What is an Intent in Android?

An intent is to perform an action on the screen. It is mostly used to start activity, send broadcast receiver,start services and send message between two activities. There are two intents available in android as Implicit Intents and Explicit Intents. Here is a sample example to start new activity with old activity.

Step 1 − Create a new project in Android Studio,go to File ⇒ New Project and fill all required details to create a new project.

Step 2 − Add the following code to res/layout/activity_main.xml. (First Activity layout)

```
<?xml version = "1.0" encoding = "utf-8"?>
<android.support.constraint.ConstraintLayout
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">
<LinearLayout
   android:layout_width = "match_parent"
   android:layout_height = "match_parent"
   android:gravity = "center"
   android:orientation = "vertical">
   <Button
      android:layout_width = "wrap_content"
      android:layout_height = "wrap_content"
      android:text = "Send to another activitys"
      android:id = "@+id/send"/>
</LinearLayout>
</android.support.constraint.ConstraintLayout>
```

Step 3 − Create a new layout in res/layout/ folder and add the following code to res/layout/activity_main.xml. (Second Activity layout)

```
<?xml version = "1.0" encoding = "utf-8"?>
<android.support.constraint.ConstraintLayout    xmlns:android    =
"http://schemas.android.com/apk/res/android"
   xmlns:app = "http://schemas.android.com/apk/res-auto"
   xmlns:tools = "http://schemas.android.com/tools"
   android:layout_width = "match_parent"
   android:layout_height = "match_parent"
   android:layout_centerInParent = "true"
   android:layout_centerHorizontal = "true"
   tools:context = ".SecondActivity">
   <TextView
      android:id = "@+id/data"
      android:textSize = "20sp"
      android:layout_width = "wrap_content"
      android:layout_height = "wrap_content" />
</android.support.constraint.ConstraintLayout>
```

Step 4 − Add the following code to src/MainActivity.java (First activity)

```
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      Button send = findViewById(R.id.send);
      send.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
                  Intent  send  =  new  Intent(MainActivity.this,
SecondActivity.class);
            startActivity(send);
         }
      });
   }
}
```

In the above activity we are starting new activity using startActivity(). To start activity, we need to create new intent and we have to pass current activity and new activity as shown below.

```
Intent send = new Intent(MainActivity.this, SecondActivity.class);
startActivity(send);
```

Step 4 − Create a new activity and add the following code to src/SecondActivity.java (Second Activity)

```
package com.example.andy.myapplication;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;
public class SecondActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_second);
      TextView data=findViewById(R.id.data);
      data.setText("This is second activity");
   }
}
```

Step5 − Add the following code to AndroidManifest.xml.

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

```
   <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 = ".SecondActivity"></activity>
   </application>
</manifest>
```

In the above code, we have declare MainActivity and SecondActivity as shown below.

```
<activity android:name = ".SecondActivity"></activity>
<activity android:name = ".MainActivity""></activity>
```

Let's try to run your application. I assume you have connected your actual Android Mobile device with your computer. To run the app from android studio, open one of your project's activity files and click Run icon from the toolbar. Select your mobile device as an option and then check your mobile device which will display your default screen.

Now click on Button to start new activity as shown below.

Intent Filters

You have seen how an Intent has been used to call an another activity. Android OS uses filters to pinpoint the set of Activities, Services, and Broadcast receivers that can handle the Intent with help of specified set of action, categories, data scheme associated with an Intent. You will use **<intent-filter>** element in the manifest file to list down actions, categories and data types associated with any activity, service, or broadcast receiver.

Following is an example of a part of **AndroidManifest.xml** file to specify an activity **com.example.My Application.CustomActivity** which can be invoked by either of the two mentioned actions, one category, and one data −

```
<activity android:name=".CustomActivity"
   android:label="@string/app_name">

   <intent-filter>
      <action android:name="android.intent.action.VIEW" />
      <action android:name="com.example.My Application.LAUNCH" />
      <category android:name="android.intent.category.DEFAULT" />
      <data android:scheme="http" />
   </intent-filter>

</activity>
```

Once this activity is defined along with above mentioned filters, other activities will be able to invoke this activity using either the **android.intent.action.VIEW**, or using the **com.example.My Application.LAUNCH** action provided their category is **android.intent.category.DEFAULT**.

The **<data>** element specifies the data type expected by the activity to be called and for above example our custom activity expects the data to start with the "http://"

There may be a situation that an intent can pass through the filters of more than one activity or service, the user may be asked which component to activate. An exception is raised if no target can be found.

There are following test Android checks before invoking an activity −

- A filter <intent-filter> may list more than one action as shown above but this list cannot be empty; a filter must contain at least one <action> element, otherwise it will block all intents. If more than one actions are mentioned then Android tries to match one of the mentioned actions before invoking the activity.
- A filter <intent-filter> may list zero, one or more than one categories. if there is no category mentioned then Android always pass this test but if more than one categories are mentioned then for an intent to pass the category test, every category in the Intent object must match a category in the filter.
- Each <data> element can specify a URI and a data type (MIME media type). There are separate attributes like **scheme, host, port**, and **path** for each part of the URI. An Intent object that contains both a URI and a data type passes the data type part of the test only if its type matches a type listed in the filter.

Example

Following example is a modification of the above example. Here we will see how Android resolves conflict if one intent is invoking two activities defined in , next how to invoke a custom activity using a filter and third one is an exception case if Android does not file appropriate activity defined for an intent.

Step	Description
1	You will use android studio to create an Android application and name it as *My Application* under a package *com.example.tutorialspoint7.myapplication;*.
2	Modify *src/Main/Java/MainActivity.java* file and add the code to define three listeners corresponding to three buttons defined in layout file.
3	Add a new *src/Main/Java/CustomActivity.java* file to have one custom activity which will be invoked by different intents.
4	Modify layout XML file *res/layout/activity_main.xml* to add three buttons in linear layout.
5	Add one layout XML file *res/layout/custom_view.xml* to add a simple <TextView> to show the passed data through intent.
6	Modify *AndroidManifest.xml* to add <intent-filter> to define rules for your intent to invoke custom activity.

7	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **src/MainActivity.java**.

```
package com.example.tutorialspoint7.myapplication;

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 b1,b2,b3;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      b1=(Button)findViewById(R.id.button);
      b1.setOnClickListener(new View.OnClickListener() {

         @Override
         public void onClick(View v) {
            Intent i = new
Intent(android.content.Intent.ACTION_VIEW,
               Uri.parse("http://www.example.com"));
            startActivity(i);
         }
      });

      b2 = (Button)findViewById(R.id.button2);
      b2.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            Intent i = new Intent("com.example.
               tutorialspoint7.myapplication.
                  LAUNCH",Uri.parse("http://www.example.com"));
            startActivity(i);
         }
      });

      b3 = (Button)findViewById(R.id.button3);
      b3.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            Intent i = new Intent("com.example.
               My Application.LAUNCH",
                  Uri.parse("https://www.example.com"));
            startActivity(i);
         }
```

```
        });
    }
}
```

Following is the content of the modified main activity file **src/com.example.My Application/CustomActivity.java**.

```
package com.example.tutorialspoint7.myapplication;

import android.app.Activity;
import android.net.Uri;
import android.os.Bundle;
import android.widget.TextView;

/**
 * Created by TutorialsPoint7 on 8/23/2016.
 */
public class CustomActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.custom_view);
        TextView label = (TextView) findViewById(R.id.show_data);
        Uri url = getIntent().getData();
        label.setText(url.toString());
    }
}
```

Following will be the content of **res/layout/activity_main.xml** file −

```
<?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:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"

tools:context="com.example.tutorialspoint7.myapplication.MainActi
vity">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Intent Example"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:textSize="30dp" />

    <TextView
```

```
      android:id="@+id/textView2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Tutorials point"
      android:textColor="#ff87ff09"
      android:textSize="30dp"
      android:layout_below="@+id/textView1"
      android:layout_centerHorizontal="true" />

   <ImageButton
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/imageButton"
      android:src="@drawable/abc"
      android:layout_below="@+id/textView2"
      android:layout_centerHorizontal="true" />

   <EditText
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/editText"
      android:layout_below="@+id/imageButton"
      android:layout_alignRight="@+id/imageButton"
      android:layout_alignEnd="@+id/imageButton" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Start Browser"
      android:id="@+id/button"
      android:layout_alignTop="@+id/editText"
      android:layout_alignLeft="@+id/imageButton"
      android:layout_alignStart="@+id/imageButton"
      android:layout_alignEnd="@+id/imageButton" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Start browsing with launch action"
      android:id="@+id/button2"
      android:layout_below="@+id/button"
      android:layout_alignLeft="@+id/button"
      android:layout_alignStart="@+id/button"
      android:layout_alignEnd="@+id/button" />
   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Exceptional condition"
      android:id="@+id/button3"
      android:layout_below="@+id/button2"
      android:layout_alignLeft="@+id/button2"
      android:layout_alignStart="@+id/button2"
      android:layout_toStartOf="@+id/editText"
```

```
      android:layout_alignParentEnd="true" />
</RelativeLayout>
```

Following will be the content of **res/layout/custom_view.xml** file −

```
<?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">
   <TextView android:id="@+id/show_data"
      android:layout_width="fill_parent"
      android:layout_height="400dp"/>
</LinearLayout>
```

Following will be the content of **res/values/strings.xml** to define two new constants −

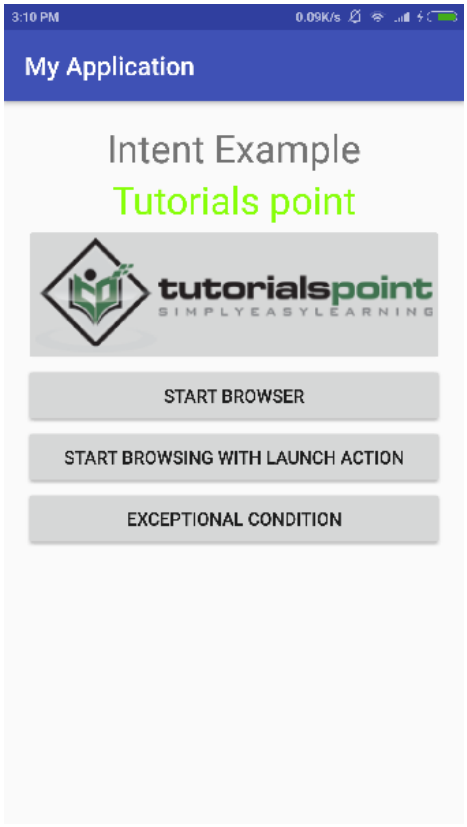
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">My Application</string>
</resources>
```

Following is the default content of **AndroidManifest.xml** −

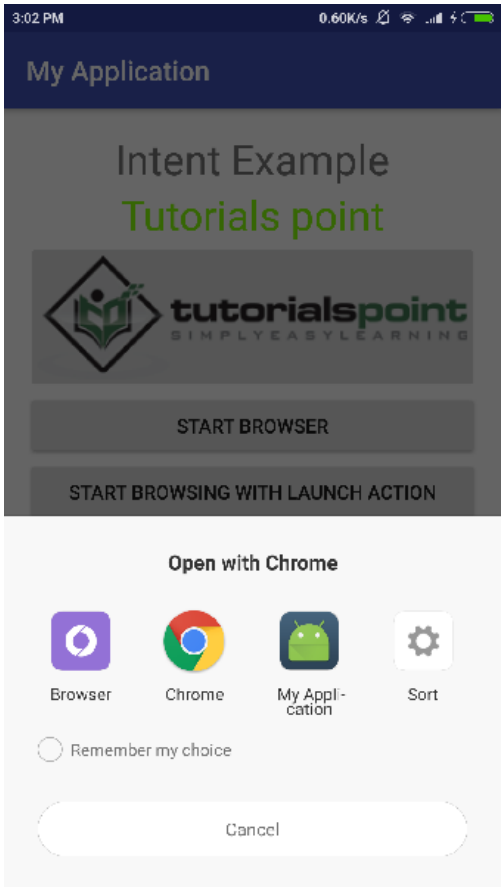
```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.tutorialspoint7.myapplication">

   <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="com.example.tutorialspoint7.myapplication.CustomAct
ivity">

         <intent-filter>
            <action android:name = "android.intent.action.VIEW"
/>
```

```
            <action android:name =
"com.example.tutorialspoint7.myapplication.LAUNCH" />
            <category android:name =
"android.intent.category.DEFAULT" />
            <data android:scheme = "http" />
         </intent-filter>

      </activity>
   </application>

</manifest>
```

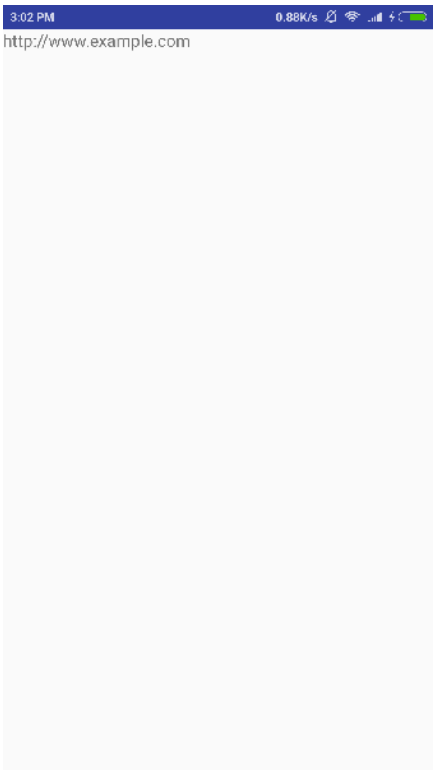
Let's try to run your **My Application** application. I assume you had created your **AVD** while doing environment setup. To run the app from Android Studio, open one of your project's activity files and click Run icon from the toolbar. Android Studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window −



Now let's start with first button "Start Browser with VIEW Action". Here we have defined our custom activity with a filter "android.intent.action.VIEW", and there is already one default activity against VIEW action defined by Android which is launching web browser, So android displays following two options to select the activity you want to launch.

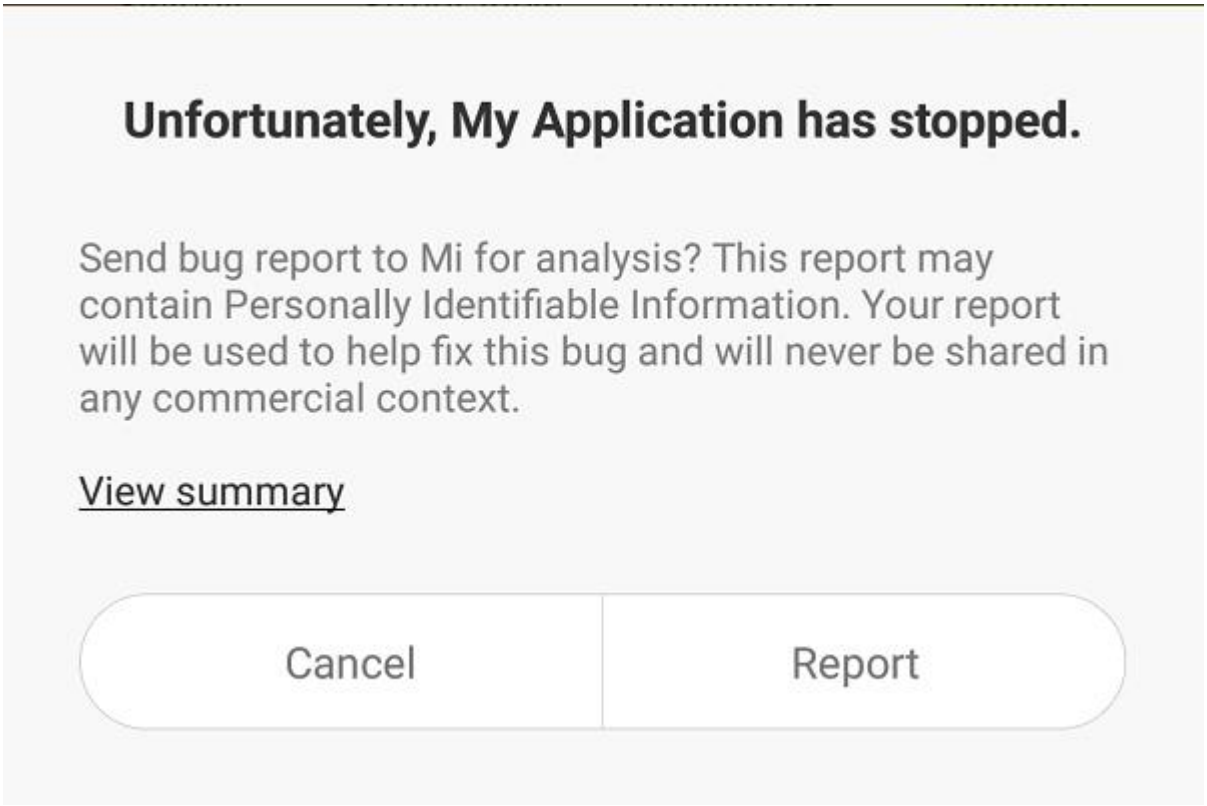


Now if you select Browser, then Android will launch web browser and open example.com website but if you select IndentDemo option then Android will launch CustomActivity which does nothing but just capture passed data and displays in a text view as follows −



Now go back using back button and click on "Start Browser with LAUNCH Action" button, here Android applies filter to choose define activity and it simply launch your custom activity

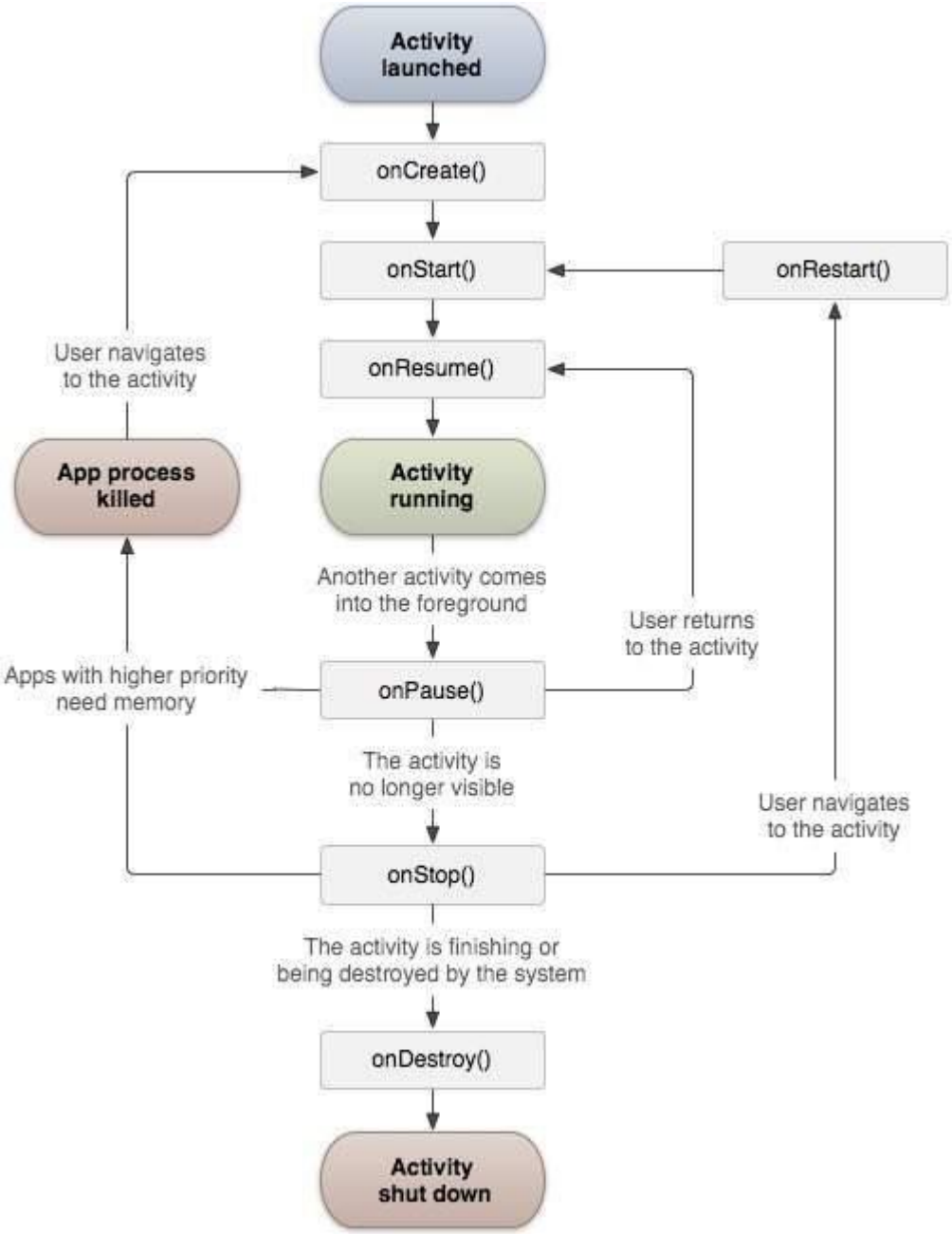
Again, go back using back button and click on "Exception Condition" button, here Android tries to find out a valid filter for the given intent but it does not find a valid activity defined because this time we have used data as **https** instead of **http** though we are giving a correct action, so Android raises an exception and shows following screen −



Activity Life Cycle

An activity represents a single screen with a user interface just like window or frame of Java.Android activity is the subclass of ContextThemeWrapper class.

If you have worked with C, C++ or Java programming language then you must have seen that your program starts from **main()** function. Very similar way, Android system initiates its program with in an **Activity** starting with a call on *onCreate()* callback method. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity as shown in the below Activity life cycle diagram: (*image courtesy : android.com*)



The Activity class defines the following call backs i.e. events. You don't need to implement all the callbacks methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.

Sr.No	Callback & Description
1	**onCreate()** This is the first callback and called when the activity is first created.
2	**onStart()** This callback is called when the activity becomes visible to the user.

3	**onResume()** This is called when the user starts interacting with the application.
4	**onPause()** The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.
5	**onStop()** This callback is called when the activity is no longer visible.
6	**onDestroy()** This callback is called before the activity is destroyed by the system.
7	**onRestart()** This callback is called when the activity restarts after stopping it.

Example

This example will take you through simple steps to show Android application activity life cycle. Follow the following steps to modify the Android application we created in *Hello World Example* chapter −

Step	Description
1	You will use Android studio to create an Android application and name it as *HelloWorld* under a package *com.example.helloworld* as explained in the *Hello World Example* chapter.
2	Modify main activity file *MainActivity.java* as explained below. Keep rest of the files unchanged.
3	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **src/com.example.helloworld/MainActivity.java**. This file includes each of the fundamental life cycle methods. The **Log.d()** method has been used to generate log messages −

```
package com.example.helloworld;

import android.os.Bundle;
import android.app.Activity;
```

```
import android.util.Log;

public class MainActivity extends Activity {
   String msg = "Android : ";

   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      Log.d(msg, "The onCreate() event");
   }

   /** Called when the activity is about to become visible. */
   @Override
   protected void onStart() {
      super.onStart();
      Log.d(msg, "The onStart() event");
   }

   /** Called when the activity has become visible. */
   @Override
   protected void onResume() {
      super.onResume();
      Log.d(msg, "The onResume() event");
   }

   /** Called when another activity is taking focus. */
   @Override
   protected void onPause() {
      super.onPause();
      Log.d(msg, "The onPause() event");
   }

   /** Called when the activity is no longer visible. */
   @Override
   protected void onStop() {
      super.onStop();
      Log.d(msg, "The onStop() event");
   }

   /** Called just before the activity is destroyed. */
   @Override
   public void onDestroy() {
      super.onDestroy();
      Log.d(msg, "The onDestroy() event");
   }
}
```

An activity class loads all the UI component using the XML file available in *res/layout* folder of the project. Following statement loads UI components from *res/layout/activity_main.xml file*:

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

An application can have one or more activities without any restrictions. Every activity you define for your application must be declared in your *AndroidManifest.xml* file and the main activity for your app must be declared in the manifest with an <intent-filter> that includes the MAIN action and LAUNCHER category as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.tutorialspoint7.myapplication">

    <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>
    </application>

</manifest>
```

If either the MAIN action or LAUNCHER category are not declared for one of your activities, then your app icon will not appear in the Home screen's list of apps.

Let's try to run our modified **Hello World!** application we just modified. I assume you had created your **AVD** while doing environment setup. To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar. Android studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display Emulator window and you should see following log messages in **LogCat** window in Android studio −

```
08-23 10:32:07.682 4480-4480/com.example.helloworld D/Android ::
The onCreate() event
08-23 10:32:07.683 4480-4480/com.example.helloworld D/Android ::
The onStart() event
08-23 10:32:07.685 4480-4480/com.example.helloworld D/Android ::
The onResume() event
```

Let us try to click lock screen button on the Android emulator and it will generate following events messages in **LogCat** window in android studio:

```
08-23 10:32:53.230 4480-4480/com.example.helloworld D/Android ::
The onPause() event
08-23 10:32:53.294 4480-4480/com.example.helloworld D/Android ::
The onStop() event
```

Let us again try to unlock your screen on the Android emulator and it will generate following events messages in **LogCat** window in Android studio:

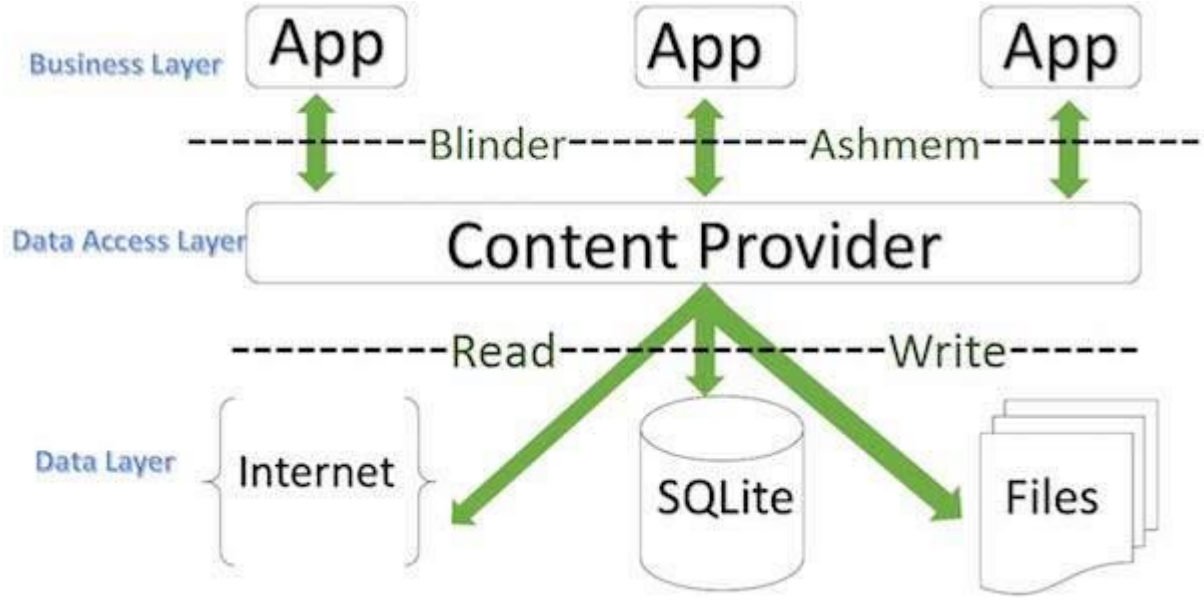
```
08-23 10:34:41.390 4480-4480/com.example.helloworld D/Android ::
The onStart() event
08-23 10:34:41.392 4480-4480/com.example.helloworld D/Android ::
The onResume() event
```

Next, let us again try to click Back button  on the Android emulator and it will generate following events messages in **LogCat** window in Android studio and this completes the Activity Life Cycle for an Android Application.

```
08-23 10:37:24.806 4480-4480/com.example.helloworld D/Android ::
The onPause() event
08-23 10:37:25.668 4480-4480/com.example.helloworld D/Android ::
The onStop() event
08-23 10:37:25.669 4480-4480/com.example.helloworld D/Android ::
The onDestroy() event
```

Content Providers

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the ContentResolver class. A content provider can use different ways to store its data and the data can be stored in a database, in files, or even over a network.



ContentProvider

sometimes it is required to share data across applications. This is where content providers become very useful.

Content providers let you centralize content in one place and have many different applications access it as needed. A content provider behaves very much like a database where you can query it, edit its content, as well as add or delete content using insert(), update(), delete(), and query() methods. In most cases this data is stored in an **SQlite** database.

A content provider is implemented as a subclass of **ContentProvider** class and must implement a standard set of APIs that enable other applications to perform transactions.

```
public class My Application extends  ContentProvider {
}
```

Content URIs

To query a content provider, you specify the query string in the form of a URI which has following format −

```
<prefix>://<authority>/<data_type>/<id>
```

Here is the detail of various parts of the URI −

Sr.No	Part & Description
1	**prefix** This is always set to content://
2	**authority** This specifies the name of the content provider, for example *contacts*, *browser* etc. For third-party content providers, this could be the fully qualified name, such as *com.tutorialspoint.statusprovider*
3	**data_type** This indicates the type of data that this particular provider provides. For example, if you are getting all the contacts from the *Contacts* content provider, then the data path would be *people* and URI would look like this*content://contacts/people*
4	**id** This specifies the specific record requested. For example, if you are looking for contact number 5 in the Contacts content provider then URI would look like this *content://contacts/people/5*.

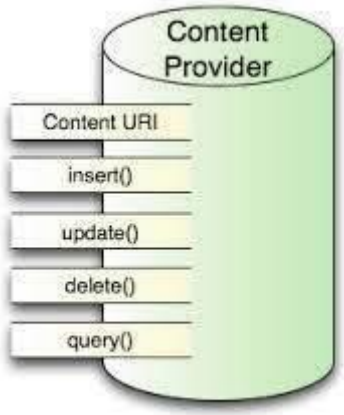
Create Content Provider

This involves number of simple steps to create your own content provider.

- First of all you need to create a Content Provider class that extends the *ContentProviderbaseclass.*

- Second, you need to define your content provider URI address which will be used to access the content.
- Next you will need to create your own database to keep the content. Usually, Android uses SQLite database and framework needs to override *onCreate()* method which will use SQLite Open Helper method to create or open the provider's database. When your application is launched, the *onCreate()* handler of each of its Content Providers is called on the main application thread.
- Next you will have to implement Content Provider queries to perform different database specific operations.
- Finally register your Content Provider in your activity file using <provider> tag.

Here is the list of methods which you need to override in Content Provider class to have your Content Provider working −



ContentProvider

- **onCreate()** This method is called when the provider is started.
- **query()** This method receives a request from a client. The result is returned as a Cursor object.
- **insert()**This method inserts a new record into the content provider.
- **delete()** This method deletes an existing record from the content provider.
- **update()** This method updates an existing record from the content provider.
- **getType()** This method returns the MIME type of the data at the given URI.

Example

This example will explain you how to create your own *ContentProvider*. So let's follow the following steps to similar to what we followed while creating *Hello World Example*−

Step	Description
1	You will use Android StudioIDE to create an Android application and name it as *My Application* under a package *com.example.MyApplication*, with blank Activity.

2	Modify main activity file *MainActivity.java* to add two new methods *onClickAddName()* and *onClickRetrieveStudents()*.
3	Create a new java file called *StudentsProvider.java* under the package *com.example.MyApplication* to define your actual provider and associated methods.
4	Register your content provider in your *AndroidManifest.xml* file using <provider.../> tag
5	Modify the default content of *res/layout/activity_main.xml* file to include a small GUI to add students records.
6	No need to change string.xml.Android studio take care of string.xml file.
7	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **src/com.example.MyApplication/MainActivity.java**. This file can include each of the fundamental life cycle methods. We have added two new methods *onClickAddName()* and *onClickRetrieveStudents()* to handle user interaction with the application.

```
package com.example.MyApplication;

import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;

import android.content.ContentValues;
import android.content.CursorLoader;

import android.database.Cursor;

import android.view.Menu;
import android.view.View;

import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {

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

```
   public void onClickAddName(View view) {
      // Add a new student record
      ContentValues values = new ContentValues();
      values.put(StudentsProvider.NAME,

((EditText)findViewById(R.id.editText2)).getText().toString());

      values.put(StudentsProvider.GRADE,

((EditText)findViewById(R.id.editText3)).getText().toString());

      Uri uri = getContentResolver().insert(
         StudentsProvider.CONTENT_URI, values);

      Toast.makeText(getBaseContext(),
         uri.toString(), Toast.LENGTH_LONG).show();
   }
   public void onClickRetrieveStudents(View view) {
      // Retrieve student records
      String URL =
"content://com.example.MyApplication.StudentsProvider";

      Uri students = Uri.parse(URL);
      Cursor c = managedQuery(students, null, null, null,
"name");

      if (c.moveToFirst()) {
         do{
            Toast.makeText(this,

c.getString(c.getColumnIndex(StudentsProvider._ID)) +
                  ", " +  c.getString(c.getColumnIndex(
StudentsProvider.NAME)) +
                     ", " + c.getString(c.getColumnIndex(
StudentsProvider.GRADE)),
            Toast.LENGTH_SHORT).show();
         } while (c.moveToNext());
      }
   }
}
```

Create new file StudentsProvider.java under *com.example.MyApplication* package and following is the content of **src/com.example.MyApplication/StudentsProvider.java** −

```
package com.example.MyApplication;

import java.util.HashMap;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.Context;
```

```
import android.content.UriMatcher;

import android.database.Cursor;
import android.database.SQLException;

import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteQueryBuilder;

import android.net.Uri;
import android.text.TextUtils;

public class StudentsProvider extends ContentProvider {
   static final String PROVIDER_NAME =
"com.example.MyApplication.StudentsProvider";
   static final String URL = "content://" + PROVIDER_NAME +
"/students";
   static final Uri CONTENT_URI = Uri.parse(URL);

   static final String _ID = "_id";
   static final String NAME = "name";
   static final String GRADE = "grade";

   private static HashMap<String, String>
STUDENTS_PROJECTION_MAP;

   static final int STUDENTS = 1;
   static final int STUDENT_ID = 2;

   static final UriMatcher uriMatcher;
   static{
      uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
      uriMatcher.addURI(PROVIDER_NAME, "students", STUDENTS);
      uriMatcher.addURI(PROVIDER_NAME, "students/#", STUDENT_ID);
   }

   /**
      * Database specific constant declarations
   */

   private SQLiteDatabase db;
   static final String DATABASE_NAME = "College";
   static final String STUDENTS_TABLE_NAME = "students";
   static final int DATABASE_VERSION = 1;
   static final String CREATE_DB_TABLE =
      " CREATE TABLE " + STUDENTS_TABLE_NAME +
         " (_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
         " name TEXT NOT NULL, " +
         " grade TEXT NOT NULL);";

   /**
      * Helper class that actually creates and manages
      * the provider's underlying data repository.
```

```
   */

   private static class DatabaseHelper extends SQLiteOpenHelper {
      DatabaseHelper(Context context){
         super(context, DATABASE_NAME, null, DATABASE_VERSION);
      }

      @Override
      public void onCreate(SQLiteDatabase db) {
         db.execSQL(CREATE_DB_TABLE);
      }

      @Override
      public void onUpgrade(SQLiteDatabase db, int oldVersion,
int newVersion) {
         db.execSQL("DROP TABLE IF EXISTS " +
STUDENTS_TABLE_NAME);
         onCreate(db);
      }
   }

   @Override
   public boolean onCreate() {
      Context context = getContext();
      DatabaseHelper dbHelper = new DatabaseHelper(context);

      /**
         * Create a write able database which will trigger its
         * creation if it doesn't already exist.
      */

      db = dbHelper.getWritableDatabase();
      return (db == null)? false:true;
   }

   @Override
   public Uri insert(Uri uri, ContentValues values) {
      /**
         * Add a new student record
      */
      long rowID = db.insert(    STUDENTS_TABLE_NAME, "",
values);

      /**
         * If record is added successfully
      */
      if (rowID > 0) {
         Uri _uri = ContentUris.withAppendedId(CONTENT_URI,
rowID);
         getContext().getContentResolver().notifyChange(_uri,
null);
         return _uri;
      }
```

```
      throw new SQLException("Failed to add a record into " +
uri);
   }

   @Override
   public Cursor query(Uri uri, String[] projection,
      String selection,String[] selectionArgs, String sortOrder)
{
      SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
      qb.setTables(STUDENTS_TABLE_NAME);

      switch (uriMatcher.match(uri)) {
         case STUDENTS:
            qb.setProjectionMap(STUDENTS_PROJECTION_MAP);
         break;

         case STUDENT_ID:
            qb.appendWhere( _ID + "=" +
uri.getPathSegments().get(1));
         break;

         default:
      }

      if (sortOrder == null || sortOrder == ""){
         /**
            * By default sort on student names
         */
         sortOrder = NAME;
      }

      Cursor c = qb.query(db,    projection,  selection,
         selectionArgs,null, null, sortOrder);
      /**
         * register to watch a content URI for changes
      */
      c.setNotificationUri(getContext().getContentResolver(),
uri);
      return c;
   }

   @Override
   public int delete(Uri uri, String selection, String[]
selectionArgs) {
      int count = 0;
      switch (uriMatcher.match(uri)){
         case STUDENTS:
            count = db.delete(STUDENTS_TABLE_NAME, selection,
selectionArgs);
         break;

         case STUDENT_ID:
```

```
            String id = uri.getPathSegments().get(1);
            count = db.delete( STUDENTS_TABLE_NAME, _ID +  " = "
+ id +
               (!TextUtils.isEmpty(selection) ? "
               AND (" + selection + ')' : ""), selectionArgs);
            break;
         default:
            throw new IllegalArgumentException("Unknown URI " +
uri);
      }

      getContext().getContentResolver().notifyChange(uri, null);
      return count;
   }

   @Override
   public int update(Uri uri, ContentValues values,
      String selection, String[] selectionArgs) {
      int count = 0;
      switch (uriMatcher.match(uri)) {
         case STUDENTS:
            count = db.update(STUDENTS_TABLE_NAME, values,
selection, selectionArgs);
         break;

         case STUDENT_ID:
            count = db.update(STUDENTS_TABLE_NAME, values,
               _ID + " = " + uri.getPathSegments().get(1) +
               (!TextUtils.isEmpty(selection) ? "
               AND (" +selection + ')' : ""), selectionArgs);
            break;
         default:
            throw new IllegalArgumentException("Unknown URI " +
uri );
      }

      getContext().getContentResolver().notifyChange(uri, null);
      return count;
   }

   @Override
   public String getType(Uri uri) {
      switch (uriMatcher.match(uri)){
         /**
            * Get all student records
         */
         case STUDENTS:
            return "vnd.android.cursor.dir/vnd.example.students";
         /**
            * Get a particular student
         */
         case STUDENT_ID:
```

```
            return
"vnd.android.cursor.item/vnd.example.students";
         default:
            throw new IllegalArgumentException("Unsupported URI:
" + uri);
      }
   }
}
```

Following will the modified content of *AndroidManifest.xml* file. Here we have added <provider.../> tag to include our content provider:

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

      <provider android:name="StudentsProvider"

android:authorities="com.example.MyApplication.StudentsProvider"/
>
   </application>
</manifest>
```

Following will be the content of **res/layout/activity_main.xml** file−

```
<?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:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context="com.example.MyApplication.MainActivity">

   <TextView
```

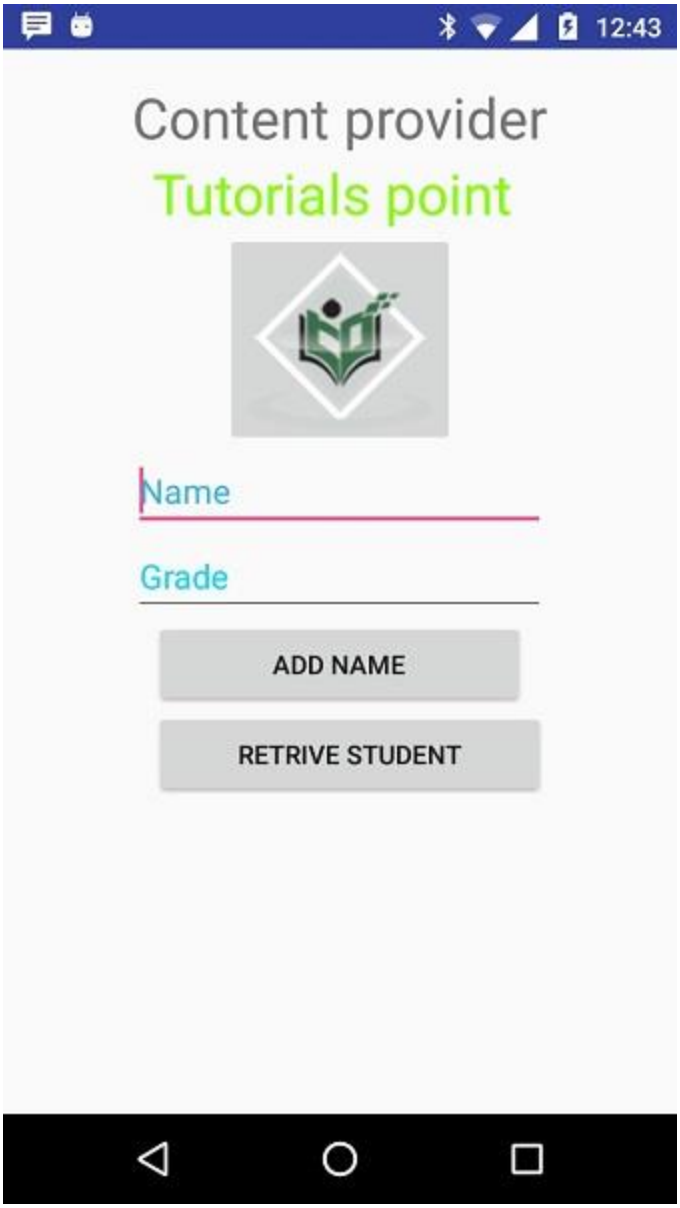
```
      android:id="@+id/textView1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Content provider"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:textSize="30dp" />

   <TextView
      android:id="@+id/textView2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Tutorials point "
      android:textColor="#ff87ff09"
      android:textSize="30dp"
      android:layout_below="@+id/textView1"
      android:layout_centerHorizontal="true" />

   <ImageButton
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/imageButton"
      android:src="@drawable/abc"
      android:layout_below="@+id/textView2"
      android:layout_centerHorizontal="true" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/button2"
      android:text="Add Name"
      android:layout_below="@+id/editText3"
      android:layout_alignRight="@+id/textView2"
      android:layout_alignEnd="@+id/textView2"
      android:layout_alignLeft="@+id/textView2"
      android:layout_alignStart="@+id/textView2"
      android:onClick="onClickAddName"/>

   <EditText
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/editText"
      android:layout_below="@+id/imageButton"
      android:layout_alignRight="@+id/imageButton"
      android:layout_alignEnd="@+id/imageButton" />

   <EditText
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/editText2"
      android:layout_alignTop="@+id/editText"
      android:layout_alignLeft="@+id/textView1"
      android:layout_alignStart="@+id/textView1"
```

```
      android:layout_alignRight="@+id/textView1"
      android:layout_alignEnd="@+id/textView1"
      android:hint="Name"
      android:textColorHint="@android:color/holo_blue_light" />

   <EditText
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/editText3"
      android:layout_below="@+id/editText"
      android:layout_alignLeft="@+id/editText2"
      android:layout_alignStart="@+id/editText2"
      android:layout_alignRight="@+id/editText2"
      android:layout_alignEnd="@+id/editText2"
      android:hint="Grade"
      android:textColorHint="@android:color/holo_blue_bright" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Retrive student"
      android:id="@+id/button"
      android:layout_below="@+id/button2"
      android:layout_alignRight="@+id/editText3"
      android:layout_alignEnd="@+id/editText3"
      android:layout_alignLeft="@+id/button2"
      android:layout_alignStart="@+id/button2"
      android:onClick="onClickRetrieveStudents"/>
</RelativeLayout>
```

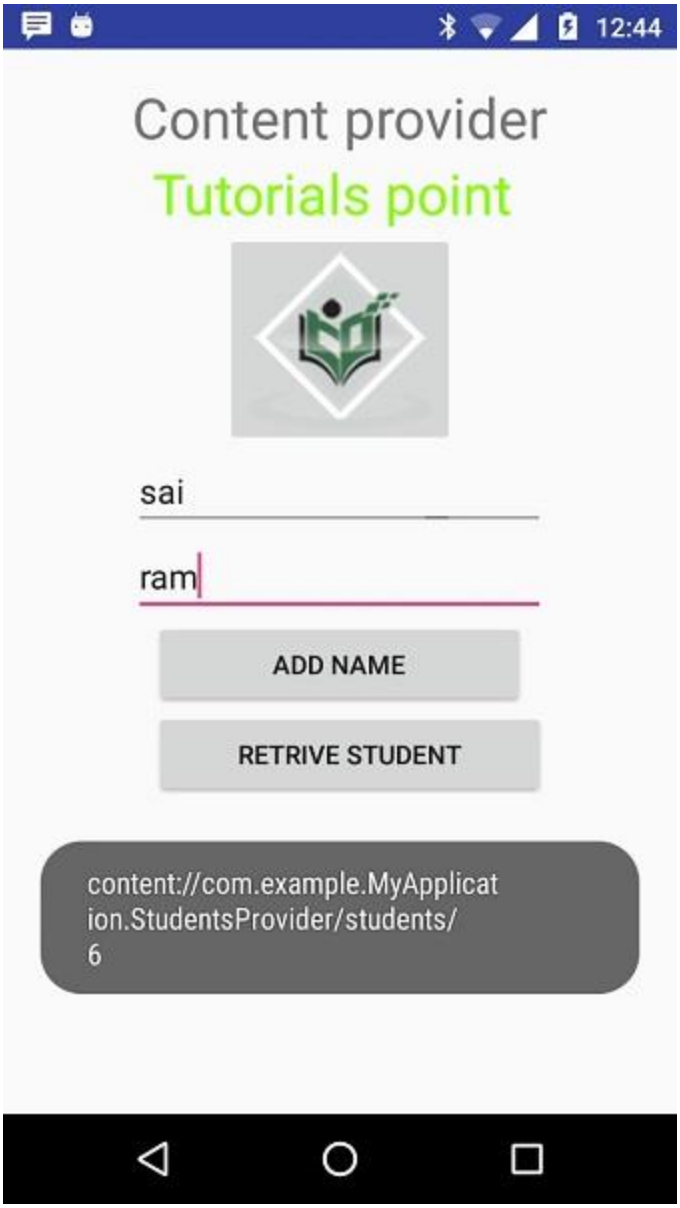
Make sure you have following content of **res/values/strings.xml** file:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">My Application</string>
</resources>;
```

Let's try to run our modified **My Application** application we just created. I assume you had created your **AVD** while doing environment set-up. To run the app from Android Studio IDE, open one of your project's activity files and click Run icon from the tool bar. Android Studio installs the app on your AVD and starts it and if everything is fine with your set-up and application, it will display following Emulator window, be patience because it may take sometime based on your computer speed −



Now let's enter student **Name** and **Grade** and finally click on **Add Name** button, this will add student record in the database and will flash a message at the bottom showing ContentProvider URI along with record number added in the database. This operation makes use of our **insert()** method. Let's repeat this process to add few more students in the database of our content provider.



Once you are done with adding records in the database, now its time to ask ContentProvider to give us those records back, so let's click **Retrieve Students** button which will fetch and display all the records one by one which is as per our the implementation of our **query()** method.

You can write activities against update and delete operations by providing callback functions in **MainActivity.java** file and then modify user interface to have buttons for update and deleted operations in the same way as we have done for add and read operations.

This way you can use existing Content Provider like Address Book or you can use Content Provider concept in developing nice database oriented applications where you can perform all sort of database operations like read, write, update and delete as explained above in the example.

Fragments

A **Fragment** is a piece of an activity which enable more modular activity design. It will not be wrong if we say, a fragment is a kind of **sub-activity**.

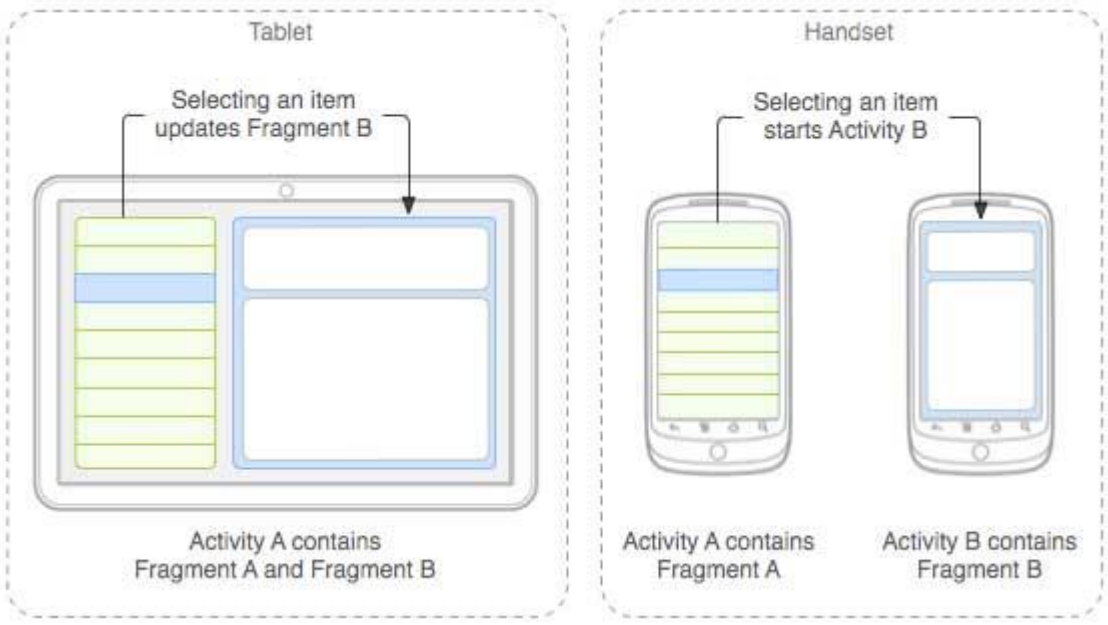
Following are important points about fragment −

- A fragment has its own layout and its own behaviour with its own life cycle callbacks.
- You can add or remove fragments in an activity while the activity is running.
- You can combine multiple fragments in a single activity to build a multi-pane UI.
- A fragment can be used in multiple activities.
- Fragment life cycle is closely related to the life cycle of its host activity which means when the activity is paused, all the fragments available in the activity will also be stopped.
- A fragment can implement a behaviour that has no user interface component.
- Fragments were added to the Android API in Honeycomb version of Android which API version 11.

You create fragments by extending **Fragment** class and You can insert a fragment into your activity layout by declaring the fragment in the activity's layout file, as a **<fragment>** element.

Prior to fragment introduction, we had a limitation because we can show only a single activity on the screen at one given point in time. So we were not able to divide device screen and control different parts separately. But with the introduction of fragment we got more flexibility and removed the limitation of having a single activity on the screen at a time. Now we can have a single activity but each activity can comprise of multiple fragments which will have their own layout, events and complete life cycle.

Following is a typical example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design.

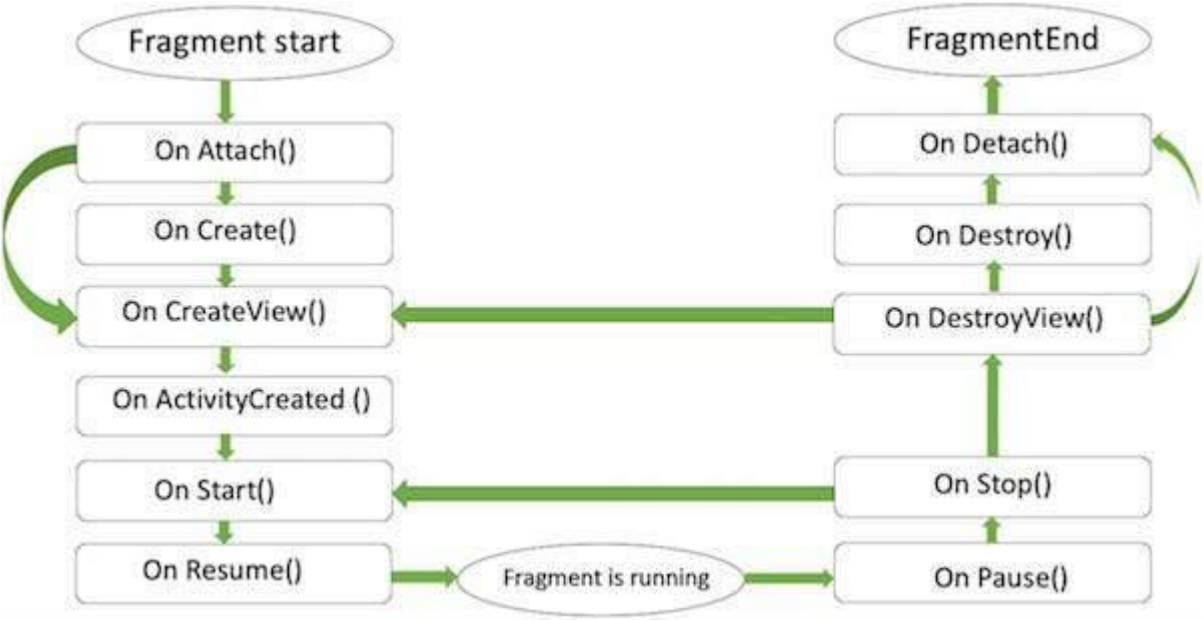


The application can embed two fragments in Activity A, when running on a tablet-sized device. However, on a handset-sized screen, there's not enough room for both

fragments, so Activity A includes only the fragment for the list of articles, and when the user selects an article, it starts Activity B, which includes the second fragment to read the article.

Fragment Life Cycle

Android fragments have their own life cycle very similar to an android activity. This section briefs different stages of its life cycle.



Fragment lifecycle

Here is the list of methods which you can to override in your fragment class −

- **onAttach()**The fragment instance is associated with an activity instance.The fragment and the activity is not fully initialized. Typically you get in this method a reference to the activity which uses the fragment for further initialization work.
- **onCreate()** The system calls this method when creating the fragment. You should initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.
- **onCreateView()** The system calls this callback when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a **View** component from this method that is the root of your fragment's layout. You can return null if the fragment does not provide a UI.
- **onActivityCreated()**The onActivityCreated() is called after the onCreateView() method when the host activity is created. Activity and fragment instance have been created as well as the view hierarchy of the activity. At this point, view can be accessed with the findViewById() method. example. In this method you can instantiate objects which require a Context object
- **onStart()**The onStart() method is called once the fragment gets visible.
- **onResume()**Fragment becomes active.
- **onPause()** The system calls this method as the first indication that the user is leaving the fragment. This is usually where you should commit any changes that should be persisted beyond the current user session.

- **onStop()**Fragment going to be stopped by calling onStop()
- **onDestroyView()**Fragment view will destroy after call this method
- **onDestroy()**onDestroy() called to do final clean up of the fragment's state but Not guaranteed to be called by the Android platform.

How to use Fragments?

This involves number of simple steps to create Fragments.

- First of all decide how many fragments you want to use in an activity. For example let's we want to use two fragments to handle landscape and portrait modes of the device.
- Next based on number of fragments, create classes which will extend the *Fragment* class. The Fragment class has above mentioned callback functions. You can override any of the functions based on your requirements.
- Corresponding to each fragment, you will need to create layout files in XML file. These files will have layout for the defined fragments.
- Finally modify activity file to define the actual logic of replacing fragments based on your requirement.

Types of Fragments

Basically fragments are divided as three stages as shown below.

- Single frame fragments − Single frame fragments are using for hand hold devices like mobiles, here we can show only one fragment as a view.
- List fragments − fragments having special list view is called as list fragment
- Fragments transaction − Using with fragment transaction. we can move one fragment to another fragment.

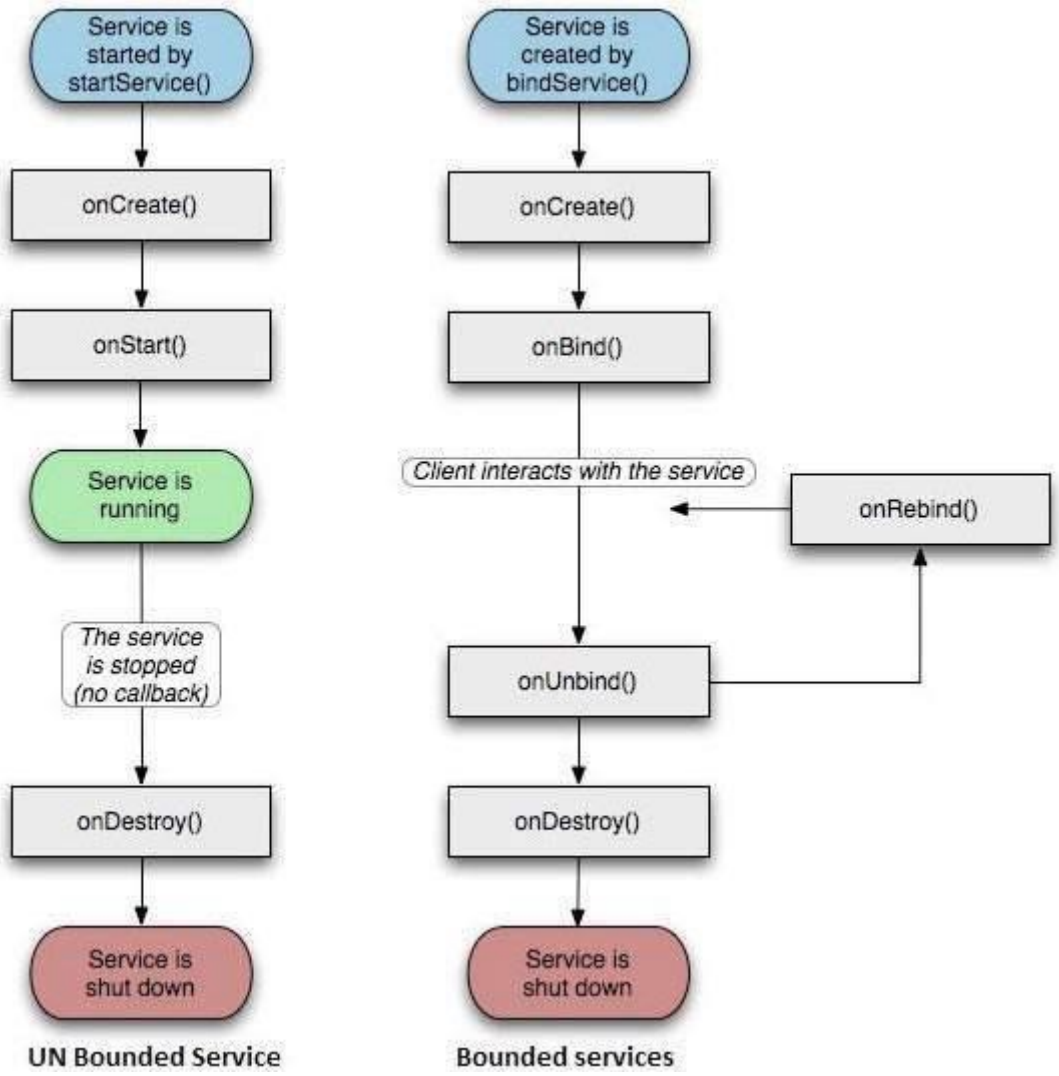
Services

A **service** is a component that runs in the background to perform long-running operations without needing to interact with the user and it works even if application is destroyed. A service can essentially take two states −

Sr.No.	State & Description
1	**Started** A service is **started** when an application component, such as an activity, starts it by calling *startService()*. Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.
2	**Bound** A service is **bound** when an application component binds to it by calling *bindService()*. A bound service offers a client-server interface that allows

	components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC).

A service has life cycle callback methods that you can implement to monitor changes in the service's state and you can perform work at the appropriate stage. The following diagram on the left shows the life cycle when the service is created with startService() and the diagram on the right shows the life cycle when the service is created with bindService(): *(image courtesy : android.com)*



To create an service, you create a Java class that extends the Service base class or one of its existing subclasses. The **Service** base class defines various callback methods and the most important are given below. You don't need to implement all the callbacks methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.

Sr.No.	Callback & Description

1	**onStartCommand()** The system calls this method when another component, such as an activity, requests that the service be started, by calling *startService()*. If you implement this method, it is your responsibility to stop the service when its work is done, by calling *stopSelf()* or *stopService()* methods.
2	**onBind()** The system calls this method when another component wants to bind with the service by calling *bindService()*. If you implement this method, you must provide an interface that clients use to communicate with the service, by returning an *IBinder* object. You must always implement this method, but if you don't want to allow binding, then you should return *null*.
3	**onUnbind()** The system calls this method when all clients have disconnected from a particular interface published by the service.
4	**onRebind()** The system calls this method when new clients have connected to the service, after it had previously been notified that all had disconnected in its *onUnbind(Intent)*.
5	**onCreate()** The system calls this method when the service is first created using *onStartCommand()* or *onBind()*. This call is required to perform one-time set-up.
6	**onDestroy()** The system calls this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, receivers, etc.

The following skeleton service demonstrates each of the life cycle methods −

```
package com.tutorialspoint;

import android.app.Service;
import android.os.IBinder;
import android.content.Intent;
import android.os.Bundle;

public class HelloService extends Service {

   /** indicates how to behave if the service is killed */
   int mStartMode;
```

```
   /** interface for clients that bind */
   IBinder mBinder;

   /** indicates whether onRebind should be used */
   boolean mAllowRebind;

   /** Called when the service is being created. */
   @Override
   public void onCreate() {

   }

   /** The service is starting, due to a call to startService()
*/
   @Override
   public int onStartCommand(Intent intent, int flags, int
startId) {
      return mStartMode;
   }

   /** A client is binding to the service with bindService() */
   @Override
   public IBinder onBind(Intent intent) {
      return mBinder;
   }

   /** Called when all clients have unbound with unbindService()
*/
   @Override
   public boolean onUnbind(Intent intent) {
      return mAllowRebind;
   }

   /** Called when a client is binding to the service with
bindService()*/
   @Override
   public void onRebind(Intent intent) {

   }

   /** Called when The service is no longer used and is being
destroyed */
   @Override
   public void onDestroy() {

   }
}
```

Example

This example will take you through simple steps to show how to create your own Android Service. Follow the following steps to modify the Android application we created in *Hello World Example* chapter −

Step	Description
1	You will use Android StudioIDE to create an Android application and name it as *My Application* under a package *com.example.tutorialspoint7.myapplication* as explained in the *Hello World Example* chapter.
2	Modify main activity file *MainActivity.java* to add *startService()* and *stopService()* methods.
3	Create a new java file *MyService.java* under the package *com.example.My Application*. This file will have implementation of Android service related methods.
4	Define your service in *AndroidManifest.xml* file using <service.../> tag. An application can have one or more services without any restrictions.
5	Modify the default content of *res/layout/activity_main.xml* file to include two buttons in linear layout.
6	No need to change any constants in *res/values/strings.xml* file. Android studio take care of string values
7	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **MainActivity.java**. This file can include each of the fundamental life cycle methods. We have added *startService()* and *stopService()* methods to start and stop the service.

```
package com.example.tutorialspoint7.myapplication;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

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

public class MainActivity extends Activity {
   String msg = "Android : ";

   /** Called when the activity is first created. */
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d(msg, "The onCreate() event");
    }

    public void startService(View view) {
        startService(new Intent(getBaseContext(),
MyService.class));
    }

    // Method to stop the service
    public void stopService(View view) {
        stopService(new Intent(getBaseContext(), MyService.class));
    }
}
```

Following is the content of **MyService.java**. This file can have implementation of one or more methods associated with Service based on requirements. For now we are going to implement only two methods *onStartCommand()* and *onDestroy()* −

```
package com.example.tutorialspoint7.myapplication;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.support.annotation.Nullable;
import android.widget.Toast;

/**
    * Created by TutorialsPoint7 on 8/23/2016.
*/

public class MyService extends Service {
    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int
startId) {
        // Let it continue running until it is stopped.
        Toast.makeText(this, "Service Started",
Toast.LENGTH_LONG).show();
        return START_STICKY;
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
```

```
      Toast.makeText(this, "Service Destroyed",
Toast.LENGTH_LONG).show();
   }
}
```

Following will the modified content of *AndroidManifest.xml* file. Here we have added <service.../> tag to include our service −

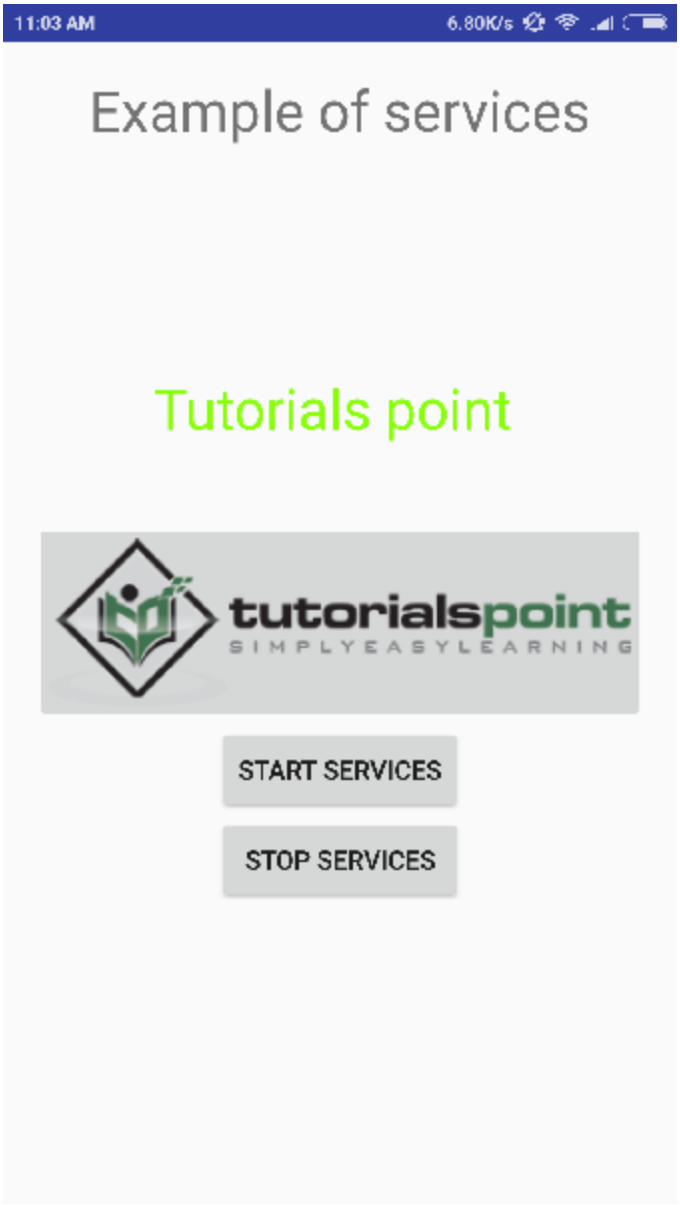
```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.tutorialspoint7.myapplication">

   <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>

      <service android:name=".MyService" />
   </application>

</manifest>
```

Following will be the content of **res/layout/activity_main.xml** file to include two buttons −

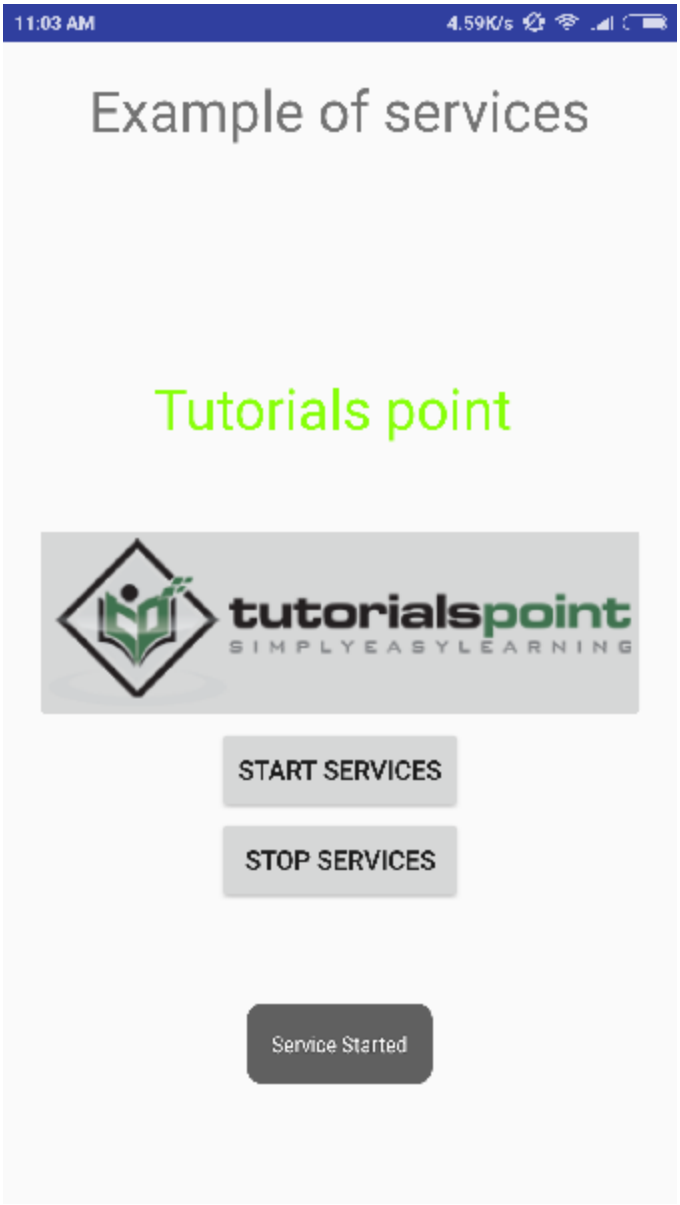
```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
   android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity">

   <TextView
      android:id="@+id/textView1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Example of services"
      android:layout_alignParentTop="true"
```

```
      android:layout_centerHorizontal="true"
      android:textSize="30dp" />

   <TextView
      android:id="@+id/textView2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Tutorials point "
      android:textColor="#ff87ff09"
      android:textSize="30dp"
      android:layout_above="@+id/imageButton"
      android:layout_centerHorizontal="true"
      android:layout_marginBottom="40dp" />

   <ImageButton
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/imageButton"
      android:src="@drawable/abc"
      android:layout_centerVertical="true"
      android:layout_centerHorizontal="true" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/button2"
      android:text="Start Services"
      android:onClick="startService"
      android:layout_below="@+id/imageButton"
      android:layout_centerHorizontal="true" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Stop Services"
      android:id="@+id/button"
      android:onClick="stopService"
      android:layout_below="@+id/button2"
      android:layout_alignLeft="@+id/button2"
      android:layout_alignStart="@+id/button2"
      android:layout_alignRight="@+id/button2"
      android:layout_alignEnd="@+id/button2" />

</RelativeLayout>
```

Let's try to run our modified **Hello World!** application we just modified. I assume you had created your **AVD** while doing environment setup. To run the app from Android studio, open one of your project's activity files and click Run icon from the tool bar. Android Studio installs the app on your AVD and starts it and if everything is fine with your set-up and application, it will display following Emulator window −



Now to start your service, let's click on **Start Service** button, this will start the service and as per our programming in *onStartCommand()* method, a message *Service Started* will appear on the bottom of the the simulator as follows −



To stop the service, you can click the Stop Service button.

Android AsyncTask example and explanation

Android AsyncTask going to do background operation on background thread and update on main thread. In android we cant directly touch background thread to main thread in android development. asynctask help us to make communication between background thread to main thread.

Methods of AsyncTask

- **onPreExecute()** − Before doing background operation we should show something on screen like progressbar or any animation to user. we can directly comminicate background operation using on doInBackground() but for the best practice, we should call all asyncTask methods .

- **doInBackground(Params)** − In this method we have to do background operation on background thread. Operations in this method should not touch on any mainthread activities or fragments.
- **onProgressUpdate(Progress…)** − While doing background operation, if you want to update some information on UI, we can use this method.
- **onPostExecute(Result)** − In this method we can update ui of background operation result.

Generic Types in Async Task

- **TypeOfVarArgParams** − It contains information about what type of params used for execution.
- **ProgressValue** − It contains information about progress units. While doing background operation we can update information on ui using onProgressUpdate().
- **ResultValue** −It contains information about result type.

This example demonstrate about how to use asyncTask in android.

Step 1 − Create a new project in Android Studio, go to File ⇒ New Project and fill all required details to create a new project.

Step 2 − Add the following code to res/layout/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:id = "@+id/rootview"
   android:layout_width = "match_parent"
   android:layout_height = "match_parent"
   android:orientation = "vertical"
   android:background = "#c1c1c1"
   android:gravity = "center_horizontal"
   tools:context = ".MainActivity">
<Button
   android:id = "@+id/asyncTask"
   android:text = "Download"
   android:layout_width = "wrap_content"
   android:layout_height = "wrap_content" />
<ImageView
   android:id = "@+id/image"
   android:layout_width = "300dp"
   android:layout_height = "300dp" />
</LinearLayout>
```

In the above xml we have created a button, when user click on the button it going to download image and append image to imageview.

Step 3 − Add the following code to src/MainActivity.java

```
package com.example.andy.myapplication;
import android.app.ProgressDialog;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import java.io.IOException;
import java.io.InputStream;
```

```
import java.net.HttpURLConnection;
import java.net.URL;
public class MainActivity extends AppCompatActivity {
   URL ImageUrl = null;
   InputStream is = null;
   Bitmap bmImg = null;
   ImageView imageView= null;
   ProgressDialog p;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      Button button=findViewById(R.id.asyncTask);
      imageView=findViewById(R.id.image);
      button.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            AsyncTaskExample asyncTask=new AsyncTaskExample();
            asyncTask.execute("https://www.tutorialspoint.com/ima
ges/tp-logo-diamond.png");
         }
      });
   }
   private class AsyncTaskExample extends AsyncTask<String,
String, Bitmap> {
      @Override
      protected void onPreExecute() {
         super.onPreExecute();
         p = new ProgressDialog(MainActivity.this);
         p.setMessage("Please wait...It is downloading");
         p.setIndeterminate(false);
         p.setCancelable(false);
         p.show();
      }
      @Override
      protected Bitmap doInBackground(String... strings) {
         try {
            ImageUrl = new URL(strings[0]);
            HttpURLConnection conn = (HttpURLConnection)
ImageUrl.openConnection();
            conn.setDoInput(true);
            conn.connect();
            is = conn.getInputStream();
            BitmapFactory.Options options = new
BitmapFactory.Options();
            options.inPreferredConfig = Bitmap.Config.RGB_565;
            bmImg = BitmapFactory.decodeStream(is, null,
options);
         } catch (IOException e) {
            e.printStackTrace();
         }
         return bmImg;
      }
```

```
      @Override
      protected void onPostExecute(Bitmap bitmap) {
         super.onPostExecute(bitmap);
         if(imageView!=null) {
            p.hide();
            imageView.setImageBitmap(bitmap);
         }else {
            p.show();
         }
      }
   }
}
```

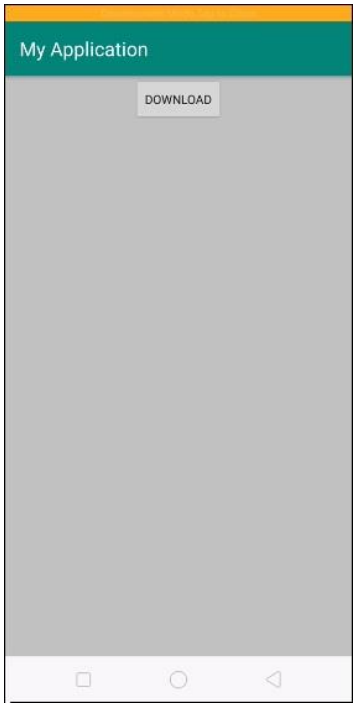
In the above code we are downloading image using asyncTask and appending image to imageview.

Step 4 − Add the following code to manifest.xml

```
<?xml version = "1.0" encoding = "utf-8"?>
<manifest xmlns:android =
"http://schemas.android.com/apk/res/android"
package = "com.example.andy.myapplication">
   <uses-permission android:name =
"android.permission.INTERNET"/>
   <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>
   </application>
</manifest>
```

In the above AndroidManifest.xml file we have added internet permission to access internet to download image.

Let's try to run your application. I assume you have connected your actual Android Mobile device with your computer. To run the app from android studio, open one of your project's activity files and click Run Eclipse Run Icon icon from the toolbar. Select your mobile device as an option and then check your mobile device which will display your default screen.



1. Now click on download button it will show progress on UI and download image at background as shown above.

2. After downloading image, it will update on UI as shown below