Experiment no: 10

Title: Creating simple project and study of android project structure and installing apk on mobile device/tablet, configuring mobile device/tablet in Android Studio with developer option and running app directly on mobile device/tablet.

Theory:

In this project, we aim to create a simple Android project, study the structure of the project, and install the APK on a mobile device or tablet. Additionally, we will configure the mobile device or tablet in Android Studio by enabling developer options and running the app directly on the device.

Key concepts:

1. **Android Project Structure**: An Android project consists of several files and folders:
   - **Java/Kotlin files**: Contains the logic of your app.
   - **res folder**: Contains all resources like XML layouts, images, and strings.
   - **Manifest file**: Contains essential information about the app, such as permissions, components, and app name.
   - **Gradle files**: Used for building and managing dependencies.
2. **Enabling Developer Options**:
   - Go to your device settings > About phone > Tap "Build Number" seven times to enable developer options.
   - In developer options, enable **USB Debugging** to connect the device with Android Studio.
3. **Running the app on a device**:
   - Connect your mobile device to your computer via a USB cable.
   - Select your device from the Android Studio device selector and run the app.

Sample code:

JAVA CODE

```
package com.example.adlexp;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }
}
```

XML CODE

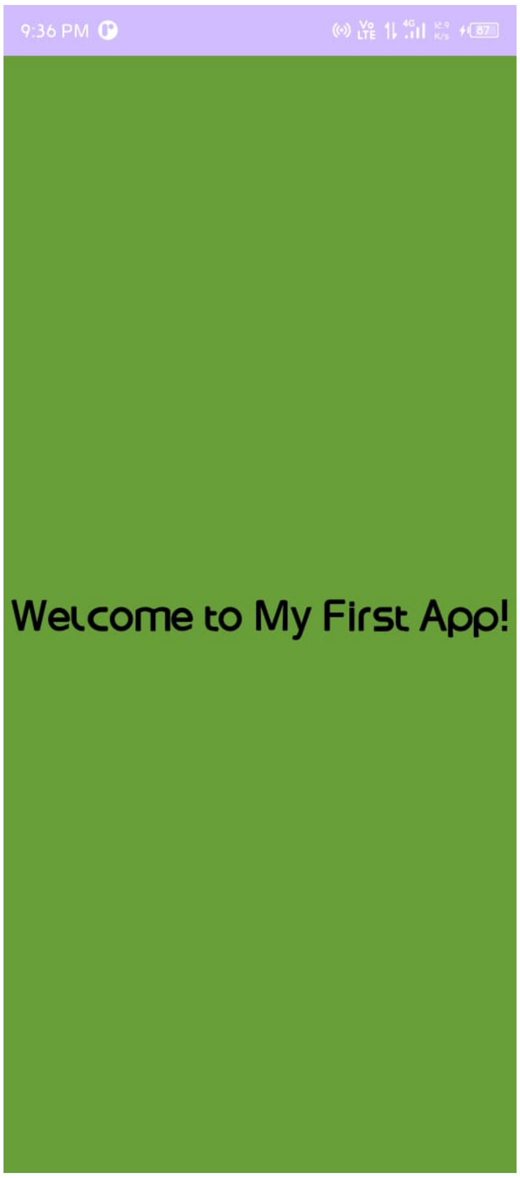
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:id="@+id/main"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:gravity="center"
   android:background="#689F38"
   tools:context=".MainActivity">

   <TextView
      android:id="@+id/welcomeText"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Welcome to My First App!"
      android:textSize="30sp"
      android:textStyle="bold"
      android:textColor="@android:color/black"/>

</LinearLayout>
```

AndroidManifest.xml

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

   <application
      android:allowBackup="true"
      android:dataExtractionRules="@xml/data_extraction_rules"
      android:fullBackupContent="@xml/backup_rules"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:roundIcon="@mipmap/ic_launcher_round"
      android:supportsRtl="true"
      android:theme="@style/Theme.ADLEXP"
      tools:targetApi="31">
      <activity
         android:name=".MainActivity"
         android:exported="true">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />

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

```
        </activity>
    </application>

</manifest>
```

Output


9:36 PM

Welcome to My First App!

Conclusion

By following the steps above, we have successfully created a simple Android project, studied the project structure, and configured a mobile device for app installation using Android Studio. We also demonstrated running the app directly on the mobile device after enabling developer options and USB debugging

Experiment no: 11

Title: Write a program to use of different layouts.

Theory

In Android development, **layouts** are used to define the user interface (UI) elements and their arrangement on the screen. There are different types of layouts available in Android, each suited to different UI designs. The most commonly used layouts include:

1. **LinearLayout**: Aligns its child elements either horizontally or vertically.
2. **RelativeLayout**: Arranges child elements relative to each other or the parent layout.
3. **ConstraintLayout**: Allows more flexible positioning and control over the child elements by defining constraints.
4. **FrameLayout**: Places child elements on top of each other, which is useful for overlaying views.
5. **GridLayout**: Positions items in a grid pattern, useful for tabular data.

In this program, we will demonstrate the use of **LinearLayout**, **RelativeLayout**, and **ConstraintLayout** by creating a simple Android UI that switches between different layouts.

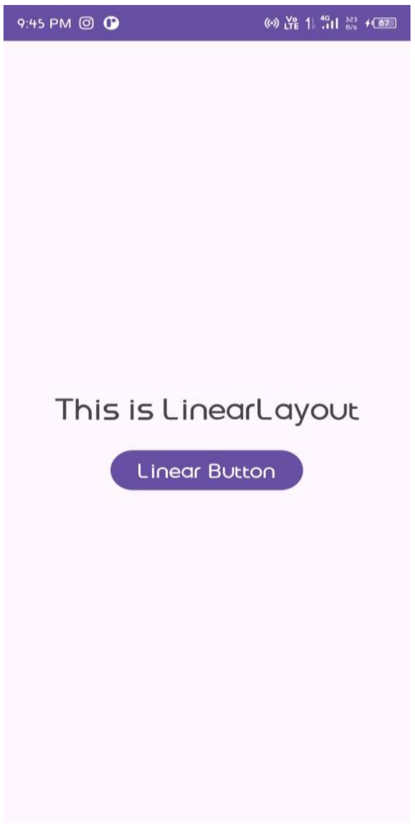
Sample code

Java Code

```
package com.example.adlexp;

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

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

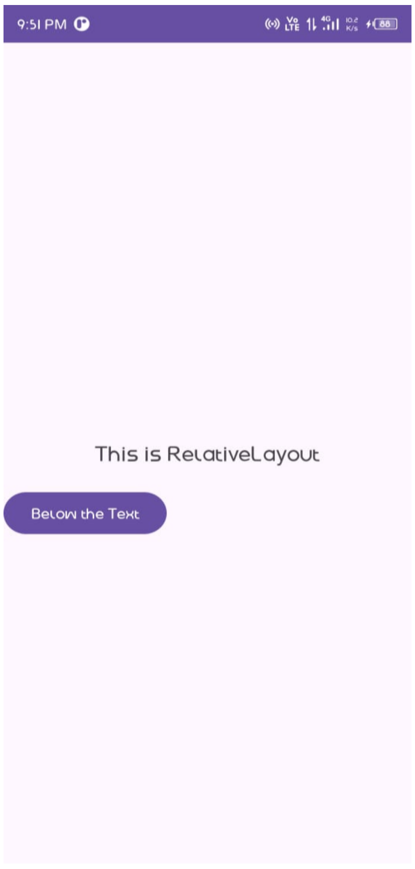
Xml Linear Layout

```
<?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:id="@+id/main"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="This is LinearLayout"
        android:textSize="30sp"
        android:padding="10pt"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Linear Button"
        android:textSize="20sp"/>

</LinearLayout>
```



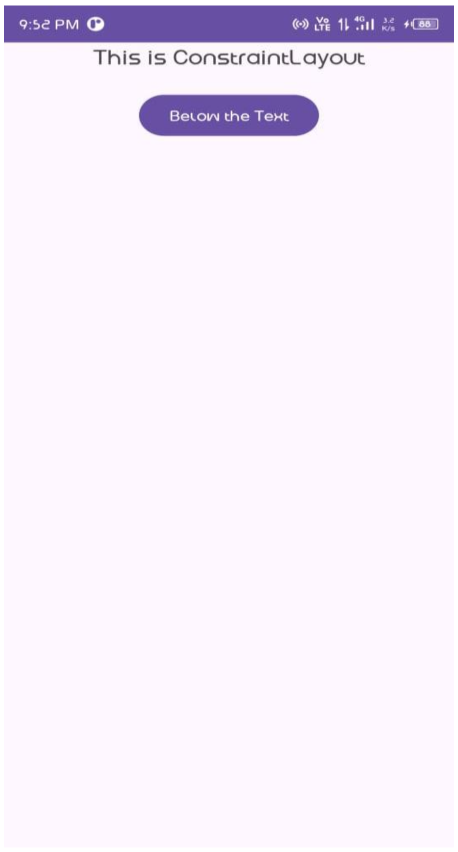
Xml Relative Layout

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

    <TextView
    android:id="@+id/relativeText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="This is RelativeLayout"
    android:layout_centerInParent="true"
    android:textSize="20sp"/>

    <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Below the Text"
    android:layout_below="@id/relativeText"
    android:layout_marginTop="20dp"/>

</RelativeLayout>
```



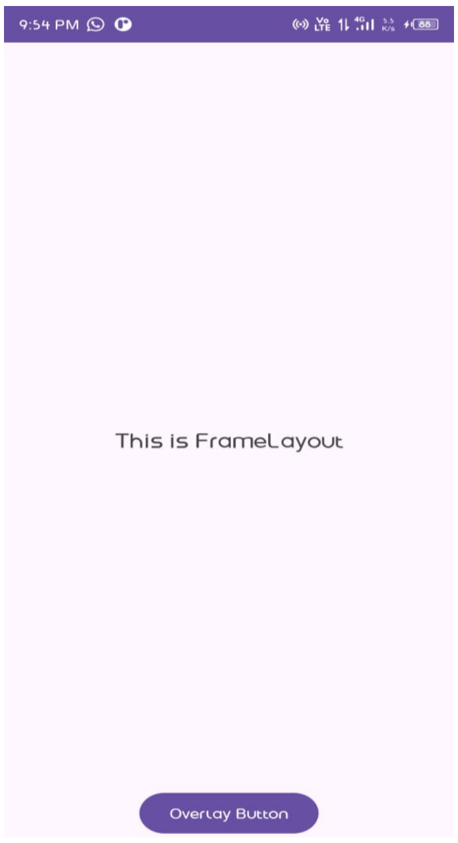
Xml Constraint Layout

```
<?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">

    <TextView
        android:id="@+id/constraintText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="This is ConstraintLayout"
        android:textSize="20sp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"/>

    <Button
        android:id="@+id/constraintButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Below the Text"
        app:layout_constraintTop_toBottomOf="@id/constraintText"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_marginTop="20dp"/>

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



Xml Frame Layout

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



```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="This is FrameLayout"
        android:layout_gravity="center"
        android:textSize="20sp"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Overlay Button"
        android:layout_gravity="bottom|center"/>

</FrameLayout>
```

Conclusion

This project demonstrates the use of different layouts in Android. We have seen how to switch between LinearLayout, RelativeLayout, and ConstraintLayout, and how each layout arranges its child elements. Understanding and using different layouts efficiently is crucial for creating flexible and responsive user interfaces in Android applications.

Experiment no: 12

Title: Write a program to use of Intents for SMS and Telephony.

Theory

1. **Intents for SMS**

An Intent in Android is a messaging object used to request an action from another app component. To send SMS using intents, the Intent.ACTION_SENDTO action is used. It opens the SMS application with the recipient number and message pre-filled.

You need the SEND_SMS permission in your manifest file.

Sample code

AndroidManifest.xml (Permission for SMS):

```
<uses-permission android:name="android.permission.SEND_SMS" />
```

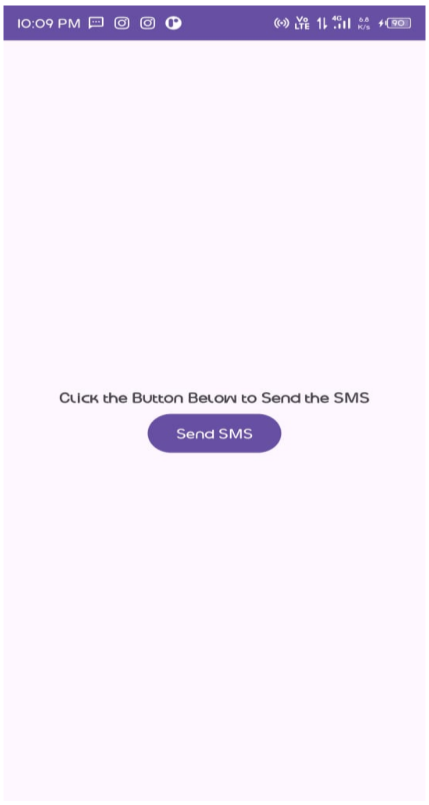
Java Code

```
package com.example.adlexp;
import android.content.Intent;
import android.net.Uri;
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);

      Button smsButton = findViewById(R.id.btnSendSMS);
      smsButton.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View view) {
            sendSMS();
         }});}

   private void sendSMS() {
      Uri uri = Uri.parse("smsto:+1234567890");  // Replace with recipient number
      Intent intent = new Intent(Intent.ACTION_SENDTO, uri);
      intent.putExtra("sms_body", "Hello! This is a test message.");  // Message content
      startActivity(intent);
   }}
```

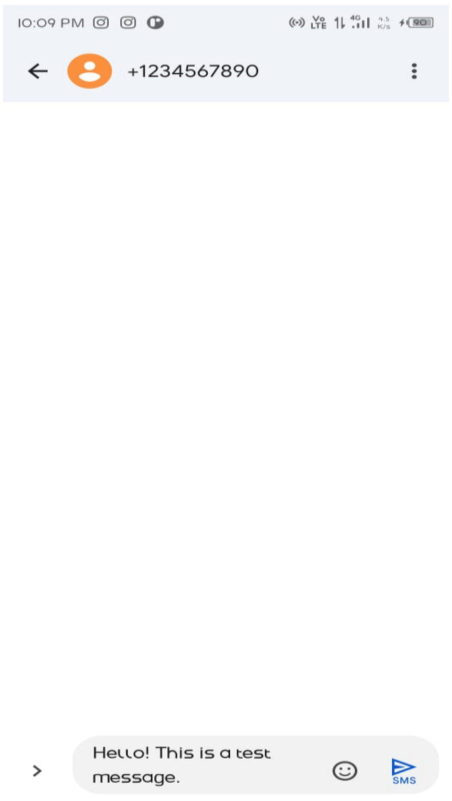
Xml Code

```
<?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:id="@+id/main"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:gravity="center"
   tools:context=".MainActivity">
   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Click the Button Below to Send the SMS"
      android:textSize="8pt"
      android:textStyle="bold"
      android:padding="10px"/>
   <Button
      android:id="@+id/btnSendSMS"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Send SMS" />
</LinearLayout>
```

Output


10:09 PM

Click the Button Below to Send the SMS


10:09 PM

+1234567890

Hello! This is a test message.

2. Intents for Telephony

Theory:

To make a phone call using intents, the Intent.ACTION_DIAL action is used. It opens the phone dialer with the phone number pre-filled. You can also use Intent.ACTION_CALL to directly initiate the call, but it requires the CALL_PHONE permission.

Sample Code

AndroidManifest.xml (Permission for Phone Calls):

```xml
<uses-permission android:name="android.permission.CALL_PHONE" />
```

Java Code

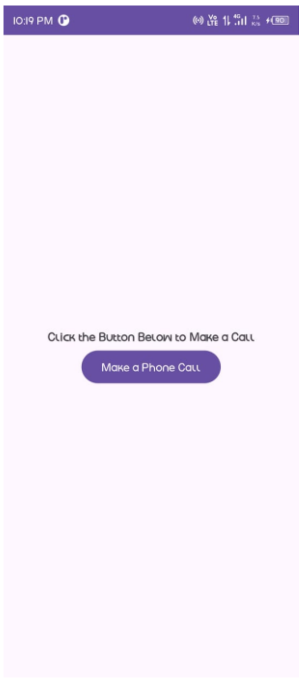
```java
package com.example.adlexp;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import androidx.appcompat.app.AppCompatActivity;

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

      Button callButton = findViewById(R.id.btnCall);
      callButton.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View view) {
            makePhoneCall();
         } }); }
   private void makePhoneCall() {
      Uri phoneNumber = Uri.parse("tel:+1234567890");  // Replace with the number
      Intent intent = new Intent(Intent.ACTION_DIAL, phoneNumber);  // Use ACTION_CALL if direct
calling is required
      startActivity(intent);
   } }
```

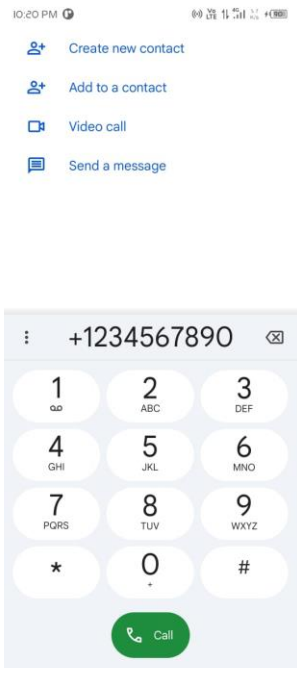
Xml Code

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:id="@+id/main"
```

```
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click the Button Below to Make a Call"
        android:textSize="8pt"
        android:textStyle="bold"
        android:padding="10px"/>

    <Button
        android:id="@+id/btnCall"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Make a Phone Call" />
</LinearLayout>
```

Output





Conclusion

Using intents, you can easily perform common tasks like sending SMS or making phone calls by interacting with built-in apps. The Intent.ACTION_SENDTO is used for SMS and Intent.ACTION_DIAL for initiating a phone call. This allows the integration of SMS and telephony functionalities into Android apps without directly handling these features

Experiment no: 13

Title: Write a program to demonstrate Buttons, Text Fields, Checkboxes, Radio Buttons, and Toggle Buttons with their event handlers.

Theory

In Android, user interactions with various UI elements such as **Buttons**, **Text Fields**, **Checkboxes**, **Radio Buttons**, and **Toggle Buttons** can be managed through event handlers like onClickListeners. These event handlers capture user actions and allow the app to respond accordingly.

- **Button**: A simple push button that can trigger a specific action when clicked.
- **Text Field (EditText)**: Allows user input via text.
- **Checkbox**: A widget that can be either checked or unchecked, often used for multi-selection.
- **Radio Button**: Allows users to select only one option from a group of options.
- **Toggle Button**: A button that has two states: ON and OFF.

Sample code

Java Code

```
package com.example.adlexp;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.EditText;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.TextView;
import android.widget.Toast;
import android.widget.ToggleButton;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    private EditText editText;
    private CheckBox checkBox1, checkBox2;
    private RadioGroup radioGroup;
    private ToggleButton toggleButton;
    private TextView resultText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        editText = findViewById(R.id.editText);
        checkBox1 = findViewById(R.id.checkBox1);
        checkBox2 = findViewById(R.id.checkBox2);
        radioGroup = findViewById(R.id.radioGroup);
        toggleButton = findViewById(R.id.toggleButton);
        resultText = findViewById(R.id.resultText);
        Button submitButton = findViewById(R.id.submitButton);

        // Handle Submit Button click
        submitButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String inputText = editText.getText().toString();
                StringBuilder result = new StringBuilder("Input: " + inputText);

                // Checkboxes handling
                if (checkBox1.isChecked()) {
                    result.append("\nSelected: Option 1");
                }
                if (checkBox2.isChecked()) {
                    result.append("\nSelected: Option 2");
                }

                // Radio Button handling
                int selectedRadioId = radioGroup.getCheckedRadioButtonId();
                if (selectedRadioId != -1) {
                    RadioButton selectedRadioButton = findViewById(selectedRadioId);
                    result.append("\nRadio selected: ").append(selectedRadioButton.getText());
                }

                // Toggle Button handling
                if (toggleButton.isChecked()) {
                    result.append("\nToggle: ON");
                } else {
                    result.append("\nToggle: OFF");
                }

                resultText.setText(result.toString());
            } });}}
```

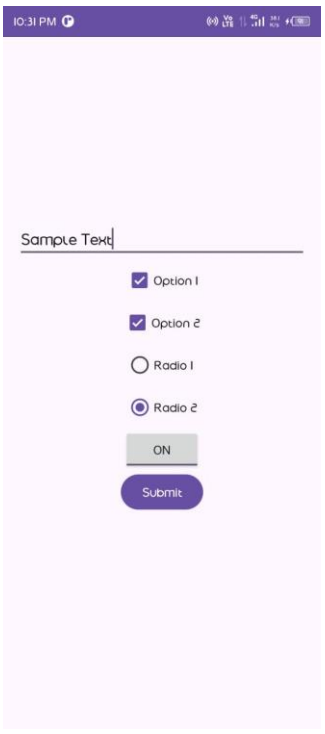
Xml Code

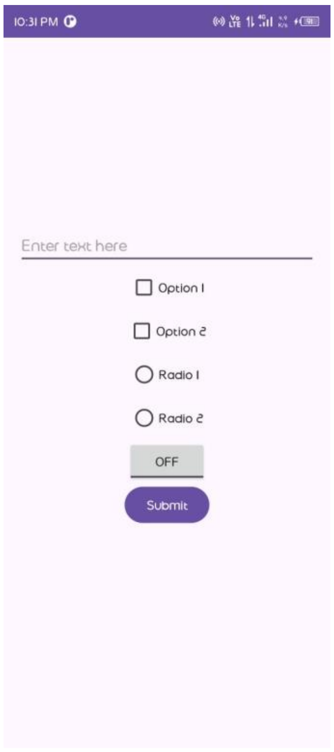
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
```

```
android:layout_height="match_parent"
android:orientation="vertical"
android:padding="16dp"
android:gravity="center"
tools:context=".MainActivity">


<!-- Text Field (EditText) -->
<EditText
   android:id="@+id/editText"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:hint="Enter text here" />

<!-- Checkboxes -->
<CheckBox
   android:id="@+id/checkBox1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Option 1" />

<CheckBox
   android:id="@+id/checkBox2"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Option 2" />

<!-- Radio Buttons Group -->
<RadioGroup
   android:id="@+id/radioGroup"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content">

   <RadioButton
      android:id="@+id/radioButton1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Radio 1" />

   <RadioButton
      android:id="@+id/radioButton2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Radio 2" />
</RadioGroup>

<!-- Toggle Button -->
<ToggleButton
```

```
        android:id="@+id/toggleButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textOff="OFF"
        android:textOn="ON" />

    <!-- Button to Submit -->
    <Button
        android:id="@+id/submitButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Submit" />

    <!-- TextView to display results -->
    <TextView
        android:id="@+id/resultText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingTop="16dp"
        android:textSize="16sp" />
</LinearLayout>
```

Output





Conclusion

This program demonstrates the use of multiple UI components such as Buttons, Text Fields, Checkboxes, Radio Buttons, and Toggle Buttons in an Android app. Event handlers like setOnClickListener allow the app to respond to user inputs and interactions effectively, displaying the results in real time.

Experiment no :14

Title : Program to demonstrate Touch Mode, Menus with their events handler.

Theory

- **Touch Mode**: In Android, **Touch Mode** refers to the mode in which the user interacts with the screen through touch. Any interaction, like tapping or swiping on the screen, is detected by the Android system using the **onTouchListener** or **gesture detectors**.
- **Menus**: Menus in Android allow users to perform actions and navigate within the app. There are three main types of menus:
    - **Options Menu**: Typically appears in the app's action bar or toolbar.
    - **Context Menu**: Appears when the user long-presses a view.
    - **Popup Menu**: Appears as a floating list of options when triggered.

Event handlers are used to detect and respond to user actions, such as selecting a menu item or touching the screen.

Sample code

Java code

```
package com.example.adlexp;


import android.os.Bundle;

import android.view.Menu;

import android.view.MenuInflater;

import android.view.MenuItem;

import android.widget.Toast;


import androidx.appcompat.app.AppCompatActivity;

import androidx.appcompat.widget.Toolbar;


public class MainActivity extends AppCompatActivity {


    @Override

    protected void onCreate(Bundle savedInstanceState) {

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

        // Set up the Toolbar
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.main_menu, menu);  // Inflate the menu from XML
        return true;
    }

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

        if (id == R.id.menu_item1) {
            Toast.makeText(this, "Menu Item 1 Selected", Toast.LENGTH_SHORT).show();
            return true;
        } else if (id == R.id.menu_item2) {
            Toast.makeText(this, "Menu Item 2 Selected", Toast.LENGTH_SHORT).show();
            return true;
        }

        return super.onOptionsItemSelected(item);
    }
}
```

XML Code

```
<?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:gravity="center"
    android:orientation="vertical"
    android:padding="16dp">

    <TextView
        android:id="@+id/touchTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Touch the screen to detect gestures!"
        android:textSize="18sp"
        android:padding="16dp" />

<androidx.appcompat.widget.Toolbar
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="?attr/colorPrimary" />

</LinearLayout>
```

main_menu.xml

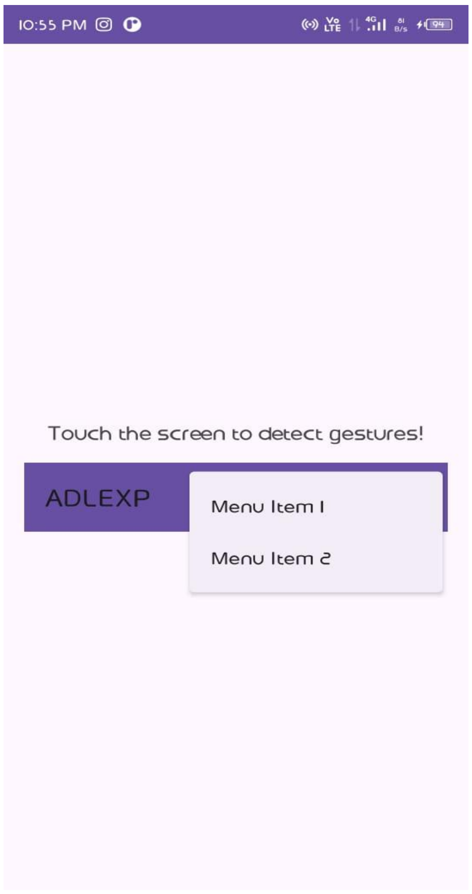
Navigate to the resourse directory

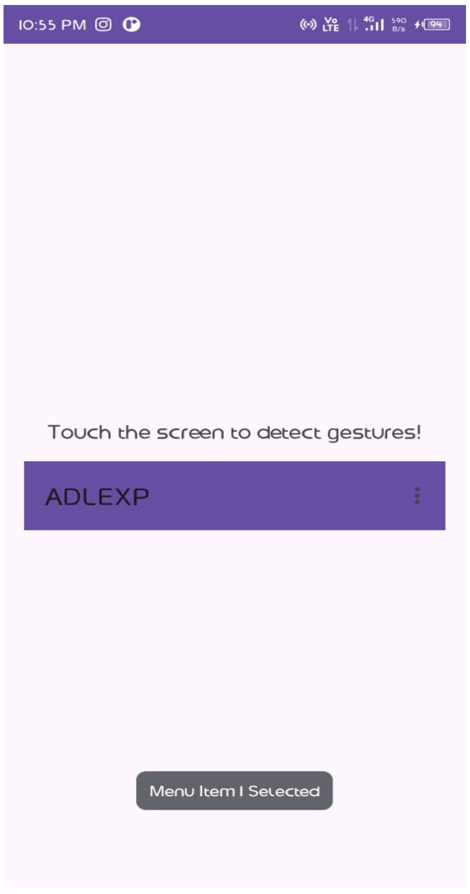
Create a menu directory

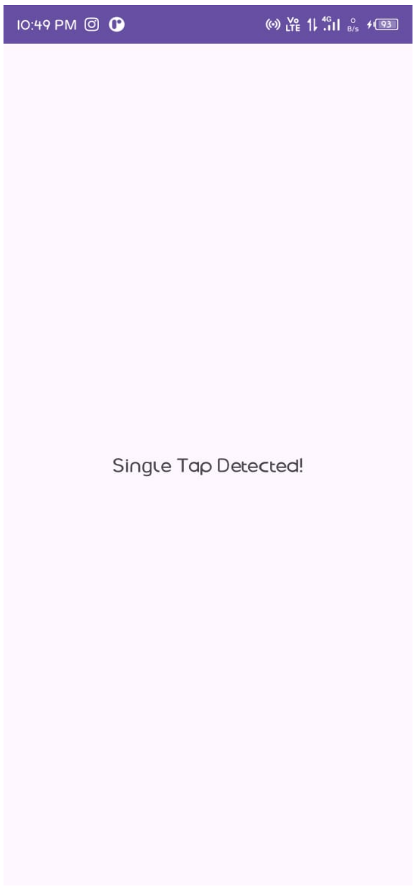
Create the main_menu.xml file

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/menu_item1"
        android:title="Menu Item 1" />
    <item
        android:id="@+id/menu_item2"
        android:title="Menu Item 2" />
</menu>
```

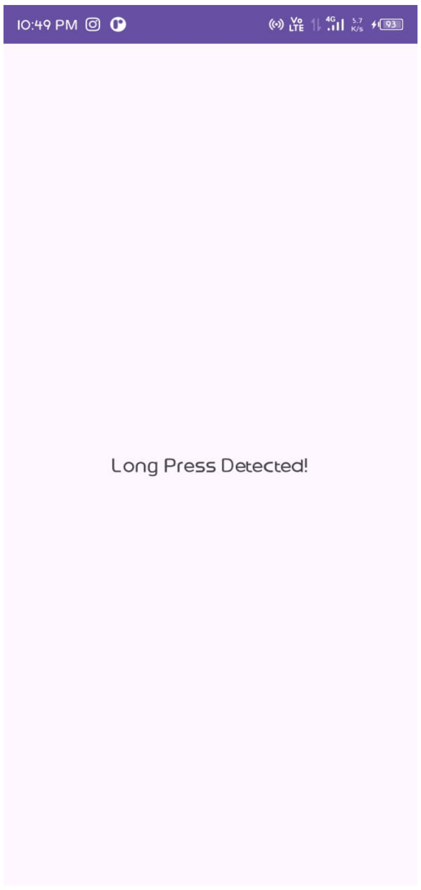
Output



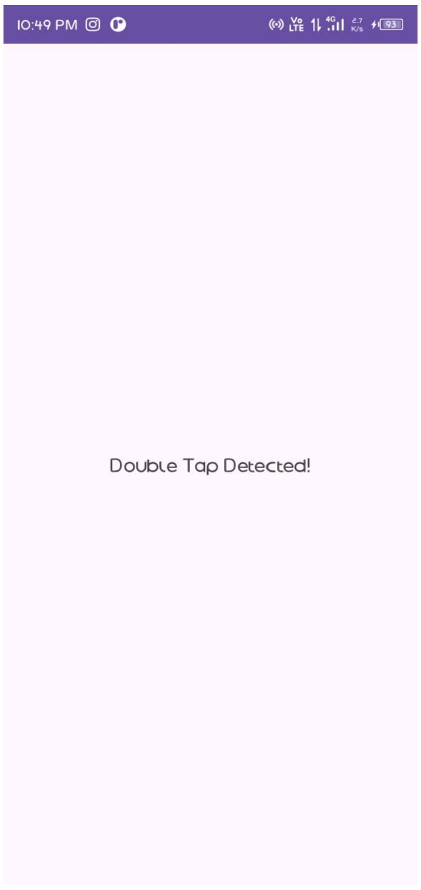



10:49 PM

Single Tap Detected!


10:49 PM

Long Press Detected!


10:49 PM

Double Tap Detected!

Conclusion

This program demonstrates the handling of touch gestures using GestureDetector and event handling for menus. The Touch Mode event listeners allow the app to respond to single taps, double taps, and long presses, while the menu event handlers allow the app to handle different menu item selections. This showcases how to create interactive user interfaces in Android that respond to user actions.

Experiment no: 15

Title: Implement an application that implements Multi-threading.

Theory

Multi-threading is a programming concept that allows multiple threads to run concurrently within a single process. In Android, it is essential for performing long-running operations (like network requests or file I/O) without blocking the main UI thread, which could lead to an unresponsive application.

Using threads in Android can be achieved through various methods, including Thread, Runnable, AsyncTask (deprecated in API level 30), or using the ExecutorService for managing thread pools. In this example, we will create a simple app that uses a Thread to perform a background task while updating the UI thread safely using Handler.

Sample code

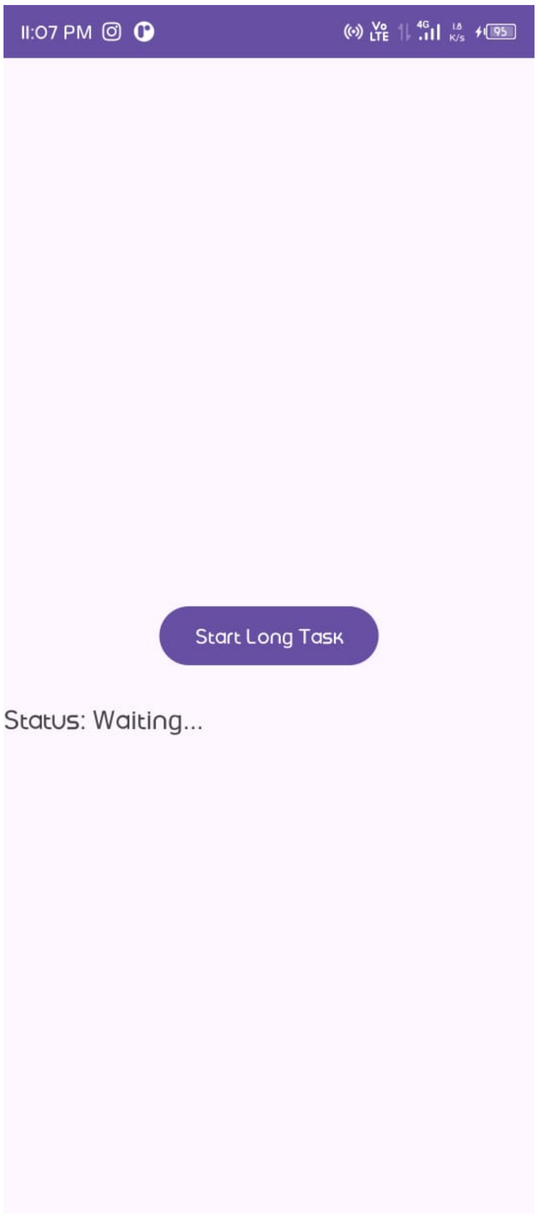
Java code

```
package com.example.adlexp;

import android.os.Bundle;
import android.os.Handler;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    private TextView statusTextView;
    private Handler handler = new Handler();

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

        statusTextView = findViewById(R.id.status_text_view);
        Button startButton = findViewById(R.id.start_button);

        startButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                startLongTask();
            }
        });}
  private void startLongTask() {
        // Update UI to show task is starting
        statusTextView.setText("Status: Task is running...");

        // Create a new thread for long-running task
        new Thread(new Runnable() {
            @Override
            public void run() {
                // Simulating long task (e.g., network operation)
                try {
                    Thread.sleep(5000); // Sleep for 5 seconds
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }

                // After the task is done, update the UI using Handler
                handler.post(new Runnable() {
                    @Override
                    public void run() {
                        statusTextView.setText("Status: Task completed!");
                    }});}
        }).start();
    }}
```

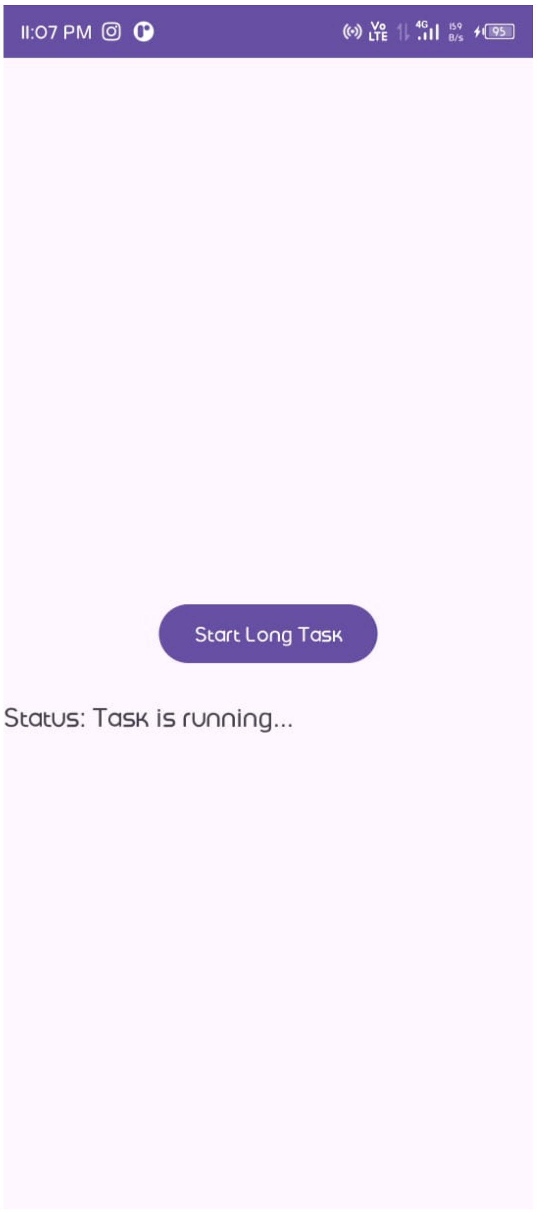
Xml Code

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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/start_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Start Long Task"
        android:layout_centerInParent="true"/>

    <TextView
        android:id="@+id/status_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/start_button"
        android:layout_marginTop="20dp"
        android:text="Status: Waiting..."
        android:textSize="18sp"/>
</RelativeLayout>
```

Output



Status: Waiting...



Status: Task is running...

Conclusion

This application demonstrates the use of multi-threading in Android by performing a long-running task on a separate thread, thereby keeping the UI responsive. By using a Handler, we safely update the UI from the background thread. This approach is fundamental in Android development, as it allows for smooth user experiences without blocking the main thread.

Experiment no: 16

Title : Write a program to study and use of SQLite database.

Theory

SQLite is a lightweight database engine that is included with Android. It provides a powerful and flexible way to manage application data in a structured format. Using SQLite, developers can store, retrieve, update, and delete data in their applications using SQL queries.

In Android, the SQLite database can be accessed through the SQLiteOpenHelper class, which helps manage database creation and version management. The database operations are performed through a SQLiteDatabase object, which allows executing SQL commands.

Sample code

Java code MainActivity

```
package com.example.adlexp;

import android.database.Cursor;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    DatabaseHelper db;
    EditText nameInput, ageInput;
    TextView displayData;

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

        db = new DatabaseHelper(this);
        nameInput = findViewById(R.id.name_input);
        ageInput = findViewById(R.id.age_input);
        displayData = findViewById(R.id.display_data);
        Button addButton = findViewById(R.id.add_button);
        Button viewButton = findViewById(R.id.view_button);

        addButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String name = nameInput.getText().toString();
                int age = Integer.parseInt(ageInput.getText().toString());
                boolean isInserted = db.insertData(name, age);
                if (isInserted) {
                    Toast.makeText(MainActivity.this, "Data Inserted", Toast.LENGTH_SHORT).show();
                } else {
                    Toast.makeText(MainActivity.this, "Data Not Inserted", Toast.LENGTH_SHORT).show();
                }
            }
        });

        viewButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Cursor cursor = db.getAllData();
                StringBuilder data = new StringBuilder();
                while (cursor.moveToNext()) {
                    data.append("ID: ").append(cursor.getString(0)).append("\n");
                    data.append("Name: ").append(cursor.getString(1)).append("\n");
                    data.append("Age: ").append(cursor.getString(2)).append("\n\n");
                }
                displayData.setText(data.toString());
            }
        });
    }
}
```

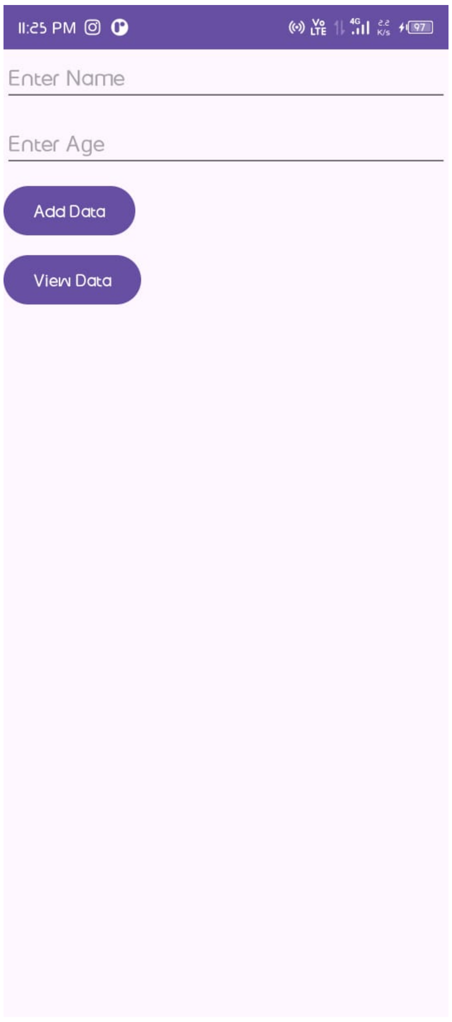
Database Code

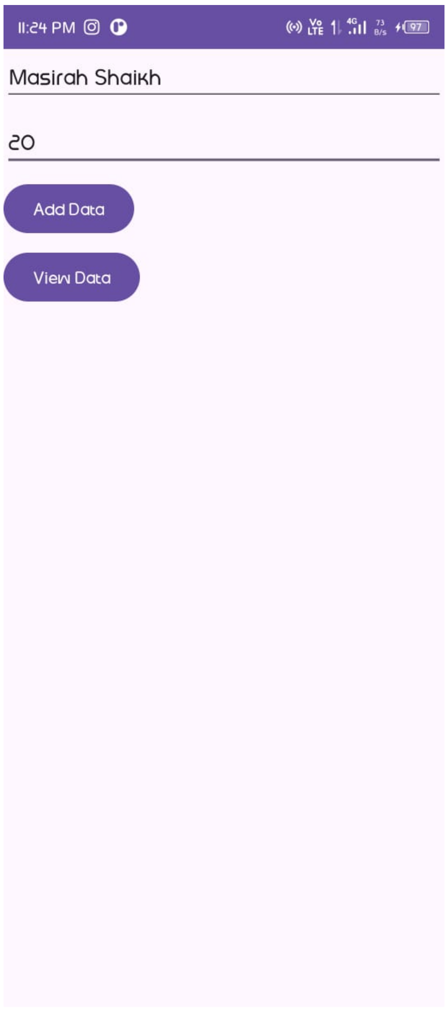
```
package com.example.adlexp;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class DatabaseHelper extends SQLiteOpenHelper {
    private static final String DATABASE_NAME = "example.db";
    private static final String TABLE_NAME = "users";
    private static final String COL_1 = "ID";
    private static final String COL_2 = "NAME";
    private static final String COL_3 = "AGE";

    // Constructor
    public DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, 1); // Ensure this matches the superclass constructor
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        // Create table SQL query
        db.execSQL("CREATE TABLE " + TABLE_NAME + " (ID INTEGER PRIMARY KEY AUTOINCREMENT,
NAME TEXT, AGE INTEGER)");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // Drop older table if it exists
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        // Create tables again
        onCreate(db);
    }

    // Insert data method
    public boolean insertData(String name, int age) {
        SQLiteDatabase db = this.getWritableDatabase(); // This should work correctly
        ContentValues contentValues = new ContentValues();
        contentValues.put(COL_2, name);
        contentValues.put(COL_3, age);
        long result = db.insert(TABLE_NAME, null, contentValues);
        return result != -1; // Returns true if data inserted successfully
    }

    // Retrieve all data method
    public Cursor getAllData() {
        SQLiteDatabase db = this.getWritableDatabase(); // This should work correctly
        return db.rawQuery("SELECT * FROM " + TABLE_NAME, null);
    }
}
```

Xml code

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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">
    <EditText
        android:id="@+id/name_input"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Enter Name" />

    <EditText
        android:id="@+id/age_input"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Enter Age"
        android:layout_below="@id/name_input"
        android:layout_marginTop="8dp" />

    <Button
        android:id="@+id/add_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Add Data"
        android:layout_below="@id/age_input"
        android:layout_marginTop="8dp" />

    <Button
        android:id="@+id/view_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="View Data"
        android:layout_below="@id/add_button"
        android:layout_marginTop="8dp" />

    <TextView
        android:id="@+id/display_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/view_button"
        android:layout_marginTop="16dp" />
</RelativeLayout>
```

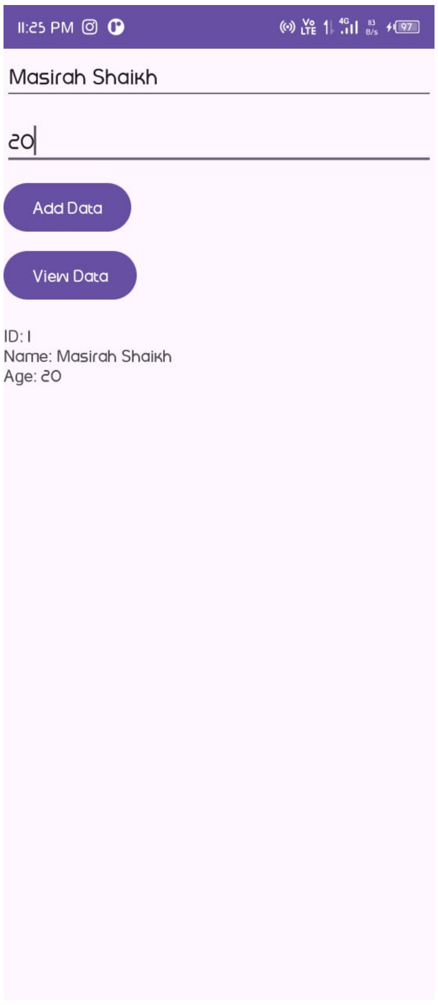
Output





Masirah Shaikh

20



Masirah Shaikh

20

```
ID: 1
Name: Masirah Shaikh
Age: 20
```

Conclusion

This application demonstrates how to use SQLite in Android for basic data management. By creating a DatabaseHelper class that extends SQLiteOpenHelper, we can easily handle database creation and version management. The main activity allows users to insert data into the database and retrieve it, showcasing the fundamental operations that can be performed with SQLite. This example serves as a foundation for more complex database interactions in Android applications.