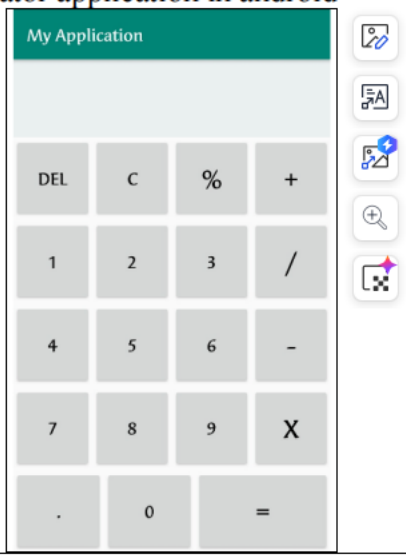
1.	Create the standard calculator application in android 	**30**
2.	Create an android application for the following menu items ,the appropriate toast should appear by clicking on the item : • Settings • Search • Compose Email • FeedBack (make Compose Email item disabled)	**10**
3.	Viva	**5**
4.	Journal	**5**

✅ Step-by-Step Summary

📌 Step 1: Create a New Project

- Open Android Studio → `New Project`
- Choose **"Empty Activity"**
- Name: `Calculator`
- Language: `Kotlin`
- Minimum SDK: `API 21 (Lollipop)` or above → click **Finish**

□ Step 2: Code Files

□ `activity_main.xml` (Design layout)

Go to: `res/layout/activity_main.xml` and replace all with:

```
xml
CopyEdit
```

```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="24dp"
    android:gravity="center"
    android:background="#FAFAFA">

    <EditText
        android:id="@+id/etFirstNumber"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Enter first number"
        android:inputType="numberDecimal"
        android:padding="12dp"
        android:layout_marginBottom="12dp" />

    <EditText
        android:id="@+id/etSecondNumber"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Enter second number"
        android:inputType="numberDecimal"
        android:padding="12dp"
        android:layout_marginBottom="12dp" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:gravity="center"
        android:layout_marginBottom="20dp">

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

        <Button
            android:id="@+id/btnSubtract"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="-"
            android:layout_marginLeft="12dp"/>

        <Button
```

```
            android:id="@+id/btnMultiply"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="×"
            android:layout_marginLeft="12dp"/>

        <Button
            android:id="@+id/btnDivide"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="÷"
            android:layout_marginLeft="12dp"/>
    </LinearLayout>

    <TextView
        android:id="@+id/tvResult"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Result: "
        android:textSize="20sp"
        android:textStyle="bold"
        android:textColor="#000000" />
</LinearLayout>
```

□ `MainActivity.kt` (Code logic)

Go to: `MainActivity.kt` and replace everything with this code:

```
kotlin
CopyEdit
package com.example.calculator

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

class MainActivity : AppCompatActivity() {

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

        val etFirstNumber = findViewById<EditText>(R.id.etFirstNumber)
        val etSecondNumber = findViewById<EditText>(R.id.etSecondNumber)
```

```
        val tvResult = findViewById<TextView>(R.id.tvResult)

        val btnAdd = findViewById<Button>(R.id.btnAdd)
        val btnSubtract = findViewById<Button>(R.id.btnSubtract)
        val btnMultiply = findViewById<Button>(R.id.btnMultiply)
        val btnDivide = findViewById<Button>(R.id.btnDivide)

        btnAdd.setOnClickListener {
            val result = calculate(etFirstNumber.text.toString(),
etSecondNumber.text.toString(), "+")
            tvResult.text = "Result: $result"
        }

        btnSubtract.setOnClickListener {
            val result = calculate(etFirstNumber.text.toString(),
etSecondNumber.text.toString(), "-")
            tvResult.text = "Result: $result"
        }

        btnMultiply.setOnClickListener {
            val result = calculate(etFirstNumber.text.toString(),
etSecondNumber.text.toString(), "*")
            tvResult.text = "Result: $result"
        }

        btnDivide.setOnClickListener {
            val result = calculate(etFirstNumber.text.toString(),
etSecondNumber.text.toString(), "/")
            tvResult.text = "Result: $result"
        }
    }

    private fun calculate(num1: String, num2: String, operator: String):
String {
        val a = num1.toDoubleOrNull()
        val b = num2.toDoubleOrNull()

        if (a == null || b == null) return "Invalid input"

        return when (operator) {
            "+" -> (a + b).toString()
            "-" -> (a - b).toString()
            "*" -> (a * b).toString()
            "/" -> if (b == 0.0) "Can't divide by 0" else (a / b).toString()
            else -> "Unknown"
        }
    }
}
```

✅ Step 3: Run the App

- Click on **Run** ▶
- Select **Emulator** or real Android device
- App will launch with input fields and buttons to do calculations.

✅ Step-by-step Implementation (Kotlin + XML)

☐ Step 1: `res/menu/menu_main.xml`

Create a folder (if not already present):
Right-click `res` → New → Android Resource Directory → type: `menu`

Inside it, create a file:
Right-click `menu` → New → Menu Resource File → Name: `menu_main.xml`

Paste this code:

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

    <item
        android:id="@+id/action_settings"
        android:title="Settings"
        android:showAsAction="never" />

    <item
        android:id="@+id/action_search"
        android:title="Search"
        android:showAsAction="never" />

    <item
        android:id="@+id/action_email"
        android:title="Compose Email"
        android:enabled="false"
        android:showAsAction="never" />

    <item
```

```
        android:id="@+id/action_feedback"
        android:title="Feedback"
        android:showAsAction="never" />

</menu>
```

□ Step 2: `activity_main.xml` (Just a basic UI)

In `res/layout/activity_main.xml`, you can leave it as default or use:

```
xml
CopyEdit
<?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:background="#FAFAFA">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Menu Example"
        android:textSize="24sp"
        android:textStyle="bold" />
</LinearLayout>
```

□ Step 3: `MainActivity.kt` (Handle Menu Actions)

Replace everything inside `MainActivity.kt` with this:

```
kotlin
CopyEdit
package com.example.menutoastapp

import android.os.Bundle
import android.view.Menu
import android.view.MenuItem
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {
```

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

    // Load the menu
    override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        menuInflater.inflate(R.menu.menu_main, menu)
        return true
    }

    // Handle menu item clicks
    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        return when (item.itemId) {
            R.id.action_settings -> {
                Toast.makeText(this, "Settings clicked",
Toast.LENGTH_SHORT).show()
                true
            }
            R.id.action_search -> {
                Toast.makeText(this, "Search clicked",
Toast.LENGTH_SHORT).show()
                true
            }
            R.id.action_feedback -> {
                Toast.makeText(this, "Feedback clicked",
Toast.LENGTH_SHORT).show()
                true
            }
            R.id.action_email -> {
                // Compose Email is disabled — won't be clickable
                true
            }
            else -> super.onOptionsItemSelected(item)
        }
    }
}
```

✅ Final Output:

- Tap the 3-dot menu in the app.
- Select **Settings, Search, or Feedback** → Shows toast.
- **Compose Email** will be **disabled**, unclickable.

1.	Create the background service android application to play the ringtone/music.	**20**
2.	Create an android application which automatically notify the user when Aeroplane mode is turned on or off using broadcast receiver.	**20**
3.	Viva	**5**
4.	Journal	**5**

⍰ Step 1: Define Permissions (in `AndroidManifest.xml`)

You need to add the permissions to play media and run a service in the background. Open **`AndroidManifest.xml`** and add the following inside the `<application>` tag:

```
xml
CopyEdit
<service android:name=".MusicService" android:enabled="true"
android:exported="false" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

□ Step 2: Create the Background Service (`MusicService.kt`)

Create a new Kotlin class named **`MusicService.kt`** to handle the background service and media playback.

```
kotlin
CopyEdit
package com.example.musicservice

import android.app.Service
import android.content.Intent
import android.media.MediaPlayer
import android.os.IBinder

class MusicService : Service() {

    private var mediaPlayer: MediaPlayer? = null

    override fun onCreate() {
        super.onCreate()
        mediaPlayer = MediaPlayer.create(this, R.raw.ringtone) // 'ringtone'
is the music you want to play
    }
```

```
    override fun onStartCommand(intent: Intent, flags: Int, startId: Int): Int
{
        mediaPlayer?.start() // Start the music
        return START_STICKY
    }

    override fun onDestroy() {
        super.onDestroy()
        mediaPlayer?.stop() // Stop the music when service is destroyed
        mediaPlayer?.release() // Release resources
    }

    override fun onBind(intent: Intent): IBinder? {
        return null
    }
}
```

□ Step 3: Add Music File (`ringtone.mp3`)

Add a **music file** (e.g., `ringtone.mp3`) in the `res/raw/` folder. If the `raw` folder doesn't exist, you can create it by right-clicking on `res` → **New** → **Android Resource Directory** → Choose `raw` as the resource type.

After creating the `raw` folder, add the **ringtone/music file** inside it.

□ Step 4: Create the UI (`activity_main.xml`)

Now, define the UI. Open **`res/layout/activity_main.xml`** and add the following XML layout to display the start and stop buttons:

```
xml
CopyEdit
<?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:background="#FAFAFA"
    android:padding="16dp">

    <Button
        android:id="@+id/btnStartService"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:text="Start Music Service" />

    <Button
        android:id="@+id/btnStopService"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Stop Music Service"
        android:layout_marginTop="20dp" />
</LinearLayout>
```

□ Step 5: Handle Button Actions in `MainActivity.kt`

Now, in your `MainActivity.kt`, manage the buttons to **start** and **stop** the service.

```
kotlin
CopyEdit
package com.example.musicservice

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

class MainActivity : AppCompatActivity() {

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

        val startServiceButton: Button = findViewById(R.id.btnStartService)
        val stopServiceButton: Button = findViewById(R.id.btnStopService)

        startServiceButton.setOnClickListener {
            val intent = Intent(this, MusicService::class.java)
            startService(intent) // Start the service to play music
        }

        stopServiceButton.setOnClickListener {
            val intent = Intent(this, MusicService::class.java)
            stopService(intent) // Stop the service and stop the music
        }
    }
}
```

□ Step 6: Update `AndroidManifest.xml` (Service Declaration)

In the **`AndroidManifest.xml`** file, you must declare the **service**. The service will run in the background to play the music.

Add the following inside the `<application>` tag:

```
xml
CopyEdit
<service android:name=".MusicService" android:enabled="true"
android:exported="false" />
```

□ Step 7: Final Testing and Running the App

1. **Run the app** on your Android device.
2. You will see **two buttons**:
   - **Start Music Service**: Starts the background music (ringtone or any music file you added).
   - **Stop Music Service**: Stops the music when clicked.

Steps to Create the Application:

1. **Create the BroadcastReceiver** to listen for the Airplane Mode state change.
2. **Create the necessary permissions** in the AndroidManifest.xml.
3. **Design the layout** with an appropriate message to show the state of Airplane Mode.
4. **Implement logic in MainActivity** to display notifications when the Airplane Mode state changes.

Full Code:

---

1. AndroidManifest.xml

Add the necessary permissions and the receiver configuration to listen for the `AIRPLANE_MODE` broadcast.

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

```
    package="com.example.airplanemodereceiver">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Airplane Mode Receiver"
        android:theme="@style/Theme.AirplaneModeReceiver">

        <receiver
            android:name=".AirplaneModeReceiver"
            android:enabled="true"
            android:exported="false"
            android:intent-filter>
            <action android:name="android.intent.action.AIRPLANE_MODE" />
        </receiver>

    </application>

</manifest>
```

---

2. AirplaneModeReceiver.kt

This class is the **BroadcastReceiver** that listens for the change in Airplane Mode and notifies the user.

```
kotlin
CopyEdit
package com.example.airplanemodereceiver

import android.content.BroadcastReceiver
import android.content.Context
import android.content.Intent
import android.widget.Toast

class AirplaneModeReceiver : BroadcastReceiver() {

    override fun onReceive(context: Context, intent: Intent) {
        val isAirplaneModeOn = intent.getBooleanExtra("state", false)

        val status = if (isAirplaneModeOn) {
            "Airplane Mode is ON"
        } else {
            "Airplane Mode is OFF"
        }

        // Show a Toast to notify the user
        Toast.makeText(context, status, Toast.LENGTH_SHORT).show()
```

```
    }
}
```

- **onReceive**: This method is called whenever the `AIRPLANE_MODE` state changes. We check whether the `state` is `true` (Airplane Mode is ON) or `false` (Airplane Mode is OFF) and display a toast message.

---

3. activity_main.xml

This XML layout is for the main screen of the app. It can just be a simple layout showing a message or a button.

```
xml
CopyEdit
<?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:background="#FAFAFA"
    android:padding="16dp">

    <TextView
        android:id="@+id/tvStatus"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Airplane Mode Status"
        android:textSize="20sp"
        android:textStyle="bold"
        android:textColor="#000000" />

</LinearLayout>
```

This layout contains a `TextView` where you can display the status of Airplane Mode, though in this app, the **Toast** will notify the user when the mode is changed.

---

4. MainActivity.kt

This Kotlin file is where the activity logic is implemented. For this case, it's enough to have an empty activity since the **BroadcastReceiver** will handle the Airplane Mode changes and display a **Toast**.

```
kotlin
CopyEdit
package com.example.airplanemodereceiver

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

class MainActivity : AppCompatActivity() {

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

Here, we're just setting the content view to the layout defined in `activity_main.xml`. The BroadcastReceiver will do the work in the background.

1.	Insert the new contents in the following resources and demonstrate their uses in the android application Android Resources: (Color, Theme, String, Drawable, Dimension, Image)	**20**
2.	Create an android application which automatically notify the user when Aeroplane mode is turned on or off using broadcast receiver.	**20**
3.	Viva	**5**
4.	Journal	**5**

1. Color Resources (`colors.xml`)

Color resources define color values that can be used across the app. These can be referenced in layouts, themes, and styles.

colors.xml

```
xml
CopyEdit
<?xml version="1.0" encoding="utf-8"?>
<resources>
<color name="black">#000000</color>
<color name="white">#FFFFFF</color>
<color name="blue">#2196F3</color>
<color name="pink">#E91E63</color>
<color name="green">#4CAF50</color>
<color name="red">#F44336</color>
<color name="grey">#9E9E9E</color>
</resources>
```

2. Theme (`styles.xml`)

The theme defines the appearance of the app and can be applied globally. You can reference color resources in the theme to set the app's primary, accent, and background colors.

styles.xml

```
xml
CopyEdit
<resources>
    <!-- Base application theme -->
     <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <item name="colorPrimary">@color/blue</item>
        <item name="colorPrimaryDark">@color/blue</item>
        <item name="colorAccent">@color/pink</item>
        <item name="android:windowBackground">@color/grey</item>
        <item name="android:textColor">@color/black</item>
    </style>
resources>
```

3. String Resources (`strings.xml`)

String resources contain all the text used in the app, like labels, button texts, and messages. They can be used in layouts or Kotlin files.

strings.xml

```
<resources>
    <string name="app_name">My Android App</string>
    <string name="welcome_message">Welcome to My App!</string>
```

```
    <string name="numbers">1, 2, 3, 4, 5</string>
    <string name="hello_message">Hello, World!</string>
</resources>
```

sources (`one.xml`)

Drawable resources can define shapes, gradients, or images used for backgrounds or other visual elements.

one.xml (In `res/drawable` folder)

```
xml
CopyEdit
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <solid android:color="@color/colorPrimary" /> <!-- Background color -->
    <corners android:radius="8dp"/> <!-- Rounded corners -->
</shape>
```

5. Dimension Resources (`dimens.xml`)

Dimension resources are used for defining sizes such as padding, margin, and text sizes in the app. This is useful to ensure consistency across different screen sizes.

dimens.xml

```
xml
CopyEdit
<resources>
    <dimen name="padding">16dp</dimen>
    <dimen name="margin">8dp</dimen>
    <dimen name="text_size">18sp</dimen>
</resources>
```

6. Image Resources

To use image resources, you simply place your images in the `res/drawable` folder. The images can then be used in layouts or in the code.

Image Example:

1. Place your image (e.g., `logo.png`) in the `res/drawable` folder.
2. Use the image in your layout XML:

```
3. xml
4. CopyEdit
5. <ImageView
6.     android:layout_width="wrap_content"
```

```
7.      android:layout_height="wrap_content"
8.      android:src="@drawable/logo"
9.      android:contentDescription="@string/app_name"/>
10.
```

7. Layout File (`activity_main.xml`)

Now, in your layout file (`activity_main.xml`), you can reference all of the resources you defined. Here's an example layout that uses color, string, dimension, and drawable resources.

activity_main.xml

```
xml
CopyEdit
<?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:background="@drawable/one"  <!-- Background using drawable -->
    android:padding="@dimen/padding"  <!-- Using dimension resource for
padding -->
    android:gravity="center"
    tools:context=".MainActivity">

    <!-- TextView using string and color resources -->
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/welcome_message"  <!-- Using string resource -->
        android:textColor="@color/colorAccent"  <!-- Using color resource -->
        android:textSize="@dimen/text_size"  <!-- Using dimension resource -->
        android:layout_marginBottom="@dimen/margin" />

    <!-- ImageView using image resource -->
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/logo"  <!-- Using image resource -->
        android:contentDescription="@string/app_name" />
</LinearLayout>
```

8. MainActivity.kt

In `MainActivity.kt`, you don't need to make any changes because the resources will automatically be referenced from the XML layout.

MainActivity.kt

```
kotlin
CopyEdit
package com.rohit.drwable

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

class MainActivity : AppCompatActivity() {

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

9. Final Explanation:

1. **Color Resources**: Used to define colors that can be applied to various UI elements like buttons, text, and backgrounds.
2. **Theme**: Customizes the overall look of the app by defining color schemes, window backgrounds, etc.
3. **String Resources**: Stores text content (e.g., app name, button text) for easy reuse and localization.
4. **Drawable Resources**: Used for backgrounds, icons, or other images. Can be simple shapes or bitmap images.
5. **Dimension Resources**: Stores common sizes like padding, margins, and text sizes for uniformity across the app.
6. **Image Resources**: Images that can be used in `ImageView` or as backgrounds.

10. How to Use These Resources

- Reference these resources using `@color`, `@string`, `@dimen`, `@drawable` in your layout and activity files.
- The advantage of using resources is that they promote reusability, maintainability, and localization support.

To create an Android application that automatically notifies the user when airplane mode is turned on or off using a `BroadcastReceiver`, here's a step-by-step approach and the full code.

Steps to Create the Application:

1. **Create a New Android Project**:
   - Open Android Studio and create a new project.
   - Choose an empty activity template for simplicity.
2. **Declare the BroadcastReceiver in the Manifest**:
   - You need to declare a `BroadcastReceiver` in your `AndroidManifest.xml` so that the system knows to listen for changes in airplane mode.
3. **Implement the BroadcastReceiver**:
   - Create a `BroadcastReceiver` to listen for changes in airplane mode (using the `Intent.ACTION_AIRPLANE_MODE_CHANGED` broadcast).
   - The receiver will notify the user (via a Toast or a notification) when airplane mode is toggled.
4. **Use `BroadcastReceiver` to Listen for Airplane Mode Changes**:
   - The `BroadcastReceiver` should check the state of airplane mode and notify the user whenever the state changes.
5. **Request Permissions (Optional)**:
   - If required, ask for permissions to check the state of airplane mode.
6. **Test the App**:
   - Run the application and test by turning the airplane mode on or off.

Full Code for the Application

1. `AndroidManifest.xml`:

```
xml
CopyEdit
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.airplanemodereceiver">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/Theme.AppCompat.DayNight">

        <!-- Declare the receiver -->
        <receiver android:name=".AirplaneModeReceiver">
            <intent-filter>
                <action android:name="android.intent.action.AIRPLANE_MODE" />
            </intent-filter>
        </receiver>
```

```
    </application>
</manifest>
```

2. `AirplaneModeReceiver.java` (The BroadcastReceiver):

```
package com.example.airplanemodereceiver;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class AirplaneModeReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        // Get the current state of airplane mode (on or off)
        boolean isAirplaneModeOn = intent.getBooleanExtra("state", false);

        // Show a Toast notification depending on the airplane mode state
        if (isAirplaneModeOn) {
            Toast.makeText(context, "Airplane Mode is ON",
Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(context, "Airplane Mode is OFF",
Toast.LENGTH_SHORT).show();
        }
    }
}
3. MainActivity.java (Main Activity for the App):
java
CopyEdit
package com.example.airplanemodereceiver;

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);
    }
}
```

4. `activity_main.xml` (Layout File):

```
xml
CopyEdit
<?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">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Airplane Mode App"
        android:textSize="24sp"
        android:layout_centerInParent="true" />
</RelativeLayout>
```

Introduction to Android, Introduction to Android Studio IDE, Application Fundamentals: Creating a project, Android components, Activities, Services, Content providers, Broadcast Receivers, Interface Overview, Creating Android Virtual device, USB debugging mode, Android Application Overview. Simple "Hello World" program

📱 1. Introduction to Android

- Android is an **open-source mobile OS** developed by **Google**.
- Based on **Linux kernel**, mainly used in **smartphones**, **tablets**, **TVs**, **wearables**, etc.
- Apps are developed in **Kotlin or Java**, and run on the **Android Runtime (ART)**.

💻 2. Introduction to Android Studio IDE

- **Android Studio** is the **official IDE** for Android development.
- Key features:
  - **Code Editor** – Smart suggestions, error checking.
  - **XML Layout Editor** – Drag-and-drop UI design.
  - **Emulator** – Test apps without real device.
  - **Gradle** – Build automation tool.
  - **SDK Manager** – Download/Update Android SDK & tools.

⚙️ 3. Application Fundamentals

✅ i) Creating a New Project in Android Studio

1. Open **Android Studio**.
2. Click **New Project**.
3. Select **Phone and Tablet > Empty Views Activity**.
4. Enter **project name**, **package name**, and **language: Kotlin**.
5. Choose **Save Location** and click **Finish**.

□ ii) Android Project Structure

- **Main Files:**
  - `MainActivity.kt` – Main Kotlin file.
  - `activity_main.xml` – Layout UI file.
  - `AndroidManifest.xml` – Declares app components and permissions.
  - `build.gradle` – Project and app-level configurations.

□ iii) Android Components

1. **Activity** – A screen with UI (e.g., login page).
2. **Service** – Runs background tasks (e.g., playing music).
3. **Content Provider** – Shares data across apps (e.g., contacts).
4. **Broadcast Receiver** – Listens to system events (e.g., battery low).

🎨 iv) Interface Overview

- UI elements like `TextView`, `Button`, `EditText` are defined in **XML layout** (in `res/layout`).

- Kotlin code in `MainActivity.kt` handles logic.

Example:

```
xml
CopyEdit
<!-- activity_main.xml -->
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello, Android!"
    android:layout_gravity="center"/>
kotlin
CopyEdit
// MainActivity.kt
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

📱 v) Creating Android Virtual Device (AVD)

1. Go to **Tools > Device Manager**.
2. Click **Create Device**.
3. Choose a device (e.g., Pixel 6) → Click **Next**.
4. Select a **System Image** (like Android 13) → Download → Next.
5. Click **Finish**, then **Run** the emulator.

🔌 vi) USB Debugging Mode

1. On your phone: Open **Settings > About Phone**.
2. Tap **Build Number 7 times** to enable **Developer Options**.
3. Go to **Developer Options** → Enable **USB Debugging**.
4. Connect phone via USB → Allow debugging → Run app from Android Studio.

📄 4. Android Application Overview

- An Android app includes:
    - **Activities**: UI Screens

    - **Services**: Background processing
    - **Broadcast Receivers**: Listen to system events
    - **Content Providers**: Share data
- Built using **Kotlin (logic)** and **XML (layout)**.

👏 5. Simple "Hello World" Program (Kotlin + XML)

✅ Steps:

1. **New Project** → Choose **Empty Views Activity**.
2. Select **Kotlin** as language.
3. Add the following in `activity_main.xml`:

```
4. xml
5. CopyEdit
6. <?xml version="1.0" encoding="utf-8"?>
7. <LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
8.     android:layout_width="match_parent"
9.     android:layout_height="match_parent"
10.    android:gravity="center"
11.    android:orientation="vertical">
12.
13.    <TextView
14.        android:id="@+id/helloText"
15.        android:text="Hello, World!"
16.        android:textSize="24sp"
17.        android:layout_width="wrap_content"
18.        android:layout_height="wrap_content" />
19.</LinearLayout>
20.
```

4. Kotlin code in `MainActivity.kt`:

```
5. kotlin
6. CopyEdit
7. package com.example.helloworld
8.
9. import android.os.Bundle
10.import androidx.appcompat.app.AppCompatActivity
11.
12.class MainActivity : AppCompatActivity() {
13.    override fun onCreate(savedInstanceState: Bundle?) {
14.        super.onCreate(savedInstanceState)
15.        setContentView(R.layout.activity_main)
16.    }
17.}
18.
```

5. Click **Run** → Choose Emulator or USB Device → App launches with **Hello, World!**

Programming Activities and fragments

Activity Life Cycle, Activity methods, Multiple Activities, Life Cycle of fragments and multiple fragments

1. Activity in Android

An **Activity** represents a single screen in an Android app. It manages **UI components** and **user interactions**.

🔄 Activity Lifecycle

Every Activity follows a lifecycle managed by the Android OS:

Lifecycle Method	Description
`onCreate()`	Called when the activity is first created. Initialize UI elements.
`onStart()`	Called when the activity becomes visible.
`onResume()`	Called when the activity starts interacting with the user.
`onPause()`	Called when another activity comes to the foreground.
`onStop()`	Called when the activity is no longer visible.
`onDestroy()`	Called before the activity is destroyed.
`onRestart()`	Called when the activity is restarted after stopping.

📈 Activity Lifecycle Diagram

```
scss
CopyEdit
onCreate() → onStart() → onResume()
                          ↓
              (User Interaction)
                          ↓
     onPause() → onStop() → onDestroy()
```

📱💻 Example: Basic Activity in Kotlin

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        Log.d("Activity Lifecycle", "onCreate called")
    }

    override fun onStart() {
        super.onStart()
    }

    override fun onResume() {
        super.onResume()
    }

    override fun onPause() {
        super.onPause()
    }

    override fun onStop() {
        super.onStop()
    }

    override fun onDestroy() {
        super.onDestroy()
    }
}
```

---

2. Activity Methods

- `startActivity(intent)`: Starts a new activity.
- `startActivityForResult(intent, requestCode)`: Starts another activity and expects a result.
- `finish()`: Closes the current activity.

✨ Example: Starting a New Activity

```
val intent = Intent(this, SecondActivity::class.java)
startActivity(intent)
```

---

3. Multiple Activities

A project may contain **multiple activities**, each handling different screens.

🔄 Example: Navigating Between Activities

1. First Activity (`MainActivity.kt`)

```
val intent = Intent(this, SecondActivity::class.java)
intent.putExtra("username", "John")
startActivity(intent)
```

2. Second Activity (`SecondActivity.kt`)

```
val username = intent.getStringExtra("username")
Toast.makeText(this, "Welcome, $username", Toast.LENGTH_SHORT).show()
```

---

4. Fragments in Android

A **Fragment** is a reusable UI component that exists within an **Activity**.

🔄 Fragment Lifecycle

Lifecycle Method	Description
`onAttach()`	Called when the fragment is attached to the activity.
`onCreate()`	Called when the fragment is created.
`onCreateView()`	Called to create the fragment's UI.
`onStart()`	Called when the fragment becomes visible.
`onResume()`	Called when the user interacts with the fragment.
`onPause()`	Called when the fragment is paused.
`onStop()`	Called when the fragment is no longer visible.
`onDestroyView()`	Called before destroying the fragment's UI.
`onDestroy()`	Called before the fragment is destroyed.
`onDetach()`	Called when the fragment is detached from the activity.

□ Example: Creating a Fragment

1. Fragment Class (`MyFragment.kt`)

```
class MyFragment : Fragment(R.layout.fragment_layout) {
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        Log.d("Fragment Lifecycle", "Fragment Created")
    }
}
```

2. Add Fragment in XML (`activity_main.xml`)

```
<fragment
    android:id="@+id/myFragment"
    android:name="com.example.MyFragment"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

3. Dynamically Add Fragment (`MainActivity.kt`)

```
val fragment = MyFragment()
supportFragmentManager.beginTransaction()
    .replace(R.id.fragmentContainer, fragment)
    .commit()
```

---

5. Multiple Fragments

You can use **multiple fragments** within the same activity.

🔁 Example: Using Two Fragments

```
1. Create Two Fragments (FragmentA.kt and FragmentB.kt)
2. Define a FrameLayout in XML (activity_main.xml)
xml
CopyEdit
<FrameLayout
    android:id="@+id/fragmentContainer"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
3. Switch Between Fragments (MainActivity.kt)
kotlin
```

```
CopyEdit
fun replaceFragment(fragment: Fragment) {
    supportFragmentManager.beginTransaction()
        .replace(R.id.fragmentContainer, fragment)
        .addToBackStack(null)
        .commit()
}

btnFragmentA.setOnClickListener {
    replaceFragment(FragmentA())
}

btnFragmentB.setOnClickListener {
    replaceFragment(FragmentB())
}
```

Coordinate, Linear, Relative, Table, Absolute, Frame, List View, Grid View.

1. LinearLayout

Arranges elements in a single direction (horizontal or vertical):

- `android:orientation="vertical"` → Stack elements vertically
- `android:orientation="horizontal"` → Arrange elements side by side

🔤 Example: Vertical Linear Layout

```
xml
CopyEdit
<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">

    <TextView
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, World!" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Click Me" />
</LinearLayout>
```

---

2. RelativeLayout

Arranges elements relative to each other using attributes like:

- `android:layout_below`
- `android:layout_alignParentRight`

📌 Example: Relative Positioning

```
xml
CopyEdit
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click Me"
        android:layout_below="@id/textView" />
</RelativeLayout>
```

---

3. FrameLayout

A single container that holds **one child view at a time**. Useful for **overlapping views**.

🖼️ Example: Overlapping Views

```
xml
CopyEdit
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:src="@drawable/background_image" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Overlay Text"
        android:textColor="#FFFFFF"
        android:layout_gravity="center" />
</FrameLayout>
```

---

4. TableLayout

Organizes UI elements in **rows and columns**, similar to an HTML table.

📋 Example: Creating a Table

```
xml
CopyEdit
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TableRow>
        <TextView android:text="Name" />
```

```
        <TextView android:text="Age" />
    </TableRow>

    <TableRow>
        <TextView android:text="John" />
        <TextView android:text="25" />
    </TableRow>

    <TableRow>
        <TextView android:text="Emma" />
        <TextView android:text="22" />
    </TableRow>
</TableLayout>
```

---

5. AbsoluteLayout *(Deprecated)*

Allows you to specify **exact locations** for its children.
❗ **Deprecated** and **not recommended** for use in modern apps.

□ Example: AbsoluteLayout

```
xml
CopyEdit
<AbsoluteLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 1"
        android:layout_x="50dp"
        android:layout_y="100dp" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 2"
        android:layout_x="200dp"
        android:layout_y="300dp" />
</AbsoluteLayout>
```

---

6. CoordinatorLayout

Used to implement **Material Design behaviors**, like Floating Action Button (FAB), Collapsing Toolbar, etc.

🎯 Example: Floating Action Button (FAB)

```
xml
CopyEdit
<androidx.coordinatorlayout.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <com.google.android.material.floatingactionbutton.FloatingActionButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_add"
        app:layout_anchor="@id/bottomBar"
        app:layout_anchorGravity="center" />
</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

---

7. ListView

Displays a **scrollable list** of items.

📋 Example: ListView in XML

```
xml
CopyEdit
<ListView
    android:id="@+id/listView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

📱💻 Example: Populating ListView (Kotlin)

```
kotlin
CopyEdit
```

```
val listView: ListView = findViewById(R.id.listView)
val items = arrayOf("Item 1", "Item 2", "Item 3")
val adapter = ArrayAdapter(this, android.R.layout.simple_list_item_1, items)
listView.adapter = adapter
```

---

8. GridView

Displays items in a **grid format** (rows and columns).

□ Example: GridView in XML

```
xml
CopyEdit
<GridView
    android:id="@+id/gridView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:numColumns="2" />
```

□💻 Example: Populating GridView (Kotlin)

```
val gridView: GridView = findViewById(R.id.gridView)
val items = arrayOf("A", "B", "C", "D")
val adapter = ArrayAdapter(this, android.R.layout.simple_list_item_1, items)
gridView.adapter = adapter
```

Using FireBase create a JSON document student with attributes: id, name and age. Create an android application to read and write the above JSON document

✅ Summary of Steps:

1. **Set up Firebase** in your Android project.
2. **Add dependencies** to `build.gradle`.
3. **Design layout** with input fields and buttons.
4. **Create `Student` data class**.
5. **Connect Firebase** to your app and implement **read/write** logic.

🔧 Step 1: Firebase Setup

1. Go to Firebase Console
2. Create a new project → Add Android app.
3. Download `google-services.json` → Put in `app/` directory.
4. Enable **Realtime Database** and set rules to:

```
json
CopyEdit
{
  "rules": {
    ".read": true,
    ".write": true
  }
}
```

🔧 Step 2: Add dependencies in `build.gradle` (App Level)

```
gradle
CopyEdit
plugins {
    id 'com.android.application'
    id 'com.google.gms.google-services'
}

dependencies {
    implementation 'com.google.firebase:firebase-database:20.3.0'
    implementation 'com.google.firebase:firebase-analytics:21.6.0'
}
```

In `build.gradle` (Project Level), add:

```
gradle
CopyEdit
classpath 'com.google.gms:google-services:4.4.1'
```

📄 Step 3: Student Data Class

Create a file: `Student.kt`

```
kotlin
CopyEdit
package com.example.firebaseapp

data class Student(
    var id: String? = "",
    var name: String? = "",
    var age: String? = ""
)
```

📱 Step 4: Layout (activity_main.xml)

```
xml
CopyEdit
<?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/editId"
        android:hint="Enter ID"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

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

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

    <Button
        android:id="@+id/btnSave"
        android:text="Save Student"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <Button
        android:id="@+id/btnRead"
        android:text="Read Student"
        android:layout_width="match_parent"
```

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

    <TextView
        android:id="@+id/textResult"
        android:text=""
        android:layout_marginTop="16dp"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

🎯 Step 5: MainActivity.kt

```
kotlin
CopyEdit
package com.example.firebaseapp

import android.os.Bundle
import android.widget.*
import androidx.appcompat.app.AppCompatActivity
import com.google.firebase.database.DatabaseReference
import com.google.firebase.database.FirebaseDatabase

class MainActivity : AppCompatActivity() {

    private lateinit var database: DatabaseReference

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

        val editId = findViewById<EditText>(R.id.editId)
        val editName = findViewById<EditText>(R.id.editName)
        val editAge = findViewById<EditText>(R.id.editAge)
        val btnSave = findViewById<Button>(R.id.btnSave)
        val btnRead = findViewById<Button>(R.id.btnRead)
        val textResult = findViewById<TextView>(R.id.textResult)

        database = FirebaseDatabase.getInstance().reference

        btnSave.setOnClickListener {
            val id = editId.text.toString()
            val name = editName.text.toString()
            val age = editAge.text.toString()

            val student = Student(id, name, age)

            database.child("student").setValue(student)
                .addOnSuccessListener {
                    Toast.makeText(this, "Student saved",
Toast.LENGTH_SHORT).show()
                }
                .addOnFailureListener {
                    Toast.makeText(this, "Failed: ${it.message}",
Toast.LENGTH_SHORT).show()
                }
        }

        btnRead.setOnClickListener {
            database.child("student").get().addOnSuccessListener {
```

```
                val student = it.getValue(Student::class.java)
                textResult.text = "ID: ${student?.id}, Name:
${student?.name}, Age: ${student?.age}"
            }.addOnFailureListener {
                Toast.makeText(this, "Failed to read: ${it.message}",
Toast.LENGTH_SHORT).show()
            }
        }
    }
}
```

🗀 Step 6: AndroidManifest.xml

```
xml
CopyEdit
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.firebaseapp">

    <application
        android:allowBackup="true"
        android:theme="@style/Theme.FirebaseApp"
        android:label="@string/app_name">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

☑ JSON in Firebase Realtime DB will look like:

```
json
CopyEdit
"student": {
    "age": "21",
    "id": "101",
    "name": "Deep"
}
```

Create an android application to demonstrate the use of sub menu the toast should be appeared by selecting the sub menu item

Project Name: `SubMenuToastApp`

🗀 1. `res/layout/activity_main.xml`

```
xml
CopyEdit
<?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:gravity="center"
    android:padding="20dp"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Tap on the menu (top right) to open submenu"
        android:textSize="18sp"
        android:textStyle="bold"
        android:textColor="#000000" />

</LinearLayout>
```

🗀 2. `res/menu/main_menu.xml`

Create this file inside `res/menu/` folder. If `menu` folder doesn't exist, right-click `res > New > Android Resource Directory` → Resource Type: `menu`.

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

    <!-- Main menu item that opens a submenu -->
    <item
        android:id="@+id/menu_settings"
        android:title="Settings">

        <menu>
            <item
                android:id="@+id/sub_wifi"
                android:title="Wi-Fi" />
            <item
                android:id="@+id/sub_bluetooth"
                android:title="Bluetooth" />
        </menu>
```

```
    </item>

</menu>
```

---

3. `MainActivity.kt`

Put this Kotlin code in your `MainActivity.kt`.

```
kotlin
CopyEdit
package com.example.submenutoastapp

import android.os.Bundle
import android.view.Menu
import android.view.MenuItem
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {

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

    // Inflate the menu
    override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        menuInflater.inflate(R.menu.main_menu, menu)
        return true
    }

    // Handle menu item clicks
    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        return when (item.itemId) {
            R.id.sub_wifi -> {
                Toast.makeText(this, "Wi-Fi selected",
Toast.LENGTH_SHORT).show()
                true
            }
            R.id.sub_bluetooth -> {
                Toast.makeText(this, "Bluetooth selected",
Toast.LENGTH_SHORT).show()
                true
            }
            else -> super.onOptionsItemSelected(item)
        }
    }
}
```

🛠 4. `AndroidManifest.xml`

This should be default, but just to be safe:

```
xml
CopyEdit
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.submenutoastapp">

    <application
        android:allowBackup="true"
        android:label="SubMenu App"
        android:icon="@mipmap/ic_launcher"
        android:theme="@style/Theme.AppCompat.Light.DarkActionBar">

        <activity
            android:name=".MainActivity"
            android:exported="true">

            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>

        </activity>
    </application>

</manifest>
```

✅ Output Preview:

- App launches with a simple message.
- Tap on the **3-dot menu (overflow menu)** at the top-right.
- Select **Wi-Fi** or **Bluetooth** under **Settings**.
- You'll see a Toast: "Wi-Fi selected" or "Bluetooth selected".