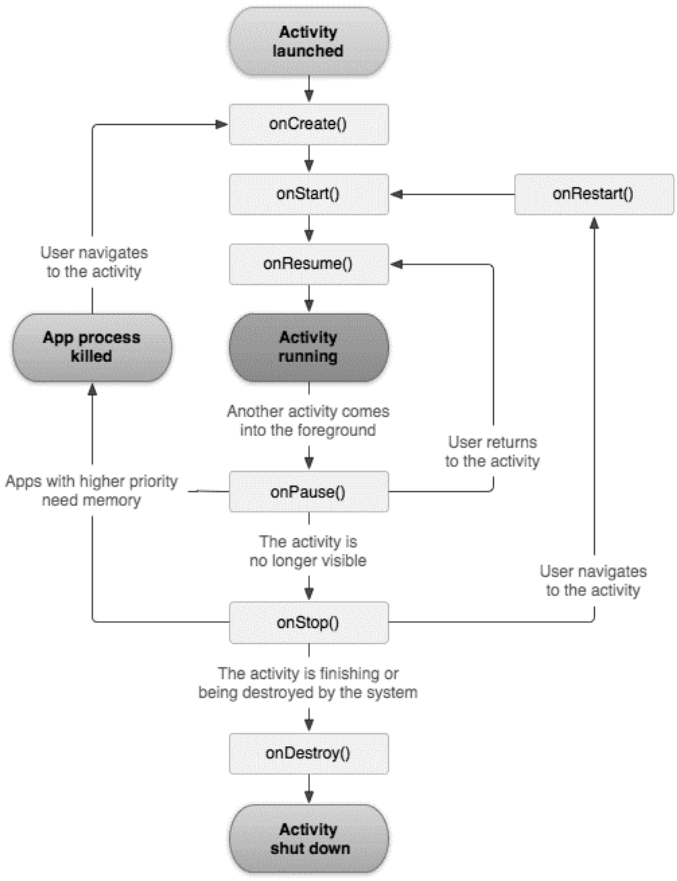
Unit 2

Android Activities, Fragments, and Intents

Working with activities

- In android, Activity represents a single screen with a user interface (UI) of an application and it will act an entry point for users to interact with an app.
- Generally, the android apps will contain multiple screens and each screen of our application will be an extension of Activity class. By using activities, we can place all our android application UI components in a single screen.
- From the multiple activities in android app, one activity can be marked as a main activity and that is the first screen to appear when we launch the application. In android app each activity can start another activity to perform different actions based on our requirements.
- For example, a contacts app which is having multiple activities, in that the main activity screen will show a list of contacts and from the main activity screen we can launch other activities that provide screens to perform tasks like add a new contact and search for the contacts. All these activities in the contact app are loosely bound to other activities but will work together to provide a better user experience.
- Generally, in android there is a minimal dependency between the activities in an app. To use activities in application we need to register those activities information in our app's manifest file (AndroidMainfest.xml) and need to manage activity life cycle properly (In a new version of android studio, new activity's registration is done automatically in a AndroidMainfest.xml)

Android Activity Lifecycle



Generally, the activities in our android application will go through a different stage in their life cycle. In android, Activity class have 7 callback methods like onCreate(), onStart(), onPause(), onRestart(), onResume(), onStop() and onDestroy() to describe how the activity will behave at different stages.

Method	Description
onCreate	called when activity is first created.
onStart	called when activity is becoming visible to the user.
onResume	called when activity will start interacting with the user.
onPause	called when activity is not visible to the user.
onStop	called when activity is no longer visible to the user.
onRestart	called after your activity is stopped, prior to start.
onDestroy	called before the activity is destroyed.

Example

It provides the details about the invocation of life cycle methods of activity. In this example, we are displaying the content on the logcat.

MainActivity.java

```
package com.example.activityexample;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d("lifecycle","onCreate invoked");
    }
    @Override
    protected void onStart() {
        super.onStart();
        Log.d("lifecycle","onStart invoked");
    }
    @Override
    protected void onResume() {
        super.onResume();
        Log.d("lifecycle","onResume invoked");
    }
    @Override
```

```
    protected void onPause() {
        super.onPause();
        Log.d("lifecycle","onPause invoked");
    }
    @Override
    protected void onStop() {
        super.onStop();
        Log.d("lifecycle","onStop invoked");
    }
    @Override
    protected void onRestart() {
        super.onRestart();
        Log.d("lifecycle","onRestart invoked");
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.d("lifecycle","onDestroy invoked");
    }
}
```

Android Fragments

In android, Fragments are the modular section of activity design and these are used to represent the behaviour of user interface (UI) in an activity. By using fragments, we can create flexible UI designs that can be adjusted based on the device screen size such as tablets, smartphones.

We can build multi-pane UI by combining multiple fragments in a single activity and we can reuse the same fragment in multiple activities. The fragment has its own lifecycle call-backs and accepts its own input events.

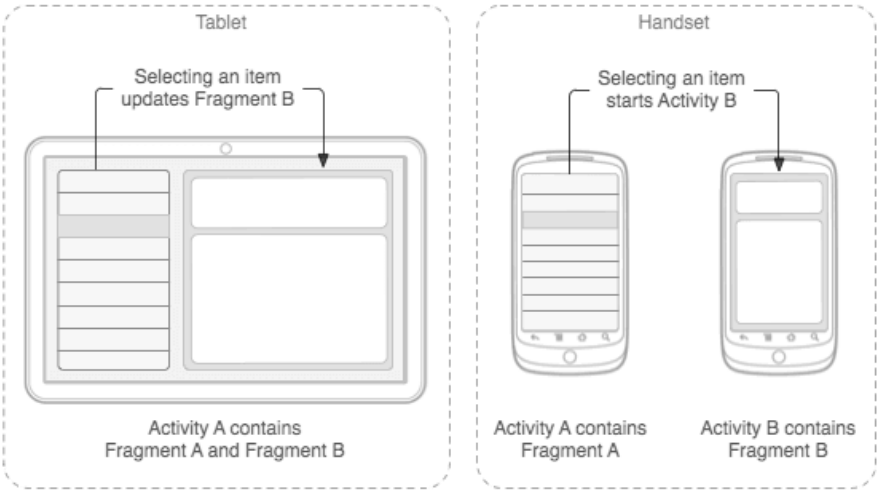
We can add or remove fragments in an activity while the activity is running. In android, the fragment will act as a sub-activity and we can reuse it in multiple activities.

Generally, in android, the fragment must be included in an activity due to that the fragment lifecycle will always be affected by the host activity life cycle. In case if we pause an activity, all the fragments related to an activity will also be stopped.

In android, we can insert the fragment into activity layout by using <fragment> element and by dividing the layout of activity into fragments, we can modify the appearance of an app design at runtime. We can also implement a fragment without having any user interface (UI).

It is an optional to use fragments into activity but by doing this it will improve the flexibility of our app UI and make it easier to adjust our app design based on the device size.

Following is the example of defining multiple fragments in single activity for the tablet design to display the details of an item which we selected in the app, but separated for mobile design.

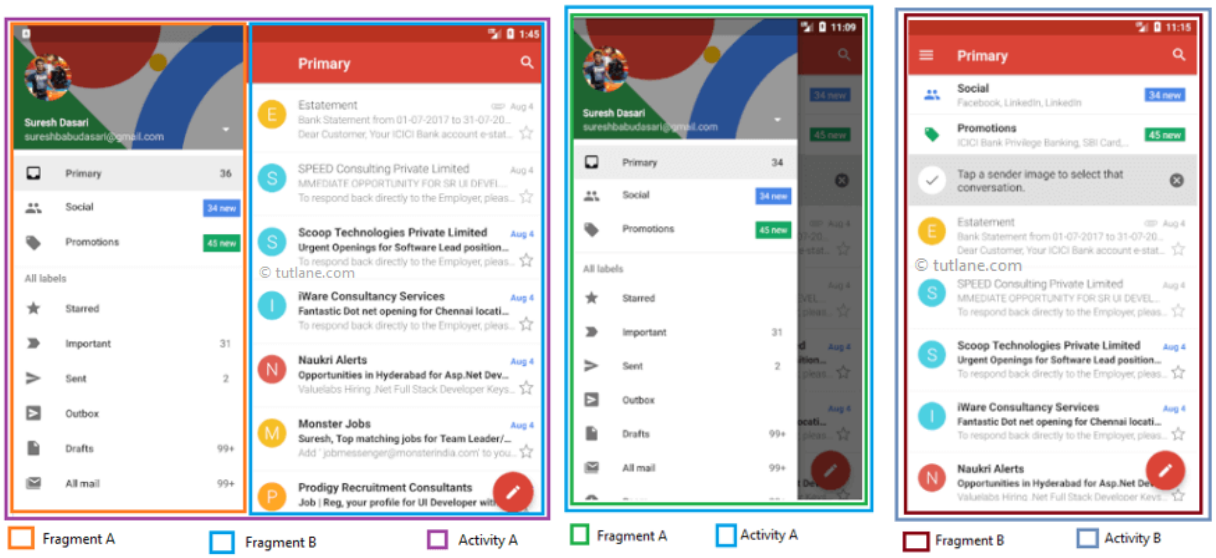


Here, in above example for Tablet we defined an Activity A with two fragments such as one is to show the list of items and second one is to show the details of item which we selected in first fragment.

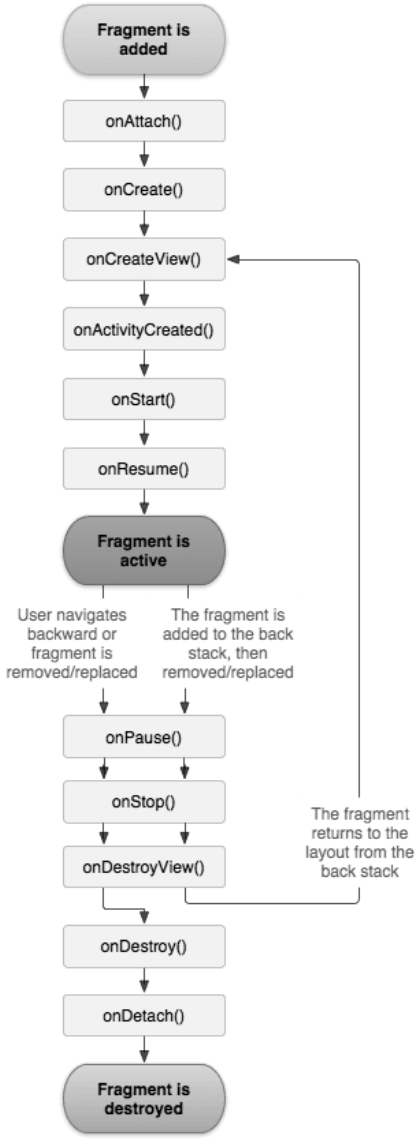
For Handset device, there is no enough space to show both the fragments in single activity, so the Activity A includes first fragment to show the list of items and the Activity B which includes another fragment to display the details of an item which is selected in Activity A.

For example, GMAIL app is designed with multiple fragments, so the design of GMAIL app will be varied based on the size of device such as tablet or mobile device.

Table View **Mobile View**



Android Fragment Life Cycle



Following are the list of methods which will perform during the lifecycle of fragment in android applications.

Method	**Description**
onAttach()	It is called when the fragment has been associated with an activity.
onCreate()	It is used to initialize the fragment.
onCreteView()	It is used to create a view hierarchy associated with the fragment.
onActivityCreated()	It is called when the fragment activity has been created and the fragment view hierarchy instantiated.
onStart()	It is used to make the fragment visible.
onResume()	It is used to make the fragment visible in an activity.

onPause()	It is called when fragment is no longer visible and it indicates that the user is leaving the fragment.
onStop()	It is called to stop the fragment using the onStop() method.
onDestoryView()	The view hierarchy associated with the fragment is being removed after executing this method.
onDestroy()	It is called to perform a final clean up of the fragments state.
onDetach()	It is called immediately after the fragment disassociated from the activity.

Example

Following is the example of creating two fragments, two buttons and showing the respective fragment when click on button in android application.

Now we need to create our own custom fragment layout files (listitems_info.xml**,** details_info.xml) in \res\layout path to display those fragments in the main layout for that right-click on your layout folder a Go to New a select Layout resource file and give name as listitems_info.xml**.**

Once we create a new file listitems_info.xml, open it and write the code like as shown below
Listitems_info.xml

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

details_info.xml

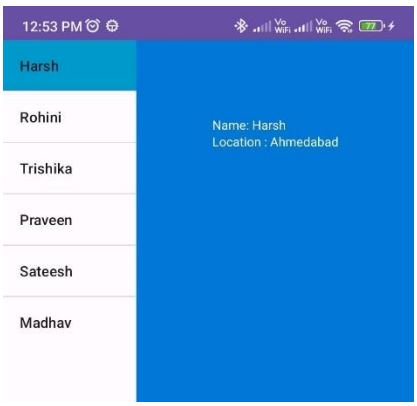
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#0079D6">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#ffffff"
        android:layout_marginTop="200px"
        android:layout_marginLeft="200px"
        android:id="@+id/Name"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="200px"
        android:textColor="#ffffff"
        android:id="@+id/Location"/>
</LinearLayout>
```

DetailsFragment.java

```
import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class DetailsFragment extends Fragment {
    TextView name,location;
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        View view = inflater.inflate(R.layout.details_info, container, false);
        name = (TextView)view.findViewById(R.id.Name);
        location = (TextView)view.findViewById(R.id.Location);
        return view;
    }
    public void change(String uname, String ulocation){
        name.setText(uname);
        location.setText(ulocation);
    }
}
```

ListMenuFragment.java

```
package com.example.fragmentexample;

import android.app.ListFragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class ListMenuFragment extends ListFragment {
    String[] users = new String[] {
"Harsh","Rohini","Trishika","Praveen","Sateesh","Madhav" };
    String[] location = new
String[]{"Ahmedabad","Surat","Vadodara","Rajkot","Gandhinagar","Jamnagar"};

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        View view =inflater.inflate(R.layout.listitems_info, container, false);
        ArrayAdapter<String> adapter = new
ArrayAdapter<String>(getActivity(),android.R.layout.simple_list_item_1, users);
        setListAdapter(adapter);
        return view;
    }
    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        DetailsFragment txt =
(DetailsFragment)getFragmentManager().findFragmentById(R.id.fragment2);
        txt.change("Name: "+ users[position],"Location : "+ location[position]);
        getListView().setSelector(android.R.color.holo_blue_dark);
    }
}
```



Project: **FragmentExample**

Android Intent

In android, Intent is a messaging object which is used to request an action from another app components such as activities, services, broadcast receivers, and content providers.

Generally, in android**,** Intents will help us to maintain the communication between app components from the same application as well as with the components of other applications.

In android, Intents are the objects of **android.content.Intent** types and intents are mainly useful to perform the following things.

Component	Description
Starting an Activity	By sending an **Intent** object to startActivity() method we can start a new Activity or existing Activity to perform required things.
Starting a Service	By sending an **Intent** object to startService() method we can start a new Service or send required instructions to an existing Service.
Delivering a Broadcast	By sending an **Intent** object to sendBroadcast() method we can deliver our messages to other app broadcast receivers.

Building an Intent Object

Generally, in android Intent object contains the information required to determine which component to start and the information about the action to be performed by the recipient component.

The Intent object in android is having following characteristics to help the android system to understand which component should start.

Component Name

- It defines the name of the component to start and by using the component name android system will deliver intent to the specific app component defined by the component name. In case if we did not define component name then the android system will decide which component should receive intent based on other intent information such as action, data, etc.
- In android, we can specify the component name for intent by using a fully qualified class name of the target component and package name, for example, **com.example.simpleActivity**. We

can set the component name by using setComponent(), setClass(), setClassName() or by using the **Intent** constructor.

Action

It defines the name of the action to be performed to start an activity. The following are some of the common actions to start an activity.

Action	Description
ACTION_VIEW	We can use this action in intent with startActivity(), when we have information that activity can show to the user.
ACTION_SEND	We can use this action in intent with startActivity(), when we have some data that the user can share through another app such as an email app, social sharing app.

We can specify the action name of intent by using setAction() or with an **Intent** constructor.

Data

It specifies a type of data to an intent filter. When we create an intent, it's important to specify the type of data (MIME type) in addition to its URI. By specifying a MIME type of data, it helps the android system to decide which is the best component to receive our intent.

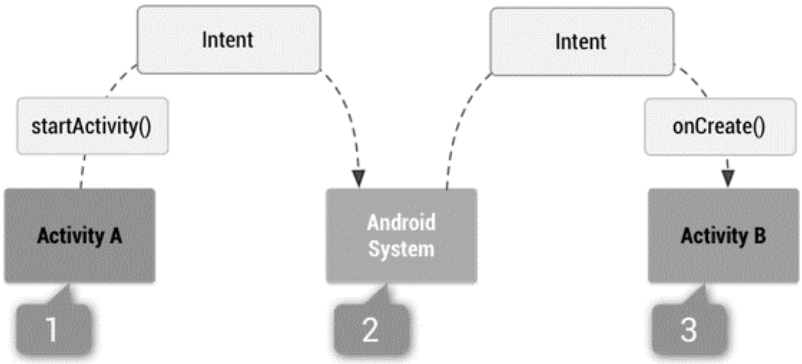
Category

- Generally, the android category is optional for intents and it specifies the additional information about the type of component that should handle an intent.
- We can specify a category for intent by using addCategory().
- The above properties (Component Name, Action, Data, and Category) will represent the characteristics of an intent. By using these properties, the android system will easily decide which app component to start.

Implicit Intent

In android, **Implicit** Intents will not specify any name of the component to start instead, it declares an action to perform and it allows a component from other apps to handle it. For example, by using implicit intents we can request another app to show the location details of the user or etc.

Following is the pictorial representation of how Implicit intents send a request to the android system to start another activity.



If you observe the above image Activity A creates an intent with the required action and sends it to an android system using the **startActivity()** method. The android system will search for an intent filter that matches the intent in all apps. Whenever the match found the system starts matching activity (**Activity B**) by invoking the **onCreate()** method.

In android when we create implicit intents, the android system will search for matching components by comparing the contents of intent with intent filters which defined in the **manifest** file of other apps on the device. If the matching component found, the system starts that component and sends it to the Intent object. In case, if multiple intent filters are matched then the system displays a dialog so that the user can pick which app to use.

In android, an **Intent Filter** is an expression in the app's **manifest** file and it is used to specify the type of intents that the component would like to receive. In case if we create an **Intent Filter** for activity, there is a possibility for other apps to start our activity by sending a certain type of intent otherwise the activity can be started only by an explicit intent.

Example for **implicit intent** in the android application.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.intentexample.MainActivity">
    <EditText
        android:layout_width="350dp"
        android:layout_height="wrap_content"
        android:id="@+id/urlText"
        android:layout_marginLeft="25dp"
        android:layout_marginTop="100dp"
        android:ems="10" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/btnNavigate"
        android:layout_below="@+id/urlText"
        android:text="Navigate"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

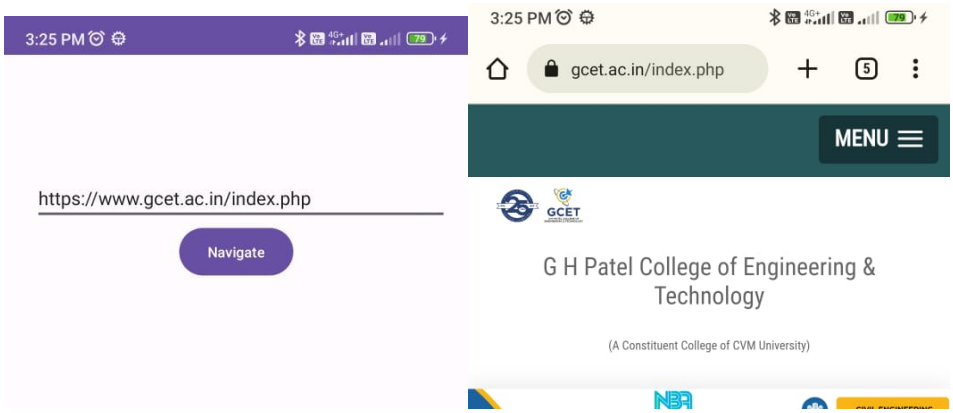
MainActiviy.java

```
package com.example.intentexample;

import android.content.Intent;
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
import android.view.View;
import android.widget.*;
import android.net.Uri;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final EditText editText = (EditText)findViewById(R.id.urlText);
        Button btn = (Button) findViewById(R.id.btnNavigate);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
            public void onClick(View v) {
                String url = editText.getText().toString();
                Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
                startActivity(intent);
            }
        });
    }
}
```

Give Following permission in AndroidMenfest.xml

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



Project: **IntentExample**

Android Explicit Intents with Examples

In android, Explicit intents explicitly specify the name of the component to be invoked by activity and we use explicit intents to start a component in our own app. For example, we can start a new activity in response to a user action using explicit intents.

By using explicit intent, we can send or share data/content from one activity to another activity based on our requirements. To create an Explicit Intent, we need to define the component name for an Intent object.

Here is the simple example of implementing an explicit intent in the android application. Here I have taken name and mobile in one activity and sending that information to another activity to display the result. For that we require two activity and two xml files.

Activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/nameText"
        android:layout_width="300dp"
        android:layout_height="50dp"
        android:layout_centerHorizontal="true"
```

```
        android:layout_marginTop="100dp"
        android:hint="Name" />

    <EditText
        android:id="@+id/mobileText"
        android:layout_width="300dp"
        android:layout_height="50dp"
        android:layout_below="@+id/nameText"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp"
        android:hint="Mobile" />

    <Button
        android:id="@+id/btnNext"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/mobileText"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp"
        android:text="Next" />
</RelativeLayout>
```

MainActivity.java

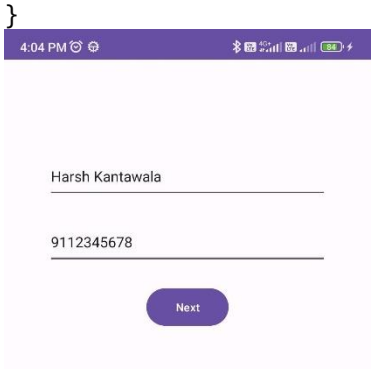
```
package com.example.explicitintent;

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

public class MainActivity extends AppCompatActivity {

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

        final EditText nameEditText = findViewById(R.id.nameText);
        final EditText mobileEditText = findViewById(R.id.mobileText);
        Button nextButton = findViewById(R.id.btnNext);

        nextButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String name = nameEditText.getText().toString();
                String mobile = mobileEditText.getText().toString();

                Intent intent = new Intent(MainActivity.this,
ResultActivity.class);
                intent.putExtra("name", name);
                intent.putExtra("mobile", mobile);
                startActivity(intent);
            }
        });
    }
}
```

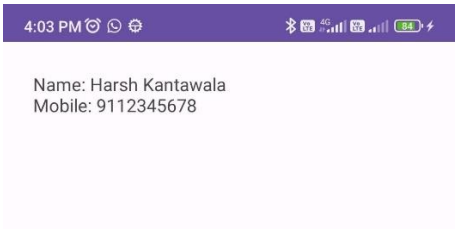
result.xml

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

    <TextView
        android:id="@+id/resultText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:layout_margin="10dp"
        />
</LinearLayout>
```

result_activity.java

```
package com.example.explicitintent;

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

public class ResultActivity extends AppCompatActivity {

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

        TextView resultTextView = findViewById(R.id.resultText);

        String name = getIntent().getStringExtra("name");
        String mobile = getIntent().getStringExtra("mobile");

        String result = "Name: " + name + "\nMobile: " + mobile;
        resultTextView.setText(result);
    }
}
```





Project: **ExplicitIntent**