cursor.getString(cursor.getColumnIndex("name"));
        Log.d("SQLite", "ID: " + id + ", Name: " + name);
    }
    cursor.close();
}
```

Step	Action
1	Create a class extending `SQLiteOpenHelper`.
2	Implement `onCreate()` and `onUpgrade()` methods.
3	Use `getWritableDatabase()` and `getReadableDatabase()` for DB operations.
4	Use `insert()`, `update()`, `delete()`, and `rawQuery()` / `query()` to manipulate data.

---

44) Explain insert operation with the help of example in SQLite database.

Ans:

Insert Operation in SQLite Database (Android)

In Android, we can insert data into an SQLite database in two ways:

- Using the insert() method.
- Using a raw SQL query with execSQL().

Ex:

Let's say we have a table called users with columns:

id (INTEGER PRIMARY KEY AUTOINCREMENT)

name (TEXT)

Step-by-Step Example

- **1. Create a SQLiteOpenHelper class**

```
public class MyDatabaseHelper extends SQLiteOpenHelper {
    public MyDatabaseHelper(Context context) {
        super(context, "MyDatabase.db", null, 1);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        // Create table
        db.execSQL("CREATE TABLE users (id INTEGER PRIMARY KEY AUTOINCREMENT,
name TEXT)");
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS users");
        onCreate(db);
    }
}
```

2. Insert Data (Using insert() method)

java

```
public void insertUser(String name) {
    SQLiteDatabase db = dbHelper.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put("name", name);  // Column name and value
    long rowId = db.insert("users", null, values);
    if (rowId != -1) {
        Log.d("Insert", "User inserted successfully with ID: " + rowId);
```

```
    } else {
        Log.e("Insert", "Insertion failed.");
    }
}
```

◇ **3. Insert Data (Using raw SQL with execSQL())**

```
java
public void insertUserRaw(String name) {
    SQLiteDatabase db = dbHelper.getWritableDatabase();
    String query = "INSERT INTO users (name) VALUES ('" + name + "')";
    db.execSQL(query);
}
```

---

45)Discuss the procedure to Browse SQLite Database using DB Browser Tool.

Ans:

Browsing SQLite Database Using DB Browser Tool

- DB Browser for SQLite is a free, open-source tool that allows you to visually browse, edit, and manage SQLite database files (`.db` files).

Steps to Browse an SQLite Database:

1. Install DB Browser for SQLite:

- Download from the official site:

https://sqlitebrowser.org/

- Install it on your system (Windows, macOS, Linux).

2. Locate Your Database File:

In Android, the database is usually stored in:

/data/data/<your.app.package.name>/databases/

3. Extract the `.db` File :

To access the database from a physical or emulator device:

Using Android Studio:

1. Run the app on an emulator or device.

2. Open Device File Explorer from Android Studio.

3. Navigate to:

/data/data/your.package.name/databases/

4. Right-click the `.db` file and choose "Save As..." to download it to your computer.

4. Open the Database in DB Browser

1. Launch DB Browser for SQLite.

2. Click on Open Database.

3. Select your `.db` file that you extracted or created.

4. The tool will now display:

- All tables in the database.

- Their schema (structure).

- Existing data (rows).

5. Browse and Manage the Data

- Browse Data tab: View, edit, or delete records.

- Execute SQL tab: Run custom SQL queries like:

Sql syntax:

```
SELECT * FROM users;
```

Step	Description
1	Install DB Browser for SQLite
2	Locate or extract `.db` file from Android
3	Open the file in DB Browser
4	Browse tables, view data, run queries

46a) List and explain Data Types used in SQLite database.

Ans:

SQLite is a lightweight database engine that supports dynamic typing, which means the data type of

a column is more flexible compared to traditional relational databases like MySQL or PostgreSQL.

While SQLite does allow flexibility in how data is stored, it uses type affinity to guide how data

should be treated in specific columns. Understanding the available data types in SQLite is crucial for

efficiently managing your database.

SQLite primarily supports 5 fundamental storage classes for storing data:

1. INTEGER

2. REAL

3. TEXT

4. BLOB

5. NULL

These data types are tied to the storage class that SQLite uses to store data. The actual storage used

depends on the affinity of the column and the type of data stored in it.

SQLite Data Types Explained

1. INTEGER

• Description: The INTEGER type is used to store whole numbers (positive or negative).

• Size: SQLite will use the most appropriate size (1, 2, 4, 8 bytes) depending on the value

stored.

• Examples: 1, 100, -5, 0

• Use Cases:

o Storing IDs (primary keys).

- Storing counters or amounts.

• Type Affinity: If you insert a number as a TEXT (like "5"), SQLite will automatically convert it to an INTEGER if needed.

2. REAL

• Description: The REAL data type is used for floating-point numbers or decimal values. It is

commonly used to store values that require precision, such as prices or scientific

measurements.

• Size: Always stored as 8 bytes (64-bit).

• Examples: 3.14, -0.99, 100.25

• Use Cases:

- Storing scientific data, percentages, or prices.
- Storing data that requires decimal places (e.g., 12.99 for a product price).

• Type Affinity: If you insert a string representing a number like "12.99", SQLite will treat it as a REAL if necessary.

3. TEXT

• Description: The TEXT data type stores strings (i.e., sequences of characters). It's used for

text data such as names, descriptions, or any field that holds alphanumeric data.

• Size: Variable-length, stored as UTF-8 or UTF-16 encoding.

• Examples: "John Doe", "Hello World", "Address"

• Use Cases:

- Storing names, email addresses, or any textual content.
- Storing long text fields (e.g., DESCRIPTION of a product).

• Type Affinity: If a number is stored as text (e.g., "100"), SQLite will still allow it, though it will treat the value as TEXT and not INTEGER.

4. BLOB

• Description: BLOB stands for Binary Large Object. It is used for storing binary data such as

images, files, or any kind of non-textual data (e.g., encoded files, PDFs, images, etc.).

• Size: Variable-sized, depending on the data.

• Examples: An image file, a video file, a PDF document.

• Use Cases:

o Storing images or audio files directly in the database.

o Storing encrypted data or serialized objects.

• Type Affinity: Unlike other data types, BLOB has no specific format (it stores data as is).

5. NULL

• Description: The NULL data type is used to represent missing or undefined values.

• Size: No storage is used for a NULL value.

• Examples: NULL (represents no value or unknown).

• Use Cases:

o Storing data that is optional or unknown.

o Used when a field has no value (e.g., a user's middle name which may not be provided).

• Type Affinity: A NULL value can be inserted into any column, regardless of the column's type affinity.

```
CREATE TABLE students (id INTEGER PRIMARY KEY, name TEXT, marks REAL,
profile_pic BLOB, remarks TEXT);
```

b) Mention applications of SQLite Database.

Ans:

Applications of SQLite Database:

1.Mobile Applications

- Used as the default local database in **Android** and **iOS** apps.

- Stores user data, settings, messages, and offline content.

2. Web Browsers

- Browsers like Google Chrome and Mozilla Firefox use SQLite to store:

- Bookmarks

- History

- Cookies

- Saved passwords

3. Desktop Applications

- Used in software like Skype, Adobe Lightroom, and Dropbox for storing user data and configuration.

4. Embedded Systems

- Ideal for IoT devices, smart TVs, routers, and car navigation systems due to its lightweight nature.

5. Testing and Prototyping

- Useful for developers to quickly create and test database-driven apps without setting up a full database server.

6. Caching in Applications

- Acts as a local cache to store temporary data and reduce server load in mobile/web apps.

7. Application File Format

- Some apps use SQLite databases as a file format to store complex data (e.g., Adobe Lightroom catalog files).

---

47) Discuss the process of creating tables in SQLite database and write the steps to view the table in database.

Ans:

Procedure to Create Tables in SQLite Database (Android):

1. Create a class that extends `SQLiteOpenHelper`.

2. Override the `onCreate()` method to write the `CREATE TABLE` SQL command.

3. Use `db.execSQL()`inside `onCreate()` to execute the table creation query.

4. Initialize the database using `getWritableDatabase()` or `getReadableDatabase()` in your activity.

5. The database and table are automatically created on the first run.

Java Syntax for creating Table(SQLiteOpenHelper Class):

```
public class MyDatabaseHelper extends SQLiteOpenHelper {
    private static final String DATABASE_NAME = "MyDatabase.db";
    private static final int DATABASE_VERSION = 1;
    public MyDatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        String createTable = "CREATE TABLE users (" +
                             "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
                             "name TEXT, " +
                             "email TEXT)";
        db.execSQL(createTable); // Executes the SQL to create the table
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS users");
        onCreate(db);
    }
}
```

Procedure to View Tables in SQLite Database:

1. Run the app to create the database.
2. Open Device File Explorer in Android Studio.

3. Navigate to:

`/data/data/<your.package.name>/databases/`

4. Right-click and download the `.db` file to your computer.

5. Open the file using DB Browser for SQLite.

6. Go to the "Browse Data" or"Database Structure" tab to view tables and their data.

---

48) Explain update operation with the help of example in SQLite database.

Ans:

Update Operation in SQLite Database (Android)

The **UPDATE** statement is used to modify existing rows in a table.

1. Using update() Method

This is the **recommended** and **safe** way to update data in SQLite because it uses ContentValues, which helps to prevent **SQL injection** attacks.

Syntax:

```
SQLiteDatabase db = dbHelper.getWritableDatabase();

ContentValues values = new ContentValues();

values.put("column_name", newValue);  // Set the new value for the column

db.update("table_name", values, "condition_column = ?", new
String[]{String.valueOf(condition_value)});
```

2. Using Raw SQL (execSQL())

This method involves directly executing a raw SQL query. Although it's **faster**, it's less secure because it doesn't handle parameters automatically, which opens up the potential for **SQL injection** if user input is directly included in the query.

Syntax:

```
String query = "UPDATE table_name SET column_name = 'new_value' WHERE
condition_column = value";

db.execSQL(query);
```

Method	Description
`update()`	Safe, easy-to-use method with `ContentValues`. Good for single updates.
`execSQL()`	Executes raw SQL directly. Faster but more error-prone, especially with user input (SQL injection risk).
`ContentValues + rawQuery()`	Not common but useful for indirect updates when combined with queries.
`SQLiteStatement`	Parameterized queries for direct SQL updates with better performance and security than raw SQL.
`Transaction`	Used for batch updates, improving performance and ensuring atomicity.

Ex:

```
public boolean updateUser(int id, String name, String email) {
   SQLiteDatabase db = this.getWritableDatabase();
   // Update query
   String query = "UPDATE users SET name = '" + name + "', email = '" + email + "' WHERE id = " + id;
   // Execute the query
   db.execSQL(query);
   return true;  // return true if update is successful
}
```

---

49) Explain delete operation with the help of example in SQLite database.

Ans:

DELETE Operation in SQLite

The **DELETE** statement in SQLite is used to **remove one or more rows** from a table. It can either delete specific rows based on a condition or delete all rows in the table.

1) DELETE Using SQL Query with execSQL()

This method uses raw SQL queries to perform a delete operation.

Java Syntax:

```
public boolean deleteUser(int id) {
   SQLiteDatabase db = this.getWritableDatabase();
   String query = "DELETE FROM users WHERE id = " + id;
   db.execSQL(query);  // Execute the delete query
   return true;
}
```

- **execSQL()**: Executes a raw SQL query. It works well for simple operations.

2) DELETE Using SQLiteStatement (Prepared Statement)

This method uses a prepared statement to perform the delete operation. It's more secure than raw SQL because it allows you to bind parameters to avoid SQL injection.

Java Syntax:

```
public boolean deleteUser(int id) {
   SQLiteDatabase db = this.getWritableDatabase();
   String deleteQuery = "DELETE FROM users WHERE id = ?";
   SQLiteStatement stmt = db.compileStatement(deleteQuery);
   stmt.bindLong(1, id);  // Bind the user id parameter
   stmt.executeUpdateDelete();  // Execute the delete operation
   return true;
}
```

- **SQLiteStatement**: This method allows you to bind values to the query securely, protecting against SQL injection.

Ex:

```
public boolean deleteUser(int id) {
   SQLiteDatabase db = this.getWritableDatabase();
```

```
    // Define the WHERE clause and arguments
    String whereClause = "id = ?";
    String[] whereArgs = new String[] { String.valueOf(id) };
    // Perform the delete operation
    int rowsDeleted = db.delete("users", whereClause, whereArgs);
    // Return true if the row was deleted, false otherwise
    return rowsDeleted > 0;
}
```

- **whereClause**: Defines the condition for which rows should be deleted (in this case, where id = ?).
- **whereArgs**: Array that contains the value to bind to the ? placeholder (the id).
- **db.delete()**: Deletes rows that match the condition, and returns the number of rows deleted.
- **Return value**: Returns true if one or more rows were deleted, otherwise false.

---

50) Write the syntax of the following operations in SQLite Database
a)Create, read
b)Update, delete

Ans:

Use above questions for this answer