

# **GURU NANAK DEV ENGINEERING COLLEGE LUDHIANA**

## **Mobile Application Development Laboratory File LPEIT – 116**



**SUBMITTED TO:**

Pf. RANJODH KAUR

**SUBMITTED BY:**

AANKITA BHAMETIYA

2121001/2104456

D4 IT A1

# CONTENTS

S. No.	Topic	Page No.
1	Android Development Environment: To study design aspects of development environment like Android, IOS	
2	Android Development Environment: To setup Android studio 2 and study its basic components	
3	Android User Interface Design: To study various XML files needed for interface design.	
4	Android User Interface Design: To implement different type of layouts like relative, grid, linear and table.	
5	Apps Interactivity in Android: To incorporate element of interactivity using Android Fragment and Intent Class	
6	Persistent Data Storage: To perform database connectivity of android app using SQLite	
7	Android Services and Threads: To implement the concept of multithreading using Android Service class	
8	Android Security and Debugging: To implement concept of permission and perform request for permission to access different hardware components of mobile	
9	Android Security and Debugging: To perform debugging and testing of android app using tools like Logcat, Android debug bridge, DDMS44	

## **Practical 1: Android Development Environment: To study design aspects of development environment like Android, ios.**

### **Android Development Environment:-**

1. Programming Languages
  - Java: The primary language for Android development for many years.
  - Kotlin: Officially supported by Google and preferred for new Android applications due to its modern features and interoperability with Java.
2. Integrated Development Environment (IDE)
  - Android Studio: The official IDE for Android development, based on IntelliJ IDEA. It includes features like code completion, debugging, and a powerful layout editor.
3. Software Development Kit (SDK)
  - Android SDK: Provides the tools necessary to build, test, and debug Android applications. Includes the Android emulator, platform tools, and various libraries.
4. Build System
  - Gradle: The build automation tool used in Android Studio to compile, package, and manage dependencies for Android projects.
5. Emulator
  - Android Emulator: Part of the Android SDK, it allows developers to run and test applications on virtual devices with different configurations and Android versions.
6. User Interface Design
  - XML Layouts: UI components are typically defined in XML files.
  - Jetpack Compose: A modern toolkit for building native UI using declarative programming.
7. Frameworks and Libraries
  - Android Jetpack: A set of libraries, tools, and guidance to help developers write highquality apps more easily. Includes components like LiveData, ViewModel, Room, Navigation, and WorkManager.
8. Testing
  - Junit: For unit testing.
  - Espresso: For UI testing.
  - Robolectric: For running tests on the JVM without an emulator.

### **ios Development Environment:-**

1. Programming Languages
  - Objective-C: An older language used for iOS development.
  - Swift: The modern language for iOS development, designed to be safe, fast, and expressive.
2. Integrated Development Environment (IDE)
  - Xcode: The official IDE for iOS development, which includes a code editor, debugger, and Interface Builder for designing user interfaces.
3. Software Development Kit (SDK)
  - iOS SDK: Provides the necessary tools and frameworks for developing iOS applications. Includes libraries like UIKit, Foundation, and Core Data.
4. Build System
  - Xcode Build System: Integrated into Xcode for compiling, packaging, and managing dependencies.
5. Simulator
  - iOS Simulator: Allows developers to test and debug applications on virtual devices with different iOS versions and device configurations.
6. User Interface Design
  - Storyboard and XIB files: Interface Builder within Xcode is used to design UIs visually.
  - SwiftUI: A modern declarative framework for building UI across all Apple platforms.
7. Frameworks and Libraries
  - Cocoa Touch: The application development environment for iOS, including frameworks like UIKit, Foundation, and Core Graphics.
  - Combine: A framework for handling asynchronous events by combining event processing operators.
8. Testing
  - XCTest: For unit and UI testing.
  - Quick/Nimble: Popular third-party frameworks for behavior-driven development (BDD).

### Comparative Analysis

Aspect	Android Development	iOS Development
Development Tools	Android Studio	Xcode
Programming Languages	Java, Kotlin	Swift, Objective-C
Frameworks	Jetpack, Retrofit, Dagger	SwiftUI, Combine, Alamofire

UI Design Tools	Layout Editor	Interface Builder
Virtual Device	Android Virtual Device (AVD)	iOS Simulator
Dependency Management	Gradle	CocoaPods, Swift Package Manager
Performance	Good with flexibility	Excellent with optimized hardware integration

## Practical 2: Android Development Environment: To setup Android studio2 and study its basic components

### Android Studio System Requirements for Windows

Microsoft Windows 7/8/10 (32-bit or 64-bit)

- 4 GB RAM minimum, 8 GB RAM recommended (plus 1 GB for the Android Emulator)
- 2 GB of available disk space minimum, 4 GB recommended (500 MB for IDE plus 1.5 GB for Android SDK and emulator system image)
- 1280 x 800 minimum screen resolution

### Steps to Install Android Studio on Windows

- Step 1: Head over to this link to get the Android Studio executable or zip file.
- Step 2: Click on the Download Android Studio Button



Android Studio provides the fastest tools for building apps on every type of Android device.

DOWNLOAD ANDROID STUDIO

4.1.3 for Windows 64-bit (896 MiB)

Download Android Studio

Before downloading, you must agree to the following terms and conditions.

Terms and Conditions

This is the Android Software Development Kit License Agreement

1. Introduction

1.1 The Android Software Development Kit (referred to in the License Agreement as the "SDK" and specifically including the Android system files, packaged APIs, and Google APIs add-ons) is licensed to you subject to the terms of the License Agreement. The License Agreement forms a legally binding contract between you and Google in relation to your use of the SDK.

1.2 "Android" means the Android software stack for devices, as made available under the Android Open Source Project, which is located at the following URL: <http://source.android.com/>, as updated from time to time.

1.3 A "compatible implementation" means any Android device that (i) complies with the Android Compatibility Definition document, which can be found at the Android compatibility website (<http://source.android.com/compatibility>) and which may be updated from time to time; and (ii) successfully passes the Android

☐ I have read and agree with the above terms and conditions

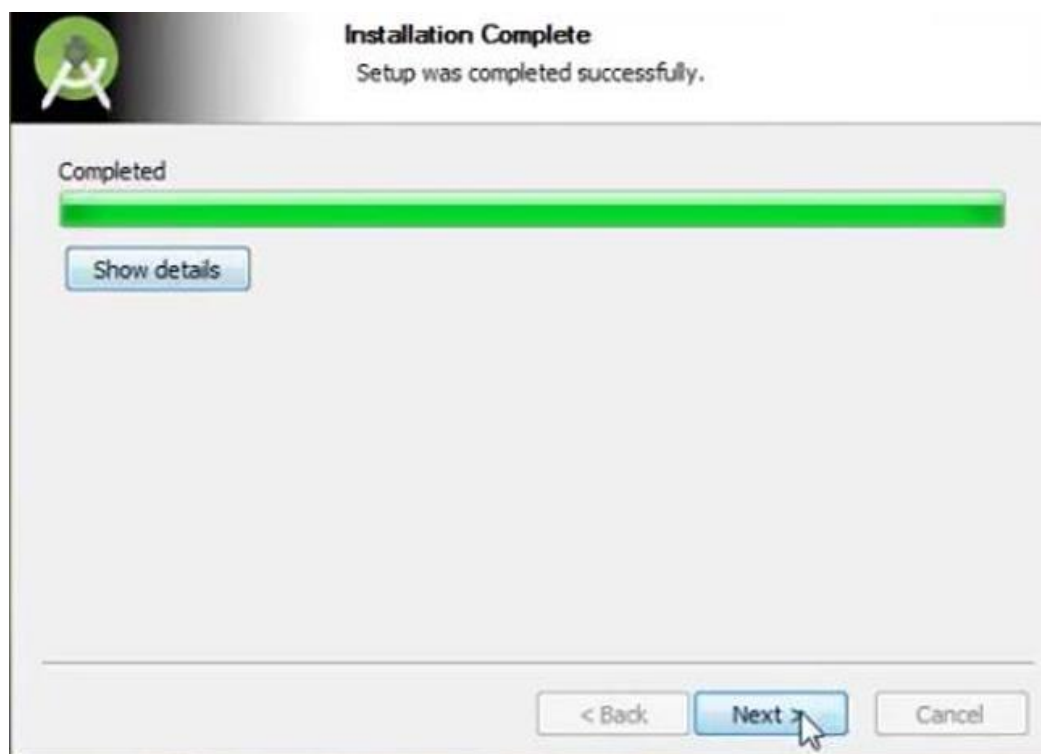
DOWNLOAD ANDROID STUDIO FOR WINDOWS

`android-studio-ide-173.4819257-windows.exe`

- Step 3: After the downloading has finished, open the file from downloads and run it. It will prompt the following dialog box.

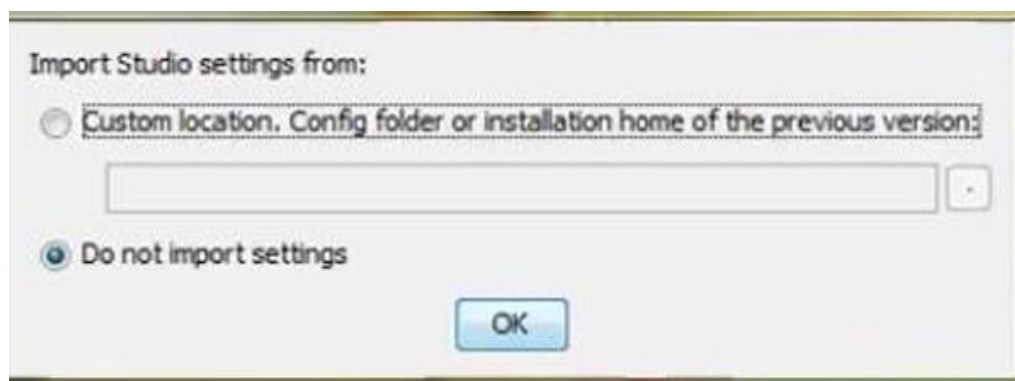


- Step 4: It will start the installation, and once it is completed, it will be like the image shown below.





- Step 5: Once “Finish” is clicked, it will ask whether the previous settings need to be imported [if the android studio had been installed earlier], or not. It is better to choose the ‘Don’t import Settings option’.



- Step 6: This will start the Android Studio.





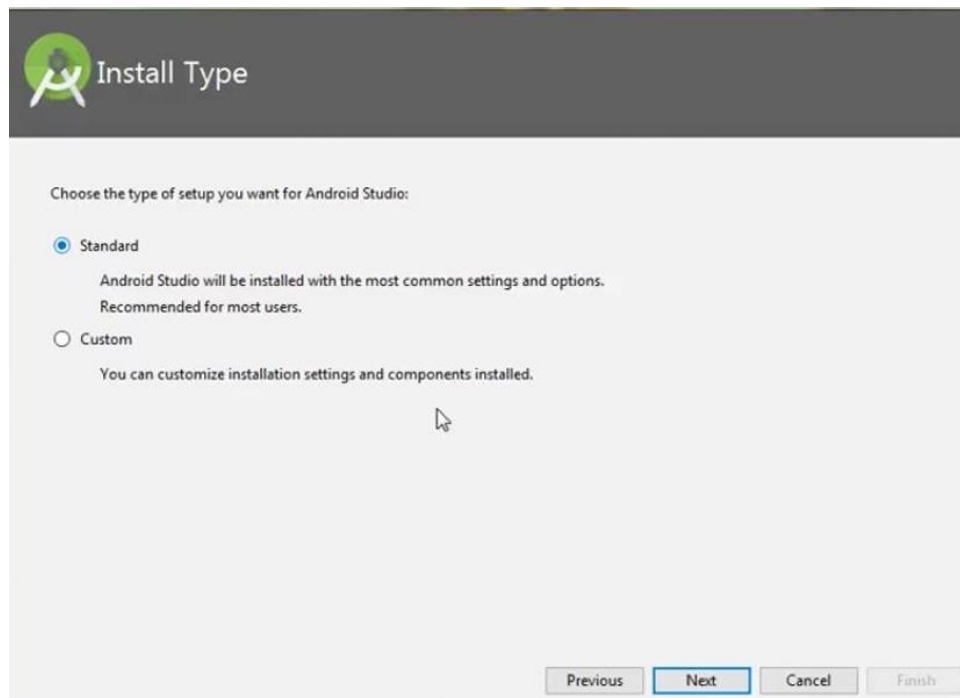
Meanwhile, it will be finding the available SDK components.



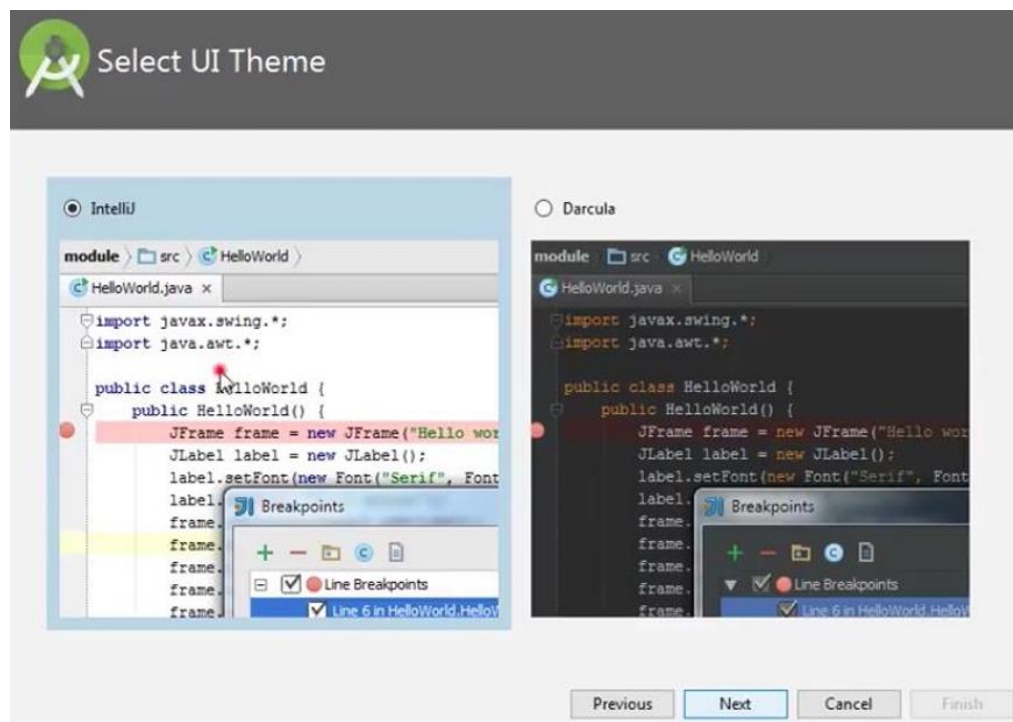
- Step 7: After it has found the SDK components, it will redirect to the Welcome dialog box.



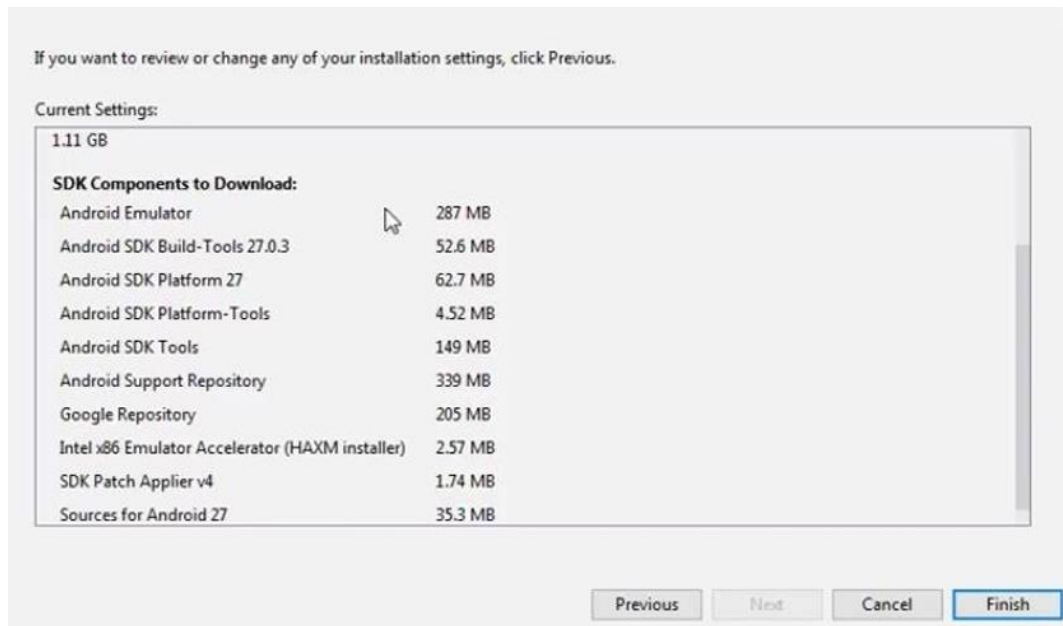
Click on Next.



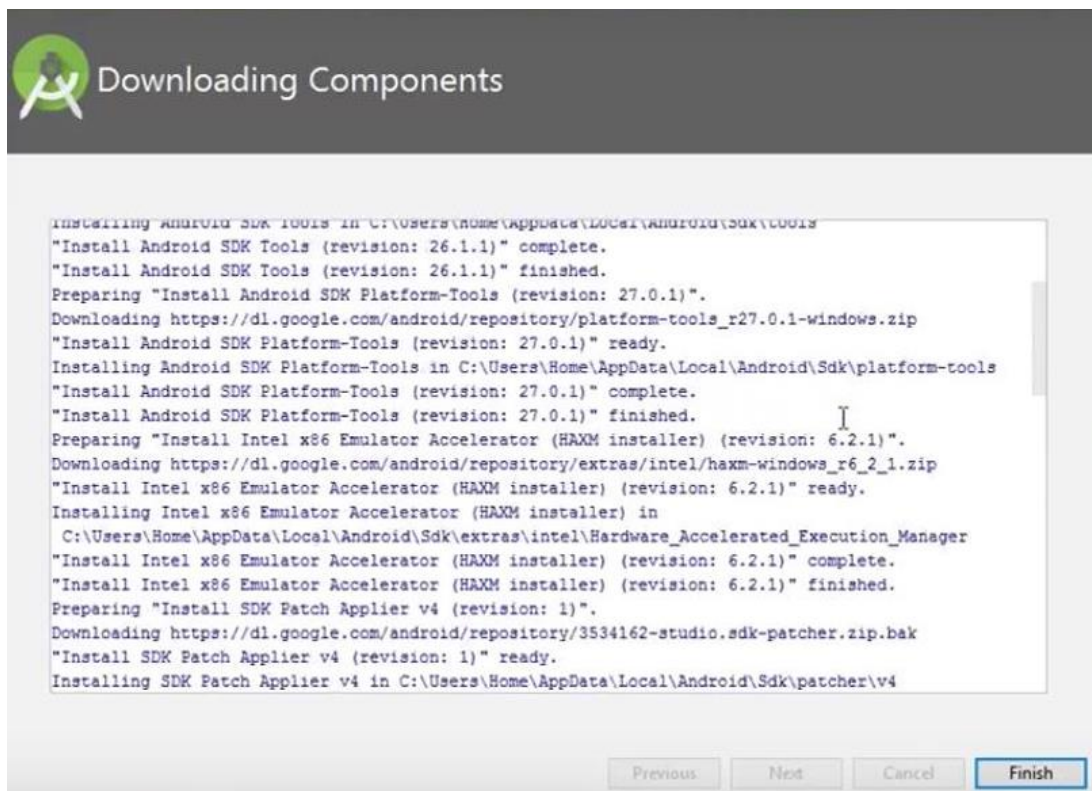
Choose Standard and click on Next. Now choose the theme, whether the Light theme or the Dark one. The light one is called the IntelliJ theme whereas the dark theme is called Dracula. Choose as required.



- Step 8: Now it is time to download the SDK components.



Click on Finish. Components begin to download let it complete.









The Android Studio has been successfully configured. Now it's time to launch and build apps. Click on the Finish button to launch it.


- Step 9: Click on Start a new Android Studio project to build a new app.



# Android Studio

Version 3.1.2

-  Start a new Android Studio project
-  Open an existing Android Studio project
-  Check out project from Version Control ▾
-  Profile or debug APK
-  Import project (Gradle, Eclipse ADT, etc.)
-  Import an Android code sample

 Configure ▾     Get Help ▾

## **Practical 3: Android User Interface Design: To study various XML files needed for interface design.**

When designing the user interface (UI) for an Android application, several XML files are typically used. These files define the layout, styles, strings, dimensions, and other aspects of the UI. Here's an overview of the most common XML files:

### 1. Layout Files

Layout files define the structure and appearance of the UI components in an activity or fragment.

- activity\_main.xml: Defines the main layout for an activity.
- fragment\_example.xml: Defines the layout for a fragment.

Common layout elements include:

- LinearLayout: Arranges its children in a single column or row.
- RelativeLayout: Positions its children relative to each other or to the parent.
- ConstraintLayout: Offers more flexibility and is recommended for complex layouts.
- FrameLayout: Designed to block out an area on the screen to display a single item.

Example of a simple layout file (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">

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

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click Me" />

</LinearLayout>
```

### 2. Resource Files

These files store various types of resources used in the app, such as strings, colors, dimensions, and styles.

- strings.xml: Stores all the string resources used in the app.
- colors.xml: Defines the color resources.
- dimens.xml: Specifies dimension resources like padding and margins.

- styles.xml: Contains style definitions to maintain a consistent look and feel.

Example of strings.xml:

```
<resources>
    <string name="app_name">MyApp</string>
    <string name="hello_world">Hello, World!</string>
    <string name="button_click_me">Click Me</string>
</resources>
```

### 3. Manifest File

The AndroidManifest.xml file provides essential information about the app to the Android system, including activities, permissions, services, and broadcast receivers.

Example of AndroidManifest.xml:

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/AppTheme">

        <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>
```

### 4. Drawable Resources

Drawable resources are graphics that can be drawn to the screen. These can be defined as XML files (e.g., shapes, colors, state lists) or as image files (e.g., PNG, JPEG).

Example of a shape drawable (res/drawable/rounded\_button.xml):

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <corners android:radius="8dp" />
    <solid android:color="@color/colorPrimary" />
    <padding android:left="16dp" android:top="8dp" android:right="16dp" android:bottom="8dp" />
</shape>
```



## 5. Menu Resources

Menu resource files define the contents of app menus.

Example of a menu resource (res/menu/menu\_main.xml):

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/action_settings"
        android:title="@string/action_settings"
        android:orderInCategory="100"
        android:showAsAction="never" />
</menu>
```

## 6. Values Resources

In addition to strings, colors, and dimensions, the values folder can contain other types of resources:

- bools.xml: Stores boolean resources.
- integers.xml: Stores integer resources.
- arrays.xml: Defines array resources.

Example of arrays.xml:

```
<resources>
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
    </string-array>
</resources>
```

These XML files collectively define the visual and interactive aspects of an Android application's user interface. Understanding and effectively utilizing these files is crucial for creating well-structured and maintainable Android apps.

## **Practical 4: Android User Interface Design: To implement different type of layouts like relative, grid, linear and table.**

1) Develop a program to implement constraint layout to display Hello World on screen:-

**Code:**

```
//Activity_main.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"
```

```
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent">
```

```
<!-- TextView for "Hello World" -->
```

```
<TextView
```

```
    android:id="@+id/helloWorldText"
```

```
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
```

```
    android:text="Hello World"
```

```
    android:textSize="24sp"
```

```
    android:textColor="#000000"
```

```
    app:layout_constraintTop_toTopOf="parent"
```

```
    app:layout_constraintBottom_toBottomOf="parent"
```

```
    app:layout_constraintStart_toStartOf="parent"
```

```
    app:layout_constraintEnd_toEndOf="parent" />
```

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

```
// MainActivity.java
```



```

package com.example.practical1helloworld;

import android.os.Bundle;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

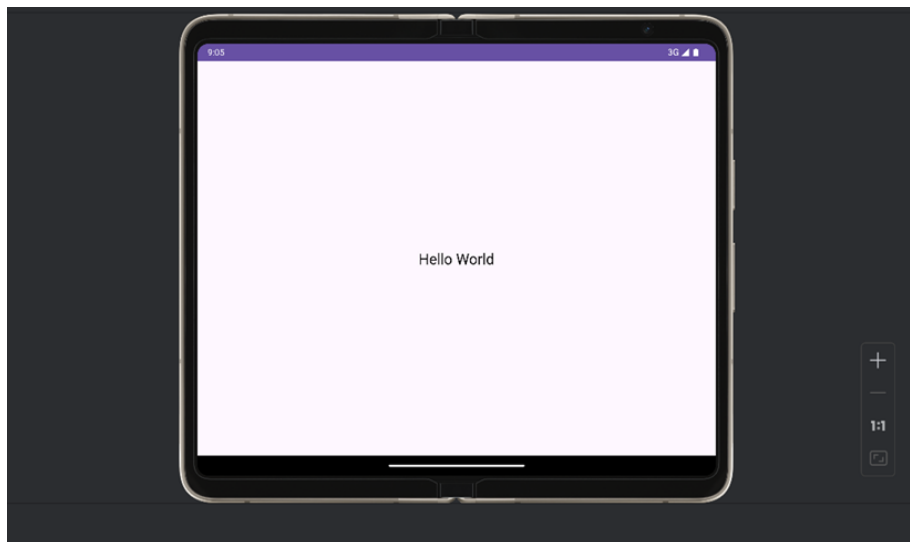
        setContentView(R.layout.activity_main);

    }

}

```

**OUTPUT:**



2) Develop a program to implement linear layout to display send message and registration form (vertical):-

**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:orientation="vertical"
```

```
android:padding="16dp"
```

```
android:id="@+id/main">
```

```
<!--To Field -->
```

```
<EditText
```

```
    android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
```

```
    android:hint="To"
```

```
    android:height="50dp" />
```

```
<!-- Subject Field -->
```

```
<EditText
```

```
    android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
```

```
    android:hint="Subject"
```

```
    android:layout_marginTop="16dp"
```

```
    android:height="50dp" />
```

```
<!-- Message Field -->
```

```
<EditText
```

```
    android:layout_width="match_parent"
```

```
    android:layout_height="0dp"
```

```
    android:layout_weight="1"
```

```
    android:hint="Message"
```

```
        android:gravity="top"

        android:inputType="textMultiLine"

        android:height="50dp"

        android:layout_marginTop="16dp" />

<!-- Send Button -->

<Button

        android:id="@+id/button_send"

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:text="SEND"

        android:layout_gravity="end"

        android:layout_marginTop="16dp" />

</LinearLayout>
```

#### OUTPUT:



### 3) Develop a program to implement relative layout to display Login and sign up form:-

#### **Code:**

```
//Activity_main.xml
```

```
<?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">
```

```
<!-- Login Form -->
```

```
<EditText
```

```
    android:id="@+id/edtUsername"
```

```
    android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
```

```
    android:hint="Username"
```

```
    android:layout_marginTop="100dp"
```

```
    android:padding="10dp"
```

```
    android:inputType="textPersonName"/>
```

```
<EditText
```

```
    android:id="@+id/edtPassword"
```

```
    android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
```

```
    android:hint="Password"
```

```
    android:layout_below="@id/edtUsername"
```

```
    android:layout_marginTop="20dp"
```

```
    android:padding="10dp"
```

```
    android:inputType="textPassword"/>
```

<Button

```
    android:id="@+id/btnLogin"

    android:layout_width="wrap_content"

    android:layout_height="wrap_content"

    android:text="Login"

    android:layout_below="@id/edtPassword"

    android:layout_marginTop="20dp"

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

<!-- Sign Up Form -->

<TextView

```
    android:id="@+id/txtSignUp"

    android:layout_width="wrap_content"

    android:layout_height="wrap_content"

    android:text="Don't have an account? Sign Up"

    android:layout_below="@id/btnLogin"

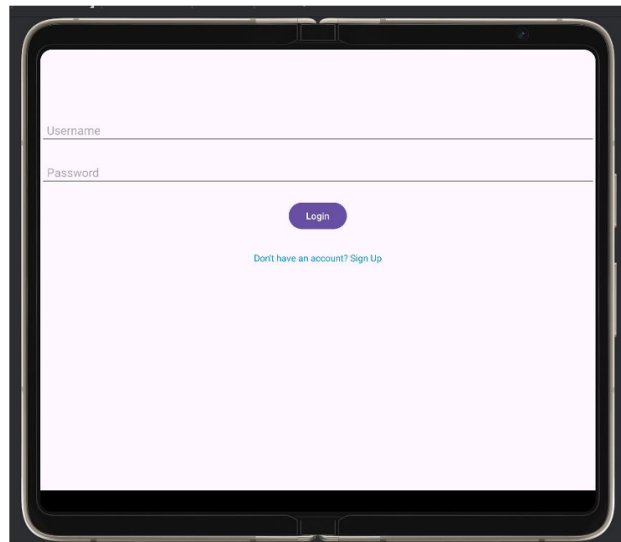
    android:layout_marginTop="30dp"

    android:layout_centerHorizontal="true"

    android:textColor="@android:color/holo_blue_dark"/>
```

</RelativeLayout>

**Output:**



```
//Activity_main.xml
```

```
<?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">
```

```
<!-- Username EditText -->
```

```
<EditText
```

```
    android:id="@+id/username"
```

```
    android:layout_width="match_parent"
```

```
    android:layout_height="48dp"
```

```
    android:hint="Username"
```

```
    android:inputType="textPersonName"
```

```
    android:layout_alignParentTop="true"
```

```
    android:layout_marginTop="40dp"/>
```

```
<!-- Email EditText -->
```

```
<EditText
```

```
        android:id="@+id/email"

        android:layout_width="match_parent"

        android:layout_height="48dp"

        android:hint="@string/email"

        android:inputType="textEmailAddress"

        android:layout_below="@id/username"

        android:layout_marginTop="20dp"/>
```

```
<!-- Password EditText -->
```

```
<EditText
```

```
        android:id="@+id/password"

        android:layout_width="match_parent"

        android:layout_height="48dp"

        android:hint="Password"

        android:inputType="textPassword"

        android:layout_below="@id/email"

        android:layout_marginTop="20dp"/>
```

```
<!-- Sign Up Button -->
```

```
<Button
```

```
        android:id="@+id/signUpButton"

        android:layout_width="match_parent"

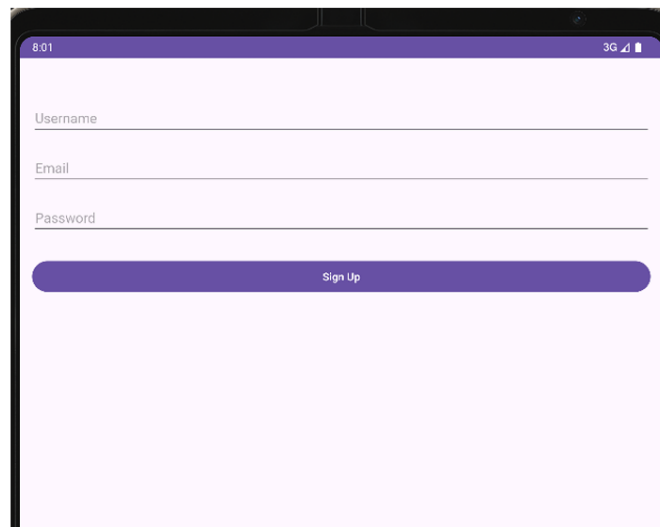
        android:layout_height="wrap_content"

        android:text="Sign Up"

        android:layout_below="@id/password"

        android:layout_marginTop="30dp"/>
```

```
</RelativeLayout>
```

**Output:**

4) Develop a program to implement table layout to display calculator:-

**Code:**

```
//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"
```

```
    android:padding="16dp"
```

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

```
<!-- Display Screen for the Calculator -->
```

```
<EditText
```

```
    android:id="@+id/display"
```

```
    android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
```



```
android:autofillHints=""

android:textSize="30sp"

android:inputType="none"

android:focusable="false"

android:gravity="end"

android:layout_marginBottom="20dp"

tools:ignore="LabelFor" />
```

```
<!-- TableLayout for Calculator Buttons -->
```

```
<TableLayout
```

```
    android:layout_width="match_parent"

    android:layout_height="wrap_content">
```

```
<!-- First Row -->
```

```
<TableRow style="?android:attr/buttonBarStyle">
```

```
    <Button
```

```
        android:id="@+id/button7"

        style="?android:attr/buttonBarButtonStyle"

        android:layout_width="0dp"

        android:layout_height="wrap_content"

        android:layout_weight="1"

        android:text="@string/_7" />
```

```
    <Button
```

```
        android:id="@+id/button8"
```

```
style="?android:attr/buttonBarButtonStyle"

android:layout_width="0dp"

android:layout_height="wrap_content"

android:layout_weight="1"

android:text="@string/_8" />
```

```
<Button

    android:id="@+id/button9"

    style="?android:attr/buttonBarButtonStyle"

    android:layout_width="0dp"

    android:layout_height="wrap_content"

    android:layout_weight="1"

    android:text="@string/_9" />
```

```
<Button

    android:id="@+id/buttonDiv"

    style="?android:attr/buttonBarButtonStyle"

    android:layout_width="0dp"

    android:layout_height="wrap_content"

    android:layout_weight="1"

    android:text="@string/_h" />
```

```
</TableRow>
```

```
<!-- Second Row -->
```

```
<TableRow style="?android:attr/buttonBarStyle">
```

```
    <Button
```

```
android:id="@+id/button4"

style="?android:attr/buttonBarButtonStyle"

android:layout_width="0dp"

android:layout_height="wrap_content"

android:layout_weight="1"

android:text="@string/_4" />
```

<Button

```
android:id="@+id/button5"

style="?android:attr/buttonBarButtonStyle"

android:layout_width="0dp"

android:layout_height="wrap_content"

android:layout_weight="1"

android:text="@string/_5" />
```

<Button

```
android:id="@+id/button6"

style="?android:attr/buttonBarButtonStyle"

android:layout_width="0dp"

android:layout_height="wrap_content"

android:layout_weight="1"

android:text="@string/_6" />
```

<Button

```
android:id="@+id/buttonMul"
```

```
        style="?android:attr/buttonBarButtonStyle"

        android:layout_width="0dp"

        android:layout_height="wrap_content"

        android:layout_weight="1"

        android:text="@string/_k" />

</TableRow>
```

```
<!-- Third Row -->
```

```
<TableRow style="?android:attr/buttonBarStyle">
```

```
<Button

    android:id="@+id/button1"

    style="?android:attr/buttonBarButtonStyle"

    android:layout_width="0dp"

    android:layout_height="wrap_content"

    android:layout_weight="1"

    android:text="@string/_1" />
```

```
<Button

    android:id="@+id/button2"

    style="?android:attr/buttonBarButtonStyle"

    android:layout_width="0dp"

    android:layout_height="wrap_content"

    android:layout_weight="1"

    android:text="@string/_2" />
```

```
<Button

    android:id="@+id/button3"

    style="?android:attr/buttonBarButtonStyle"

    android:layout_width="0dp"

    android:layout_height="wrap_content"

    android:layout_weight="1"

    android:text="@string/_3" />
```

```
<Button

    android:id="@+id/buttonSub"

    style="?android:attr/buttonBarButtonStyle"

    android:layout_width="0dp"

    android:layout_height="wrap_content"

    android:layout_weight="1"

    android:text="@string/_l" />
```

```
</TableRow>
```

```
<!-- Fourth Row -->
```

```
<TableRow style="?android:attr/buttonBarStyle">
```

```
<Button

    android:id="@+id/button0"

    style="?android:attr/buttonBarButtonStyle"

    android:layout_width="0dp"
```

android:layout\_height="wrap\_content"

android:layout\_weight="1"

android:text="@string/\_0" />

<Button

android:id="@+id/buttonClear"

style="?android:attr/buttonBarButtonStyle"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

android:layout\_weight="1"

android:text="@string/c" />

<Button

android:id="@+id/buttonEqual"

style="?android:attr/buttonBarButtonStyle"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

android:layout\_weight="1"

android:text="@string/p" />

<Button

android:id="@+id/buttonAdd"

style="?android:attr/buttonBarButtonStyle"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

android:layout\_weight="1"

android:text="@string/a" />

</TableRow>

</TableLayout>

</LinearLayout>

**Output:**



## **Practical 5: Apps Interactivity in Android: To incorporate element of interactivity using Android Fragment and Intent Class.**

```
// MainActivity.java
```

```
package com.cscorner.experiment5;
```

```
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);
```

```
        // Load the fragment into the container
```

```
        getSupportFragmentManager()
```

```
            .beginTransaction()
```

```
            .replace(R.id.fragment_container, new MyFragment())
```

```
            .commit();
```

```
    }
```

```
}
```

```
//Activity_main.xml
```

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

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

```
    android:id="@+id/fragment_container"
```

```
    android:layout_width="match_parent"
```

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



```
//fragment_my.xml
```

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

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

```
    android:id="@+id/fragment_layout"
```

```
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
```

```
    android:gravity="center"
```

```
    android:orientation="vertical">
```

```
<Button
```

```
    android:id="@+id/fragment_button"
```

```
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
```

```
    android:text="Open SecondActivity" />
```

```
</LinearLayout>
```

```
//SecondActivity.java
```

```
package com.cscorner.experiment5;
```

```
import android.os.Bundle;
```

```
import androidx.appcompat.app.AppCompatActivity;
```

```
public class SecondActivity extends AppCompatActivity {
```

```
    @Override
```

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

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

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

```
    }
```

```
}
```

```
//activity_second.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:gravity="center"
```

```
    android:orientation="vertical">
```

```
    <TextView
```

```
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
```

```
        android:text="This is SecondActivity"
```

```
        android:textSize="18sp" />
```

```
</LinearLayout>
```

```
//MyFragment.java
```

```
<?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">
```

```
    <TextView
```

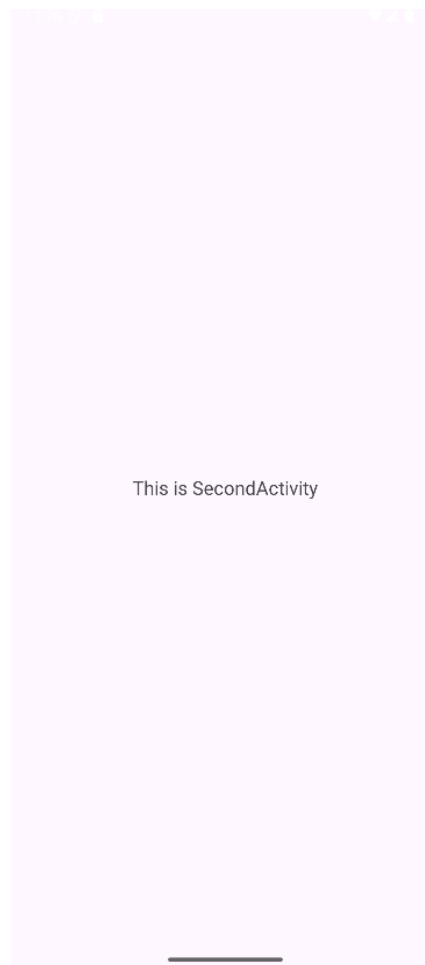
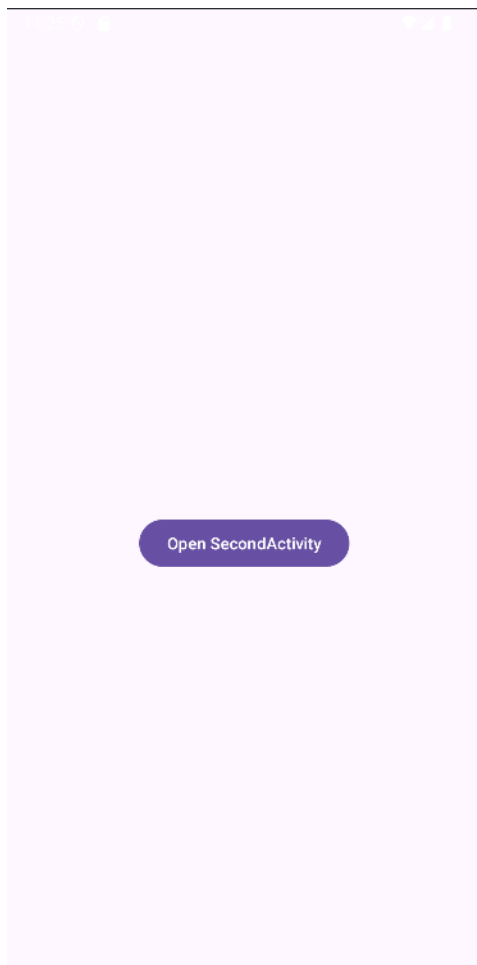
```
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"

        android:text="This is SecondActivity"

        android:textSize="18sp" />
</LinearLayout>
```

**Output:**



## **Practical 6: Persistent Data Storage: To perform database connectivity of android app using SQLite.**

Activity\_main.xml

```
<?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:layout_width="match_parent" android:layout_height="match_parent"
tools:context="ty.practical6.MainActivity">

<Button android:id="@+id/btnSendEmail" android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginTop="92dp" android:onClick="sendEmail"
android:text="Compose Email
android:layout_alignTop="@+id/txtMessage" android:layout_centerHorizontal="true"
/>

<EditText android:id="@+id/txtMessage" android:layout_width="wrap_content"
android:layout_height="wrap_content" android:ems="10" android:hint="Message"
android:inputType="textMultiLine" android:singleLine="true"
android:layout_marginTop="48dp" android:layout_below="@+id/txtSubject"
android:layout_centerHorizontal="true" />

<EditText android:id="@+id/txtEmailTo" android:layout_width="wrap_content"
android:layout_height="wrap_content" android:ems="10" android:hint="To"
android:inputType="textEmailAddress" android:layout_marginTop="22dp"
android:layout_alignParentTop="true" android:layout_alignStart="@+id/txtSubject"
/>
```

```
<EditText android:id="@+id/txtSubject" android:layout_width="wrap_content"
android:layout_height="wrap_content" android:layout_alignStart="@+id/txtMessage"
android:layout_below="@+id/txtEmailTo" android:layout_marginTop="43dp"
android:ems="10"
android:hint="Subject" android:inputType="text" />
</RelativeLayout>
```

Source Code:

MainActivity.java

```
package ty.practical6;

import android.content.Intent;

import android.net.Uri;

import android.os.Bundle;

import android.support.v7.app.AppCompatActivity;

import android.view.View;

import android.widget.EditText;

import android.widget.Toast;

public class MainActivity extends AppCompatActivity

{ @Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_main);

}

public void sendEmail(View v) {

EditText txtEmailTo = (EditText) findViewById(R.id.txtEmailTo);
```

```
EditText txtSubject = (EditText) findViewById(R.id.txtSubject);

EditText txtMessage = (EditText) findViewById(R.id.txtMessage);

String[] TO = {txtEmailTo.getText().toString()};

String[] CC = {"");

String subject = txtSubject.getText().toString();

String msg = txtMessage.getText().toString();

Intent emailIntent = new Intent(Intent.ACTION_SEND);

emailIntent.setData(Uri.parse("mailto:"));

emailIntent.setType("text/plain");

emailIntent.putExtra(Intent.EXTRA_EMAIL, TO);

emailIntent.putExtra(Intent.EXTRA_CC, CC);

emailIntent.putExtra(Intent.EXTRA_SUBJECT, subject);

emailIntent.putExtra(Intent.EXTRA_TEXT, msg);

try {

startActivity(Intent.createChooser(emailIntent, "Send mail..."));

finish();

}

catch (android.content.ActivityNotFoundException ex)

{

Toast.makeText(MainActivity.this, "No email client app installed.",

Toast.LENGTH_SHORT).show();

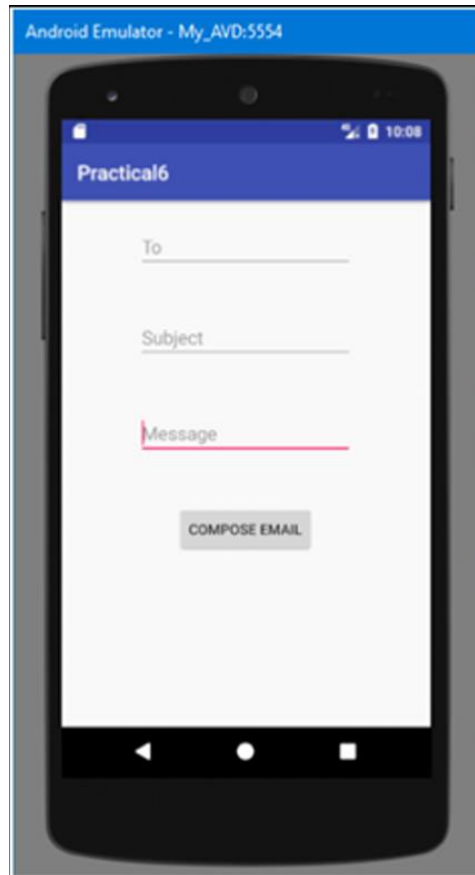
}

}

}
```

On clicking COMPOSE EMAIL button a list of apps will be displayed select a relevant email client app e.g. Gmail and the contents given as input here will be passed to Gmail app's email compose screen.

**OUTPUT:**



## **Practical 7: Android Services and Threads: To implement the concept of multithreading using Android Service class.**

Introduction: Multi-threading is defined as a feature through which we can run two or more concurrent threads of a process. In this a process, the common data is shared among all these threads also known as sub-processes exclusively. In android there are many ways through which multi-threading can be established in the application.

Objective:

- Understanding the basic concept of multithreading.
- Understanding of Handler class in android
- Understanding of Runnable Interface.

Multi-Threading In Android: Multi-Threading in Android is a unique feature through which more than one threads execute together without hindering the execution of other threads.

Multi-Threading in Android is not different from conventional multi-Threading. A class can be thought of as a process having its method as it's sub-processes or threads. All these methods can run concurrently by using feature of Multi-Threading. In android, multi Threading can be achieved through the use of many in-built classes. Out of them, Handler class is most commonly used.

Handler Class In Android: Handler class come from the Package android.os.Handler package and is most commonly used for multi-threading in android. Handler class provide sending and receiving feature for messages between different threads and handle the thread execution which is associated with that instance of Handler class. In android class, every thread is associated with an instance of Handler class and it allows the thread to run along with other threads and communicate with them through messages.

Runnable Interface:

Runnable interface is used in multi-threading to be called in a loop when the thread starts. It



is a type of thread that executes the statement in its body or calls other methods for a specified or infinite number of times. This Runnable interface is used by the Handler class to execute the multi-threading, i.e., to execute one or more thread in specified time. Runnable is an interface which is implemented by the class desired to support multithreading and that class must implement its abstract method public void run(). Run() method is the core of multithreading as it includes the statement or calls to other methods that the thread needs to be made for multithreading.

**CODE:**

```
//mainactivity.java

package com.cscorner.experiment7;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

import android.widget.TextView;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    private TextView outputText;

    private Button startThreadButton;

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);

        // Initialize UI components

        outputText = findViewById(R.id.outputText);
```

```

startThreadButton = findViewById(R.id.startThreadButton);

startThreadButton.setOnClickListener(new View.OnClickListener() {

    @Override

    public void onClick(View v) {

        // Start the thread when the button is clicked

        startThread();

    }

});

}

private void startThread() {

    // Create and start a new thread

    new Thread(new Runnable() {

        @Override

        public void run() {

            // Simulating a task that runs in a loop

            for (int i = 1; i <= 10; i++) {

                final int count = i;

                try {

                    Thread.sleep(1000); // Sleep for 1 second

                } catch (InterruptedException e) {

                    e.printStackTrace();

                }

            }

            runOnUiThread(new Runnable() {

                @Override

                public void run() {

```

```

        outputText.setText("Current count: " + count);
    }

});

}

runOnUiThread(new Runnable() {

    @Override

    public void run() {

        outputText.setText("Thread Finished!");

    }

});

}

}).start(); // Start the thread

}

}

```

//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">

    <TextView

        android:id="@+id/outputText"

```

```
android:layout_width="wrap_content"

android:layout_height="wrap_content"

android:text="Thread not started"

android:textSize="18sp"

android:gravity="center"

android:layout_marginBottom="16dp" />
```

```
<Button
```

```
    android:id="@+id/startThreadButton"

    android:layout_width="wrap_content"

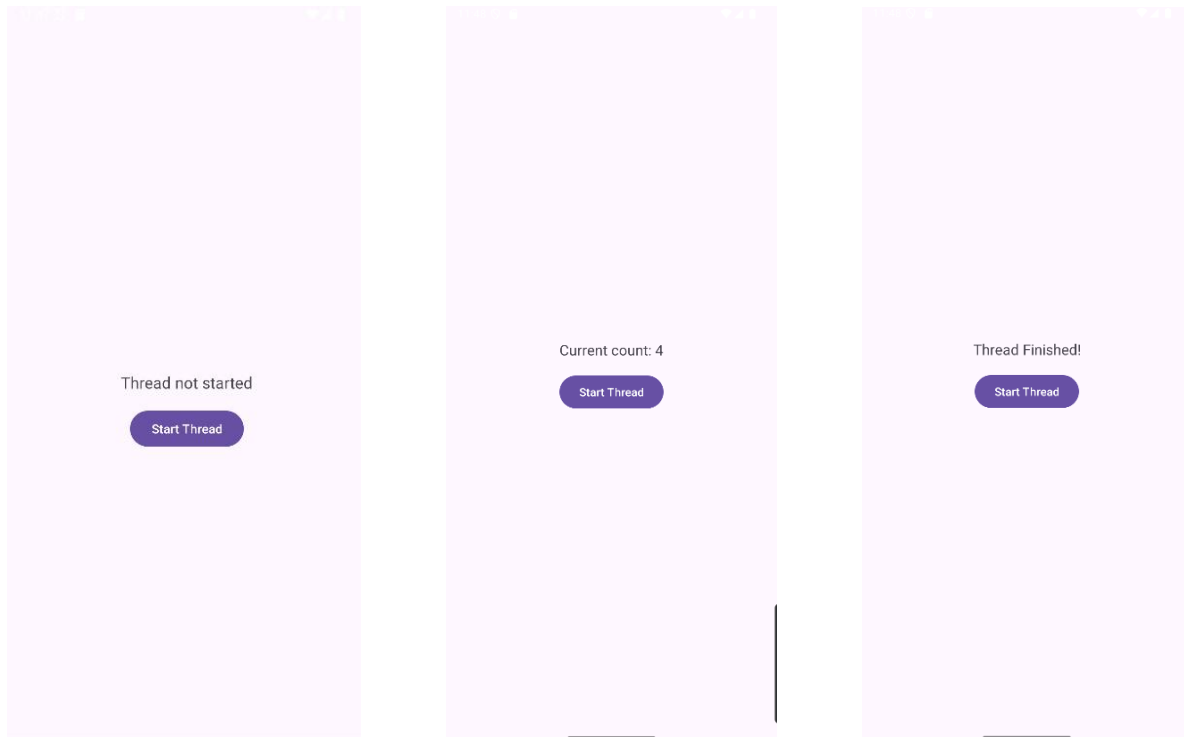
    android:layout_height="wrap_content"

    android:layout_gravity="center"

    android:text="Start Thread" />
```

```
</LinearLayout>
```

## OUTPUT:



## **Practical 8 To implement concept of permission and perform request for permission to access different hardware components of mobile.**

Permissions : The purpose of a permission is to protect the privacy of an Android user.

Android apps must request permission to access sensitive user data (such as contacts and SMS), as well as certain system features (such as camera and internet). Depending on the feature, the system might grant the permission automatically or might prompt the user to approve the request.

Permission approval : An app must publicize the permissions it requires by including tags in the app manifest. For example, an app that needs to access location would have this line in the manifest:

```
//mainactivity.java

package com.cscorner.experiment8;

import android.Manifest;

import android.content.pm.PackageManager;

import android.os.Bundle;

import android.widget.Button;

import android.widget.TextView;

import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

import androidx.core.app.ActivityCompat;

import androidx.core.content.ContextCompat;

public class MainActivity extends AppCompatActivity {

    private static final int LOCATION_PERMISSION_REQUEST_CODE = 1;

    private TextView permissionStatusText;

    private Button requestPermissionButton;
```

@Override

```
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_main);

    permissionStatusText = findViewById(R.id.permissionStatusText);

    requestPermissionButton = findViewById(R.id.requestPermissionButton);

    if (ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED &&

        ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_COARSE_LOCATION)
!= PackageManager.PERMISSION_GRANTED) {

        permissionStatusText.setText("Location permission not granted.");

        requestPermissionButton.setEnabled(true);

    } else {

        permissionStatusText.setText("Location permission granted!");

        requestPermissionButton.setEnabled(false); // Disable button if permission is already granted

    }

    requestPermissionButton.setOnClickListener(v -> ActivityCompat.requestPermissions(this,

        new                                     String[]{Manifest.permission.ACCESS_FINE_LOCATION,
Manifest.permission.ACCESS_COARSE_LOCATION},

        LOCATION_PERMISSION_REQUEST_CODE));

}
```

@Override

```
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {

    super.onRequestPermissionsResult(requestCode, permissions, grantResults);

    if (requestCode == LOCATION_PERMISSION_REQUEST_CODE) {

        if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
```

```

        // Permission granted, update the UI

        permissionStatusText.setText("Location permission granted!");

        Toast.makeText(this, "Location permission granted!", Toast.LENGTH_SHORT).show();

    } else {

        // Permission denied, update the UI

        permissionStatusText.setText("Location permission denied.");

        Toast.makeText(this, "Location permission denied.", Toast.LENGTH_SHORT).show();

    }

}

}

}

}

```

//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"

    android:gravity="center">

    <TextView

        android:id="@+id/permissionStatusText"

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:text="Location permission status will appear here."

```

```

        android:textSize="18sp"

        android:gravity="center"

        android:layout_marginBottom="16dp"/>

<Button

    android:id="@+id/requestPermissionButton"

    android:layout_width="wrap_content"

    android:layout_height="wrap_content"

    android:text="Request Location Permission"

    android:layout_gravity="center"

    android:enabled="true"/>

</LinearLayout>

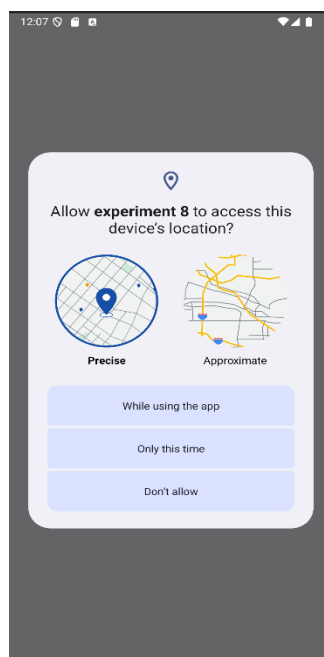
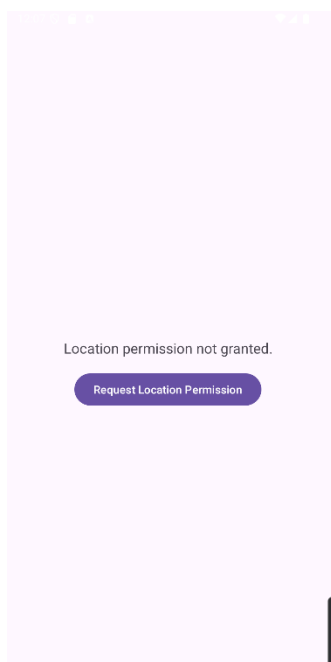
//AndroidManifest.xml

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

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

```

## OUTPUT:





## **Practical 9 To perform debugging and testing of android app using tools like Logcat, Android debug bridge, DDMS.**

Debugging and Testing:

Debugging is the process of finding and fixing errors (bugs) or unexpected behaviour in our code. All code has bugs, from incorrect behaviour in our app, to behaviour that excessively consumes memory or network resources, to actual app freezing or crashing.

Bugs can result for many reasons:

- Errors in design or implementation
- Android framework limitations (or bugs)
- Missing requirements or assumptions for how the app should work
- Device limitations (or bugs)

Use the debugging, testing, and profiling capabilities in Android Studio to help us reproduce,

find, and resolve all of these problems. Those capabilities include:

- Logcat
- Android Debug Bridge
- DDMS (Dalvik Debug Monitor Server)

LOGCAT:

We can use log class to send messages to the Android system log, and view those messages in Android Studio in the Logcat pane. The Android SDK includes a useful logging utility class called `android.util.Log`. The class allows us to log messages categorized based severity; each type of logging message has its own message. Here is a listing of the message types, and their respective method calls, ordered from lowest to highest priority:

- The `Log.v()` method is used to log verbose messages.
- The `Log.d()` method is used to log debug messages.
- The `Log.i()` method is used to log informational messages.
- The `Log.w()` method is used to log warnings.
- The `Log.e()` method is used to log errors.
- The `Log.wtf()` method is used to log events that should never happen

("wtf" being an abbreviation for "What a Terrible Failure", of course).

We can think of this method as the equivalent of Java's assert method.

Android Debug Bridge:

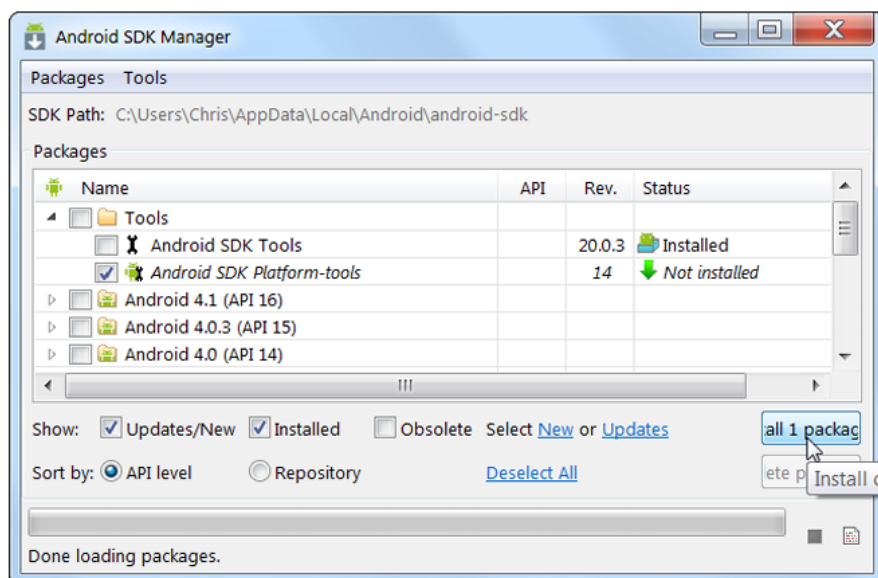
Android Debug Bridge (adb) is a versatile command-line tool that lets us communicate with a device. ADB is a part of Android SDK. The adb command facilitates a variety of device actions, such as installing and debugging apps, and it provides access to a Unix shell that we can use to run a variety of commands on a device. It is a client-server program that includes three components:

- A client, which sends commands. The client runs on development machine. We can invoke a client from a command-line terminal by issuing an adb command.
- A daemon (adbd), which runs commands on a device. The daemon runs as a background process on each device.
- A server, which manages communication between the client and the daemon. The server runs as a background process on your development machine.

Enable adb debugging:

We have to use USB debugging under Developer Options for using adb with a device. We can now connect our device with USB. We can verify that our device is connected by executing adb devices from the android\_sdk/platform-tools/ directory.

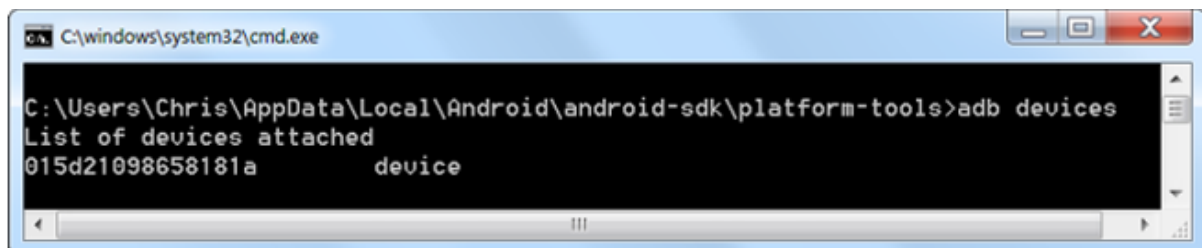
Setup Android SDK



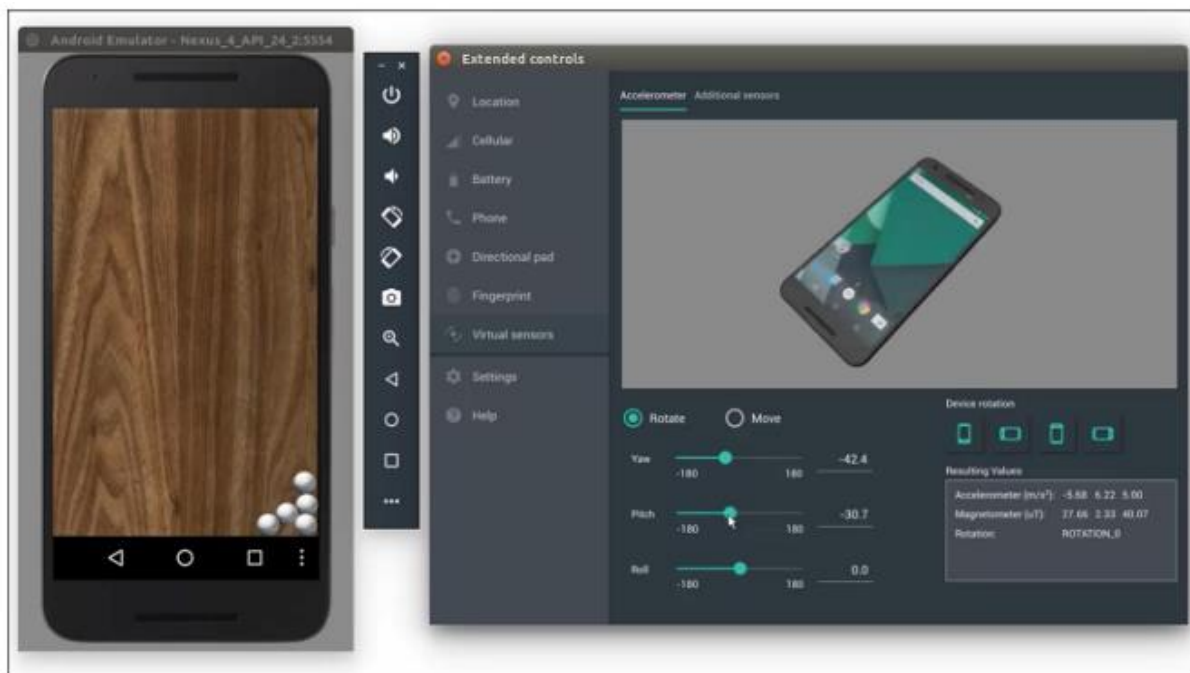
Enable USB debugging:



Test ADB connection: adb devices



Now, we can run ADB commands to use the tool in console for both Debugging and Testing of the application. Further details and guide to the ADB can be seen [here](#). DDMS (Dalvik Debug Monitor Server): The DDMS is a debugging tool used in the Android platform. The Dalvik Debug Monitor Service is downloaded as part of the Android SDK. Some of the services provided by the DDMS are port forwarding, on-device screen capture, on-device thread and heap monitoring, and radio state information. It allows developers to spot bugs in applications running on either an emulator or an actual Android device. Its feature, known as Emulator Control allows developers to simulate phone states and activities. For example, it can simulate different types of networks which can have different network characteristics such as speed and latency. This debugging tool can be integrated into the Eclipse IDE by adding the ADT (Android Development Tools) plug-in. Otherwise, it can be accessed from the command line and will automatically connect to any running emulator.



DDMS is deprecated. Its features have been replaced by other new features. Instead of this, we use Android Profiler in Android Studio 3.0 and higher to profile your app's CPU, memory, and network usage. To perform other debugging tasks, such as sending commands to a connected device to set up port-forwarding, transfer files, or take screenshots, then use the Android Debug Bridge (adb), Android Emulator, Device File Explorer, or Debugger window.