Chapter 3

3. Activity, Fragments, Intent and Services

3.1. Activity

In android, **Activity** represents a single screen with a user interface (UI) of an application and it will act as an entry point for users to interact with an app.

Android apps can contain multiple screens and each screen of the application is 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 a 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 provides a screens to perform a tasks like add a new contact and search for the contacts. All these activities in 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.

To use activities in our application we need to define activities with required attributes in manifest file (**AndroidMainfest.xml**) like as shown below

```
 <?xml version="1.0" encoding="utf-8"?>
<manifest …..>
    <application …..>
        <activity android:name=".MainActivity" >
          …….
          …….
        </activity>
     …….
</application>
</manifest>
```

The activity attribute **android:name** will represent the name of class and we can also add multiple attributes like icon, label, theme, permissions, etc. to an activity element based on our requirements.

In android application, activities can be implemented as a subclass of **Activity** class like as shown below.

```
import android.app.Activity;
import android.os.Bundle;
public class MainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Android Activity Lifecycle

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

- ✓ onCreate() — Called when the activity is first created
- ✓ onStart() — Called when the activity becomes visible to the user
- ✓ onResume() — Called when the activity starts interacting with the user
- ✓ onPause() — Called when the current activity is being paused and the previous activity is being resumed
- ✓ onStop() — Called when the activity is no longer visible to the user
- ✓ onDestroy() — Called before the activity is destroyed by the system (either manually or by the system to conserve memory)
- ✓ onRestart() — Called when the activity has been stopped and is restarting again

By default, the activity created for you contains the onCreate() event. Within this event handler is the code that helps to display the UI elements of your screen. The following figure shows life cycle of an activity.

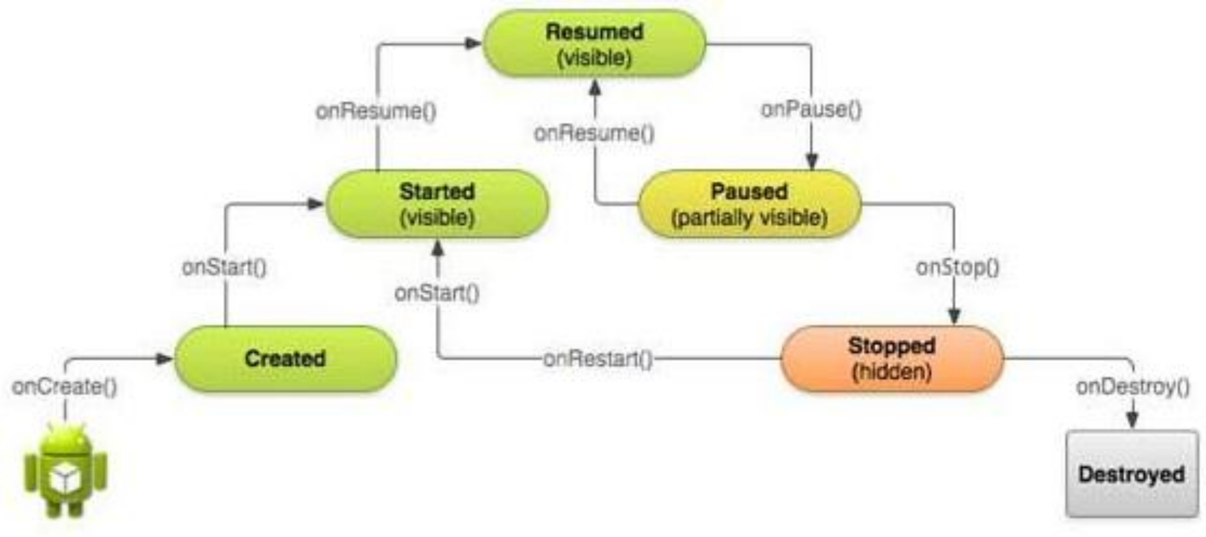

Figure 3.1 Life cycle of an activity

The best way to understand the various stages of an activity is to create a new project, implement the various events, and then subject the activity to various user interactions.

Understanding the Life Cycle of an Activity

1. Create a new Android project and name it **Activity101**.
2. In the Activity101Activity.java file, add the following statements in bold:

```
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
public class Activity101Activity extends Activity {
    String tag = “Lifecycle”;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Log.d(tag, “In the onCreate() event”);
    }
       public void onStart(){
                super.onStart();
                Log.d(tag, “In the onStart() event”);   }
       public void onRestart() {
                super.onRestart();
                Log.d(tag, “In the onRestart() event”);    }
       public void onResume(){
                super.onResume();
                Log.d(tag, “In the onResume() event”);    }
       public void onPause(){
                super.onPause();
                Log.d(tag, “In the onPause() event”);    }
       public void onStop(){
                super.onStop();
                Log.d(tag, “In the onStop() event”);    }
       public void onDestroy() {
                super.onDestroy();
                Log.d(tag, “In the onDestroy() event”);    }
}
```

3. Press F11 to debug the application on the Android emulator.
4. When the activity is first loaded, you should see something very similar to the following in the LogCat window (click the Debug perspective):

```
11-16 06:25:59.396: D/Lifecycle(559): In the onCreate() event
11-16 06:25:59.396: D/Lifecycle(559): In the onStart() event
11-16 06:25:59.396: D/Lifecycle(559): In the onResume() event
```

5. If you click the Back button on the Android emulator, the following is printed:

```
11-16 06:29:26.665: D/Lifecycle(559): In the onPause() event
11-16 06:29:28.465: D/Lifecycle(559): In the onStop() event
11-16 06:29:28.465: D/Lifecycle(559): In the onDestroy() event
```

6. Click the Home button and hold it there. Click the Activities icon and observe the following:

```
11-16 06:31:08.905: D/Lifecycle(559): In the onCreate() event
11-16 06:31:08.905: D/Lifecycle(559): In the onStart() event
11-16 06:31:08.925: D/Lifecycle(559): In the onResume() event
```

7. Click the Phone button on the Android emulator so that the activity is pushed to the background. Observe the output in the LogCat window:

```
11-16 06:32:00.585: D/Lifecycle(559): In the onPause() event
11-16 06:32:05.015: D/Lifecycle(559): In the onStop() event
```

8. Notice that the onDestroy() event is not called, indicating that the activity is still in memory. Exit the phone dialer by clicking the Back button. The activity is now visible again. Observe the output in the LogCat window:

```
11-16 06:32:50.515: D/Lifecycle(559): In the onRestart() event
11-16 06:32:50.515: D/Lifecycle(559): In the onStart() event
11-16 06:32:50.515: D/Lifecycle(559): In the onResume() event
```

The onRestart() event is now fi red, followed by the onStart() and onResume() methods.

How It Works

As you can see from this simple example, an activity is destroyed when you click the Back button. This is crucial to know, as whatever state the activity is currently in will be lost; hence, you need to write additional code in your activity to preserve its state when it is destroyed. At this point, note that the onPause() method is called in both scenarios — when an activity is sent to the background, as well as when it is killed when the user presses the Back button.

When an activity is started, the onStart() and onResume()methods are always called, regardless of whether the activity is restored from the background or newly created. When an activity is created for the first time, the onCreate() method is called.

From the preceding example, you can derive the following guidelines:

- ✓ Use the onCreate() method to create and instantiate the objects that you will be using in your application.
- ✓ Use the onResume() method to start any services or code that needs to run while your activity is in the foreground.
- ✓ Use the onPause() method to stop any services or code that does not need to run when your activity is not in the foreground.
- ✓ Use the onDestroy() method to free up resources before your activity is destroyed.

Note: *Even if an application has only one activity and the activity is killed, the application will still be running in memory.*

3.2. Fragments

In the previous section you learned what an activity is and how to use it. In a small-screen device (such as a smartphone), an activity typically fills the entire screen, displaying the various views that make up the user interface of an application. The activity is essentially a container for views.

However, when an activity is displayed in a large-screen device, such as on a tablet, it is somewhat out of place. Because the screen is much bigger, all the views in an activity must be arranged to make full use of the increased space, resulting in complex changes to the view hierarchy. A better approach is to have "mini-activities," each containing its own set of views. During runtime, an activity can contain one or more of these mini-activities, depending on the screen orientation in which the device is held. In Android 3.0 and later, these mini-activities are known as **fragments.**

Think of a fragment as another form of activity. You create fragments to contain views, just like activities. Fragments are always embedded in an activity. For example, Figure 3.2.1 shows two fragments. Fragment 1 might contain a ListView showing a list of book titles. Fragment 2 might contain some TextViews and ImageViews showing some text and images.

Now imagine the application is running on an Android tablet in portrait mode (or on an Android smartphone). In this case, Fragment 1 may be embedded in one activity, while Fragment 2 may be embedded in another activity (see Figure 3.2.2). When users select an item in the list in Fragment 1,Activity 2 will be started.

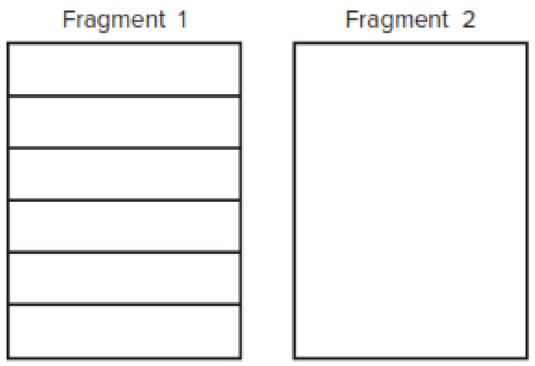

Figure 3.2.1 Fragments in an activity

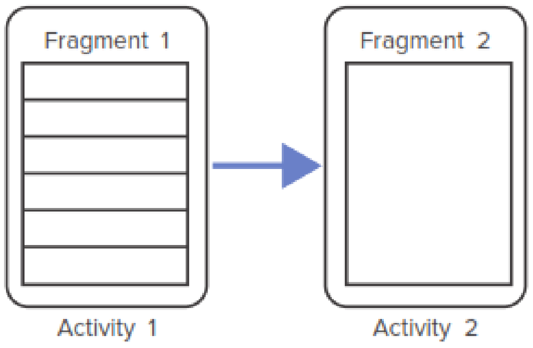

Figure 3.2.2 Fragments in different activities

If the application is now displayed in a tablet in landscape mode, both fragments can be embedded within a single activity, as shown in Figure 3.2.3.

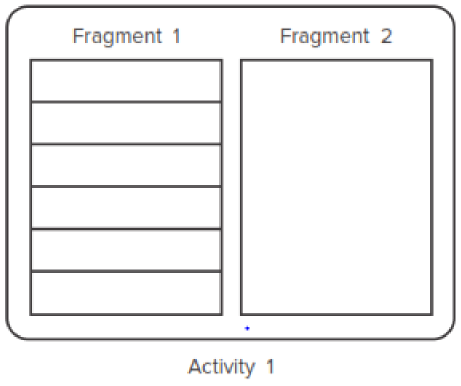

Figure 3.2.3 Fragments in landscape mode

From this discussion, it becomes apparent that fragments present a versatile way in which you can create the user interface of an Android application.

Fragments form the atomic unit of your user interface, and they can be dynamically added (or removed) to activities in order to create the best user experience possible for the target device.

Android Fragment Life Cycle

The following is pictorial representation of android fragment life cycle while its activity is running.

The 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.
onCreateView()	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 onStop() method.
onDestoryView()	The view hierarchy which 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 a two fragments, two buttons and showing the respective fragment when click on button in android application.

1. Create a new android application using android studio and give name as **Fragments**.
2. 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 main layout for that right click on your layout folder → Go to **New** → 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>
```

3. Same way create another file **details_info.xml**, open it and write the code like as shown below

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>
```

4. Now we need to create our own custom fragment class files (**ListMenuFragment.java**, **DetailsFragment.java**) in **\java\com.example.fragmentsexample** path to bind and display data in fragments for that right click on your application folder →Go to New → select **Java Class** and give name as **DetailsFragment.java**. Once we create a new file **DetailsFragment.java**, open it and write the code like as shown below

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);
    }
}
```

5. If you observe above code we extended class with Fragment and used **LayoutInflater** to show the details of fragment. We defined a function **change()** to change the text in textview. Same way create another file **ListMenuFragment.java**, open it and write the code like as shown below

   ListMenuFragment.java

```
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[] { "Abebe","Samson","Kirubel","Helen" };
    String[] location = new String[]{"Gondar","Bahir Dar","Addis Ababa","Debre
Markos"};
    @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);
    }
}
```

6. If you observe above code we extended our class using **ListFragment** and we defined two arrays of strings users, location which contains names and locations. We defined **onListItemClick** event to update the name and location in **DetailsFragment** based on the list item which we clicked. Now we need to display our fragments horizontally side by side in main layout for that open activity_main.xml file and write code like as shown below

   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="horizontal"
    tools:context="com.example.fragmentsexample.MainActivity">
    <fragment
        android:layout_height="match_parent"
        android:layout_width="350px"
        class="com.example.fragmentsexample.ListMenuFragment"
        android:id="@+id/fragment"/>
    <fragment
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        class="com.example.fragmentsexample.DetailsFragment"
        android:id="@+id/fragment2"/>
</LinearLayout>
```

We are not going to make any modifications for our main activity file (**MainActivity.java**) and manifest file (**AndroidMainfest.xml**).

3.3. Intents

Android uses Intent for communicating between the components (such as activities, services, broadcast receivers and content providers) of an Application and also from one application to another application.
For example: Intent facilitates you to redirect your activity to another activity on occurrence of any event. By calling, startActivity() you can perform this task.

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

In the above example, foreground activity is getting redirected to another activity i.e. SecondActivity.java. getApplicationContext() returns the context for your foreground activity.

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

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

Types of Intents:

There are two types of intents: **Explicit Intent** and **Implicit Intent**

Explicit Intent:

Explicit Intents are used to connect the application internally. In Explicit we use the name of component which will be affected by Intent. For Example: If we know class name then we can navigate the app from One Activity to another activity using Intent. In the similar way we can start a service to download a file in background process. Explicit Intent works internally within an application to perform navigation and data transfer. The below given code snippet will help you understand the concept of Explicit Intents

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

Here **SecondActivity** is the JAVA class name where the activity will now be navigated.

Implicit Intent:

In Implicit Intents we do need to specify the name of the component. We just specify the Action which has to be performed and further this action is handled by the component of another application. The basic example of implicit Intent is to open any web page.

Let’s take an example to understand Implicit Intents more clearly. We have to open a website using intent in your application. See the code snippet given below

```
Intent intentObj = new Intent(Intent.ACTION_VIEW);
intentObj.setData(Uri.parse("https://www.google.com"));
```

```
startActivity(intentObj);
```

Unlike Explicit Intent you do not use any class name to pass through Intent(). In this example we have just specified an action. Now when we run this code then Android will automatically start your web browser and it will open Google home page.

Example:

The example will show you both implicit and explicit Intent together. Let's implement Intent for a very basic use. In the below example we will Navigate from one Activity to another and open a web homepage of Google using Intent.

Create a project in Android Studio and named it "Intents". Make an activity, which would consists Java file; **MainActivity.java** and an xml file for User interface which would be **activity_main.xml**

Step 1: Let's design the UI of **activity_main.xml**:

- ✓ First design the text view displaying basic details of the App
- ✓ Second design the two button of Explicit Intent Example and Implicit Intent Example

Below is the complete code of **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"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="If you click on Explicit example we will navigate to second
        activity within App and if you click on Implicit example Google Homepage will
        open in Browser"
        android:id="@+id/textView2"
        android:clickable="false"
        android:layout_alignParentTop="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="42dp"
        android:background="#3e7d02"
        android:textColor="#ffffff" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Explicit Intent Example"
        android:id="@+id/explicit_Intent"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="147dp" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Implicit Intent Example"
        android:id="@+id/implicit_Intent"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

Step 2: Design the UI of second activity **activity_second.xml**

Now let's design UI of another activity where user will navigate after he click on Explicit Example button. Go to layout folder, create a new activity and name it activity_second.xml.

- ✓ In this activity we will simply use TextView to tell user he is now on second activity.

Below is the complete code of **activity_second.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"
android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:background="#CCEEAA"
    tools:context="com.example.android.intents.SecondActivity">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="This is Second Activity"
        android:id="@+id/textView"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

Step 3: Implement onClick event for Implicit and Explicit Button inside **MainActivity.java**

Now we will use setOnClickListener() method to implement OnClick event on both the button. Implicit button will open AbhiAndroid.com homepage in browser and Explicit button will move to **SecondActivity.java**.

Below is the complete code of **MainActivity**.java

```
import android.content.Intent;
import android.net.Uri;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
public class MainActivity extends AppCompatActivity {
    Button explicit_btn, implicit_btn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        explicit_btn = (Button)findViewById(R.id.explicit_Intent);
        implicit_btn = (Button) findViewById(R.id.implicit_Intent);
        //implement Onclick event for Explicit Intent
        explicit_btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new  Intent(getBaseContext(), SecondActivity.class);
                startActivity(intent);
            }
        });
        //implement onClick event for Implicit Intent
        implicit_btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(Intent.ACTION_VIEW);
                intent.setData(Uri.parse("https://www.google.com"));
                startActivity(intent);
            }
        });
    } }
```

Step 4: Create A New JAVA class name **SecondActivity**

Now we need to create another SecondActivity.java which will simply open the layout of **activity_second.xml**. Also we will use Toast to display message that he is on second activity.

Below is the complete code of SecondActivity.java:

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Toast;
public class SecondActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
        Toast.makeText(getApplicationContext(), "We are moved to second Activity”,
Toast.LENGTH_LONG).show();
    }
}
```

Step 5: Manifest file:

Make sure Manifest file has both the **MainActivity** and **SecondActivity** listed it. Also here **MainActivity** is our main activity which will be launched first. So make sure intent-filter is correctly added just below **MainActivity**.

Below is the code of Manifest file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.intents" >
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="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>
        <activity android:name=".SecondActivity" >
        </activity>
    </application>
</manifest>
```

Testing:

- ✓ Now run the above program in your Emulator.
- ✓ First Click on Explicit Intent Example. The SecondActivity will be open within the App.
- ✓ Now go back in Emulator and click on Implicit Intent Example. The google.com homepage will open in Browser (make sure you have internet)

Benefits of Intent in Android

Android uses Intents for facilitating communication between its components like Activities, Services and Broadcast Receivers.

For an Activity: Every screen in Android application represents an activity. To start a new activity you need to pass an Intent object to startActivity() method. This Intent object helps to start a new activity and passing data to the second activity.

For Services: Services work in background of an Android application and it does not require any user Interface. Intents could be used to start a Service that performs one-time task(for example: Downloading some file) or for starting a Service you need to pass Intent to startService() method.

For Broadcast Receivers: There are various messages that an app receives; these messages are called as Broadcast Receivers. (For example, a broadcast message could be initiated to intimate that the file downloading is completed and ready to use). Android system initiates some broadcast message on several events, such as System Reboot, Low Battery warning message etc.

For Android Applications: Whenever you need to navigate to another activity of your app or you need to send some information to next activity then we can always prefer to Intents for doing so.

3.4. Android Services

Android service is a component that is *used to perform operations on the background* such as playing music, handle network transactions, interacting content providers etc. It doesn't have any UI (user interface). The service runs in the background indefinitely even if application is destroyed. Moreover, service can be bounded by a component to perform interactivity and inter process communication (IPC). The android.app.Service is subclass of ContextWrapper class.

Life Cycle of Android Service

There can be two forms of a service. The lifecycle of service can follow two different paths: started or bound.

- ✓ Started
- ✓ Bound

1) **Started Service**

A service is started when component (like activity) calls **startService()** method, now it runs in the background indefinitely. It is stopped by **stopService()** method. The service can stop itself by calling the **stopSelf()** method.

2) **Bound Service**

A service is bound when another component (e.g. client) calls **bindService()** method. The client can unbind the service by calling the **unbindService()** method. The service cannot be stopped until all clients unbind the service.

Example:

Let's see the example of service in android that plays an audio in the background. Audio will not be stopped even if you switch to another activity. To stop the audio, you need to stop the service.

1. Create an android project and design the **activity_main.xml as follows**

```
<?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="example.com.androidservice.MainActivity">
      <Button
        android:id="@+id/buttonStart"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="74dp"
        android:text="Start Service" />
      <Button
        android:id="@+id/buttonStop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="Stop Service" />
      <Button
        android:id="@+id/buttonNext"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="63dp"
        android:text="Next Page" />
</RelativeLayout>
```

2. Create the **activity_next.xml** file

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/androi
d"
    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="example.com.androidservice.NextPage">
      <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="200dp"
        android:text="Next Page"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

3. Now create the service implementation class by inheriting the Service class and overriding its callback methods. **MyService.java**

```
import android.app.Service;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.IBinder;
import android.support.annotation.Nullable;
import android.widget.Toast;
  public class MyService extends Service {
    MediaPlayer myPlayer;
    @Nullable
    @Override
    public IBinder onBind(Intent intent) {          return null;      }
    @Override
    public void onCreate() {
        Toast.makeText(this, "Service Created", Toast.LENGTH_LONG).show();
        myPlayer = MediaPlayer.create(this, R.raw.sun);
        myPlayer.setLooping(false); // Set looping
    }
    @Override
    public void onStart(Intent intent, int startid) {
        Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
        myPlayer.start();
    }
    @Override
    public void onDestroy() {
        Toast.makeText(this, "Service Stopped", Toast.LENGTH_LONG).show();
        myPlayer.stop();
    }  }
```

4. Now create the MainActivity class to perform event handling. Here, we are writing the code to start and stop service. Additionally, calling the second activity on buttonNext. **MainActivity.java**

```
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
  public class MainActivity extends AppCompatActivity implements View.OnClickListener
{
    Button buttonStart, buttonStop,buttonNext;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        buttonStart = findViewById(R.id.buttonStart);
        buttonStop = findViewById(R.id.buttonStop);
        buttonNext =  findViewById(R.id.buttonNext);
        buttonStart.setOnClickListener(this);
        buttonStop.setOnClickListener(this);
        buttonNext.setOnClickListener(this);
      }
    public void onClick(View src) {
        switch (src.getId()) {
            case R.id.buttonStart:
                startService(new Intent(this, MyService.class));
                break;
            case R.id.buttonStop:
                stopService(new Intent(this, MyService.class));
                break;
            case R.id.buttonNext:
                Intent intent=new Intent(this,NextPage.class);
                startActivity(intent);
                break;
        }
    }
}
```

5. Now, create another activity.

NextPage.java

```
 import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
public class NextPage extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_next);
    }
}
```

6. Finally, declare the service in the manifest file. Let's see the complete AndroidManifest.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="example.com.androidservice">
      <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="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>
        <activity android:name=".NextPage"></activity>
        <service
            android:name=".MyService"
            android:enabled="true" />
    </application>
  </manifest>
```