
Unit-V Activity and Multimedia with Databases

Course Outcome:

Configure Android environment and development tools.

Unit Outcomes:

- 5a. Apply the given Intents and service in Application development.
 - 5b. Use Fragment to generate the given multiple activities.
 - 5c. Develop programs to play the given multimedia.
 - 5d. Write the query to perform the given database management operation.
-

Contents:

- 5.1 Intent, Intent Filter
 - 5.2 Activity Lifecycle; Broadcast lifecycle
 - 5.3 Content Provider; Fragments
 - 5.4 Service: Features Of service, the Android platform service, defining new service, Service Lifecycle, Permission, example of service
 - 5.5 Android System Architecture, Multimedia framework, Play Audio and Video, Text to speech, Sensors, Async tasks
 - 5.6 Audio Capture, Camera
 - 5.7 Bluetooth, Animation
 - 5.8 SQLite Database, necessity of SQLite, Creation and connection of the database, extracting value from cursors, Transactions.
-

Intent Tutorial in Android And Types

Android uses Intent for communicating between the components of an Application and also from one application to another application.

Intent are the objects which is used in android for passing the information among Activities in an Application and from one app to another also. Intent are used for communicating between the Application components and it also provides the connectivity between two apps.

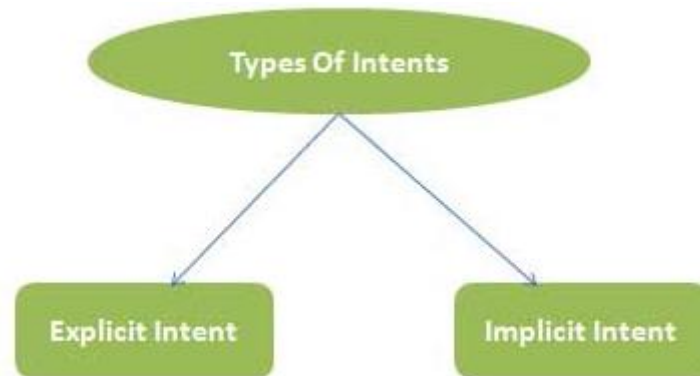
For example: Intent facilitate you to redirect your activity to another activity on occurrence of any event. By calling, `startActivity()` you can perform this task.

```
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.

Types of Intents:

Intents are of two types: Explicit Intent and Implicit Intent



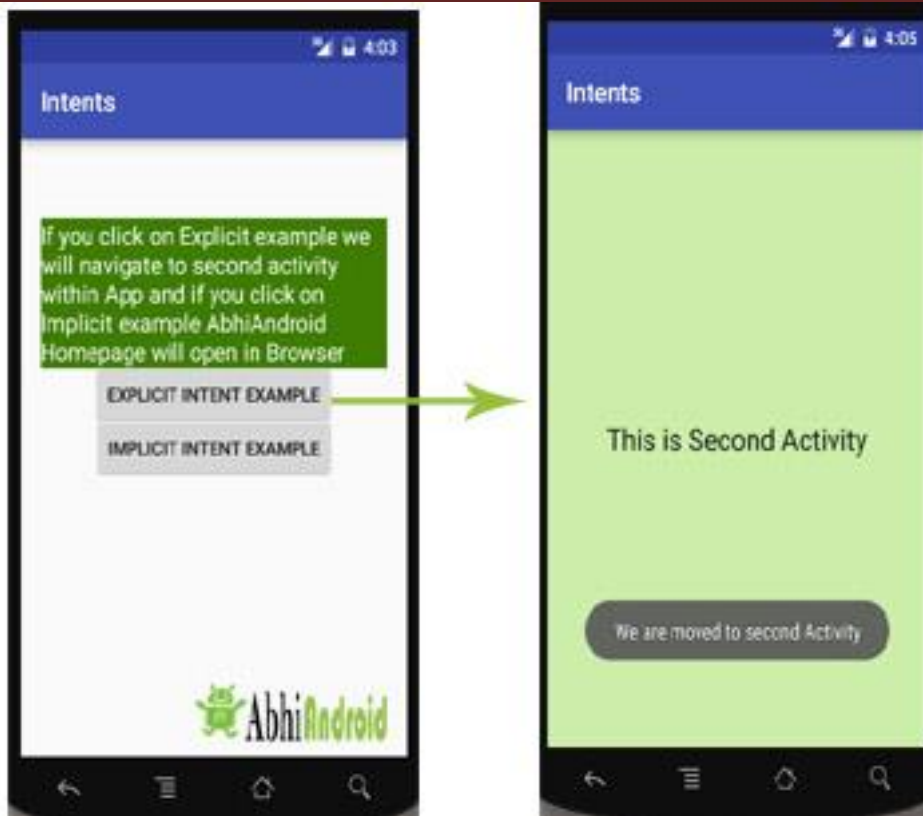
Explicit Intent:

- 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 Intents work 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. Example with code in the end of this post will make it more clear.



Implicit Intent:

- In Implicit Intents we do not 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.abhiandroid.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 will run this code then Android will automatically start your web browser and it will open AbhiAndroid home page.

*Intent Example****In Android:***

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 AbhiAndroid using Intent. **The example will show you both implicit and explicit Intent together. Below is the final output:**

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_horizo
ntal_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 with  
in App and if you click on Implicit example AbhiAndroid 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 clicks 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_hori_
    zontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:background="#CCEEEA"
    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

```
package com.example.android.intents;

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(getApplicationContext(), 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.abhiandroid.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:

```
package com.example.android.intents;  
  
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" >

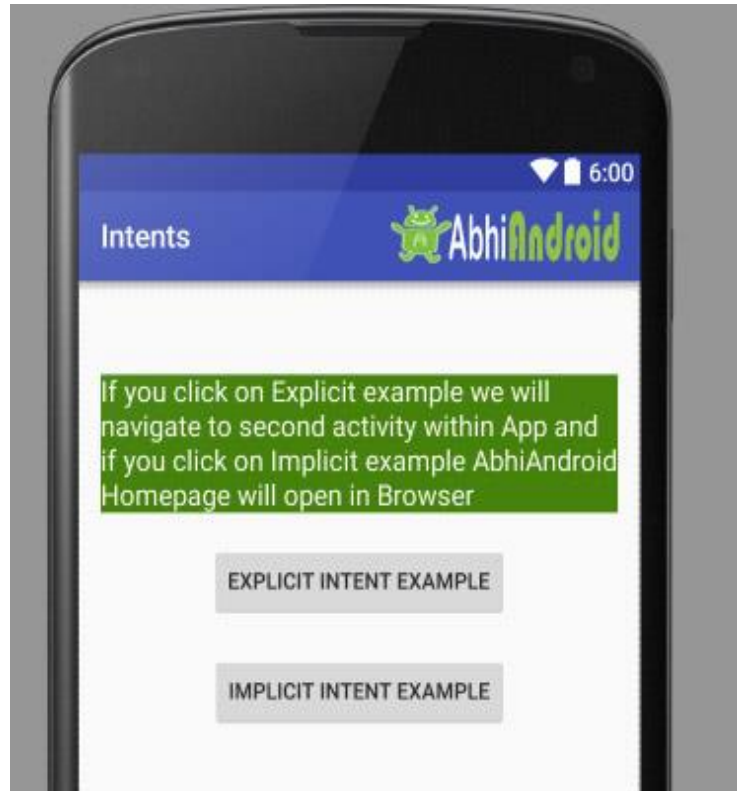
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" >
        <activity android:name=".MainActivity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".SecondActivity" >

    </activity>
</application>
```

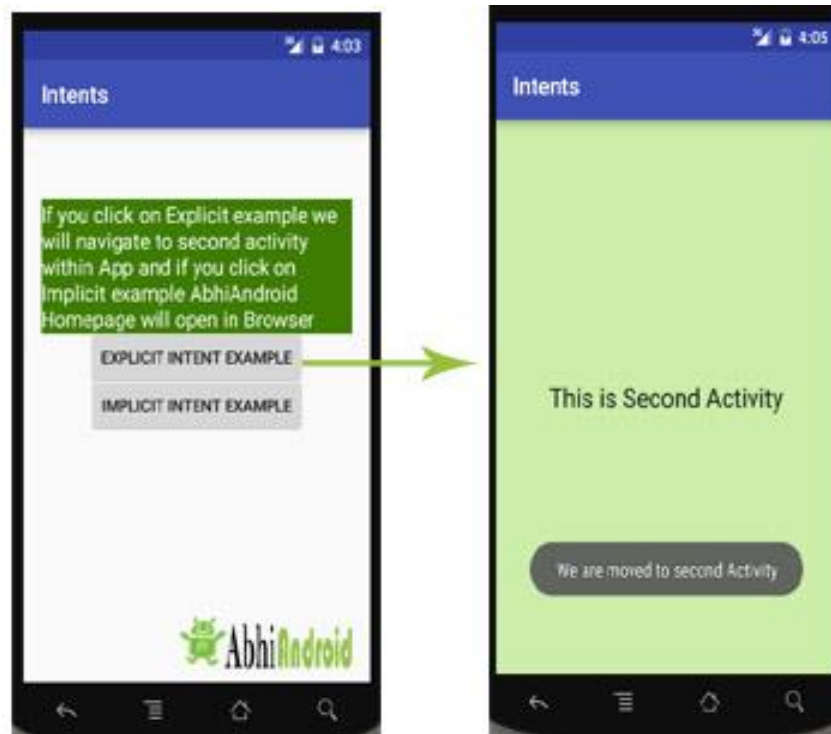
```
</manifest>
```

Output:

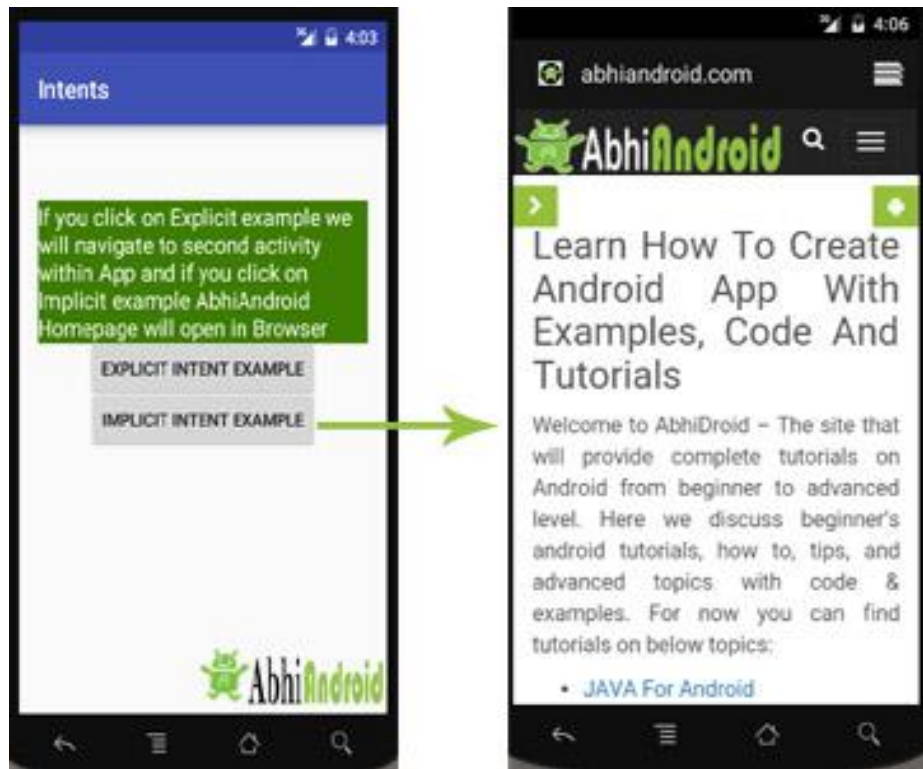
Now run the above program in your Emulator. The App will look like this:



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 AbhiAndroid.com homepage will open in Browser (make sure you have internet):



Intent Uses In Android:

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

Intent 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.

Intent 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.

Intent for Broadcast Receivers:

There are various message 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.

Importance of using Intents in 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.

Intents are really easy to handle and it facilitates communication of components and activities of your application. Moreover you can communicate to another application and send some data to another application using Intents.

Intent Filter in Android Manifest

Intent Filter are the components which decide the behavior of an intent. As we have read in our previous tutorial of Intent about the navigation of one activity to another, that can be achieved by declaring intent filter. We can declare an Intent Filter for an Activity in manifest file.

Intent filters specify the type of intents that an Activity, service or Broadcast receiver can respond to. It declares the functionality of its parent component (i.e. activity, services or broadcast receiver). It declares the capability of any activity or services or a broadcast receiver.

Intent Filter Code Inside Android Manifest:

The code of Intent Filter is used inside `AndroidManifest.xml` file for an activity. You can see it: open manifests folder >> open `AndroidManifest.xml` file



Syntax of Intent Filters:

Intent filter is declared inside Manifest file for an Activity.

```
<!--Here Name of Activity is .MainActivity, image of icon name stored in drawable folder, label present inside string name is label-->
```

```
<!--If anything is different please customize the code according to your app-->
```

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

```
<intent-filter android:icon="@drawable/icon"
    android:label="@string/label"
```

```
>
```

```
<action android:name="android.intent.action.MAIN" />
```

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

```
</intent-filter>
```

```
</activity>
```

Important Note: If you are using above code make sure below things are correct:

icon: This is displayed as icon for activity. You can check or save png image of name icon in drawable folder. If icon image of any other name is stored please replace @drawable/icon with that image name i.e. @drawable/image_name.

label: The label / title that appears at top in Toolbar of that particular Activity. You can check or edit label name by opening String XML file present inside Values folder (Values -> Strings). Replace @string/label to @string/yourlabel.

priority: This is another important attribute in Intent filter used in broadcasting. We will explain it in future topics of broadcasting.

Attributes of Intent Filter:

Below are the attributes of Intent filter:

1. android:icon

An icon represents the activity, service or broadcast receiver when a user interact with it or when it appears to user in an application. To set an icon you need to give reference of drawable resource as declared android:icon="@drawable/icon".

How to add custom icon in your App:

Step 1: Open your android project folder in computer / system

Step 2: Open app>> src >> main >> res >> drawable >> here save custom icon image with name

Step 3: Replace @drawable/icon with @drawable/your_image name in the below code

```
android:icon="@drawable/icon"
```

Important Note: If you have not declared any icon for your activity then android sets the icon of parent component by default. And if no icon is defined by parent component then icon is set as the default icon by <application> element (declared in Manifest file).

2. android:label

A label represents the title of an activity on the toolbar. You can have different Labels for different Activities as per your requirement or choice. The label should be set as a reference to a string resource. However, you can also use a raw string to set a label as declared in the below given code snippet

```
android:label = "@string/label"
```

or

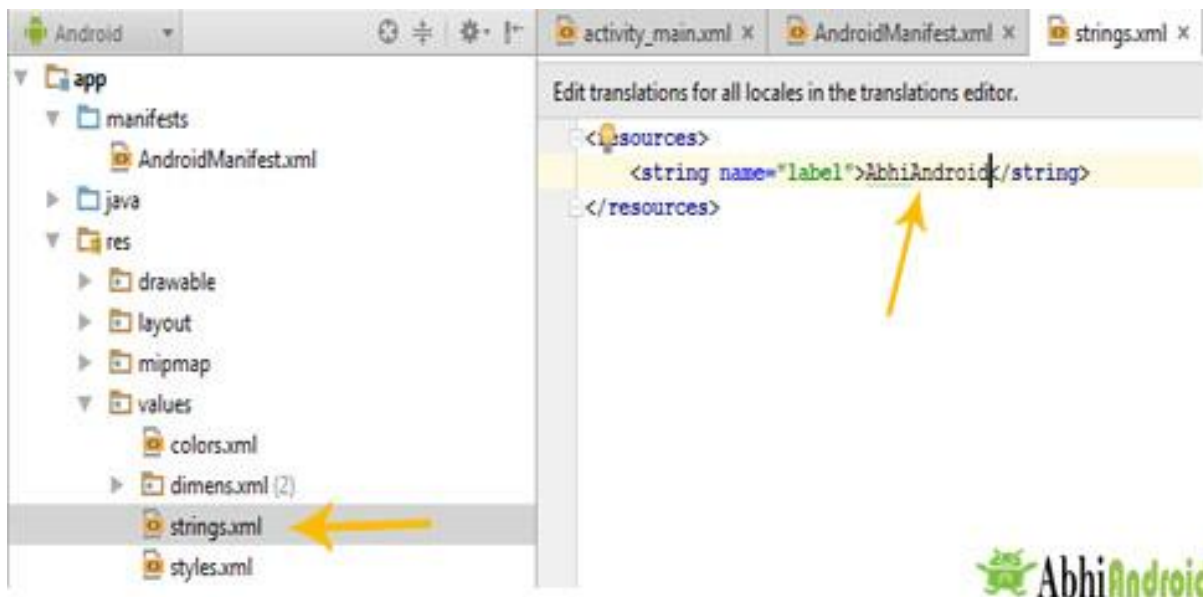
```
android:label = "New Activity"
```

How To Create Custom Label in App:

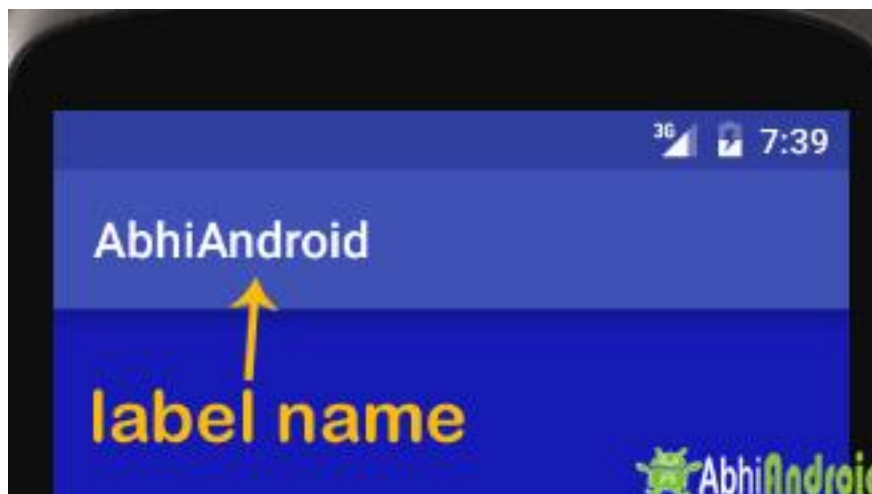
Step 1: Click on values values

Step 2: Open Strings.xml file present inside values

Step 3: Edit String value to your custom name. For example,



Step 4: Now run the App and you will see your custom name in Toolbar. Below you can see AbhiAndroid printed.



Important Note: Just like icon attribute, if you have not declared any label for your activity then it will be same as your parent activity. However if you haven't declared any label to parent activity then label is set by <application> element' label attribute. Below is sample code of Intent filter for an App having two activity:

```
<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>

<!--Add code here for first activity-->

</intent-filter>
</activity>
<activity android:name=".Main2Activity">
<intent-filter>
<!--Add code here for second activity-->
</intent-filter>
</activity>
</application>
```

Elements In Intent Filter:

There are following three elements in an intent filter:

1. Action
2. Data
3. Category

Important Note: Every intent filter must contain action element in it. Data and category element is optional for it.

1. Action:

It represent an activities action, what an activity is going to do. It is declared with the name attribute as given below

```
<action android:name = "string" />
```

An Intent Filter element must contain one or more action element. Action is a string that specifies the action to perform. You can declare your own action as given below. But we usually use action constants defined by Intent class.

```
<intent-filter>
<action android:name="com.example.android.intentfilters.Main2Activity" />
</intent-filter>
```

There are few common actions for starting an activity like ACTION_VIEW

ACTION_VIEW: This is used in an Intent with `startActivity()`. This helps when you redirect to see any website, photos in `gallery` app or an address to view in a map app.

For example: As we have done in previous topic of Intent, we started a website using implicit intent in that Intent we used ACTION_VIEW element to view website in the web browser.

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

There are more actions similar to above like, ACTION_SEND, ACTION_MAIN, ACTION_WEB_SEARCH and many more.

2. Data:

There are two forms in which you can pass the data, using URI(Uniform Resource Identifiers) or MIME type of data. For understanding the concept of URI in better manner check the link.

The syntax of data attribute is as follows:

```
<data android:scheme="string"
android:host="string"
android:port="string"
android:path="string"
android:pathPattern="string"
android:pathPrefix="string"
android:mimeType="string" />
```

This specifies the format of data associated with an Intent which you use with component. As explained in above code snippet, data passed through Intent is a type of URI. Check the below given table for better clarification

ACTION	DATA	MEANING
Intent.ACTION_CALL	tel:phone_number	Opens phone application and calls phone number
Intent.ACTION_DIAL	tel:phone_number	Opens phone application and dials (but doesn't call) phone_number
Intent.ACTION_DIAL	voicemail:	Opens phone application and dials (but doesn't call) the voice mail number.
Intent.ACTION_VIEW	geo:lat,long	Opens the maps Application centered on (lat, long).
Intent.ACTION_VIEW	geo:0,0?q=address	Opens the maps application centered on the

		specified address.
Intent.ACTION_VIEW	http://url https://url	Opens the browser application to the specified address.
Intent.ACTION_WEB_SEARCH	plain_text	Opens the browser application and uses Google search for given string

3. Category:

This attribute of Intent filter dictates the behavior or nature of an Intent. There is a string which contains some additional information about the intent which will be handled by a component. The syntax of category is as follows:

```
<category android:name="string" />
```

Most of Intents do not require a category for example: CATEGORY_BROWSABLE, CATEGORY_LAUNCHER.

BROWSABLE – Browsable category, activity allows itself to be opened with web browser to open the reference link provided in data.

LAUNCHER – Launcher category puts an activity on the top of stack, whenever application will start, the activity containing this category will be opened first.

```
<intent-filter>
<!--Code here-->
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
<intent-filter>
<!--Code here-->
<category android:name="android.intent.category.BROWSABLE" />
</intent-filter>
```

Intent Filter Example:

Let's take an example of Intent filter for an Activity. In this example, we will make an Activity a launcher activity by declaring the intent filter. By launcher we mean this Activity will be launched first out of all activities in App.

Below code snippet gives the details for the same. You have to make changes in Android Manifest file to make your activity launcher activity.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.android.intentfilters">
<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=".Main2Activity">
<intent-filter>
<action android:name="com.example.android.intentfilters.Main2Activity" />
<category android:name="android.intent.category.DEFAULT" />
</intent-filter>
</activity>
</application>
</manifest>
```

Important Note: In above mentioned code snippet we are using two activities which you have to declare it in the Manifest that which activity has to launch first when your application starts. In above code we have made MainActivity as a Launcher Activity by declaring Intent Filter:

```

<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" >

</activity>
    <activity android:name=".SecondActivity" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

```

This code decide which activity launch first if more than one activity is used




Activity Lifecycle With Example In Android – Tutorial, Code And Importance

Activity Lifecycle: Activity is one of the building blocks of Android OS. In simple words Activity is a screen that user interact with. *Every Activity in android has lifecycle like created, started, resumed, paused, stopped or destroyed. These different states are known as Activity Lifecycle.* In other words we can say Activity is a class pre-written in Java Programming.

Below is Activity Lifecycle Table:

Short description of Activity Lifecycle example:

onCreate() – Called when the activity is first created

onStart() – Called just after it's creation or by restart method after onStop(). Here Activity start becoming visible to user

onResume() – Called when Activity is visible to user and user can interact with it

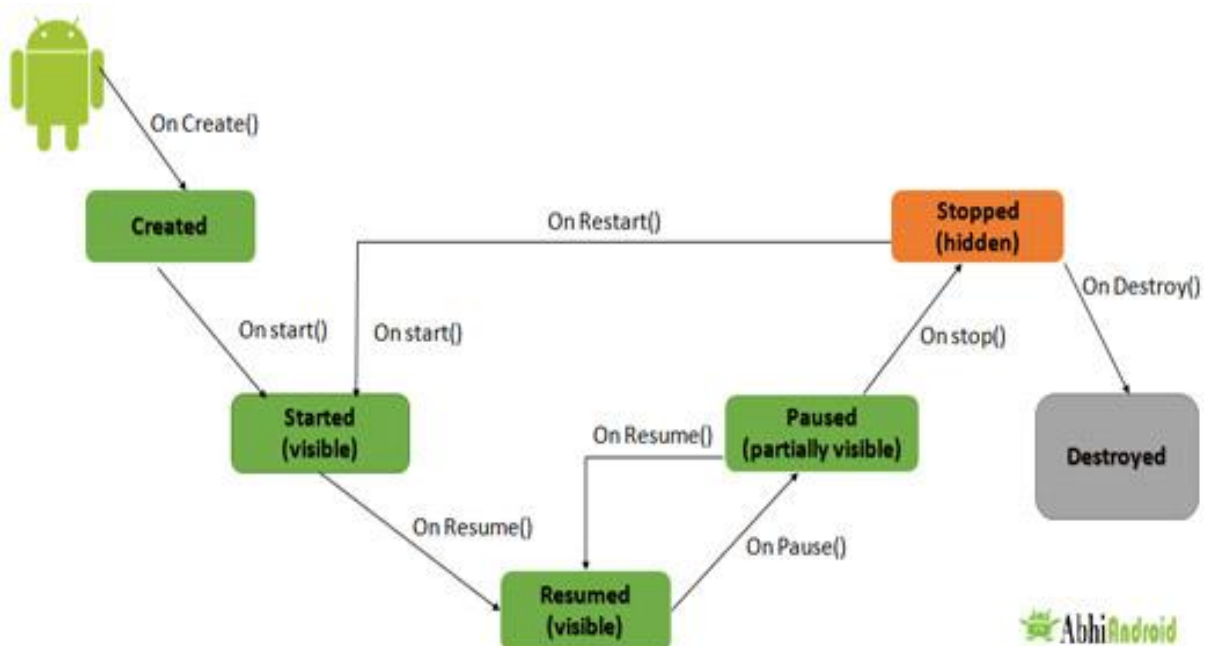
onPause() – Called when Activity content is not visible because user resume previous activity

onStop() – Called when activity is not visible to user because some other activity takes place of it

onRestart() – Called when user comes on screen or resume the activity which was stopped

onDestroy – Called when Activity is not in background

Below Activity Lifecycle Diagram Shows Different States:



Different Types of Activity Lifecycle States:

Activity have different states or it's known as Activity life cycle. All life cycle methods aren't required to override but it's quite important to understand them. Lifecycles methods can be overridden according to requirements.

LIST OF ACTIVITY LIFECYCLE METHODS OR STATES:

Activity Created: onCreate(Bundle savedInstanceState):

onCreate() method is called when activity gets memory in the OS. **To use create state we need to override onCreate(Bundle savedInstanceState) method.** Now there will be question in mind what is Bundle here, so Bundle is a data repository object that can store any kind of primitive data and this object will be null until some data isn't saved in that.

- When an Activity first call or launched then onCreate(Bundle savedInstanceState) method is responsible to create the activity.
- When ever orientation(i.e. from horizontal to vertical or vertical to horizontal) of activity gets changed or when an Activity gets forcefully terminated by any Operating System then savedInstanceState i.e. object of Bundle Class will save the state of an Activity.
- It is best place to put initialization code.

Learn More About onCreate(Bundle savedInstanceState) With Example

Activity Started: onStart():

onStart() method is called just after it's creation. In other case Activity can also be started by calling restart method i.e after activity stop. So this means onStart() gets called by Android OS when user switch between applications. For example, if a user was using Application A and then a notification comes and user clicked on notification and moved to

Application B, in this case Application A will be paused. And again if a user again click on app icon of Application A then Application A which was stopped will again gets started.

Learn More About [onStart\(\) With Example](#)

Activity Resumed: onResume():

Activity resumed is that situation when it is actually visible to user means the data displayed in the activity is visible to user. In lifecycle it always gets called after activity start and in most use case after activity paused (**onPause**).

Activity Paused: onPause():

Activity is called paused when it's content is not visible to user, in most case onPause() method called by Android OS when user press Home button (Center Button on Device) to make hide.

Activity also gets paused before stop called in case user press the back navigation button. The activity will go in paused state for these reasons also if a notification or some other dialog is overlaying any part (top or bottom) of the activity (screen). Similarly, if the other screen or dialog is transparent then user can see the screen but cannot interact with it. For example, if a call or notification comes in, the user will get the opportunity to take the call or ignore it.

Learn More About [onPause\(\) With Example](#)

Activity Stopped: onStop():

Activity is called stopped when it's not visible to user. Any activity gets stopped in case some other activity takes place of it. For example, if a user was on screen 1 and click on some button and moves to screen 2. In this case Activity displaying content for screen 1 will be stopped.

Every activity gets stopped before destroy in case of when user press back navigation button. So Activity will be in stopped state when hidden or replaced by other activities that have been launched or switched by user. In this case application will not present anything useful to the user directly as it's going to stop.

Learn More About [onStop\(\) With Example](#)

Activity Restarted: onRestart():

Activity is called in restart state after stop state. So activity's onRestart() function gets called when user comes on screen or resume the activity which was stopped. In other words, when Operating System starts the activity for the first time onRestart() never gets called. It gets called only in case when activity is resumes after stopped state.

Activity Destroyed: onDestroy():

Any activity is known as in destroyed state when it's not in background. There can different cases at what time activity get destroyed.

First is if user pressed the back navigation button then activity will be destroyed after completing the lifecycle of pause and stop.

In case if user press the home button and app moves to background. User is not using it no more and it's being shown in recent apps list. So in this case if system required resources need to use somewhere else then OS can destroy the Activity.

After the Activity is destroyed if user again click the app icon, in this case activity will be recreated and follow the same lifecycle again. Another use case is with Splash Screens if there is call to finish() method from onCreate() of an activity then OS can directly call onDestroy() with calling onPause() and onStop().

Activity Lifecycle Example:

In the below example we have used the below JAVA and Android topics:

JAVA Topics Used: Method Overriding, static variable, package, Inheritance, method and class.

Android Topic Used: We have used Log class which is used to printout message in Logcat. One of the important use of Log is in debugging.

First we will create a new Android Project and name the activity as HomeActivity. In our case we have named our App project as Activity Lifecycle Example.

We will initialize a static String variable with the name of the underlying class using getSimpleName() method. In our case HOME_ACTIVITY_TAG is the name of the String variable which store class name HomeActivity.

```
private static final String HOME_ACTIVITY_TAG = HomeActivity.class.getSimpleName();
```

Now we will create a new method which will print message in Logcat.

```
private void showLog(String text){  
  
    Log.d(HOME_ACTIVITY_TAG, text);  
  
}
```

Now we will override all activity lifecycle method in Android and use showLog() method which we creating for printing message in Logcat.

Complete JAVA code of HomeActivity.java:

```
package com.abhiandroid.activitylifecycleexample;  
  
import android.os.Bundle;  
import android.support.design.widget.FloatingActionButton;  
import android.support.design.widget.Snackbar;  
import android.support.v7.app.AppCompatActivity;  
import android.support.v7.widget.Toolbar;  
import android.util.Log;  
import android.view.View;  
import android.view.Menu;  
import android.view.MenuItem;
```

```
public class HomeActivity extends AppCompatActivity {  
    private static final String HOME_ACTIVITY_TAG = HomeActivity.class.getSimpleName()  
    0;  
  
    private void showLog(String text){  
        Log.d(HOME_ACTIVITY_TAG, text);  
    }  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        showLog("Activity Created");  
    }  
  
    @Override  
    protected void onRestart(){  
        super.onRestart();//call to restart after onStop  
        showLog("Activity restarted");  
    }  
  
    @Override  
    protected void onStart() {  
        super.onStart();//soon be visible  
        showLog("Activity started");  
    }  
  
    @Override  
    protected void onResume() {  
        super.onResume();//visible
```

```
        showLog("Activity resumed");
    }

    @Override
    protected void onPause() {
        super.onPause();//invisible
        showLog("Activity paused");
    }

    @Override
    protected void onStop() {
        super.onStop();
        showLog("Activity stopped");
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        showLog("Activity is being destroyed");
    }
}
```

When creating an Activity we need to register this in [AndroidManifest.xml](#) file. Now question is why need to register? **Its actually because manifest file has the information which Android OS read very first.** When registering an activity other information can also be defined within manifest like Launcher Activity (An activity that should start when user click on app icon).

Here is declaration example in [AndroidManifest.xml](#) file

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

    <application
```

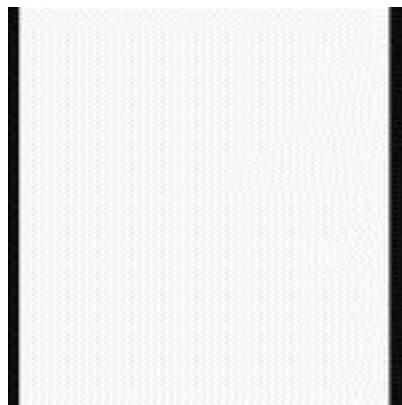


```
android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:supportsRtl="true"
android:theme="@style/AppTheme">
<activity
    android:name=".HomeActivity"
    android:label="@string/app_name"
    android:theme="@style/AppTheme.NoActionBar">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
</application>

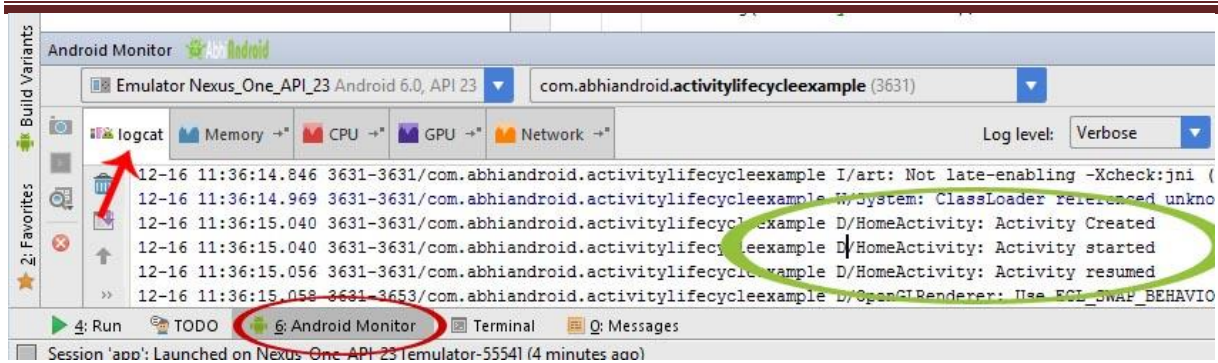
</manifest>
```

Output Of Activity Lifecycle:

When you will run the above program you will notice a blank white screen will open up in Emulator. *You might be wondering where is default Hello world screen. Actually we have removed it by overriding onCreate() method.* Below is the blank white screen that will pop up.



Now go to Logcat present inside Android Monitor: Scroll up and you will notice three methods which were called: Activity Created, Activity started and Activity resumed.



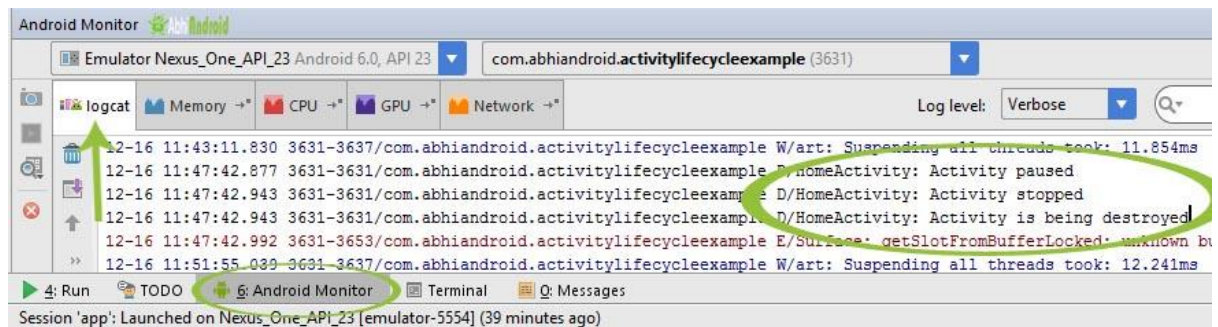
So this clears:

- first onCreate() method was called when activity was created
- second onStart() method was called when activity start becoming visible to user
- Finally onResume() method was called when activity is visible to user and user can interact with it

Now press the back button on the Emulator and exit the App:



Go to Logcat again and scroll down to bottom. You will see 3 more methods were called: Activity paused, Activity stopped and Activity is being destroyed.



So this clears:

- onPause() method was called when user resume previous activity
- onStop() method was called when activity is not visible to user
- Last onDestroy() method was called when Activity is not in background

Important Note: In the above example onRestart() won't be called because there was no situation when we can resume the onStart() method again. In future example we will show you onRestart() in action as well.

Importance Of Activity Life Cycle:

Activity is the main component of Android Application, as every screen is an activity so to create any simple app first we have to start with Activities. Every lifecycle method is quite important to implement according to requirements, However onCreate(Bundle state) is always needed to implement to show or display some content on screen.

BroadcastReceivers

In android, **Broadcast Receiver** is a component which will allow android system or other apps to deliver events to the app like sending a low battery message or screen turned off message to the app. The apps can also initiate broadcasts to let other apps know that required data available in a device to use it.

Generally, we use Intents to deliver broadcast events to other apps and Broadcast Receivers use status bar notifications to let user know that broadcast event occurs.

In android, Broadcast Receiver is implemented as a subclass of **BroadcastReceiver** and each broadcast is delivered as an Intent object.

We can register an app to receive only few broadcast messages based on our requirements. When a new broadcast received, the system will check for specified broadcasts have subscribed or not based on that it will routes the broadcasts to the apps.

Receiving Broadcasts

In android, we can receive broadcasts by registering in two ways.

One way is by registering a broadcasts using android application manifest file (**AndroidManifest.xml**). We need to specify **<receiver>** element in apps manifest file like as shown below.

```
<receiver android:name=".SampleBroadcastReceiver">
  <intent-filter>
    <action android:name="android.intent.action.BOOT_COMPLETED"/>
  </intent-filter>
</receiver>
```

The above statement will fire the defined system broadcast event whenever the boot process is completed.

Another way is to register a receiver dynamically via **Context.registerReceiver()** method. To register broadcast receiver we need to extend our class using **BroadcastReceiver** and need to implement a **onReceive(Context, Intent)** method like as shown below.

```
public class MainActivity extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, log, Toast.LENGTH_LONG).show();
    }
}
```

Suppose if we register for **ACTION_BOOT_COMPLETED** event, whenever the boot process is completed the broadcast receiver's method **onReceive()** method will be invoked.

Sending Broadcasts

In android, we can send a broadcasts in apps using three different ways, those are

Method	Description
sendOrderedBroadcast(Intent, String)	This method is used to send broadcasts to one receiver at a time.
sendBroadcast(Intent)	This method is used to send broadcasts to all receivers in an undefined order.
LoadBroadcastManager.sendBroadcast	This method is used to send broadcasts to receivers that are in the same app as the sender.

Broadcasting Custom Intents

In android we can create our own custom broadcasts using intents. Following is the simple code snippet of sending a broadcast by creating an intent using **sendBroadcast(Intent)** method.

```
Intent sintent = new Intent();
intent.setAction("com.tutlane.broadcast.MY_NOTIFICATION");
intent.putExtra("data","Welcome to Tutlane");
sendBroadcast(intent);
```

If you observe above code snippet we create a custom Intent "**sintent**". We need to register our intent action in android manifest file like as shown below

```
<receiver android:name=".SampleBroadcastReceiver">
    <intent-filter>
        <action android:name=" com.tutlane.broadcast.MY_NOTIFICATION"/>
    </intent-filter>
</receiver>
```

This is how we can create our own custom broadcasts using Intents in android applications.

System Broadcasts

In android, several system events are defined as final static fields in the Intent class. Following are the some of system events available in android applications.

Event	Description
android.intent.action.BOOT_COMPLETED	It raise an event, once boot completed.
android.intent.action.POWER_CONNECTED	It is used to trigger an event when power connected to the device.
android.intent.action.POWER_DISCONNECTED	It is used to trigger an event when power got disconnected from the device.
android.intent.action.BATTERY_LOW	It is used to call an event when battery is low on device.
android.intent.action.BATTERY_OKAY	It is used to call an event when battery is OKAY again.
android.intent.action.REBOOT	It call an event when the device rebooted again.

Likewise we have many system events available in android application.

Android BroadcastReceiver Example

To setup a Broadcast Receiver in our android application we need to do following things.

- Create a BroadcastReceiver
- Register a BroadcastReceiver

Create a new android application using android studio and give names as **BroadcastReceiver**. In case if you are not aware of creating an app in android studio check this article [Android Hello World App](#).

Now we need to create our own broadcast content file **MyBroadcastReceiver.java** in `\src\main\java\com.tutlane.broadcastreceiver` path to define our actual provider and associated methods for that right click on your application folder Go to **New** select **Java Class** and give name as **MyBroadcastReceiver.java**.

Once we create a new file **MyBroadcastReceiver.java**, open it and write the code like as shown below

MyBroadcastReceiver.java

```
package com.tutlane.broadcastreceiver;
import android.content.BroadcastReceiver;
```

```
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

/**
 * Created by Tutlane on 01-08-2017.
 */

public class MyBroadcastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent){
        CharSequence iData = intent.getCharSequenceExtra("msg");
        Toast.makeText(context,"Tutlane Received Message: "+iData,Toast.LENGTH_LONG).s
how();
    }
}
```

Now open **activity_main.xml** file from `\src\main\res\layout` path and write the following code.

activity_main.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">
    <EditText
        android:id="@+id/txtMsg"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="100dp"
        android:layout_marginTop="180dp"
        android:ems="10"/>
    <Button
        android:id="@+id/btnShow"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClickShowBroadcast"
        android:layout_marginLeft="130dp"
        android:text="Show Broadcast"/>
</LinearLayout>
```

Now open **MainActivity.java** file from `\java\com.tutlane.broadcastreceiver` path and write following to implement custom broadcast intents.

MainActivity.java

```
package com.tutlane.broadcastreceiver;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
```

```

import android.view.View;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void onClickShowBroadcast(View view){
        EditText st = (EditText)findViewById(R.id.txtMsg);
        Intent intent = new Intent();
        intent.putExtra("msg",(CharSequence)st.getText().toString());
        intent.setAction("com.tutlane.CUSTOM_INTENT");
        sendBroadcast(intent);
    }
}

```

Now we need to register our broadcast receiver in android manifest file (**AndroidManifest.xml**) using **<receiver>** attribute like as shown below.

AndroidManifest.xml

```

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

    <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>
        <receiver android:name="MyBroadcastReceiver">
            <intent-filter>
                <action android:name="com.tutlane.CUSTOM_INTENT">
            </action>
            </intent-filter>
        </receiver>
    </application>
</manifest>

```

Output of Android BroadcastReceiver Example

When we run above example in android emulator we will get a result like as shown below



Content Providers

In android, **Content Provider** will act as a central repository to store the applications data in one place and make that data available for different applications to access whenever it's required.

In android, we can configure Content Providers to allow other applications securely access and modify our app data based on our requirements.

Generally, the **Content Provider** is a part of an android application and it will act as more like relational database to store the app data. We can perform a multiple operations like insert, update, delete and edit on the data stored in content provider using **insert()**, **update()**, **delete()** and **query()** methods.

In android, we can use content provider whenever we want to share our app data with other apps and it allow us to make a modifications to our application data without effecting other applications which depends on our app.

In android, content provider is having different ways to store app data. The app data can be stored in a SQLite database or in files or even over a network based on our requirements. By using content providers we can manage data such as audio, video, images and personal contact information.

We have a different type of access permissions available in content provider to share the data. We can set a restrict access permissions in content provider to restrict data access limited to only our application and we can configure different permissions to read or write a data.

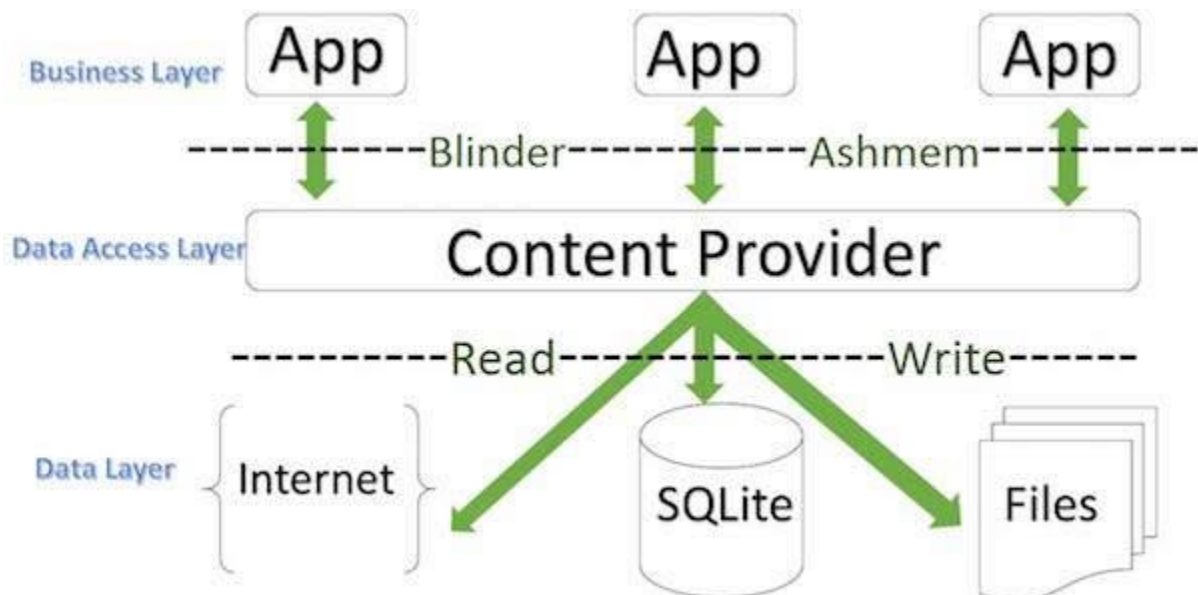
Access Data from Content Provider

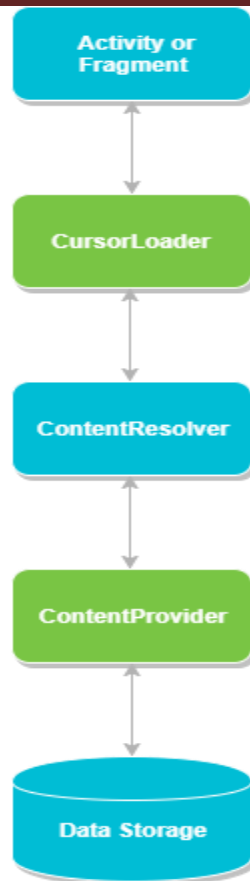
To access a data from content provider, we need to use `ContentResolver` object in our application to communicate with the provider as a client. The `ContentResolver` object will communicate with the provider object (`ContentProvider`) which is implemented by instance of class.

Generally, in android to send a request from UI to `ContentResolver` we have another object called **CursorLoader** which is used to run the query asynchronously in background. In android application the UI components such as Activity or Fragment will call a **CursorLoader** to query and get a required data from `ContentProvider` using `ContentResolver`.

The `ContentProvider` object will receive a data requests from client, performs the requested actions (create, update, delete, retrieve) and return the result.

Following is the pictorial representation of requesting an operation from UI using Activity or Fragment to get the data from `ContentProvider` object.





This is how the interaction will happen between android application UI and content providers to perform required actions to get a data.

Content URIs

In android, **Content URI** is an URI which is used to query a content provider to get the required data. The Content URIs will contain the name of entire provider (**authority**) and the name that points to a table (**path**).

Generally the format of URI in android applications will be like as shown below

```
content://authority/path
```

Following are the details about various parts of an URI in android application.

content:// - The string **content://** is always present in the URI and it is used to represent the given URI is a content URI.

authority - It represents the name of content provider, for example phone, contacts, etc. and we need to use fully qualified name for third party content providers like com.tutlane.contactprovider

path - It represents the table's path.

The ContentResolver object use the URI's **authority** to find the appropriate provider and send the query objects to the correct provider. After that ContentProvider uses the **path** of content URI to choose the right table to access.

Following is the example of simple example of URI in android applications.

```
content://contacts_info/users
```

Here the string **content://** is used to represent URI is a content URI, **contacts_info** string is the name of provider's authority and **users** string is the table's path.

Creating a Content Provider

To create a content provider in android applications we should follow below steps.

- We need to create a content provider class that extends the ContentProvider base class.
- We need to define our content provider URI to access the content.
- The ContentProvider class defines a six abstract methods (insert(), update(), delete(), query(), getType()) which we need to implement all these methods as a part of our subclass.
- We need to register our content provider in AndroidManifest.xml using **<provider>** tag.

Following are the list of methods which need to implement as a part of ContentProvider class.



query() - It receives a request from the client. By using arguments it will get a data from requested table and return the data as a **Cursor** object.

insert() - This method will insert a new row into our content provider and it will return the content URI for newly inserted row.

update() - This method will update an existing rows in our content provider and it return the number of rows updated.

delete() - This method will delete the rows in our content provider and it return the number of rows deleted.



getType() - This method will return the MIME type of data to given content URI.

onCreate() - This method will initialize our provider. The android system will call this method immediately after it creates our provider.

Android Content Provider Example

Following is the example of using **Content Provider** in android applications. Here we will create our own content provider to insert and access data in android application.

Create a new android application using android studio and give names as ContentProvider. In case if you are not aware of creating an app in android studio check this article [Android Hello World App](#).

Now we need to create our own content provider file **UserProvider.java** in `\src\main\java\com.tutlane.contentprovider` path to define our actual provider and associated methods for that right click on your application folder  Go to **New**  select **Java Class** and give name as **UserProvider.java**.

Once we create a new file **UserProvider.java**, open it and write the code like as shown below

UserProvider.java

```
package com.tutlane.contentprovider;

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.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteException;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteQueryBuilder;
import android.net.Uri;
import java.util.HashMap;

/**
```

* Created by sureshdasari on 29-07-2017.

*/

```
public class UsersProvider extends ContentProvider {
    static final String PROVIDER_NAME = "com.tutlane.contentprovider.UserProvider";
    static final String URL = "content://" + PROVIDER_NAME + "/users";
    static final Uri CONTENT_URI = Uri.parse(URL);

    static final String id = "id";
    static final String name = "name";
    static final int uriCode = 1;
    static final UriMatcher uriMatcher;
    private static HashMap<String, String> values;
    static {
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI(PROVIDER_NAME, "users", uriCode);
        uriMatcher.addURI(PROVIDER_NAME, "users/*", uriCode);
    }

    @Override
    public String getType(Uri uri) {
        switch (uriMatcher.match(uri)) {
            case uriCode:
                return "vnd.android.cursor.dir/users";
            default:
                throw new IllegalArgumentException("Unsupported URI: " + uri);
        }
    }

    @Override
    public boolean onCreate() {
        Context context = getContext();
        DatabaseHelper dbHelper = new DatabaseHelper(context);
        db = dbHelper.getWritableDatabase();
        if (db != null) {
            return true;
        }
        return false;
    }

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

        switch (uriMatcher.match(uri)) {
            case uriCode:
                qb.setProjectionMap(values);
                break;
        }
    }
}
```

```
        default:
            throw new IllegalArgumentException("Unknown URI " + uri);
        }
    if (sortOrder == null || sortOrder == "") {
        sortOrder = id;
    }
    Cursor c = qb.query(db, projection, selection, selectionArgs, null,
        null, sortOrder);
    c.setNotificationUri(getContext().getContentResolver(), uri);
    return c;
}
@Override
public Uri insert(Uri uri, ContentValues values) {
    long rowID = db.insert(TABLE_NAME, "", values);
    if (rowID > 0) {
        Uri _uri = ContentUris.withAppendedId(CONTENT_URI, rowID);
        getContext().getContentResolver().notifyChange(_uri, null);
        return _uri;
    }
    throw new SQLiteException("Failed to add a record into " + uri);
}
@Override
public int update(Uri uri, ContentValues values, String selection,
    String[] selectionArgs) {
    int count = 0;
    switch (uriMatcher.match(uri)) {
        case uriCode:
            count = db.update(TABLE_NAME, values, selection, selectionArgs);
            break;
        default:
            throw new IllegalArgumentException("Unknown URI " + uri);
    }
    getContext().getContentResolver().notifyChange(uri, null);
    return count;
}
@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
    int count = 0;
    switch (uriMatcher.match(uri)) {
        case uriCode:
            count = db.delete(TABLE_NAME, selection, selectionArgs);
            break;
        default:
            throw new IllegalArgumentException("Unknown URI " + uri);
    }
    getContext().getContentResolver().notifyChange(uri, null);
    return count;
}
private SQLiteDatabase db;
static final String DATABASE_NAME = "EmpDB";
```

```

static final String TABLE_NAME = "Employees";
static final int DATABASE_VERSION = 1;
static final String CREATE_DB_TABLE = " CREATE TABLE " + TABLE_NAME
    + " (id INTEGER PRIMARY KEY AUTOINCREMENT, "
    + " name TEXT NOT NULL);";

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 " + TABLE_NAME);
        onCreate(db);
    }
}
}

```

Now open an **activity_main.xml** file from `\src\main\res\layout` path and write the 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"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Name"
    android:layout_marginLeft="100dp"
    android:layout_marginTop="100dp"/>
<EditText
    android:id="@+id/txtName"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="100dp"
    android:ems="10"/>
<Button
    android:id="@+id/btnAdd"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onClickAddDetails"

```

```
        android:layout_marginLeft="100dp"
        android:text="Add User"/>

<Button
    android:id="@+id/btnRetrieve"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onClickShowDetails"
    android:layout_marginLeft="100dp"
    android:text="Show Users"/>
<TextView
    android:id="@+id/res"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="100dp"
    android:clickable="false"
    android:ems="10"/>
</LinearLayout>
```

Now open **MainActivity.java** file and write the code like as shown below

MainActivity.java

```
package com.tutlane.contentprovider;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.net.Uri;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.View;
import android.view.inputmethod.InputMethodManager;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        InputMethodManager imm = (InputMethodManager) getSystemService(Context.INPUT_METHOD_SERVICE);
        imm.hideSoftInputFromWindow(getCurrentFocus().getWindowToken(), 0);
        return true;
    }
}
```



```

    }
    public void onClickAddDetails(View view) {
        ContentValues values = new ContentValues();
        values.put(UsersProvider.name, ((EditText) findViewById(R.id.txtName)).getText().toString());
        getContentResolver().insert(UsersProvider.CONTENT_URI, values);
        Toast.makeText(getApplicationContext(), "New Record Inserted", Toast.LENGTH_LONG).show();
    }

    public void onClickShowDetails(View view) {
        // Retrieve employee records
        TextView resultView= (TextView) findViewById(R.id.res);
        Cursor cursor = getContentResolver().query(Uri.parse("content://com.tutlane.contentprovider.UserProvider/users"), null, null, null, null);
        if(cursor.moveToFirst()) {
            StringBuilder strBuild=new StringBuilder();
            while (!cursor.isAfterLast()) {
                strBuild.append("\n"+cursor.getString(cursor.getColumnIndex("id"))+ "-" + cursor.getString(cursor.getColumnIndex("name")));
                cursor.moveToNext();
            }
            resultView.setText(strBuild);
        }
        else {
            resultView.setText("No Records Found");
        }
    }
}

```

We need to include this newly created content provider in android manifest file (**ActivityManifest.xml**) using **<provider>** attribute like as shown below.

AndroidManifest.xml

```

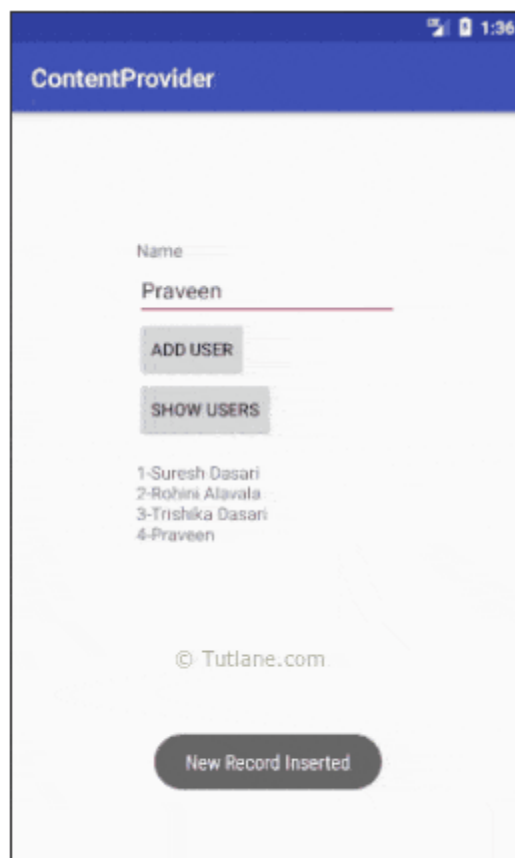
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.tutlane.contentprovider">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

```

```
<provider
  android:authorities="com.tutlane.contentprovider.UserProvider"
  android:name=".UsersProvider">
</provider>
</application>
</manifest>
```

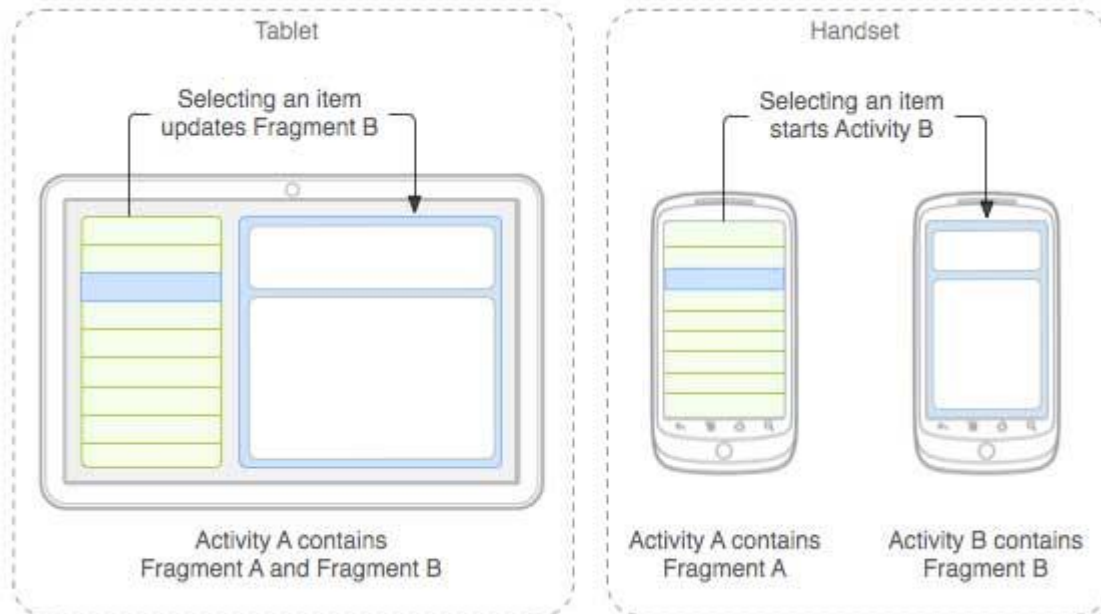
Output of Android Content Provider Example

When we run above example in android emulator we will get a result like as shown below



Fragment In Android Studio

In Android, Fragment is a part of an activity which enables more modular activity design. It will not be wrong if we say a fragment is a kind of sub-activity. It represents behaviour or a portion of user interface in an Activity. We can combine multiple Fragments in Single Activity to build a multi panel UI and reuse a Fragment in multiple Activities. We always need to embed Fragment in an activity and the fragment lifecycle is directly affected by the host activity's lifecycle.



We can create Fragments by extending Fragment class or by inserting a Fragment into our Activity layout by declaring the Fragment in the activity's layout file, as a <fragment> element. We can manipulate each Fragment independently, such as add or remove them.

While performing Fragment Transaction we can add a Fragment into back stack that's managed by the Activity. back stack allow us to reverse a Fragment transaction on pressing Back button of device. For Example if we replace a Fragment and add it in back stack then on pressing the Back button on device it display the previous Fragment.

Some Important Points about Fragment in Android:

1. Fragments were added in Honeycomb version of Android i.e API version 11.
2. We can add, replace or remove Fragment's in an Activity while the activity is running. For performing these operations we need a Layout(Relative Layout, FrameLayout or any other layout) in xml file and then replace that layout with the required Fragment.
3. Fragments has its own layout and its own behaviour with its own life cycle callbacks.
4. Fragment can be used in multiple activities.
5. We can also combine multiple Fragments in a single activity to build a multi-plane UI.

Need of Fragments in Android:

Before the introduction of Fragment's we can only show a single Activity on the screen at one given point of time so we were not able to divide the screen and control different parts separately. With the help of Fragment's we can divide the screens in different parts and controls different parts separately.

By using Fragments we can comprise multiple Fragments in a single Activity. Fragments have their own events, layouts and complete life cycle. It provide flexibility and also removed the limitation of single Activity on the screen at a time.

Basic Fragment Code in XML:

```
<fragment  
android:id="@+id/fragments"  
android:layout_width="match_parent"  
android:layout_height="match_parent" />
```

Create A Fragment Class In Android Studio:

For creating a Fragment firstly we extend the Fragment class, then override key lifecycle methods to insert our app logic, similar to the way we would with an Activity class. While creating a Fragment we must use onCreateView() callback to define the layout and in order to run a Fragment.

```
import android.os.Bundle;  
import android.support.v4.app.Fragment;  
import android.view.LayoutInflater;  
import android.view.ViewGroup;  
  
public class FirstFragment extends Fragment {  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container,  
        Bundle savedInstanceState) {  
        // Inflate the layout for this fragment  
        return inflater.inflate(R.layout.fragment_first, container, false);  
    }  
}
```

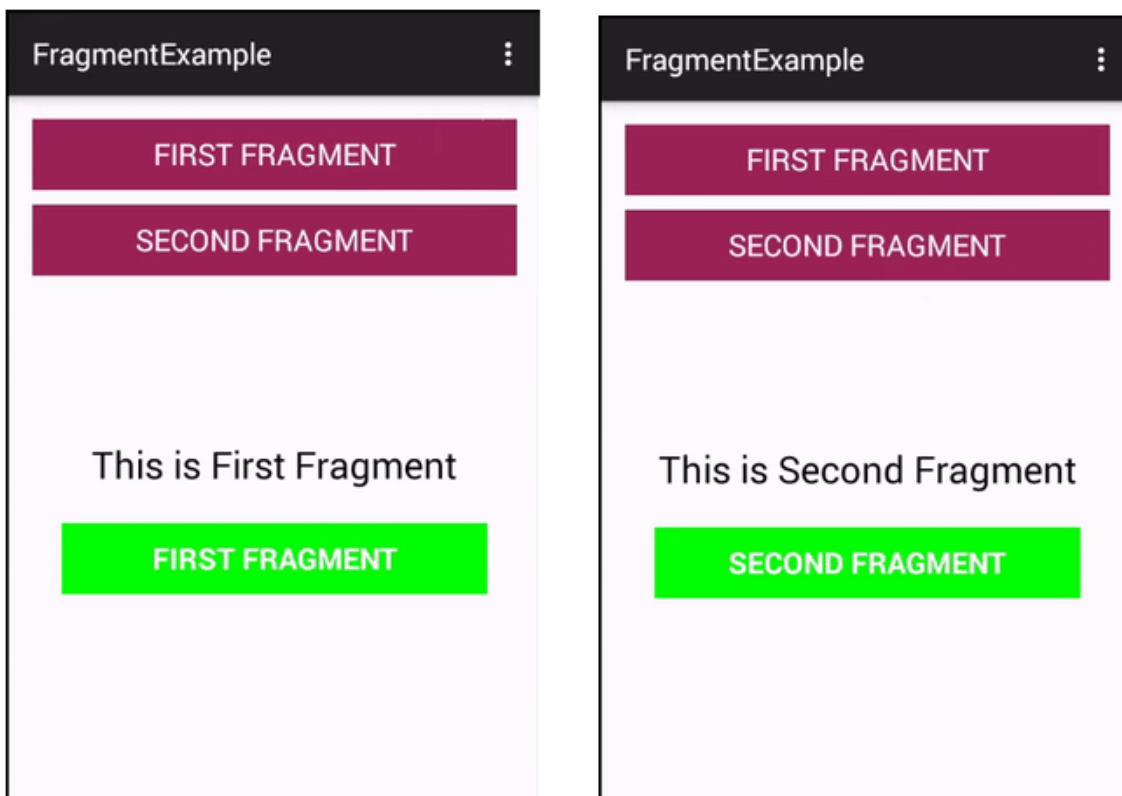
Here the inflater parameter is a LayoutInflater used to inflate the layout, container parameter is the parent ViewGroup (from the activity's layout) in which our Fragment layout will be inserted.

The savedInstanceState parameter is a Bundle that provides data about the previous instance of the Fragment. The inflate() method has three arguments first one is the resource layout which we want to inflate, second is the ViewGroup to be the parent of the inflated layout. Passing the container is important in order for the system to apply layout parameters to the root view of the inflated layout, specified by the parent view in which it's going and the third parameter is a boolean value indicating whether the inflated layout should be attached to the ViewGroup (the second parameter) during inflation.

Implementation of Fragment in Android Require Honeycomb (3.0) or Later:

Fragments were added in in Honeycomb version of Android i.e API version 11. There are some primary classes related to Fragment's are:

- 1. FragmentActivity:** The base class for all activities using compatibility based Fragment (and loader) features.
- 2. Fragment:** The base class for all Fragment definitions
- 3. FragmentManager:** The class for interacting with Fragment objects inside an activity
- 4. FragmentTransaction:** The class for performing an atomic set of Fragment operations such as Replace or Add a Fragment.

**Fragment Example 1 In Android Studio:**

Below is the example of Fragment's. In this example we create two Fragments and load them on the click of Button's. We display two Button's and a FrameLayout in our Activity and perform setOnClickListener event on both Button's. On the click of First Button we replace the First Fragment and on click of Second Button we replace the Second Fragment with the layout(FrameLayout). In the both Fragment's we display a TextView and a Button and onclick of Button we display the name of the Fragment with the help of Toast.

Below you can download code, see final output and read step by step explanation:

Step 1: Create a new project and name it FragmentExample

Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

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

```
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
android: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=".MainActivity">
<!-- display two Button's and a FrameLayout to replace the Fragment's -->
<Button
android:id="@+id/firstFragment"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:background="@color/button_background_color"
android:text="First Fragment"
android:textColor="@color/white"
android:textSize="20sp" />

<Button
android:id="@+id/secondFragment"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginTop="10dp"
android:background="@color/button_background_color"
android:text="Second Fragment"
android:textColor="@color/white"
android:textSize="20sp" />

<FrameLayout
android:id="@+id/frameLayout"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_marginTop="10dp" />
```

```
</LinearLayout>
```

Step 3: Open src -> package -> MainActivity.java

In this step we open MainActivity and add the code for initiate the Button's. After that we perform setOnClickListener event on both Button's. On the click of First Button we replace the First Fragment and on click of Second Button we replace the Second Fragment with the layout(FrameLayout). For replacing a Fragment with FrameLayout firstly we create a Fragment Manager and then begin the transaction using Fragment Transaction and finally replace the Fragment with the layout i.e FrameLayout.

```
package com.abhiandroid.fragmentexample;

import android.app.Fragment;
import android.app.FragmentManager;
import android.app.FragmentTransaction;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

    Button firstFragment, secondFragment;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // get the reference of Button's
        firstFragment = (Button) findViewById(R.id.firstFragment);
        secondFragment = (Button) findViewById(R.id.secondFragment);

        // perform setOnClickListener event on First Button
        firstFragment.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
```

```
// load First Fragment
loadFragment(new FirstFragment());
}
});

// perform setOnClickListener event on Second Button
secondFragment.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
// load Second Fragment
loadFragment(new SecondFragment());
}
});

}

private void loadFragment(Fragment fragment) {
// create a FragmentManager
FragmentManager fm = getFragmentManager();
// create a FragmentTransaction to begin the transaction and replace the Fragment
FragmentTransaction fragmentTransaction = fm.beginTransaction();
// replace the FrameLayout with new Fragment
fragmentTransaction.replace(R.id.frameLayout, fragment);
fragmentTransaction.commit(); // save the changes
}
}
```

Step 4: Now we need 2 fragments and 2 xml layouts. So create two fragments by right click on your package folder and create classes and name them as FirstFragment and SecondFragment and add the following code respectively.

FirstFragment.class

In this Fragment firstly we inflate the layout and get the reference of Button. After that we perform setOnClickListener event on Button so whenever a user click on the button a message "First Fragment" is displayed on the screen by using a Toast.

```
package com.abhiandroid.fragmentexample;
```



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

public class FirstFragment extends Fragment {

    View view;
    Button firstButton;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        view = inflater.inflate(R.layout.fragment_first, container, false);
        // get the reference of Button
        firstButton = (Button) view.findViewById(R.id.firstButton);
        // perform setOnClickListener on first Button
        firstButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // display a message by using a Toast
                Toast.makeText(getActivity(), "First Fragment", Toast.LENGTH_LONG).show();
            }
        });
        return view;
    }
}
```

SecondFragment.class

In this Fragment firstly we inflate the layout and get the reference of Button. After that we perform `setOnClickListener` event on Button so whenever a user click on the button a message "Second Fragment" is displayed on the screen by using a Toast.

```
package com.abhiandroid.fragmentexample;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.Toast;

public class SecondFragment extends Fragment {
    View view;
    Button secondButton;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        view = inflater.inflate(R.layout.fragment_second, container, false);
        // get the reference of Button
        secondButton = (Button) view.findViewById(R.id.secondButton);
        // perform setOnClickListener on second Button
        secondButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // display a message by using a Toast
                Toast.makeText(getActivity(), "Second Fragment", Toast.LENGTH_LONG).show();
            }
        });
        return view;
    }
}
```

```
}  
}
```

Step 5: Now create 2 xml layouts by right clicking on res/layout -> New -> Layout Resource File and name them as fragment_first and fragment_second and add the following code in respective files.

Here we will design the basic simple UI by using TextView and Button in both xml's.

fragment_first.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
xmlns:tools="http://schemas.android.com/tools"  
android:layout_width="match_parent"  
android:layout_height="match_parent"  
tools:context="com.abhiandroid.fragmentexample.FirstFragment">  
  
<!--TextView and Button displayed in First Fragment -->  
<TextView  
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:layout_centerHorizontal="true"  
android:layout_marginTop="100dp"  
android:text="This is First Fragment"  
android:textColor="@color/black"  
android:textSize="25sp" />  
  
<Button  
android:id="@+id/firstButton"  
android:layout_width="fill_parent"  
android:layout_height="wrap_content"  
android:layout_centerInParent="true"  
android:layout_marginLeft="20dp"  
android:layout_marginRight="20dp"  
android:background="@color/green"  
android:text="First Fragment"  
android:textColor="@color/white"
```

```
android:textSize="20sp"  
android:textStyle="bold" />  
</RelativeLayout>
```

fragment_second.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
xmlns:tools="http://schemas.android.com/tools"  
android:layout_width="match_parent"  
android:layout_height="match_parent"  
tools:context="com.abhiandroid.fragmentexample.SecondFragment">  
  
<!--TextView and Button displayed in Second Fragment -->  
<TextView  
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:layout_centerHorizontal="true"  
android:layout_marginTop="100dp"  
android:text="This is Second Fragment"  
android:textColor="@color/black"  
android:textSize="25sp" />  
  
<Button  
android:id="@+id/secondButton"  
android:layout_width="fill_parent"  
android:layout_height="wrap_content"  
android:layout_centerInParent="true"  
android:layout_marginLeft="20dp"  
android:layout_marginRight="20dp"  
android:background="@color/green"  
android:text="Second Fragment"  
android:textColor="@color/white"  
android:textSize="20sp"  
android:textStyle="bold" />
```

```
</RelativeLayout>
```

Step 6: Open res ->values ->colors.xml

In this step we define the color's that used in our xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<!-- color's used in our project -->
<color name="black">#000</color>
<color name="green">#0f0</color>
<color name="white">#fff</color>
<color name="button_background_color">#925</color>
</resources>
```

Step 7: Open AndroidManifest.xml

In this step we show the Android Manifest file in which do nothing because we need only one Activity i.e MainActivity which is already defined in it. In our project we create two Fragment's but we don't need to define the Fragment's in manifest because Fragment is a part of an Activity.

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

<application
android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:theme="@style/AppTheme" >
<activity
android:name=".MainActivity"
android:label="@string/app_name" >
<intent-filter>
<action android:name="android.intent.action.MAIN" />

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

```
</intent-filter>  
</activity>  
</application>  
</manifest>
```

Step 8: Now run the App and you will two button. Clicking on first button shows First Fragment and on click of Second Button shows the Second Fragment which is actually replacing layout (FrameLayout).

Android Service

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 has 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).

Features of Service

- Service is an Android Component without an UI
- It is used to perform long running operations in background. Services run indefinitely unless they are explicitly stopped or destroyed
- It can be started by any other application component. Components can even in fact bind to a service to perform Interprocess- Communication
- It can still be running even if the application is killed unless it stops itself by calling *stopself()* or is stopped by a Android component by calling *stopService()*.
- If not stopped it goes on running unless is terminated by Android due to resource shortage
- The `android.app.Service` is subclass of `ContextWrapper` class.

Android platform service

The Android platform provides and runs predefined system services and every Android application can use them, given the right permissions. These system services are usually exposed via a specific Manager class. Access to them can be gained via the `getSystemService()` method.

Permission

The purpose of a *permission* is to protect the privacy of an Android user. Android apps must request permission to access sensitive user data (such as contacts and SMS), as well as certain system features (such as camera and internet). Depending on the feature, the system might grant the permission automatically or might prompt the user to approve the request.

Add permissions to the manifest

On all versions of Android, to declare that your app needs a permission, put a `<uses-permission>` element in your app manifest, as a child of the top-level `<manifest>` element. For example, an app that needs to access the internet would have this line in the manifest:

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

    <uses-permission android:name="android.permission.INTERNET"/>
    <!-- other permissions go here -->

    <application ...>
        ...
    </application>
</manifest>
```

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.

1. Started
2. Bound

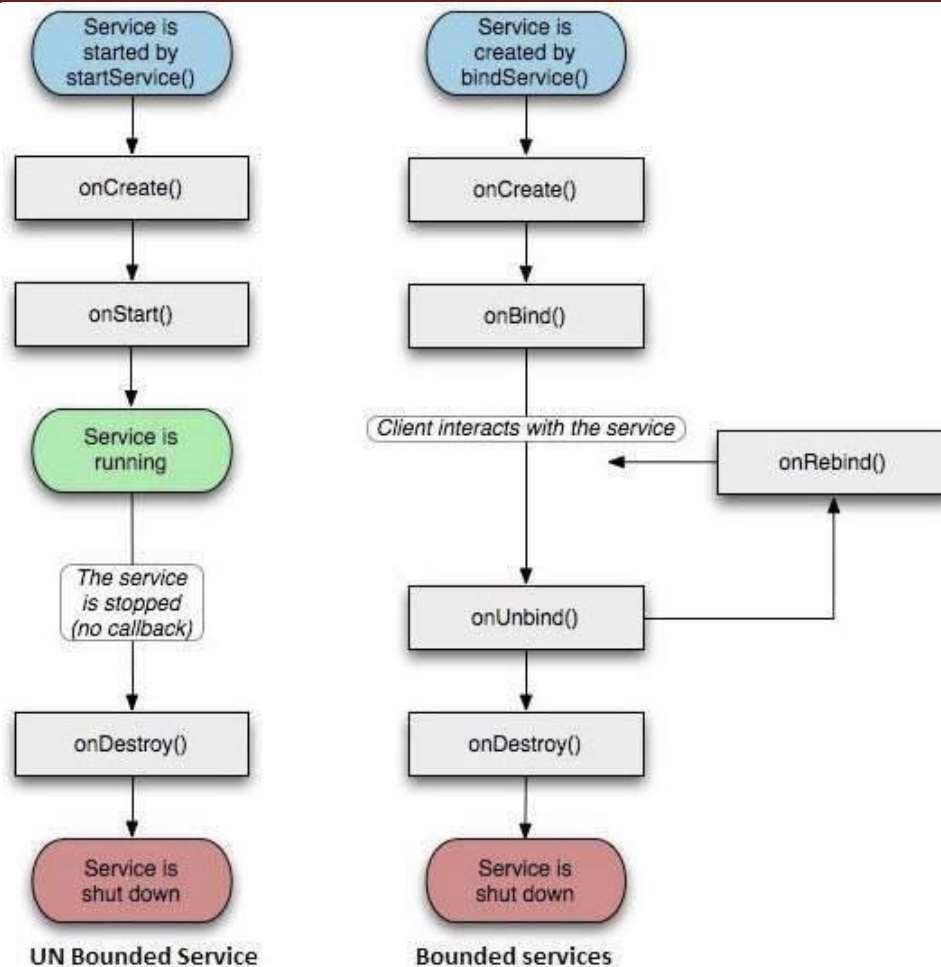
1) Started Service

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.



Like any other components service also has callback methods. These will be invoked while the service is running to inform the application of its state. Implementing these in your custom service would help you in performing the right operation in the right state.

onCreate()

This is the first callback which will be invoked when any component starts the service. If the same service is called again while it is still running this method won't be invoked. Ideally one time setup and initializing should be done in this callback.

onStartCommand()

This callback is invoked when service is started by any component by calling `startService()`. It basically indicates that the service has started and can now run indefinitely.

onBind()

This is invoked when any component starts the service by calling `onBind`.

onUnbind()

This is invoked when all the clients are disconnected from the service.

onRebind()

This is invoked when new clients are connected to the service. It is called after onUnbind

onDestroy()

This is a final clean up call from the system. This is invoked just before the service is being destroyed. Could be very useful to cleanup any resources such as threads, registered listeners, or receivers.

Android 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.

activity_main.xml

Drag the 3 buttons from the palette, now the activity_main.xml will look like this:

File: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="example.javatpoint.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>

```

activity_next.xml

It is the layout file of next activity.

File: activity_next.xml

It contains only one textview displaying the message Next Page

```

<?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"
    tools:context="example.javatpoint.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>

```

Service class

Now create the service implementation class by inheriting the Service class and overriding its callback methods.

File: MyService.java

```
package example.javatpoint.com.androidservice;

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();
    }
}
```

Activity class

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.

File: MainActivity.java

```
package example.javatpoint.com.androidservice;

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;
        }
    }
}
```

NextPage class

Now, create another activity.

File: NextPage.java

```
package example.javatpoint.com.androidservice;
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);
    }
}
```

Declare the Service in the AndroidManifest.xml file

Finally, declare the service in the manifest file.

File: AndroidManifest.xml

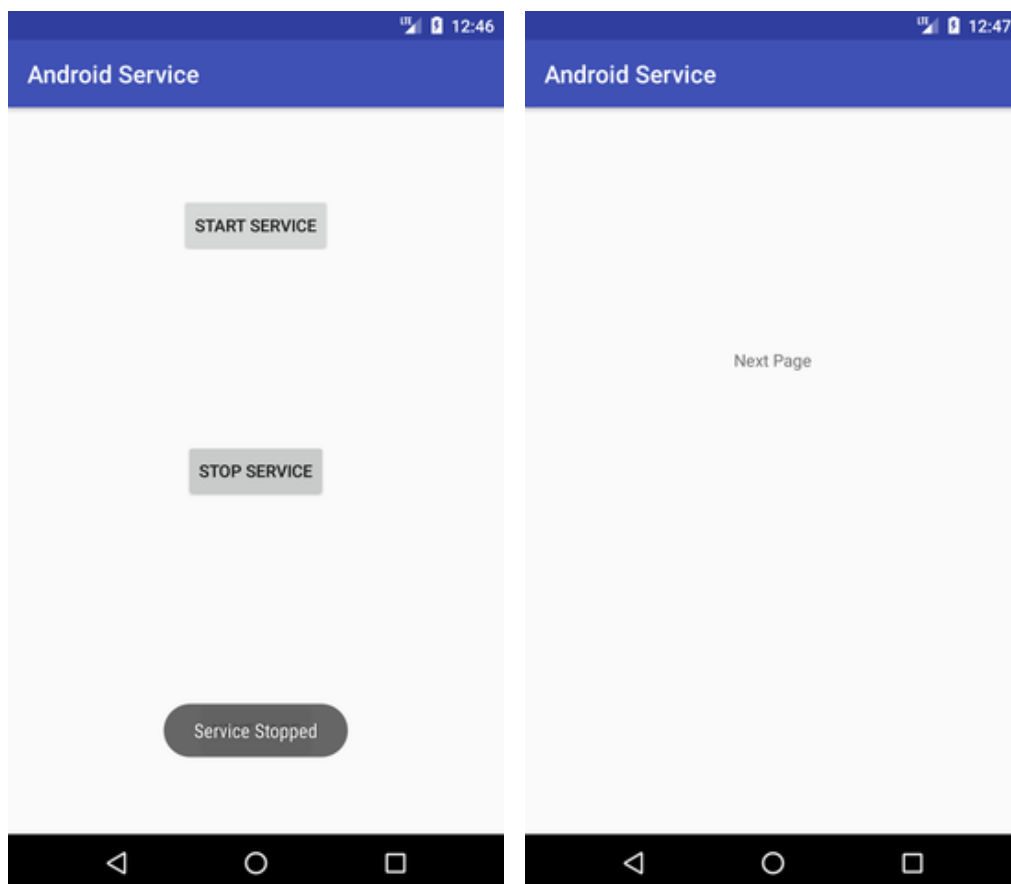
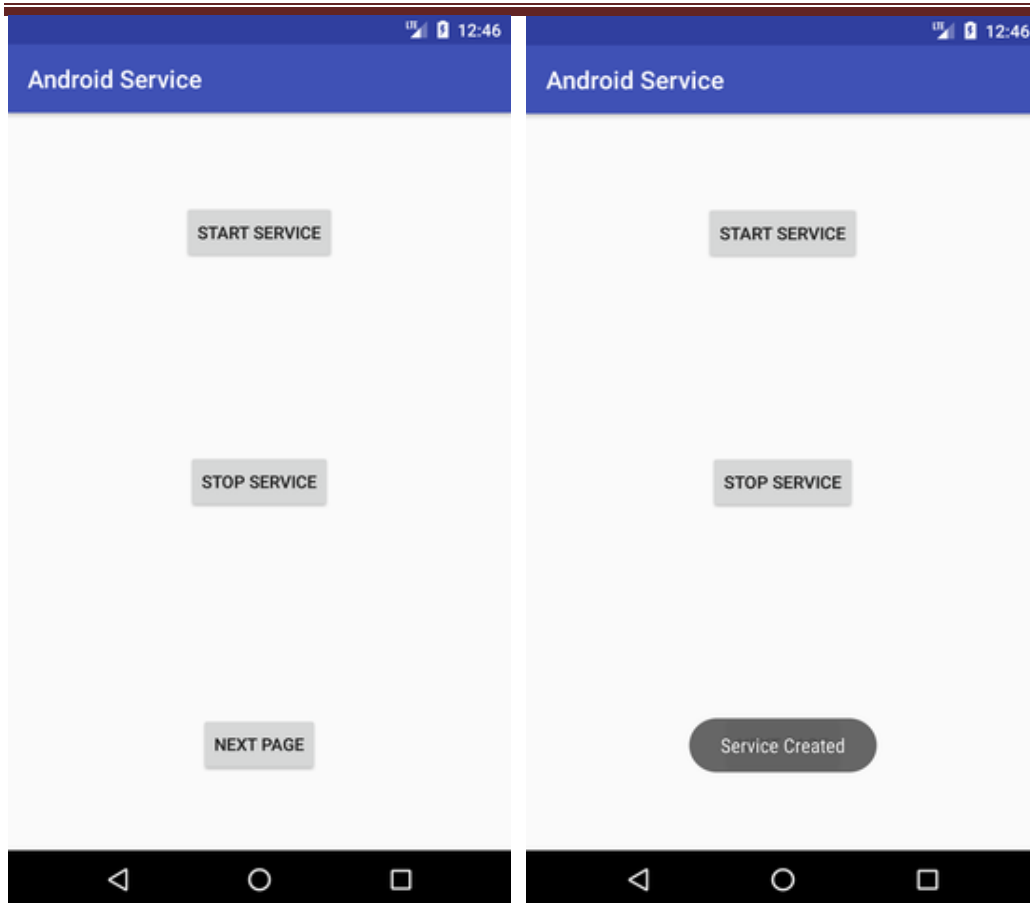
Let's see the complete AndroidManifest.xml file

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

Output:



Android System Architecture

The android system consists of five layers and each layer consists of some core components. Figure 1 shows the architecture of android. From top to down, the core components are: Applications, Application Framework, Native C libraries, Android Runtime Environment (JVM), HAL (Hardware Abstract Layer), Linux Kernel.

1) Applications. Application layer consists of many core Java-based applications, such as calendar, web browser, SMS application, E-mail, etc.

2) Application Framework. Application framework consists of many components and Java classes to allow android application developers to develop various kinds of applications. By using Java language, it hides the internal implementation of system core functions and provides the developers an easy-use API. Basically, it includes Java core class and some special components of android. Some typical components are as follows: View (List, Grids), Content Provider, Resource Manager, Activity Manager.

3) Native C Libraries. In Native C library layer, it consists of many C/C++ libraries. And the core functions of android are implemented by those libraries. Some typical core libraries are as follows: Bionic C lib, OpenCore, SQLite, Surface Manager, WebKit, 3D library.

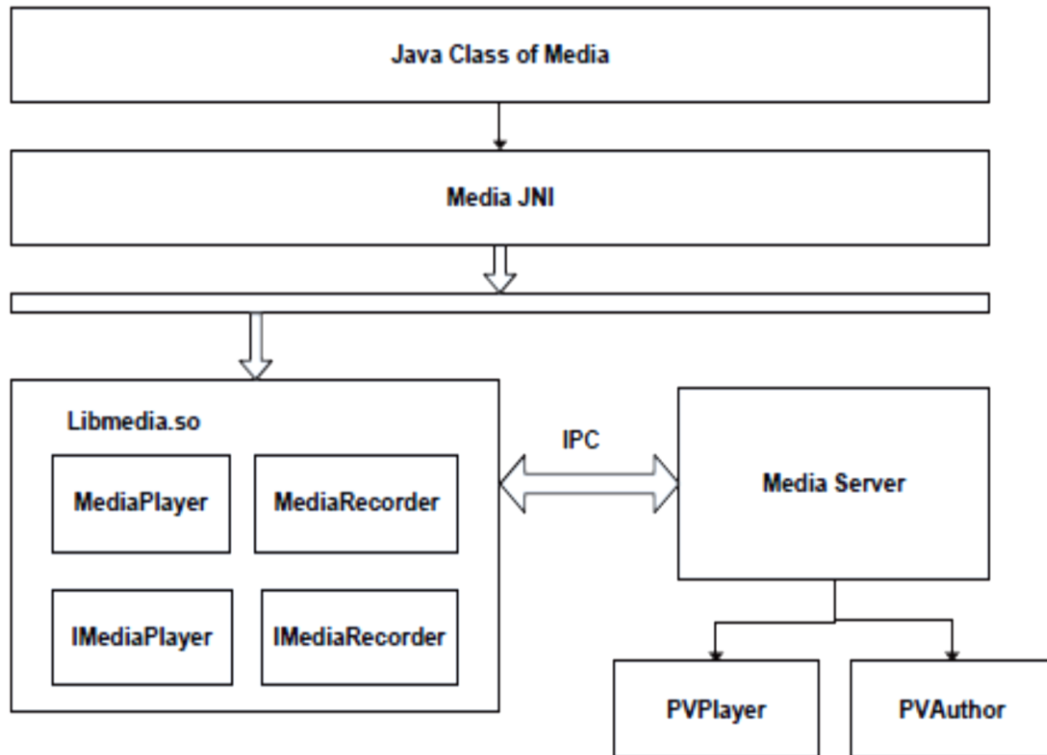
4) Android Runtime Environment. Runtime environment consists of Dalvik Java virtual machine and some implementations of Java core libraries.

5) HAL. This layer abstracts different kinds of hardwares and provides an unified program interface to Native C libraries. HAL can make Android port on different platforms more easily.

6) Linux Kernel. Android's core system functions (e.g., safety management, RAM management, process management, network stack) depend on Linux kernels.

Android Multimedia framework

The android multimedia system includes multimedia applications, multimedia framework, OpenCore engine and hardware abstract for audio/video input/output devices. And the goal of the android multimedia framework is to provide a consistent interface for Java services. The multimedia framework consists of several core dynamic libraries such as libmediajni, libmedia, libmediaplayservice and so on. A general multimedia framework architecture is shown in Figure



Java classes call the Native C library Libmedia through Java JNI (Java Native Interface). Libmedia library communicates with Media Server guard process through Android’s Binder IPC (inter process communication) mechanism. Media Server process creates the corresponding multimedia service according to the Java multimedia applications. The whole communication between Libmedia and Media Server forms a Client/Server model. In Media Server guard process, it calls OpenCore multimedia engine to realize the specific multimedia processing functions. And the OpenCore engine refers to the PVPlayer and PVAuthor.

Play Audio and Video

We can play and control the audio files in android by the help of **MediaPlayer class**.

Here, we are going to see a simple example to play the audio file. In the next page, we will see the example to control the audio playback like start, stop, pause etc.

MediaPlayer class

The **android.media.MediaPlayer** class is used to control the audio or video files.

Methods of MediaPlayer class

There are many methods of MediaPlayer class. Some of them are as follows:

Method	Description
public void setDataSource(String path)	Sets the data source (file path or http url) to use.
public void prepare()	Prepares the player for playback synchronously.

public void start()	It starts or resumes the playback.
public void stop()	It stops the playback.
public void pause()	It pauses the playback.
public boolean isPlaying()	Checks if media player is playing.
public void seekTo(int millis)	Seeks to specified time in miliseconds.
public void setLooping(boolean looping)	Sets the player for looping or non-looping.
public boolean isLooping()	Checks if the player is looping or non-looping.
public void selectTrack(int index)	It selects a track for the specified index.
public int getCurrentPosition()	Returns the current playback position.
public int getDuration()	Returns duration of the file.
public void setVolume(float leftVolume, float rightVolume)	Sets the volume on this player.

Activity class

Let's write the code of to play the audio file. Here, we are going to play maine.mp3 file located inside the sdcard/Music directory.

File: MainActivity.java

```
package com.example.audiomediaplayer1;

import android.media.MediaPlayer;
import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.MediaController;
import android.widget.VideoView;

public class MainActivity extends Activity {

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

        MediaPlayer mp=new MediaPlayer();
        try{
            mp.setDataSource("/sdcard/Music/maine.mp3");//Write your location here
            mp.prepare();
            mp.start();

        }catch(Exception e){e.printStackTrace();}

    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
```

```

// Inflate the menu; this adds items to the action bar if it is present.
getMenuInflater().inflate(R.menu.activity_main, menu);
return true;
}
}

```

Android MediaPlayer Example of controlling the audio

Let's see a simple example to start, stop and pause the audio play.

activity_main.xml

Drag three buttons from palette to start, stop and pause the audio play. Now the xml file will look like this:

File: MainActivity.java

```

<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=".MainActivity" >

```

```

<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_marginTop="30dp"
    android:text="Audio Controller" />

```

```

<Button
    android:id="@+id/button1"
    style="?android:attr/buttonStyleSmall"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView1"
    android:layout_below="@+id/textView1"
    android:layout_marginTop="48dp"
    android:text="start" />

```

```
<Button
  android:id="@+id/button2"
  style="?android:attr/buttonStyleSmall"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignTop="@+id/button1"
  android:layout_toRightOf="@+id/button1"
  android:text="pause" />
```

```
<Button
  android:id="@+id/button3"
  style="?android:attr/buttonStyleSmall"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignTop="@+id/button2"
  android:layout_toRightOf="@+id/button2"
  android:text="stop" />
```

```
</RelativeLayout>
```

Activity class

Let's write the code to start, pause and stop the audio player.

File: MainActivity.java

```
package com.example.audioplay;

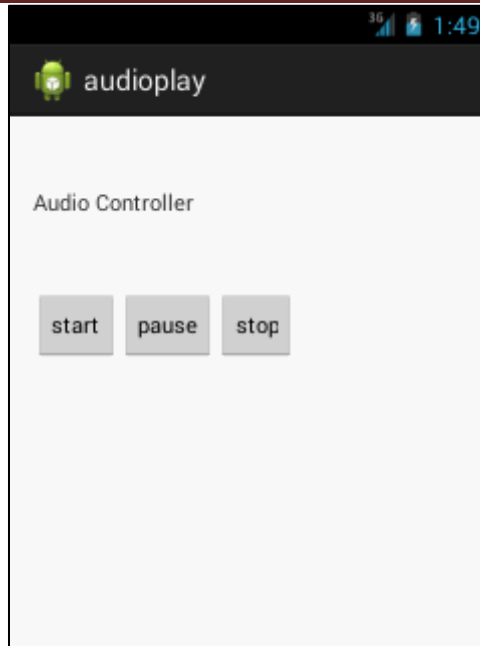
import android.media.MediaPlayer;
import android.os.Bundle;
import android.os.Environment;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class MainActivity extends Activity {
  Button start,pause,stop;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
```

```
start=(Button)findViewById(R.id.button1);
pause=(Button)findViewById(R.id.button2);
stop=(Button)findViewById(R.id.button3);
//creating media player
final MediaPlayer mp=new MediaPlayer();
try{
    //you can change the path, here path is external directory(e.g. sdcard) /Music/main
.mp3
    mp.setDataSource(Environment.getExternalStorageDirectory().getPath()+"/Music/main
e.mp3");

    mp.prepare();
} catch (Exception e){e.printStackTrace();}

start.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        mp.start();
    }
});
pause.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        mp.pause();
    }
});
stop.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        mp.stop();
    }
});
}
}
```



Android Video Player Example

By the help of **MediaController** and **VideoView** classes, we can play the video files in android.

MediaController class

The **android.widget.MediaController** is a view that contains media controls like play/pause, previous, next, fast-forward, rewind etc.

VideoView class

The **android.widget.VideoView** class provides methods to play and control the video player. The commonly used methods of VideoView class are as follows:

Method	Description
public void setMediaController(MediaController controller)	Sets the media controller to the video view.
public void setVideoURI (Uri uri)	Sets the URI of the video file.
public void start()	Starts the video view.
public void stopPlayback()	Stops the playback.
public void pause()	Pauses the playback.
public void suspend()	Suspends the playback.
public void resume()	Resumes the playback.
public void seekTo(int millis)	Seeks to specified time in milliseconds.

activity_main.xml

Drag the VideoView from the palette, now the activity_main.xml file will like this:

File: activity_main.xml

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

<VideoView

```
    android:id="@+id/videoView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_centerVertical="true" />
```

</RelativeLayout>

Activity class

Let's write the code of to play the video file. Here, we are going to play 1.mp4 file located inside the sdcard/media directory.

File: MainActivity.java

```
package com.example.video1;
```

```
import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.MediaController;
import android.widget.VideoView;
```

```
public class MainActivity extends Activity {
```

```
    @Override
```

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

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

```
        VideoView videoView =(VideoView)findViewById(R.id.videoView1);
```

```

//Creating MediaController
MediaController mediaController= new MediaController(this);
mediaController.setAnchorView(videoView);

//specify the location of media file
Uri uri=Uri.parse(Environment.getExternalStorageDirectory().getPath()+"/media/1.m
p4");

//Setting MediaController and URI, then starting the videoView
videoView.setMediaController(mediaController);
videoView.setVideoURI(uri);
videoView.requestFocus();
videoView.start();
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
// Inflate the menu; this adds items to the action bar if it is present.
getMenuInflater().inflate(R.menu.activity_main, menu);
return true;
}
}

```

Text To Speech

Android allows you convert your text into voice. Not only you can convert it but it also allows you to speak text in variety of different languages.

Android provides **TextToSpeech** class for this purpose. In order to use this class, you need to instantiate an object of this class and also specify the **initListener**. Its syntax is given below –

```

private EditText write;
ttobj=new TextToSpeech(getApplicationContext(), new TextToSpeech.OnInitListener() {
@Override
public void onInit(int status) {
}
});

```

In this listener, you have to specify the properties for TextToSpeech object , such as its language ,pitch e.t.c. Language can be set by calling **setLanguage()** method. Its syntax is given below –

```
ttobj.setLanguage(Locale.UK);
```

The method setLanguage takes an Locale object as parameter. The list of some of the locales available are given below –

Sr.No.	Locale
1	US
2	CANADA_FRENCH
3	GERMANY
4	ITALY
5	JAPAN
6	CHINA

Once you have set the language, you can call **speak** method of the class to speak the text. Its syntax is given below –

```
ttobj.speak(toSpeak, TextToSpeech.QUEUE_FLUSH, null);
```

Apart from the speak method, there are some other methods available in the TextToSpeech class. They are listed below –

Sr.No	Method & description
1	addSpeech(String text, String filename) This method adds a mapping between a string of text and a sound file.
2	getLanguage() This method returns a Locale instance describing the language.
3	isSpeaking() This method checks whether the TextToSpeech engine is busy speaking.
4	setPitch(float pitch) This method sets the speech pitch for the TextToSpeech engine.
5	setSpeechRate(float speechRate) This method sets the speech rate.
6	shutdown() This method releases the resources used by the TextToSpeech engine.
7	stop() This method stop the speak.

Example

The below example demonstrates the use of TextToSpeech class. It crates a basic application that allows you to set write text and speak it.

To experiment with this example , you need to run this on an actual device.

Steps	Description
1	You will use Android studio to create an Android application under a package com.example.sairamkrishna.myapplication.
2	Modify src/MainActivity.java file to add TextToSpeech code.
3	Modify layout XML file res/layout/activity_main.xml add any GUI component if required.
4	Run the application and choose a running android device and install the application on it and verify the results.

Here is the content of **src/MainActivity.java**.

```
package com.example.sairamkrishna.myapplication;
```



```
import android.app.Activity;
import android.os.Bundle;
import android.speech.tts.TextToSpeech;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import java.util.Locale;
import android.widget.Toast;

public class MainActivity extends Activity {
    TextToSpeech t1;
    EditText ed1;
    Button b1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ed1=(EditText)findViewById(R.id.editText);
        b1=(Button)findViewById(R.id.button);

        t1=new TextToSpeech(getApplicationContext(), new TextToSpeech.OnInitListener() {
            @Override
            public void onInit(int status) {
                if(status != TextToSpeech.ERROR) {
                    t1.setLanguage(Locale.UK);
                }
            }
        });

        b1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String toSpeak = ed1.getText().toString();
                Toast.makeText(getApplicationContext(), toSpeak,Toast.LENGTH_SHORT).show();
                t1.speak(toSpeak, TextToSpeech.QUEUE_FLUSH, null);
            }
        });
    }

    public void onPause(){
        if(t1 !=null){
            t1.stop();
            t1.shutdown();
        }
        super.onPause();
    }
}
```

Here is the content of **activity_main.xml**

In the following code **abc** indicates the logo of tutorialspoint.com

```
<?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"
    tools:context=".MainActivity"
    android:transitionGroup="true">

    <TextView android:text="Text to Speech" android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textview"
        android:textSize="35dp"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Tutorials point"
        android:id="@+id/textView"
        android:layout_below="@+id/textview"
        android:layout_centerHorizontal="true"
        android:textColor="#ff7aff24"
        android:textSize="35dp" />

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView"
        android:src="@drawable/abc"
        android:layout_below="@+id/textView"
        android:layout_centerHorizontal="true"
        android:theme="@style/Base.TextAppearance.AppCompat" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText"
        android:layout_below="@+id/imageView"
        android:layout_marginTop="46dp"
        android:hint="Enter Text"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:layout_alignParentLeft="true"
```

```

    android:layout_alignParentStart="true"
    android:textColor="#ff7aff10"
    android:textColorHint="#ffff23d1" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Text to Speech"
        android:id="@+id/button"
        android:layout_below="@+id/editText"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="46dp" />

</RelativeLayout>

```

Here is the content of **Strings.xml**.

```

<resources>
    <string name="app_name">My Application</string>
</resources>

```

Here is the content of **AndroidManifest.xml**

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sairamkrishna.myapplication" >
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >


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

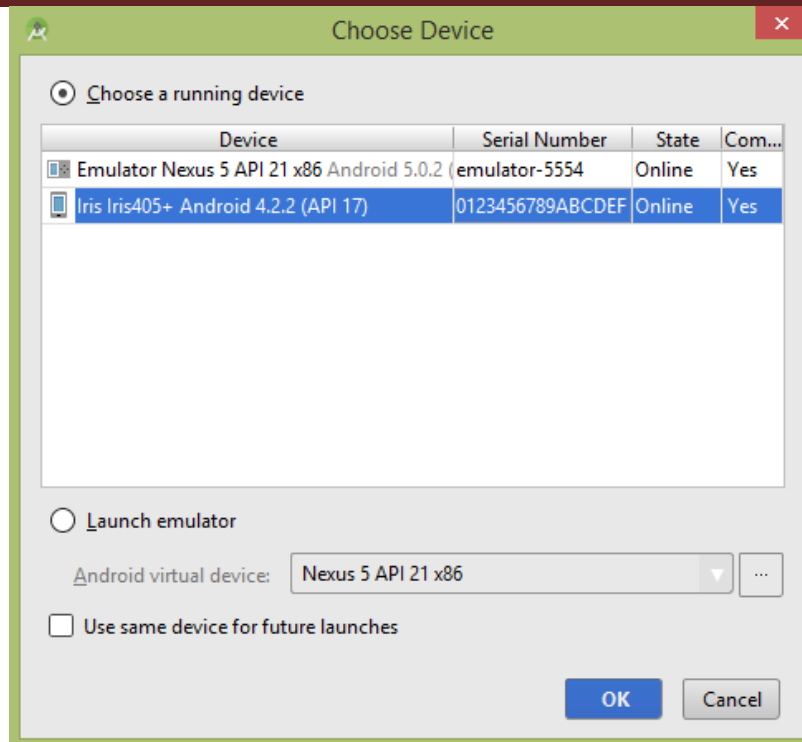
            <intent-filter>
                <action android:name="android.intent.action.MAIN" >
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>

        </activity>

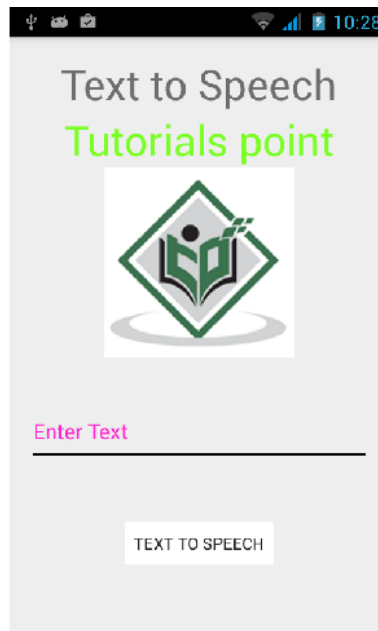
    </application>
</manifest>

```

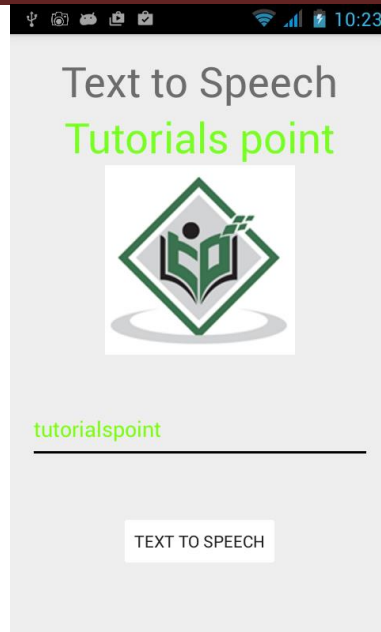
Let's try to run your application. I assume you have connected your actual Android Mobile device with your computer. To run the app from Android studio, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, android studio will display following window to select an option where you want to run your Android application.



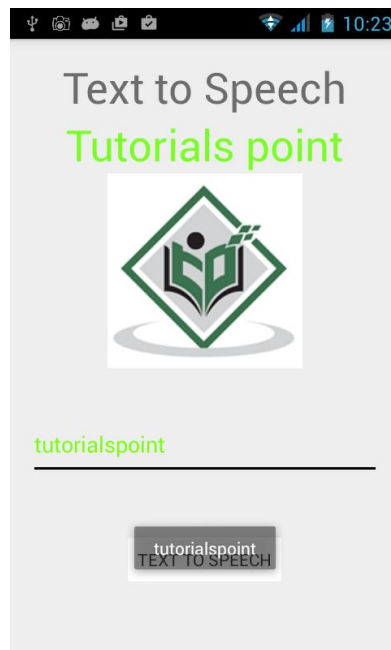
Select your mobile device as an option and then check your mobile device which will display following screen.



Now just type some text in the field and click on the text to speech button below. A notification would appear and text will be spoken. It is shown in the image below –



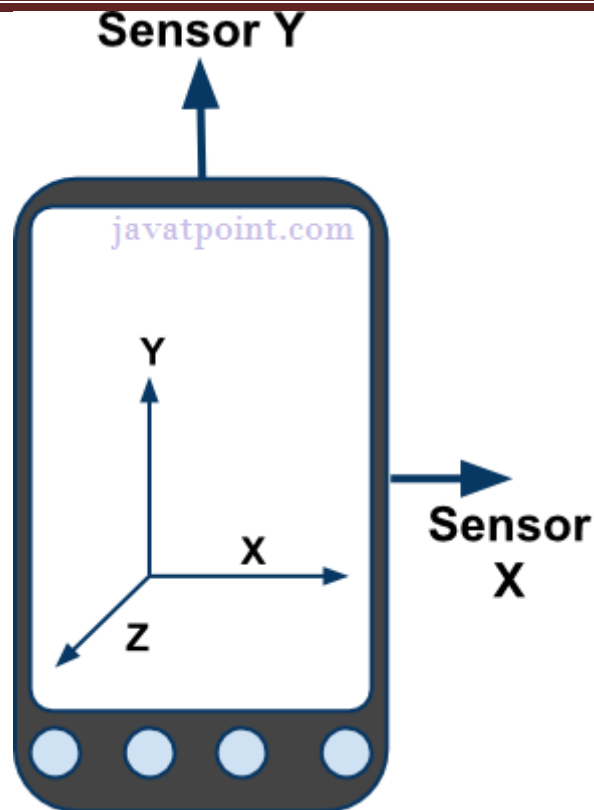
Now type something else and repeat the step again with different locale. You will again hear sound. This is shown below –



Android Sensor

Sensors can be used to monitor the three-dimensional device movement or change in the environment of the device.

Android provides sensor api to work with different types of sensors.



Types of Sensors

Android supports three types of sensors:

1) Motion Sensors

These are used to measure acceleration forces and rotational forces along with three axes.

2) Position Sensors

These are used to measure the physical position of device.

3) Environmental Sensors

These are used to measure the environmental changes such as temperature, humidity etc.

Android Sensor API

Android sensor api provides many classes and interface. The important classes and interfaces of sensor api are as follows:

1) SensorManager class

The **android.hardware.SensorManager** class provides methods :

- to get sensor instance,

- to access and list sensors,
- to register and unregister sensor listeners etc.

You can get the instance of `SensorManager` by calling the method `getSystemService()` and passing the `SENSOR_SERVICE` constant in it.

```
SensorManager sm = (SensorManager) getSystemService(SENSOR_SERVICE);
```

2) Sensor class

The `android.hardware.Sensor` class provides methods to get information of the sensor such as sensor name, sensor type, sensor resolution, sensor type etc.

3) SensorEvent class

Its instance is created by the system. It provides information about the sensor.

4) SensorEventListener interface

It provides two call back methods to get information when sensor values (x,y and z) change or sensor accuracy changes.

Public and abstract methods	Description
<code>void onAccuracyChanged(Sensor sensor, int accuracy)</code>	it is called when sensor accuracy is changed.
<code>void onSensorChanged(SensorEvent event)</code>	it is called when sensor values are changed.

Android simple sensor app example

Let's see the two sensor examples.

1. A sensor example that prints x, y and z axis values. Here, we are going to see that.
2. A sensor example that changes the background color when device is shuffled. Click for changing background color of activity sensor example

activity_main.xml

There is only one textview in this file.

File: activity_main.xml

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

```

<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:layout_marginLeft="92dp"
    android:layout_marginTop="114dp"
    android:text="TextView" />

```

```
</RelativeLayout>
```

Activity class

Let's write the code that prints values of x axis, y axis and z axis.

File: MainActivity.java

```

package com.example.sensorsimple;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Toast;
import android.hardware.SensorManager;
import android.hardware.SensorEventListener;
import android.hardware.SensorEvent;
import android.hardware.Sensor;
import java.util.List;
public class MainActivity extends Activity {
    SensorManager sm = null;
    TextView textView1 = null;
    List list;

    SensorEventListener sel = new SensorEventListener(){
        public void onAccuracyChanged(Sensor sensor, int accuracy) {}
        public void onSensorChanged(SensorEvent event) {
            float[] values = event.values;
            textView1.setText("x: "+values[0]+"\\ny: "+values[1]+"\\nz: "+values[2]);
        }
    };

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

```



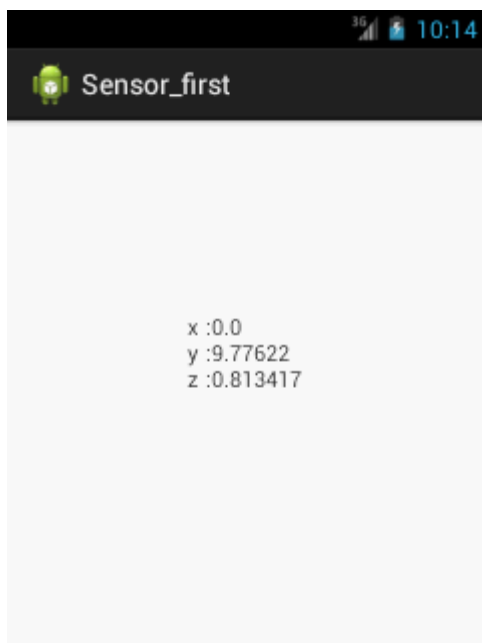
```
/* Get a SensorManager instance */
sm = (SensorManager) getSystemService(SENSOR_SERVICE);

textView1 = (TextView) findViewById(R.id.textView1);

list = sm.getSensorList(Sensor.TYPE_ACCELEROMETER);
if(list.size()>0){
    sm.registerListener(sel, (Sensor) list.get(0), SensorManager.SENSOR_DELAY_NORMAL
);
}
else{
    Toast.makeText(getApplicationContext(), "Error: No Accelerometer.", Toast.LENGTH_LONG).s
how();
}
}

@Override
protected void onStop() {
    if(list.size()>0){
        sm.unregisterListener(sel);
    }
    super.onStop();
}
}
```

Output:



Android Sensor Example

In this example, we are going to create a sensor application that changes the background color of activity when device is shaken.

For understanding about sensor basics, visit the previous page that provides details about sensor api.

Android Sensor Example

activity_main.xml

File: activity_main.xml

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

    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Shake to switch color" />

</RelativeLayout>
```

Activity class

File: MainActivity.java

```
package com.example.sensor;

import android.app.Activity;
import android.graphics.Color;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class MainActivity extends Activity implements SensorEventListener{
    private SensorManager sensorManager;
    private boolean isColor = false;
```

```

private View view;
private long lastUpdate;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    view = findViewById(R.id.textView);
    view.setBackgroundColor(Color.GREEN);

    sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
    lastUpdate = System.currentTimeMillis();
}
//overriding two methods of SensorEventListener
@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {}
@Override
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
        getAccelerometer(event);
    }
}

private void getAccelerometer(SensorEvent event) {
    float[] values = event.values;
    // Movement
    float x = values[0];
    float y = values[1];
    float z = values[2];

    float accelationSquareRoot = (x * x + y * y + z * z)
        / (SensorManager.GRAVITY_EARTH * SensorManager.GRAVITY_EARTH);

    long actualTime = System.currentTimeMillis();
    Toast.makeText(getApplicationContext(),String.valueOf(accelationSquareRoot)+" "+
        SensorManager.GRAVITY_EARTH,Toast.LENGTH_SHORT).show();

    if (accelationSquareRoot >= 2) //it will be executed if you shuffle
    {

        if (actualTime - lastUpdate < 200) {
            return;
        }
        lastUpdate = actualTime;//updating lastUpdate for next shuffle
    }
}

```

```
if (isColor) {  
    view.setBackgroundColor(Color.GREEN);  
  
} else {  
    view.setBackgroundColor(Color.RED);  
}  
isColor = !isColor;  
}  
}
```

```
@Override  
protected void onResume() {  
    super.onResume();  
    // register this class as a listener for the orientation and  
    // accelerometer sensors  
    sensorManager.registerListener(this,sensorManager.getDefaultSensor(Sensor.TYPE_AC  
CELEROMETER),  
        SensorManager.SENSOR_DELAY_NORMAL);  
}
```

```
@Override  
protected void onPause() {  
    // unregister listener  
    super.onPause();  
    sensorManager.unregisterListener(this);  
}  
}
```



AsyncTask

Android AsyncTask is an abstract class provided by Android which gives us the liberty to perform heavy tasks in the background and keep the UI thread light thus making the application more responsive.

Android application runs on a single thread when launched. Due to this single thread model tasks that take longer time to fetch the response can make the application non-responsive. To avoid this we use android AsyncTask to perform the heavy tasks in background on a dedicated thread and passing the results back to the UI thread. Hence use of AsyncTask in android application keeps the UI thread responsive at all times.

The basic methods used in an android AsyncTask class are defined below :

-

The three generic types used in an android AsyncTask class are given below :

- **Params** : The type of the parameters sent to the task upon execution
- **Progress** : The type of the progress units published during the background computation
- **Result** : The type of the result of the background computation

Android AsyncTask Example

To start an AsyncTask the following snippet must be present in the MainActivity class :

```
MyTask myTask = new MyTask();  
myTask.execute();
```

In the above snippet we've used a sample classname that extends AsyncTask and execute method is used to start the background thread.

Note:

- The AsyncTask instance must be created and invoked in the UI thread.
- The methods overridden in the AsyncTask class should never be called. They're called automatically
- AsyncTask can be called only once. Executing it again will throw an exception

In this tutorial we'll implement an AsyncTask that makes a process to go to sleep for a given period of time as set by the user.

Android AsyncTask Example Code

The xml layout is defined in the activity_main.xml and its given below:

activity_main.xml

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

    <TextView
        android:id="@+id/tv_time"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="10pt"
        android:textColor="#444444"
        android:layout_alignParentLeft="true"
        android:layout_marginRight="9dip"
        android:layout_marginTop="20dip"
        android:layout_marginLeft="10dip"
        android:text="Sleep time in Seconds:" />
    <EditText
        android:id="@+id/in_time"
        android:layout_width="150dip"
        android:layout_height="wrap_content"
        android:background="@android:drawable/editbox_background"
        android:layout_toRightOf="@id/tv_time"
        android:layout_alignTop="@id/tv_time"
        android:inputType="number"
        />
    <Button
        android:id="@+id/btn_run"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Run Async task"
        android:layout_below="@+id/in_time"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="64dp" />
    <TextView
        android:id="@+id/tv_result"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="7pt"
        android:layout_below="@+id/btn_run"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

In the above layout we've used a predefined drawable as the border of the EditText.

MainActivity.java

```
package com.journaldev.asyncTask;

import android.app.ProgressDialog;
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.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    private Button button;
    private EditText time;
    private TextView finalResult;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        time = (EditText) findViewById(R.id.in_time);
        button = (Button) findViewById(R.id.btn_run);
        finalResult = (TextView) findViewById(R.id.tv_result);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                AsyncTaskRunner runner = new AsyncTaskRunner();
                String sleepTime = time.getText().toString();
                runner.execute(sleepTime);
            }
        });
    }

    private class AsyncTaskRunner extends AsyncTask<String, String, String> {

        private String resp;
        ProgressDialog progressDialog;

        @Override
        protected String doInBackground(String... params) {
            publishProgress("Sleeping..."); // Calls onProgressUpdate()
            try {
                int time = Integer.parseInt(params[0])*1000;

                Thread.sleep(time);
                resp = "Slept for " + params[0] + " seconds";
            } catch (InterruptedException e) {
```

```
e.printStackTrace();
    resp = e.getMessage();
} catch (Exception e) {
    e.printStackTrace();
    resp = e.getMessage();
}
return resp;
}

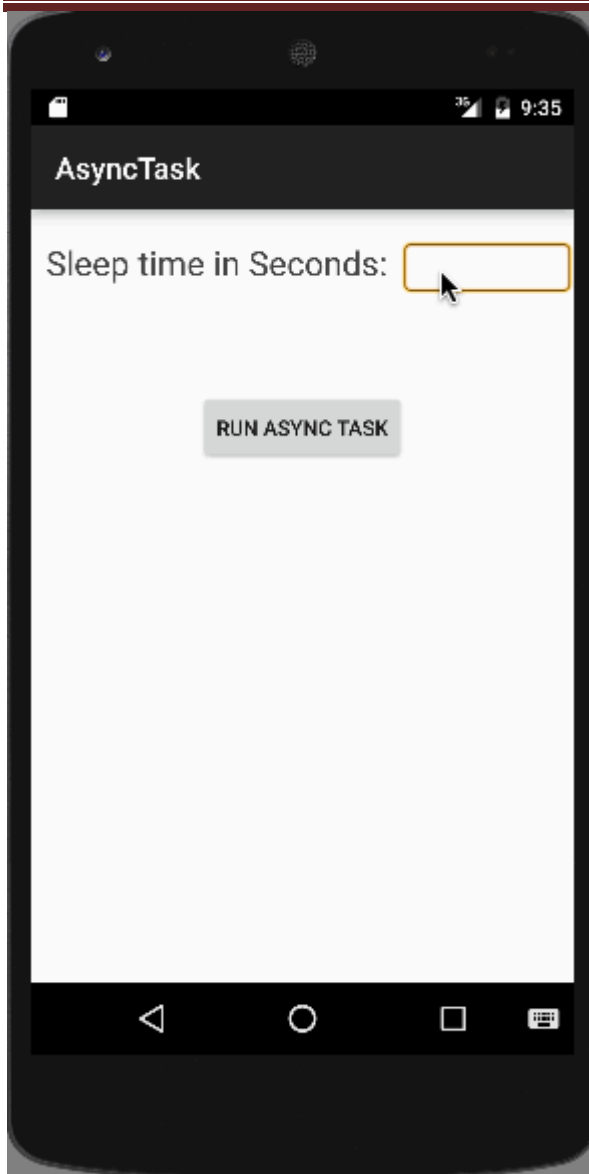
@Override
protected void onPostExecute(String result) {
    // execution of result of Long time consuming operation
    progressDialog.dismiss();
    finalResult.setText(result);
}

@Override
protected void onPreExecute() {
    progressDialog = ProgressDialog.show(MainActivity.this,
        "ProgressDialog",
        "Wait for "+time.getText().toString()+" seconds");
}

@Override
protected void onProgressUpdate(String... text) {
    finalResult.setText(text[0]);
}
}
}
```

In the above code we've used `AsyncTaskRunner` class to perform the `AsyncTask` operations. The time in seconds is passed as a parameter to the class and a `ProgressDialog` is displayed for the given amount of time.

The images given below are the outputs produced by the project where the time set by the user is 5 seconds.



Audio Capture

Android has a built in microphone through which you can capture audio and store it , or play it in your phone. There are many ways to do that but the most common way is through MediaRecorder class.

Android provides MediaRecorder class to record audio or video. In order to use MediaRecorder class ,you will first create an instance of MediaRecorder class. Its syntax is given below.

```
MediaRecorder myAudioRecorder = new MediaRecorder();
```

Now you will set the source , output and encoding format and output file. Their syntax is given below.

```
myAudioRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);  
myAudioRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);  
myAudioRecorder.setAudioEncoder(MediaRecorder.OutputFormat.AMR_NB);  
myAudioRecorder.setOutputFile(outputFile);
```

After specifying the audio source and format and its output file, we can then call the two basic methods prepare and start to start recording the audio.

```
myAudioRecorder.prepare();
myAudioRecorder.start();
```

Apart from these methods , there are other methods listed in the MediaRecorder class that allows you more control over audio and video recording.

Sr.No	Method & description
1	setAudioSource() This method specifies the source of audio to be recorded
2	setVideoSource() This method specifies the source of video to be recorded
3	setOutputFormat() This method specifies the audio format in which audio to be stored
4	setAudioEncoder() This method specifies the audio encoder to be used
5	setOutputFile() This method configures the path to the file into which the recorded audio is to be stored
6	stop() This method stops the recording process.
7	release() This method should be called when the recorder instance is needed.

Example

This example provides demonstration of MediaRecorder class to capture audio and then MediaPlayer class to play that recorded audio.

To experiment with this example , you need to run this on an actual device.

Steps	Description
1	You will use Android studio IDE to create an Android application and name it as AudioCapture under a package com.example.sairamkrishna.myapplication.
2	Modify src/MainActivity.java file to add AudioCapture code
3	Modify layout XML file res/layout/activity_main.xml add any GUI component if required.
4	Modify AndroidManifest.xml to add necessary permissions.
5	Run the application and choose a running android device and install the application on it and verify the results.

Here is the content of **src/MainActivity.java**

```
package com.example.sairamkrishna.myapplication;

import android.media.MediaPlayer;
import android.media.MediaRecorder;

import android.os.Environment;
import android.support.v7.app.AppCompatActivity;
```

```
import android.os.Bundle;
import android.view.View;

import android.widget.Button;
import android.widget.Toast;

import java.io.IOException;
import java.util.Random;

import static android.Manifest.permission.RECORD_AUDIO;
import static android.Manifest.permission.WRITE_EXTERNAL_STORAGE;

import android.support.v4.app.ActivityCompat;
import android.content.pm.PackageManager;
import android.support.v4.content.ContextCompat;

public class MainActivity extends AppCompatActivity {

    Button buttonStart, buttonStop, buttonPlayLastRecordAudio,
        buttonStopPlayingRecording ;
    String AudioSavePathInDevice = null;
    MediaRecorder mediaRecorder ;
    Random random ;
    String RandomAudioFileName = "ABCDEFGHJKLMNOP";
    public static final int RequestPermissionCode = 1;
    MediaPlayer mediaPlayer ;

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

        buttonStart = (Button) findViewById(R.id.button);
        buttonStop = (Button) findViewById(R.id.button2);
        buttonPlayLastRecordAudio = (Button) findViewById(R.id.button3);
        buttonStopPlayingRecording = (Button) findViewById(R.id.button4);

        buttonStop.setEnabled(false);
        buttonPlayLastRecordAudio.setEnabled(false);
        buttonStopPlayingRecording.setEnabled(false);

        random = new Random();

        buttonStart.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                if(checkPermission()) {

                    AudioSavePathInDevice =
```

```
Environment.getExternalStorageDirectory().getAbsolutePath() + "/" +
    CreateRandomAudioFileName(5) + "AudioRecording.3gp";

MediaRecorderReady();

try {
    mediaRecorder.prepare();
    mediaRecorder.start();
} catch (IllegalStateException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

buttonStart.setEnabled(false);
buttonStop.setEnabled(true);

Toast.makeText(MainActivity.this, "Recording started",
    Toast.LENGTH_LONG).show();
} else {
    requestPermission();
}

}
});

buttonStop.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        mediaRecorder.stop();
        buttonStop.setEnabled(false);
        buttonPlayLastRecordAudio.setEnabled(true);
        buttonStart.setEnabled(true);
        buttonStopPlayingRecording.setEnabled(false);

        Toast.makeText(MainActivity.this, "Recording Completed",
            Toast.LENGTH_LONG).show();
    }
});

buttonPlayLastRecordAudio.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) throws IllegalArgumentException,
        SecurityException, IllegalStateException {

        buttonStop.setEnabled(false);
        buttonStart.setEnabled(false);
        buttonStopPlayingRecording.setEnabled(true);
```

```
        mediaPlayer = new MediaPlayer();
        try {
            mediaPlayer.setDataSource(AudioSavePathInDevice);
            mediaPlayer.prepare();
        } catch (IOException e) {
            e.printStackTrace();
        }

        mediaPlayer.start();
        Toast.makeText(MainActivity.this, "Recording Playing",
            Toast.LENGTH_LONG).show();
    }
});

buttonStopPlayingRecording.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        buttonStop.setEnabled(false);
        buttonStart.setEnabled(true);
        buttonStopPlayingRecording.setEnabled(false);
        buttonPlayLastRecordAudio.setEnabled(true);

        if(mediaPlayer != null){
            mediaPlayer.stop();
            mediaPlayer.release();
            MediaRecorderReady();
        }
    }
});
}

public void MediaRecorderReady(){
    mediaRecorder=new MediaRecorder();
    mediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
    mediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
    mediaRecorder.setAudioEncoder(MediaRecorder.OutputFormat.AMR_NB);
    mediaRecorder.setOutputFile(AudioSavePathInDevice);
}

public String CreateRandomAudioFileName(int string){
    StringBuilder stringBuilder = new StringBuilder( string );
    int i = 0 ;
    while(i < string ) {
        stringBuilder.append(RandomAudioFileName.
            charAt(random.nextInt(RandomAudioFileName.length())));

        i++;
    }
}
```

```

return stringBuilder.toString();
}

private void requestPermission() {
    ActivityCompat.requestPermissions(MainActivity.this, new
        String[]{WRITE_EXTERNAL_STORAGE, RECORD_AUDIO}, RequestPermissionCode);
}

@Override
public void onRequestPermissionsResult(int requestCode,
    String permissions[], int[] grantResults) {
    switch (requestCode) {
        case RequestPermissionCode:
            if (grantResults.length > 0) {
                boolean StoragePermission = grantResults[0] ==
                    PackageManager.PERMISSION_GRANTED;
                boolean RecordPermission = grantResults[1] ==
                    PackageManager.PERMISSION_GRANTED;

                if (StoragePermission && RecordPermission) {
                    Toast.makeText(MainActivity.this, "Permission Granted",
                        Toast.LENGTH_LONG).show();
                } else {
                    Toast.makeText(MainActivity.this, "Permission
                        Denied", Toast.LENGTH_LONG).show();
                }
            }
            break;
    }
}

public boolean checkPermission() {
    int result = ContextCompat.checkSelfPermission(getApplicationContext(),
        WRITE_EXTERNAL_STORAGE);
    int result1 = ContextCompat.checkSelfPermission(getApplicationContext(),
        RECORD_AUDIO);
    return result == PackageManager.PERMISSION_GRANTED &&
        result1 == PackageManager.PERMISSION_GRANTED;
}
}

```

Here is the content of **activity_main.xml**

In the below code **abc** indicates the logo of tutorialspoint

```

<?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">
```

```
<ImageView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:id="@+id/imageView"  
    android:layout_alignParentTop="true"  
    android:layout_centerHorizontal="true"  
    android:src="@drawable/abc"/>
```

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Record"  
    android:id="@+id/button"  
    android:layout_below="@+id/imageView"  
    android:layout_alignParentLeft="true"  
    android:layout_marginTop="37dp"  
>
```

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="STOP"  
    android:id="@+id/button2"  
    android:layout_alignTop="@+id/button"  
    android:layout_centerHorizontal="true"  
>
```

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Play"  
    android:id="@+id/button3"  
    android:layout_alignTop="@+id/button2"  
    android:layout_alignParentRight="true"  
    android:layout_alignParentEnd="true"  
>
```

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="STOP PLAYING RECORDING "  
    android:id="@+id/button4"  
    android:layout_below="@+id/button2"  
    android:layout_centerHorizontal="true"  
    android:layout_marginTop="10dp"  
>
```

```
</RelativeLayout>
```

Here is the content of **Strings.xml**

```
<resources>
  <string name="app_name">My Application</string>
</resources>
```

Here is the content of **AndroidManifest.xml**

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

  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
  <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.STORAGE" />


  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <activity
      android:name="com.example.sairamkrishna.myapplication.MainActivity"
      android:label="@string/app_name" >

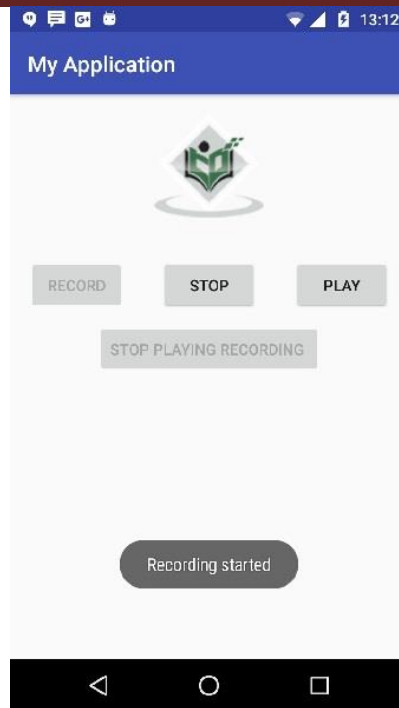
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>

    </activity>

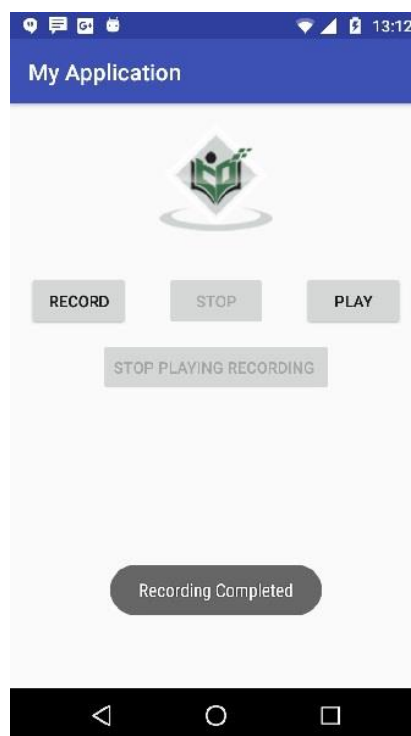
  </application>
</manifest>
```

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. Before starting your application, Android studio will display following images.

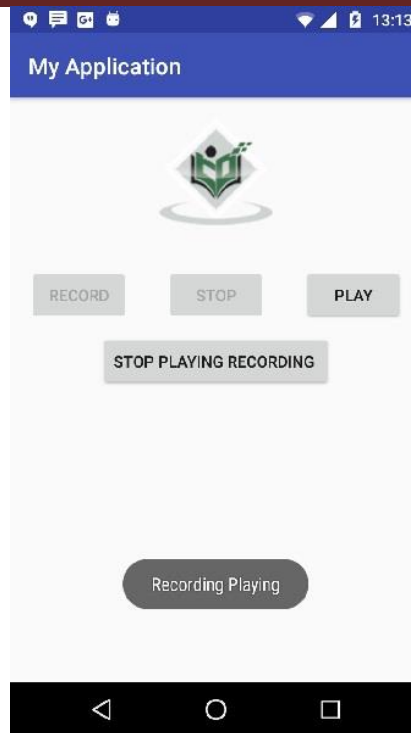
Now by default you will see stop and play button disable. Just press the Record button and your application will start recording the audio. It will display the following screen.



Now just press stop button and it will save the recorded audio to external sd card. When you click on stop button , the following screen would appear.



Now just press the play button and recorded audio will just start playing on the device. The following message appears when you click on play button.



Camera In Android

In Android, Camera is a hardware device that allows capturing pictures and videos in your applications. Follow this tutorial to easily understand how to use a camera in your own Android App.

Android provides the facility to work on camera by 2 ways:

1. By Camera Intent
2. By Camera API

1 Using Camera By Using Camera Application

We can capture pictures without using the instance of Camera class. Here you will use an intent action type of `MediaStore.ACTION_IMAGE_CAPTURE` to launch an existing Camera application on your phone. In Android MediaStore is a type of DataBase which stores pictures and videos in android.

```
Intent cameraIntent = new  
Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
```

2 Using Camera By using Camera Api

This class is used for controlling device cameras. It can be used to take pictures when you are building a camera application.

Camera API works in following ways:

1.Camera Manager: This is used to get all the cameras available in the device like front camera back camera each having the camera id.

2.CameraDevice: You can get it from Camera Manager class by its id.

3.CaptureRequest: You can create a capture request from camera device to capture images.

4.CameraCaptureSession: To get capture request's from Camera Device create a CameraCaptureSession.

5.CameraCaptureSession.CaptureCallback: This is going to provide the Capture session results.

Camera Permission Declarations In Manifest

First, you should declare the Camera requirement in your Manifest file if Camera is compulsory for your application and you don't want your application to be installed on a device that does not support Camera.

Before you start development on your application you have to make sure that your Manifest has appropriate declarations in it that will allow you to use Camera feature in your Application.

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

Android camera app example by camera intent

activity_main.xml

Drag one imageview and one button from the pallette, now the xml file will look like this:

File: activity_main.xml

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

<Button

```
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"
    android:text="Take a Photo" >
```

</Button>

<ImageView

```
    android:id="@+id/imageView1"
```

```

android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:layout_above="@+id/button1"
android:layout_alignParentTop="true"
android:src="@drawable/ic_launcher" >
</ImageView>
</RelativeLayout>

```

Activity class

Let's write the code to capture image using camera and displaying it on the image view.

File: MainActivity.java

```

package com.example.simplecamera;

import android.app.Activity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;

public class MainActivity extends Activity {
    private static final int CAMERA_REQUEST = 1888;
    ImageView imageView;
    public void onCreate(Bundle savedInstanceState) {

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

        imageView = (ImageView) this.findViewById(R.id.imageView1);
        Button photoButton = (Button) this.findViewById(R.id.button1);

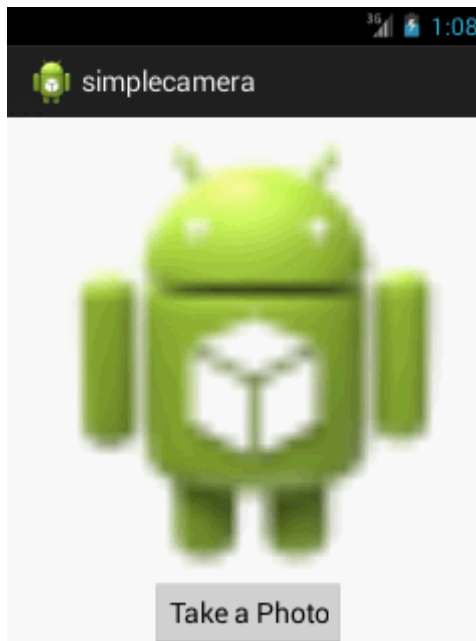
        photoButton.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                Intent cameraIntent = new Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
                startActivityForResult(cameraIntent, CAMERA_REQUEST);
            }
        });
    }
}

```

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
    if (requestCode == CAMERA_REQUEST) {  
        Bitmap photo = (Bitmap) data.getExtras().get("data");  
        imageView.setImageBitmap(photo);  
    }  
}  
  
@Override  
public boolean onCreateOptionsMenu(Menu menu) {  
    // Inflate the menu; this adds items to the action bar if it is present.  
    getMenuInflater().inflate(R.menu.activity_main, menu);  
    return true;  
}  
}
```

Output:



Android Bluetooth Tutorial

Bluetooth is a way to exchange data with other devices wirelessly. Android provides Bluetooth API to perform several tasks such as:

- scan bluetooth devices
- connect and transfer data from and to other devices
- manage multiple connections etc.

Android Bluetooth API

The android.bluetooth package provides a lot of interfaces classes to work with bluetooth such as:

- BluetoothAdapter
- BluetoothDevice
- BluetoothSocket
- BluetoothServerSocket
- BluetoothClass
- BluetoothProfile
- BluetoothProfile.ServiceListener
- BluetoothHeadset
- BluetoothA2dp
- BluetoothHealth
- BluetoothHealthCallback
- BluetoothHealthAppConfiguration

android-preferences-example

BluetoothAdapter class

By the help of BluetoothAdapter class, we can perform fundamental tasks such as initiate device discovery, query a list of paired (bonded) devices, create a BluetoothServerSocket instance to listen for connection requests etc.

Constants of BluetoothAdapter class

BluetoothAdapter class provides many constants. Some of them are as follows:

- String ACTION_REQUEST_ENABLE
- String ACTION_REQUEST_DISCOVERABLE
- String ACTION_DISCOVERY_STARTED
- String ACTION_DISCOVERY_FINISHED

Methods of BluetoothAdapter class

Commonly used methods of BluetoothAdapter class are as follows:

- **static synchronized BluetoothAdapter getDefaultAdapter()** returns the instance of BluetoothAdapter.
- **boolean enable()** enables the bluetooth adapter if it is disabled.
- **boolean isEnabled()** returns true if the bluetooth adapter is enabled.

- **boolean disable()** disables the bluetooth adapter if it is enabled.
- **String getName()** returns the name of the bluetooth adapter.
- **boolean setName(String name)** changes the bluetooth name.
- **int getState()** returns the current state of the local bluetooth adapter.
- **Set<BluetoothDevice> getBondedDevices()** returns a set of paired (bonded) BluetoothDevice objects.
- **boolean startDiscovery()** starts the discovery process.

Android Bluetooth Example: enable, disable and make discoverable bluetooth programmatically

You need to write few lines of code only, to enable or disable the bluetooth.

activity_main.xml

Drag one textview and three buttons from the palette, now the activity_main.xml file will like this:

File: activity_main.xml

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

```
<TextView android:text=""
    android:id="@+id/out"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
</TextView>
```

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:layout_marginLeft="30dp"
    android:layout_marginTop="49dp"
    android:text="TURN_ON" />
```

```
<Button
    android:id="@+id/button2"
```

```

android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignLeft="@+id/button1"
android:layout_below="@+id/button1"
android:layout_marginTop="27dp"
android:text="DISCOVERABLE" />

```

<Button

```

android:id="@+id/button3"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignLeft="@+id/button2"
android:layout_below="@+id/button2"
android:layout_marginTop="28dp"
android:text="TURN_OFF" />

```

</RelativeLayout>*Provide Permission*

You need to provide following permissions in AndroidManifest.xml file.

```

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

```

The full code of AndroidManifest.xml file is given below.

File: AndroidManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.bluetooth"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="16" />

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

    <application
        android:allowBackup="true"

```

```

android:icon="@drawable/ic_launcher"
android:label="@string/app_name"
android:theme="@style/AppTheme" >
<activity
    android:name="com.example.bluetooth.MainActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

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

</manifest>

```

Activity class

Let's write the code to enable, disable and make bluetooth discoverable.

File: MainActivity.java

```

package com.example.bluetooth;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends Activity {
    private static final int REQUEST_ENABLE_BT = 0;
    private static final int REQUEST_DISCOVERABLE_BT = 0;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

```

```

final TextView out=(TextView)findViewById(R.id.out);
final Button button1 = (Button) findViewById(R.id.button1);
final Button button2 = (Button) findViewById(R.id.button2);
final Button button3 = (Button) findViewById(R.id.button3);
final BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
if (mBluetoothAdapter == null) {
    out.append("device not supported");
}
button1.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        if (!mBluetoothAdapter.isEnabled()) {
            Intent enableBluetooth = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(enableBluetooth, REQUEST_ENABLE_BT);
        }
    }
});
button2.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View arg0) {
        if (!mBluetoothAdapter.isDiscovering()) {
            //out.append("MAKING YOUR DEVICE DISCOVERABLE");
            Toast.makeText(getApplicationContext(), "MAKING YOUR DEVICE DISCOVERABLE
",
            Toast.LENGTH_LONG);

            Intent enableBluetooth = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVER
ABLE);
            startActivityForResult(enableBluetooth, REQUEST_DISCOVERABLE_BT);

        }
    }
});
button3.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View arg0) {
        mBluetoothAdapter.disable();
        //out.append("TURN_OFF BLUETOOTH");
        Toast.makeText(getApplicationContext(), "TURNING_OFF BLUETOOTH", Toast.LENGTH
H_LONG);

    }
});
}

```

@Override

```
public boolean onCreateOptionsMenu(Menu menu) {  
    // Inflate the menu; this adds items to the action bar if it is present.  
    getMenuInflater().inflate(R.menu.activity_main, menu);  
    return true;  
}  
  
}
```

Android Animation

Animation in android apps is the process of creating motion and shape change.

Android Animation Example XML

We create a resource directory under the res folder names **anim** to keep all the xml files containing the animation logic. Following is a sample xml file showing an android animation code logic.

sample_animation.xml

```
<?xml version="1.0" encoding="utf-8"?>  
  
<scale xmlns:android="https://schemas.android.com/apk/res/android"  
    android:interpolator="@android:anim/accelerate_decelerate_interpolator"  
    android:duration="300"  
    android:fillAfter="true"  
    android:fromXScale="0.0"  
    android:fromYScale="0.0"  
    android:toXScale="1.0"  
    android:toYScale="1.0" />
```

- **android:interpolator** : It is the rate of change in animation. We can define our own interpolators using the time as the constraint. In the above xml code an inbuilt interpolator is assigned
- **android:duration** : Duration of the animation in which the animation should complete. It is 300ms here. This is generally the ideal duration to show the transition on the screen.

The start and end of the animation are set using:

android:fromTRANSFORMATION

android:toTRANSFORMATION

- **TRANSFORMATION** : is the transformation that we want to specify. In our case we start with an x and y scale of 0 and end with an x and y scale of 1
- **android:fillAfter** : property specifies whether the view should be visible or hidden at the end of the animation. We've set it visible in the above code. If it sets to false, the element changes to its previous state after the animation
- **android:startOffset** : It is the waiting time before an animation starts. This property is mainly used to perform multiple animations in a sequential manner
- **android:repeatMode** : This is useful when you want the animation to be repeat
- **android:repeatCount** : This defines number of repetitions on animation. If we set this value to infinite then animation will repeat infinite times

Loading Animation when UI widget is clicked

Our aim is to show an animation when any widget(lets say TextView) is clicked. For that we need to use the Animation Class. The xml file that contains the animation logic is loaded using **AnimationUtils** class by calling the loadAnimation() function. The below snippet shows this implementation.

```
Animation animation;
```

```
animation = AnimationUtils.loadAnimation(getApplicationContext(),
    R.anim.sample_animation);
```

To start the animation we need to call the startAnimation() function on the UI element as shown in the snippet below:

```
sampleTextView.startAnimation(animation);
```

Here we perform the animation on a textview component by passing the type of Animation as the parameter.

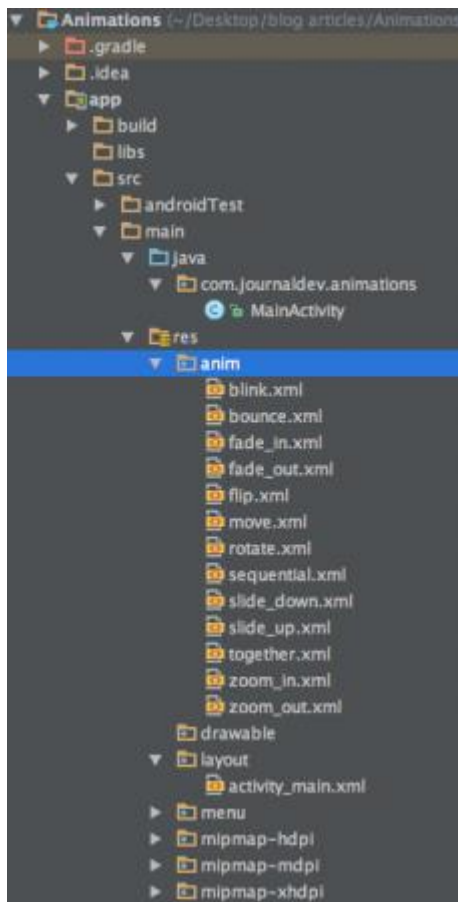
Setting the Animation Listeners

This is only needed if we wish to listen to events like start, end or repeat. For this the activity must implement **AnimationListener** and the following methods need to overridden.

- **onAnimationStart** : This will be triggered once the animation started
- **onAnimationEnd** : This will be triggered once the animation is over

- **onAnimationRepeat** : This will be triggered if the animation repeats

Android Animation Project Structure



As you can see, we've included the xml of all the major types of animations covered above.

Android Animation Examples XML Code

Here I am providing sample code for most of the common android animations.

Fade In Animation

fade_in.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
```

```
    android:fillAfter="true" >
```

```
    <alpha
```

```
    android:duration="1000"  
    android:fromAlpha="0.0"  
    android:interpolator="@android:anim/accelerate_interpolator"  
    android:toAlpha="1.0" />
```

```
</set>
```

Here **alpha** references the opacity of an object. An object with lower alpha values is more transparent, while an object with higher alpha values is less transparent, more opaque. Fade in animation is nothing but increasing alpha value from 0 to 1.

Fade Out Animation

fade_out.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"  
    android:fillAfter="true" >  
  
    <alpha  
        android:duration="1000"  
        android:fromAlpha="1.0"  
        android:interpolator="@android:anim/accelerate_interpolator"  
        android:toAlpha="0.0" />
```

```
</set>
```

Fade out android animation is exactly opposite to fade in, where we need to decrease the alpha value from 1 to 0.

Cross Fading Animation

Cross fading is performing fade in animation on one TextView while other TextView is fading out. This can be done by using fade_in.xml and fade_out.xml on the two TextViews. The code will be discussed in the MainActivity.java

Blink Animation

blink.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android">
  <alpha android:fromAlpha="0.0"
    android:toAlpha="1.0"
    android:interpolator="@android:anim/accelerate_interpolator"
    android:duration="600"
    android:repeatMode="reverse"
    android:repeatCount="infinite" />
</set>
```

Here fade in and fade out are performed infinitely in reverse mode each time.

Zoom In Animation

zoom_in.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
  android:fillAfter="true" >
  <scale
    xmlns:android="https://schemas.android.com/apk/res/android"
    android:duration="1000"
    android:fromXScale="1"
```

```
    android:fromYScale="1"  
    android:pivotX="50%"  
    android:pivotY="50%"  
    android:toXScale="3"  
    android:toYScale="3" >  
</scale>
```

```
</set>
```

We use `pivotX="50%"` and `pivotY="50%"` to perform zoom from the center of the element.

Zoom Out Animation

zoom_out.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"  
    android:fillAfter="true" >  
  
    <scale  
        xmlns:android="https://schemas.android.com/apk/res/android"  
        android:duration="1000"  
        android:fromXScale="1.0"  
        android:fromYScale="1.0"  
        android:pivotX="50%"  
        android:pivotY="50%"  
        android:toXScale="0.5"  
        android:toYScale="0.5" >  
    </scale>
```

```
</set>
```

Notice that **android:from** and **android:to** are opposite in zoom_in.xml and zoom_out.xml.

Rotate Animation

rotate.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android">
    <rotate android:fromDegrees="0"
        android:toDegrees="360"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="600"
        android:repeatMode="restart"
        android:repeatCount="infinite"
        android:interpolator="@android:anim/cycle_interpolator"/>
</set>
```

A **from/toDegrees** tag is used here to specify the degrees and a cyclic interpolator is used.

Move Animation

move.xml

```
<set
    xmlns:android="https://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/linear_interpolator"
    android:fillAfter="true">
```

```
<translate
    android:fromXDelta="0%p"
    android:toXDelta="75%p"
    android:duration="800" />
</set>
```

Slide Up Animation

slide_up.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
    android:fillAfter="true" >

    <scale
        android:duration="500"
        android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:interpolator="@android:anim/linear_interpolator"
        android:toXScale="1.0"
        android:toYScale="0.0" />

</set>
```

It's achieved by setting **android:fromYScale="1.0"** and **android:toYScale="0.0"** inside the **scale** tag.

Slide Down Animation

slide_down.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
    android:fillAfter="true">

    <scale
        android:duration="500"
        android:fromXScale="1.0"
        android:fromYScale="0.0"
        android:toXScale="1.0"
        android:toYScale="1.0" />

</set>
```

This is just the opposite of slide_up.xml.

Bounce Animation

bounce.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
    android:fillAfter="true"
    android:interpolator="@android:anim/bounce_interpolator">

    <scale
        android:duration="500"
        android:fromXScale="1.0"
        android:fromYScale="0.0"
        android:toXScale="1.0"
        android:toYScale="1.0" />
```

```
</set>
```

Here bounce interpolator is used to complete the animation in bouncing fashion.

Sequential Animation

sequential.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
```

```
    android:fillAfter="true"
```

```
    android:interpolator="@android:anim/linear_interpolator" >
```

```
<!-- Move -->
```

```
<translate
```

```
    android:duration="800"
```

```
    android:fillAfter="true"
```

```
    android:fromXDelta="0%p"
```

```
    android:startOffset="300"
```

```
    android:toXDelta="75%p" />
```

```
<translate
```

```
    android:duration="800"
```

```
    android:fillAfter="true"
```

```
    android:fromYDelta="0%p"
```

```
    android:startOffset="1100"
```

```
    android:toYDelta="70%p" />
```

```
<translate
```

```
    android:duration="800"
```

```
    android:fillAfter="true"
```

```
    android:fromXDelta="0%p"
    android:startOffset="1900"
    android:toXDelta="-75%p" />
<translate
    android:duration="800"
    android:fillAfter="true"
    android:fromYDelta="0%p"
    android:startOffset="2700"
    android:toYDelta="-70%p" />

<!-- Rotate 360 degrees -->
<rotate
    android:duration="1000"
    android:fromDegrees="0"
    android:interpolator="@android:anim/cycle_interpolator"
    android:pivotX="50%"
    android:pivotY="50%"
    android:startOffset="3800"
    android:repeatCount="infinite"
    android:repeatMode="restart"
    android:toDegrees="360" />

</set>
```

Here a different **android:startOffset** is used from the transitions to keep them sequential.

Together Animation

together.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"  
    android:fillAfter="true"  
    android:interpolator="@android:anim/linear_interpolator" >
```

```
<!-- Move -->
```

```
<scale  
    xmlns:android="https://schemas.android.com/apk/res/android"  
    android:duration="4000"  
    android:fromXScale="1"  
    android:fromYScale="1"  
    android:pivotX="50%"  
    android:pivotY="50%"  
    android:toXScale="4"  
    android:toYScale="4" >  
</scale>
```

```
<!-- Rotate 180 degrees -->
```

```
<rotate  
    android:duration="500"  
    android:fromDegrees="0"  
    android:pivotX="50%"  
    android:pivotY="50%"  
    android:repeatCount="infinite"  
    android:repeatMode="restart"  
    android:toDegrees="360" />
```

</set>

Android Animation is used to give the UI a rich look and feel. Animations in android apps can be performed through XML or android code. In this android animation tutorial we'll go with XML codes for adding animations into our application.

Android Animation

Animation in android apps is the process of creating motion and shape change. The basic ways of animation that we'll look upon in this tutorial are:

1. Fade In Animation
2. Fade Out Animation
3. Cross Fading Animation
4. Blink Animation
5. Zoom In Animation
6. Zoom Out Animation
7. Rotate Animation
8. Move Animation
9. Slide Up Animation
10. Slide Down Animation
11. Bounce Animation
12. Sequential Animation
13. Together Animation

Android Animation Example XML

We create a resource directory under the res folder names **anim** to keep all the xml files containing the animation logic. Following is a sample xml file showing an android animation code logic.

sample_animation.xml

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

```
<scale xmlns:android="https://schemas.android.com/apk/res/android"
```

```
    android:interpolator="@android:anim/accelerate_decelerate_interpolator"
```

```

android:duration="300"

android:fillAfter="true"

android:fromXScale="0.0"

android:fromYScale="0.0"

android:toXScale="1.0"

android:toYScale="1.0" />

```

- **android:interpolator** : It is the rate of change in animation. We can define our own interpolators using the time as the constraint. In the above xml code an inbuilt interpolator is assigned
- **android:duration** : Duration of the animation in which the animation should complete. It is 300ms here. This is generally the ideal duration to show the transition on the screen.

The start and end of the animation are set using:

```

android:fromTRANSFORMATION

android:toTRANSFORMATION

```

- **TRANSFORMATION** : is the transformation that we want to specify. In our case we start with an x and y scale of 0 and end with an x and y scale of 1
- **android:fillAfter** : property specifies whether the view should be visible or hidden at the end of the animation. We've set it visible in the above code. If it sets to false, the element changes to its previous state after the animation
- **android:startOffset** : It is the waiting time before an animation starts. This property is mainly used to perform multiple animations in a sequential manner
- **android:repeatMode** : This is useful when you want the animation to be repeat
- **android:repeatCount** : This defines number of repetitions on animation. If we set this value to infinite then animation will repeat infinite times

Loading Animation when UI widget is clicked

Our aim is to show an animation when any widget(lets say TextView) is clicked. For that we need to use the Animation Class. The xml file that contains the animation logic is loaded using **AnimationUtils** class by calling the loadAnimation() function. The below snippet shows this implementation.

```

Animation animation;

```

```

animation = AnimationUtils.loadAnimation(getApplicationContext(),

        R.anim.sample_animation);

```


To start the animation we need to call the `startAnimation()` function on the UI element as shown in the snippet below:

```
sampleTextView.startAnimation(animation);
```

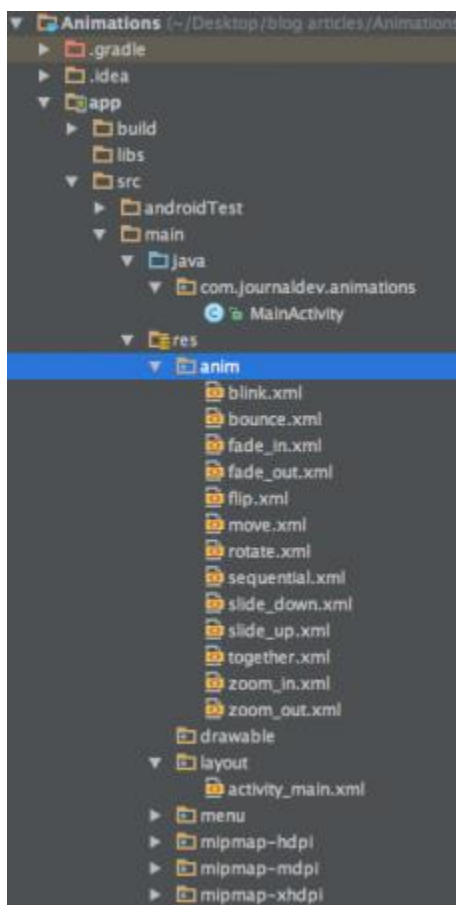
Here we perform the animation on a textview component by passing the type of Animation as the parameter.

Setting the Animation Listeners

This is only needed if we wish to listen to events like start, end or repeat. For this the activity must implement **AnimationListener** and the following methods need to be overridden.

- **onAnimationStart** : This will be triggered once the animation started
- **onAnimationEnd** : This will be triggered once the animation is over
- **onAnimationRepeat** : This will be triggered if the animation repeats

Android Animation Project Structure



As you can see, we've included the xml of all the major types of animations covered above.

Android Animation Examples XML Code

Here I am providing sample code for most of the common android animations.

Fade In Animation

fade_in.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
    android:fillAfter="true" >

    <alpha
        android:duration="1000"
        android:fromAlpha="0.0"
        android:interpolator="@android:anim/accelerate_interpolator"
        android:toAlpha="1.0" />

</set>
```

Here **alpha** references the opacity of an object. An object with lower alpha values is more transparent, while an object with higher alpha values is less transparent, more opaque. Fade in animation is nothing but increasing alpha value from 0 to 1.

Fade Out Animation

fade_out.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
    android:fillAfter="true" >

    <alpha
        android:duration="1000"
```

```
android:fromAlpha="1.0"  
android:interpolator="@android:anim/accelerate_interpolator"  
android:toAlpha="0.0" />
```

```
</set>
```

Fade out android animation is exactly opposite to fade in, where we need to decrease the alpha value from 1 to 0.

Cross Fading Animation

Cross fading is performing fade in animation on one TextView while other TextView is fading out. This can be done by using fade_in.xml and fade_out.xml on the two TextViews. The code will be discussed in the MainActivity.java

Blink Animation

blink.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android">  
  <alpha android:fromAlpha="0.0"  
    android:toAlpha="1.0"  
    android:interpolator="@android:anim/accelerate_interpolator"  
    android:duration="600"  
    android:repeatMode="reverse"  
    android:repeatCount="infinite" />  
</set>
```

Here fade in and fade out are performed infinitely in reverse mode each time.

Zoom In Animation

zoom_in.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
    android:fillAfter="true" >

<scale
    xmlns:android="https://schemas.android.com/apk/res/android"
    android:duration="1000"
    android:fromXScale="1"
    android:fromYScale="1"
    android:pivotX="50%"
    android:pivotY="50%"
    android:toXScale="3"
    android:toYScale="3" >
</scale>

</set>
```

We use `pivotX="50%"` and `pivotY="50%"` to perform zoom from the center of the element.

Zoom Out Animation

zoom_out.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
    android:fillAfter="true" >

<scale
    xmlns:android="https://schemas.android.com/apk/res/android"
    android:duration="1000"
    android:fromXScale="1.0"
```

```
    android:fromYScale="1.0"  
    android:pivotX="50%"  
    android:pivotY="50%"  
    android:toXScale="0.5"  
    android:toYScale="0.5" >  
</scale>
```

```
</set>
```

Notice that **android:from** and **android:to** are opposite in zoom_in.xml and zoom_out.xml.

Rotate Animation

rotate.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android">  
    <rotate android:fromDegrees="0"  
        android:toDegrees="360"  
        android:pivotX="50%"  
        android:pivotY="50%"  
        android:duration="600"  
        android:repeatMode="restart"  
        android:repeatCount="infinite"  
        android:interpolator="@android:anim/cycle_interpolator"/>  
</set>
```

A **from/toDegrees** tag is used here to specify the degrees and a cyclic interpolator is used.

Move Animation

move.xml

```
<set
  xmlns:android="https://schemas.android.com/apk/res/android"
  android:interpolator="@android:anim/linear_interpolator"
  android:fillAfter="true">

  <translate
    android:fromXDelta="0%p"
    android:toXDelta="75%p"
    android:duration="800" />
</set>
```

Slide Up Animation

slide_up.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
  android:fillAfter="true" >

  <scale
    android:duration="500"
    android:fromXScale="1.0"
    android:fromYScale="1.0"
    android:interpolator="@android:anim/linear_interpolator"
    android:toXScale="1.0"
    android:toYScale="0.0" />
```

```
</set>
```

It's achieved by setting **android:fromYScale="1.0"** and **android:toYScale="0.0"** inside the **scale** tag.

Slide Down Animation

slide_down.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
    android:fillAfter="true">
```

```
<scale
```

```
    android:duration="500"
```

```
    android:fromXScale="1.0"
```

```
    android:fromYScale="0.0"
```

```
    android:toXScale="1.0"
```

```
    android:toYScale="1.0" />
```

```
</set>
```

This is just the opposite of slide_up.xml.

Bounce Animation

bounce.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
    android:fillAfter="true"
    android:interpolator="@android:anim/bounce_interpolator">
```

```
<scale
    android:duration="500"
    android:fromXScale="1.0"
    android:fromYScale="0.0"
    android:toXScale="1.0"
    android:toYScale="1.0" />
```

```
</set>
```

Here bounce interpolator is used to complete the animation in bouncing fashion.

Sequential Animation

sequential.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
    android:fillAfter="true"
    android:interpolator="@android:anim/linear_interpolator" >
```

```
<!-- Move -->
```

```
<translate
    android:duration="800"
    android:fillAfter="true"
    android:fromXDelta="0%p"
    android:startOffset="300"
    android:toXDelta="75%p" />
```

```
<translate
    android:duration="800"
```

```
    android:fillAfter="true"
    android:fromYDelta="0%p"
    android:startOffset="1100"
    android:toYDelta="70%p" />
<translate
    android:duration="800"
    android:fillAfter="true"
    android:fromXDelta="0%p"
    android:startOffset="1900"
    android:toXDelta="-75%p" />
<translate
    android:duration="800"
    android:fillAfter="true"
    android:fromYDelta="0%p"
    android:startOffset="2700"
    android:toYDelta="-70%p" />

<!-- Rotate 360 degrees -->
<rotate
    android:duration="1000"
    android:fromDegrees="0"
    android:interpolator="@android:anim/cycle_interpolator"
    android:pivotX="50%"
    android:pivotY="50%"
    android:startOffset="3800"
    android:repeatCount="infinite"
    android:repeatMode="restart"
```

```
    android:toDegrees="360" />
```

```
</set>
```

Here a different **android:startOffset** is used from the transitions to keep them sequential.

Together Animation

together.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
```

```
    android:fillAfter="true"
```

```
    android:interpolator="@android:anim/linear_interpolator" >
```

```
<!-- Move -->
```

```
<scale
```

```
    xmlns:android="https://schemas.android.com/apk/res/android"
```

```
    android:duration="4000"
```

```
    android:fromXScale="1"
```

```
    android:fromYScale="1"
```

```
    android:pivotX="50%"
```

```
    android:pivotY="50%"
```

```
    android:toXScale="4"
```

```
    android:toYScale="4" >
```

```
</scale>
```

```
<!-- Rotate 180 degrees -->
```

```
<rotate
```

```
    android:duration="500"  
    android:fromDegrees="0"  
    android:pivotX="50%"  
    android:pivotY="50%"  
    android:repeatCount="infinite"  
    android:repeatMode="restart"  
    android:toDegrees="360" />
```

```
</set>
```

Here android:startOffset is removed to let them happen simultaneously.

Code

The activity_main.xml layout consists of a ScrollView and RelativeLayout (we'll discuss this in a later tutorial) in which every animation type is invoked on the text using their respective buttons. The xml file is shown below :

activity_main.xml

```
<ScrollView xmlns:android="https://schemas.android.com/apk/res/android"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent" >
```

```
<RelativeLayout
```

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

```
<Button
```

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

```
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:layout_margin="5dp"  
android:text="Fade In" />
```

```
<TextView
```

```
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:textAppearance="?android:attr/textAppearanceMedium"  
    android:text="Fade In"  
    android:id="@+id/txt_fade_in"  
    android:layout_alignBottom="@+id/btnFadeIn"  
    android:layout_alignLeft="@+id/txt_fade_out"  
    android:layout_alignStart="@+id/txt_fade_out" />
```

```
<Button
```

```
    android:id="@+id/btnFadeOut"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_margin="5dp"  
    android:layout_below="@id/btnFadeIn"  
    android:text="Fade Out" />
```

```
<Button
```

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

```
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:layout_margin="5dp"  
android:layout_below="@id/btnFadeOut"  
android:text="Cross Fade" />
```

```
<TextView
```

```
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:textAppearance="?android:attr/textAppearanceMedium"  
    android:text="Cross Fade In"  
    android:id="@+id/txt_out"  
    android:visibility="gone"  
    android:layout_gravity="center_horizontal"  
    android:layout_alignTop="@+id/txt_in"  
    android:layout_alignLeft="@+id/txt_in"  
    android:layout_alignStart="@+id/txt_in" />
```

```
<Button
```

```
    android:id="@+id/btnBlink"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_margin="5dp"  
    android:layout_below="@id/btnCrossFade"  
    android:text="Blink" />
```

```
<Button
```

```
android:id="@+id/btnZoomIn"  
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:layout_margin="5dp"  
android:layout_below="@id/btnBlink"  
android:text="Zoom In" />
```

```
<TextView
```

```
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:textAppearance="?android:attr/textAppearanceMedium"  
    android:text="Blink"  
    android:id="@+id/txt_blink"  
    android:layout_gravity="center_horizontal"  
    android:layout_alignBottom="@+id/btnBlink"  
    android:layout_alignLeft="@+id/txt_zoom_in"  
    android:layout_alignStart="@+id/txt_zoom_in" />
```

```
<Button
```

```
    android:id="@+id/btnZoomOut"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_margin="5dp"  
    android:layout_below="@id/btnZoomIn"  
    android:text="Zoom Out" />
```

```
<Button
```

```
android:id="@+id/btnRotate"  
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:layout_margin="5dp"  
android:layout_below="@id/btnZoomOut"  
android:text="Rotate" />
```

<Button

```
android:id="@+id/btnMove"  
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:layout_margin="5dp"  
android:layout_below="@id/btnRotate"  
android:text="Move" />
```

<Button

```
android:id="@+id/btnSlideUp"  
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:layout_margin="5dp"  
android:layout_below="@id/btnMove"  
android:text="Slide Up" />
```

<TextView

```
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:textAppearance="?android:attr/textAppearanceMedium"
```

```
android:text="Fade Out"
android:id="@+id/txt_fade_out"
android:layout_gravity="center_horizontal"
android:layout_alignBottom="@+id/btnFadeOut"
android:layout_alignLeft="@+id/txt_in"
android:layout_alignStart="@+id/txt_in" />
```

```
<Button
```

```
android:id="@+id/btnSlideDown"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_margin="5dp"
android:layout_below="@id/btnSlideUp"
android:text="Slide Down" />
```

```
<Button
```

```
android:id="@+id/btnBounce"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_margin="5dp"
android:layout_below="@id/btnSlideDown"
android:text="Bounce" />
```

```
<Button
```

```
android:id="@+id/btnSequential"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
```

```
android:layout_margin="5dp"  
android:layout_below="@id/btnBounce"  
android:text="Sequential Animation" />
```

```
<Button
```

```
    android:id="@+id/btnTogether"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_below="@id/btnSequential"  
    android:layout_margin="5dp"  
    android:text="Together Animation" />
```

```
<TextView
```

```
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:textAppearance="?android:attr/textAppearanceMedium"  
    android:text="Cross Fade Out"  
    android:id="@+id/txt_in"  
    android:layout_gravity="center_horizontal"  
    android:layout_alignBottom="@+id/btnCrossFade"  
    android:layout_alignLeft="@+id/txt_blink"  
    android:layout_alignStart="@+id/txt_blink" />
```

```
<TextView
```

```
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:textAppearance="?android:attr/textAppearanceMedium"
```

```
android:text="Zoom In"
android:id="@+id/txt_zoom_in"
android:layout_alignBottom="@+id/btnZoomIn"
android:layout_alignLeft="@+id/txt_zoom_out"
android:layout_alignStart="@+id/txt_zoom_out" />
```

```
<TextView
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="Zoom Out"
    android:id="@+id/txt_zoom_out"
    android:layout_alignBottom="@+id/btnZoomOut"
    android:layout_toRightOf="@+id/btnSequential"
    android:layout_toEndOf="@+id/btnSequential" />
```

```
<TextView
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="Rotate"
    android:id="@+id/txt_rotate"
    android:layout_above="@+id/btnMove"
    android:layout_toRightOf="@+id/btnSequential"
    android:layout_toEndOf="@+id/btnSequential" />
```

```
<TextView
```

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textAppearance="?android:attr/textAppearanceMedium"
android:text="Move"
android:id="@+id/txt_move"
android:layout_alignBottom="@+id/btnMove"
android:layout_alignLeft="@+id/txt_slide_up"
android:layout_alignStart="@+id/txt_slide_up" />
```

```
<TextView
```

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textAppearance="?android:attr/textAppearanceMedium"
android:text="Slide Up"
android:id="@+id/txt_slide_up"
android:layout_alignBottom="@+id/btnSlideUp"
android:layout_toRightOf="@+id/btnSequential"
android:layout_toEndOf="@+id/btnSequential" />
```

```
<TextView
```

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textAppearance="?android:attr/textAppearanceMedium"
android:text="Slide Down"
android:id="@+id/txt_slide_down"
android:layout_alignBottom="@+id/btnSlideDown"
android:layout_alignLeft="@+id/txt_slide_up"
```

```
android:layout_alignStart="@+id/txt_slide_up" />
```

```
<TextView
```

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

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

```
    android:textAppearance="?android:attr/textAppearanceMedium"
```

```
    android:text="Bounce"
```

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

```
    android:layout_alignBottom="@+id/btnBounce"
```

```
    android:layout_alignLeft="@+id/txt_slide_down"
```

```
    android:layout_alignStart="@+id/txt_slide_down" />
```

```
<TextView
```

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

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

```
    android:textAppearance="?android:attr/textAppearanceMedium"
```

```
    android:text="Sequential"
```

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

```
    android:layout_alignBottom="@+id/btnSequential"
```

```
    android:layout_alignLeft="@+id/txt_bounce"
```

```
    android:layout_alignStart="@+id/txt_bounce" />
```

```
<TextView
```

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

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

```
    android:textAppearance="?android:attr/textAppearanceMedium"
```

```
    android:text="Together"
```

```
    android:id="@+id/txt_tog"
    android:layout_alignBottom="@+id/btnTogether"
    android:layout_toRightOf="@+id/btnSequential"
    android:layout_toEndOf="@+id/btnSequential" />
```

```
</RelativeLayout>
```

```
</ScrollView>
```

To sum up, a RelativeLayout, as the name suggests the arrangement of UI Components is relative to each other.

The MainActivity.java file contains the onClick Listeners for every button related to its animation type. It's source code is given below.

```
package com.journaldev.animations;
```

```
import android.app.Activity;
```

```
import android.content.Intent;
```

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

```
import android.view.View;
```

```
import android.view.animation.Animation;
```

```
import android.view.animation.AnimationUtils;
```

```
import android.widget.Button;
```

```
import android.widget.TextView;
```

```
public class MainActivity extends Activity {
```

```
    Button btnFadeIn, btnFadeOut, btnCrossFade, btnBlink, btnZoomIn,
```

```
        btnZoomOut, btnRotate, btnMove, btnSlideUp, btnSlideDown,
```

```
        btnBounce, btnSequential, btnTogether;
```

Animation

animFadeIn,animFadeOut,animBlink,animZoomIn,animZoomOut,animRotate

,animMove,animSlideUp,animSlideDown,animBounce,animSequential,animTogether,animCrossFadeIn,animCrossFadeOut;

TextView

txtFadeIn,txtFadeOut,txtBlink,txtZoomIn,txtZoomOut,txtRotate,txtMove,txtSlideUp,
txtSlideDown,txtBounce,txtSeq,txtTog,txtIn,txtOut;

@Override

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
  
    btnFadeIn = (Button) findViewById(R.id.btnFadeIn);  
    btnFadeOut = (Button) findViewById(R.id.btnFadeOut);  
    btnCrossFade = (Button) findViewById(R.id.btnCrossFade);  
    btnBlink = (Button) findViewById(R.id.btnBlink);  
    btnZoomIn = (Button) findViewById(R.id.btnZoomIn);  
    btnZoomOut = (Button) findViewById(R.id.btnZoomOut);  
    btnRotate = (Button) findViewById(R.id.btnRotate);  
    btnMove = (Button) findViewById(R.id.btnMove);  
    btnSlideUp = (Button) findViewById(R.id.btnSlideUp);  
    btnSlideDown = (Button) findViewById(R.id.btnSlideDown);  
    btnBounce = (Button) findViewById(R.id.btnBounce);  
    btnSequential = (Button) findViewById(R.id.btnSequential);  
    btnTogether = (Button) findViewById(R.id.btnTogether);  
    txtFadeIn=(TextView)findViewById(R.id.txt_fade_in);  
    txtFadeOut=(TextView)findViewById(R.id.txt_fade_out);  
}
```

```
txtBlink=(TextView)findViewById(R.id.txt_blink);
txtZoomIn=(TextView)findViewById(R.id.txt_zoom_in);
txtZoomOut=(TextView)findViewById(R.id.txt_zoom_out);
txtRotate=(TextView)findViewById(R.id.txt_rotate);
txtMove=(TextView)findViewById(R.id.txt_move);
txtSlideUp=(TextView)findViewById(R.id.txt_slide_up);
txtSlideDown=(TextView)findViewById(R.id.txt_slide_down);
txtBounce=(TextView)findViewById(R.id.txt_bounce);
txtSeq=(TextView)findViewById(R.id.txt_seq);
txtTog=(TextView)findViewById(R.id.txt_tog);
txtIn=(TextView)findViewById(R.id.txt_in);
txtOut=(TextView)findViewById(R.id.txt_out);
animFadeIn = AnimationUtils.loadAnimation(getApplicationContext(),
    R.anim.fade_in);

animFadeIn = AnimationUtils.loadAnimation(getApplicationContext(),
    R.anim.fade_in);

// fade in
btnFadeIn.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        txtFadeIn.setVisibility(View.VISIBLE);
        txtFadeIn.startAnimation(animFadeIn);
    }
});
```

```
animFadeOut = AnimationUtils.loadAnimation(getApplicationContext(),
    R.anim.fade_out);

// fade out
btnFadeOut.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        txtFadeOut.setVisibility(View.VISIBLE);
        txtFadeOut.startAnimation(animFadeOut);
    }
});

animCrossFadeIn = AnimationUtils.loadAnimation(getApplicationContext(),
    R.anim.fade_in);

animCrossFadeOut = AnimationUtils.loadAnimation(getApplicationContext(),
    R.anim.fade_out);

// cross fade
btnCrossFade.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        txtOut.setVisibility(View.VISIBLE);
        // start fade in animation
        txtOut.startAnimation(animCrossFadeIn);

        // start fade out animation
        txtIn.startAnimation(animCrossFadeOut);
    }
});
```

```
animBlink = AnimationUtils.loadAnimation(getApplicationContext(),
    R.anim.blink);

// blink

btnBlink.setOnClickListener(new View.OnClickListener() {

    @Override

    public void onClick(View v) {

        txtBlink.setVisibility(View.VISIBLE);

        txtBlink.startAnimation(animBlink);

    }

});

animZoomIn = AnimationUtils.loadAnimation(getApplicationContext(),

    R.anim.zoom_in);

// Zoom In

btnZoomIn.setOnClickListener(new View.OnClickListener() {

    @Override

    public void onClick(View v) {

        txtZoomIn.setVisibility(View.VISIBLE);

        txtZoomIn.startAnimation(animZoomIn);

    }

});

animZoomOut = AnimationUtils.loadAnimation(getApplicationContext(),

    R.anim.zoom_out);

// Zoom Out

btnZoomOut.setOnClickListener(new View.OnClickListener() {

    @Override

    public void onClick(View v) {
```

```
txtZoomOut.setVisibility(View.VISIBLE);

txtZoomOut.startAnimation(animZoomOut);

}

});

animRotate = AnimationUtils.loadAnimation(getApplicationContext(),
    R.anim.rotate);

// Rotate
btnRotate.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        txtRotate.startAnimation(animRotate);
    }
});

animMove = AnimationUtils.loadAnimation(getApplicationContext(),
    R.anim.move);

// Move
btnMove.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        txtMove.startAnimation(animMove);
    }
});

animSlideUp = AnimationUtils.loadAnimation(getApplicationContext(),
    R.anim.slide_up);

// Slide Up
btnSlideUp.setOnClickListener(new View.OnClickListener() {
```

```
@Override

public void onClick(View v) {

    txtSlideUp.startAnimation(animSlideUp);

}

});

animSlideDown = AnimationUtils.loadAnimation(getApplicationContext(),

    R.anim.slide_down);

// Slide Down

btnSlideDown.setOnClickListener(new View.OnClickListener() {

    @Override

    public void onClick(View v) {

        txtSlideDown.startAnimation(animSlideDown);

    }

});

animBounce = AnimationUtils.loadAnimation(getApplicationContext(),

    R.anim.bounce);

// Slide Down

btnBounce.setOnClickListener(new View.OnClickListener() {

    @Override

    public void onClick(View v) {

        txtBounce.startAnimation(animBounce);

    }

});

animSequential = AnimationUtils.loadAnimation(getApplicationContext(),

    R.anim.sequential);

// Sequential

btnSequential.setOnClickListener(new View.OnClickListener() {
```

```
@Override
public void onClick(View v) {

    txtSeq.startAnimation(animSequential);
}
});

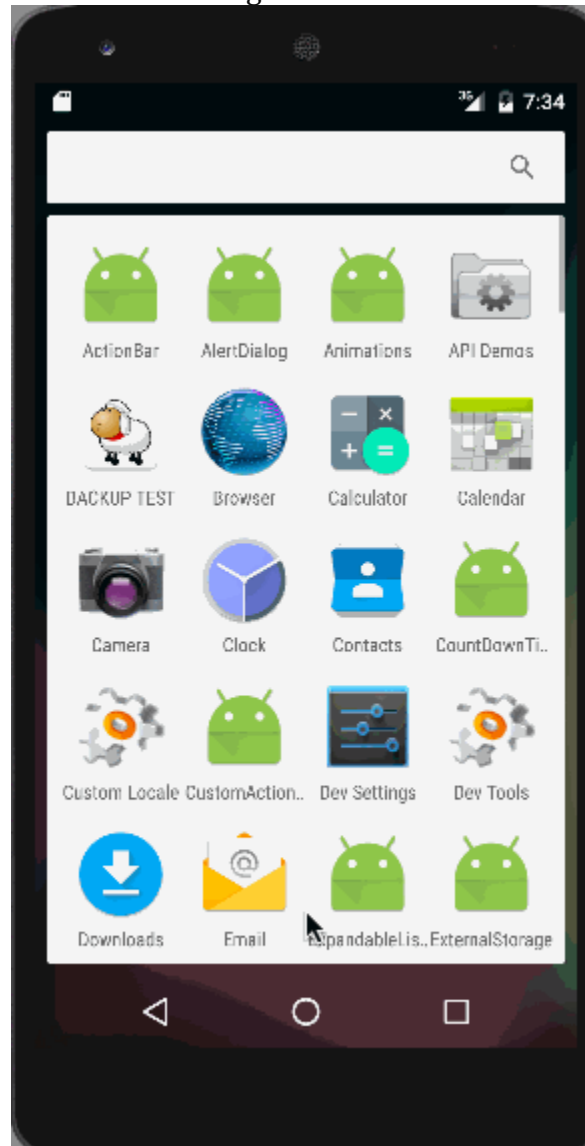
animTogether = AnimationUtils.loadAnimation(getApplicationContext(),
    R.anim.together);

// Together
btnTogether.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        txtTog.startAnimation(animTogether);
    }
});

}
}
```

As discussed before each textView animation is started by invoking the respective animation object in which the animation logic is loaded by **AnimationUtils.loadAnimation()** method. The crossFade animation consists of two TextViews in which one fades out and the other fades in.

Below is a short video showing all the animations in our application.



The together animation is seen in the image above. Note that these animation when run on an emulator wont be smooth, so it's recommended to run the application on a normal device.

This brings an end to android animation example tutorial.

Android SQLite Database

Android SQLite is the mostly preferred way to store data for android applications. For many applications, SQLite is the apps backbone whether it's used directly or via some third-party wrapper. Below is the final app we will create today using Android SQLite database.

Android SQLite

Android SQLite is a very lightweight database which comes with Android OS. Android SQLite combines a clean SQL interface with a very small memory footprint and decent speed. For Android, SQLite is “baked into” the Android runtime, so every Android application can create its own SQLite databases.

Android SQLite native API is not JDBC, as JDBC might be too much overhead for a memory-limited smartphone. Once a database is created successfully its located in **data/data//databases/** accessible from Android Device Monitor.

SQLite is a typical **relational database**, containing tables (which consists of rows and columns), indexes etc. We can create our own tables to hold the data accordingly. This structure is referred to as a **schema**.

Android SQLite SQLiteOpenHelper

Android has features available to handle changing database schemas, which mostly depend on using the SQLiteOpenHelper class.

SQLiteOpenHelper is designed to get rid of two very common problems.

1. When the application runs the first time – At this point, we do not yet have a database. So we will have to create the tables, indexes, starter data, and so on.
2. When the application is upgraded to a newer schema – Our database will still be on the old schema from the older edition of the app. We will have option to alter the database schema to match the needs of the rest of the app.

SQLiteOpenHelper wraps up these logic to create and upgrade a database as per our specifications. For that we’ll need to create a custom subclass of SQLiteOpenHelper implementing at least the following three methods.

1. **Constructor** : This takes the Context (e.g., an Activity), the name of the database, an optional cursor factory (we’ll discuss this later), and an integer representing the version of the database schema you are using (typically starting from 1 and increment later).

```
2.
public DatabaseHelper(Context context) {
    super(context, DB_NAME, null, DB_VERSION);
}
```

3. **onCreate(SQLiteDatabase db)** : It’s called when there is no database and the app needs one. It passes us a SQLiteDatabase object, pointing to a newly-created database, that we can populate with tables and initial data.

4. **onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)** : It's called when the schema version we need does not match the schema version of the database, It passes us a **SQLiteDatabase** object and the old and new version numbers. Hence we can figure out the best way to convert the database from the old schema to the new one.

We define a DBManager class to perform all database CRUD(Create, Read, Update and Delete) operations.

Opening and Closing Android SQLite Database Connection

Before performing any database operations like insert, update, delete records in a table, first open the database connection by calling **getWritableDatabase()** method as shown below:

```
public DBManager open() throws SQLException {  
    dbHelper = new DatabaseHelper(context);  
    database = dbHelper.getWritableDatabase();  
    return this;  
}
```

The **dbHelper** is an instance of the subclass of SQLiteOpenHelper.

To close a database connection the following method is invoked.

```
public void close() {  
    dbHelper.close();  
}
```

Inserting new Record into Android SQLite database table

The following code snippet shows how to insert a new record in the android SQLite database.

```
public void insert(String name, String desc) {  
    ContentValues contentValues = new ContentValues();  
    contentValues.put(DatabaseHelper.SUBJECT, name);  
    contentValues.put(DatabaseHelper.DESC, desc);
```

```
database.insert(DatabaseHelper.TABLE_NAME, null, contentValues);  
  
}
```

Content Values creates an empty set of values using the given initial size. We'll discuss the other instance values when we jump into the coding part.

Updating Record in Android SQLite database table

The following snippet shows how to update a single record.

```
public int update(long _id, String name, String desc) {  
  
    ContentValues contentValues = new ContentValues();  
    contentValues.put(DatabaseHelper.SUBJECT, name);  
    contentValues.put(DatabaseHelper.DESC, desc);  
  
    int i = database.update(DatabaseHelper.TABLE_NAME, contentValues,  
DatabaseHelper._ID + " = " + _id, null);  
  
    return i;  
  
}
```

Android SQLite – Deleting a Record

We just need to pass the id of the record to be deleted as shown below.

```
public void delete(long _id) {  
  
    database.delete(DatabaseHelper.TABLE_NAME, DatabaseHelper._ID + "=" + _id, null);  
  
}
```

Android SQLite Cursor

A Cursor represents the entire result set of the query. Once the query is fetched a call to **cursor.moveToFirst()** is made. Calling `moveToFirst()` does two things:

- It allows us to test whether the query returned an empty set (by testing the return value)
- It moves the cursor to the first result (when the set is not empty)

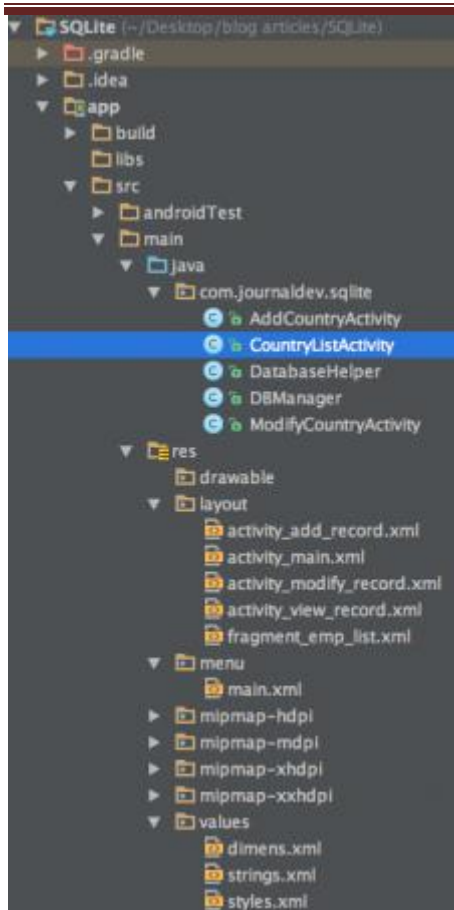
The following code is used to fetch all records:


```
public Cursor fetch() {  
    String[] columns = new String[] { DatabaseHelper._ID, DatabaseHelper.SUBJECT,  
DatabaseHelper.DESC };  
    Cursor cursor = database.query(DatabaseHelper.TABLE_NAME, columns, null, null,  
null, null, null);  
    if (cursor != null) {  
        cursor.moveToFirst();  
    }  
    return cursor;  
}
```

Another way to use a Cursor is to wrap it in a CursorAdapter. Just as ArrayAdapter adapts arrays, CursorAdapter adapts Cursor objects, making their data available to an AdapterView like a ListView.

Let's jump to our project that uses SQLite to store some meaningful data.

Android SQLite Example Project Structure



In this application we wish to create records that store Country names and their respective currencies in the form of a **ListView**. We cover all the features discussed above.

Android SQLite Project Code

The application consists of 5 classes. We begin with defining with **DatabaseHelper**, which is a subclass of SQLiteOpenHelper as follows:

DatabaseHelper.java

```
package com.journaldev.sqlite;
```

```
import android.content.Context;
```

```
import android.database.sqlite.SQLiteDatabase;
```

```
import android.database.sqlite.SQLiteOpenHelper;
```

```
public class DatabaseHelper extends SQLiteOpenHelper {
```

```
// Table Name
public static final String TABLE_NAME = "COUNTRIES";

// Table columns
public static final String _ID = "_id";
public static final String SUBJECT = "subject";
public static final String DESC = "description";

// Database Information
static final String DB_NAME = "JOURNALDEV_COUNTRIES.DB";

// database version
static final int DB_VERSION = 1;

// Creating table query
private static final String CREATE_TABLE = "create table " + TABLE_NAME + "(" + _ID
    + " INTEGER PRIMARY KEY AUTOINCREMENT, " + SUBJECT + " TEXT NOT NULL, "
    + DESC + " TEXT)";

public DatabaseHelper(Context context) {
    super(context, DB_NAME, null, DB_VERSION);
}

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

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

As discussed above we have overridden the `onCreate` and `onUpgrade` methods besides the constructor. We've assigned the names to the database and the table as `JOURNALDEV_COUNTRIES.DB` and `COUNTRIES` respectively. The index column is auto incremented whenever a new row is inserted. The column names for country and currency are "subject" and "description".

The `DBManager` class is where the `DatabaseHelper` is initialized and the CRUD Operations are defined. Below is the code for this class:

`DBManager.java`

```
package com.journaldev.sqlite;

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

public class DBManager {

    private DatabaseHelper dbHelper;

    private Context context;
```

```
private SQLiteDatabase database;
```

```
public DBManager(Context c) {  
    context = c;  
}
```

```
public DBManager open() throws SQLException {  
    dbHelper = new DatabaseHelper(context);  
    database = dbHelper.getWritableDatabase();  
    return this;  
}
```

```
public void close() {  
    dbHelper.close();  
}
```

```
public void insert(String name, String desc) {  
    ContentValues contentValues = new ContentValues();  
    contentValues.put(DatabaseHelper.SUBJECT, name);  
    contentValues.put(DatabaseHelper.DESC, desc);  
    database.insert(DatabaseHelper.TABLE_NAME, null, contentValues);  
}
```

```
public Cursor fetch() {  
    String[] columns = new String[] { DatabaseHelper._ID, DatabaseHelper.SUBJECT,  
    DatabaseHelper.DESC };  
    Cursor cursor = database.query(DatabaseHelper.TABLE_NAME, columns, null, null,  
    null, null, null);  
}
```

```
        if (cursor != null) {
            cursor.moveToFirst();
        }
        return cursor;
    }

    public int update(long _id, String name, String desc) {
        ContentValues contentValues = new ContentValues();
        contentValues.put(DatabaseHelper.SUBJECT, name);
        contentValues.put(DatabaseHelper.DESC, desc);

        int i = database.update(DatabaseHelper.TABLE_NAME, contentValues,
            DatabaseHelper._ID + " = " + _id, null);

        return i;
    }

    public void delete(long _id) {
        database.delete(DatabaseHelper.TABLE_NAME, DatabaseHelper._ID + "=" + _id, null);
    }
}
```

The CountryListActivity.java class is the activity which is launched when the application starts. Below is layout defined for it:
fragment_emp_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="https://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
```

```
<ListView
    android:id="@+id/list_view"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:dividerHeight="1dp"
    android:padding="10dp" >

</ListView>

<TextView
    android:id="@+id/empty"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerInParent="true"
    android:text="@string/empty_list_text" />

</RelativeLayout>
```

Here a ListView component is defined to include the records stored in the database. Initially the ListView would be empty hence a TextView is used to display the same.

CountryListActivity.java

```
package com.journaldev.sqlite;

import android.content.Intent;
import android.database.Cursor;
import android.os.Bundle;
import android.support.v4.widget.SimpleCursorAdapter;
import android.support.v7.app.ActionBarActivity;
```

```
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.TextView;

public class CountryListActivity extends ActionBarActivity {

    private DBManager dbManager;

    private ListView listView;

    private SimpleCursorAdapter adapter;

    final String[] from = new String[] { DatabaseHelper._ID,
        DatabaseHelper.SUBJECT, DatabaseHelper.DESC };

    final int[] to = new int[] { R.id.id, R.id.title, R.id.desc };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.fragment_emp_list);

        dbManager = new DBManager(this);
```

```
dbManager.open();

Cursor cursor = dbManager.fetch();

listView = (ListView) findViewById(R.id.list_view);
listView.setEmptyView(findViewById(R.id.empty));

adapter = new SimpleCursorAdapter(this, R.layout.activity_view_record, cursor, from,
to, 0);

adapter.notifyDataSetChanged();

listView.setAdapter(adapter);

// OnClickListiner For List Items
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {

    @Override

    public void onItemClick(AdapterView<?> parent, View view, int position, long
viewId) {

        TextView idTextView = (TextView) view.findViewById(R.id.id);

        TextView titleTextView = (TextView) view.findViewById(R.id.title);

        TextView descTextView = (TextView) view.findViewById(R.id.desc);

        String id = idTextView.getText().toString();

        String title = titleTextView.getText().toString();

        String desc = descTextView.getText().toString();

        Intent modify_intent = new Intent(getApplicationContext(),
ModifyCountryActivity.class);

        modify_intent.putExtra("title", title);
```

```
        modify_intent.putExtra("desc", desc);

        modify_intent.putExtra("id", id);

        startActivity(modify_intent);
    }
});
}

@Override

public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

@Override

public boolean onOptionsItemSelected(MenuItem item) {

    int id = item.getItemId();

    if (id == R.id.add_record) {

        Intent add_mem = new Intent(this, AddCountryActivity.class);
        startActivity(add_mem);

    }

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

```
}
```

In this activity the DBManager object is invoked to perform the CRUD Operations.

A SimpleCursorAdapter is defined to add elements to the list from the query results that are returned in an Cursor Object.

On list item click an intent is performed to open the ModifyCountryActivity class.

The menu contains an item to add a new record from the ActionBar. Here again an intent is performed to open the AddCountryActivity class. Below is menu.xml code.
menu.xml

```
<menu xmlns:android="https://schemas.android.com/apk/res/android"
      xmlns:app="https://schemas.android.com/apk/res-auto"
      xmlns:tools="https://schemas.android.com/tools"
      tools:context="com.example.sqlitesample.MainActivity" >

    <item
        android:id="@+id/add_record"
        android:icon="@android:drawable/ic_menu_add"
        android:orderInCategory="100"
        android:title="@string/add_record"
        app:showAsAction="always"/>

</menu>
```

The xml layout and code of AddCountryActivity.java file are defined below:
activity_add_record.xml

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

<LinearLayout xmlns:android="https://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

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

```
android:padding="20dp" >
```

```
<EditText
```

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

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

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

```
    android:ems="10"
```

```
    android:hint="@string/enter_title" >
```

```
    <requestFocus />
```

```
</EditText>
```

```
<EditText
```

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

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

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

```
    android:ems="10"
```

```
    android:hint="@string/enter_desc"
```

```
    android:inputType="textMultiLine"
```

```
    android:minLines="5" >
```

```
</EditText>
```

```
<Button
```

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

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

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

```
android:layout_gravity="center"  
android:text="@string/add_record" />
```

```
</LinearLayout>
```

Two EditText components that take the inputs for country and currency along with a button to add the values to the database and display it in the ListView are defined.

AddCountryActivity.java

```
package com.journaldev.sqlite;  
  
import android.app.Activity;  
import android.content.Intent;  
import android.os.Bundle;  
import android.view.View;  
import android.view.View.OnClickListener;  
import android.widget.Button;  
import android.widget.EditText;  
  
public class AddCountryActivity extends Activity implements OnClickListener {  
  
    private Button addTodoBtn;  
    private EditText subjectEditText;  
    private EditText descEditText;  
  
    private DBManager dbManager;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);

setTitle("Add Record");

setContentView(R.layout.activity_add_record);

subjectEditText = (EditText) findViewById(R.id.subject_edittext);
descEditText = (EditText) findViewById(R.id.description_edittext);

addTodoBtn = (Button) findViewById(R.id.add_record);

dbManager = new DBManager(this);
dbManager.open();
addTodoBtn.setOnClickListener(this);
}

@Override

public void onClick(View v) {
    switch (v.getId()) {
        case R.id.add_record:

            final String name = subjectEditText.getText().toString();
            final String desc = descEditText.getText().toString();

            dbManager.insert(name, desc);

            Intent main = new Intent(AddCountryActivity.this, CountryListActivity.class)
```

```
        .setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);

        startActivity(main);
        break;
    }
}

}
```

The CRUD operation performed here is adding a new record to the database.

The xml layout and code of ModifyCountryActivity.java file are defined below:

activity_modify_record.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="https://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="10dp" >

    <EditText
        android:id="@+id/subject_edittext"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="10dp"
        android:ems="10"
        android:hint="@string/enter_title" />
```

```
<EditText  
    android:id="@+id/description_edittext"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:ems="10"  
    android:hint="@string/enter_desc"  
    android:inputType="textMultiLine"  
    android:minLines="5" >  
</EditText>
```

```
<LinearLayout  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:weightSum="2"  
    android:gravity="center_horizontal"  
    android:orientation="horizontal" >
```

```
<Button  
    android:id="@+id/btn_update"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_weight="1"  
    android:text="@string/btn_update" />
```

```
<Button  
    android:id="@+id/btn_delete"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="@string/btn_delete" />
</LinearLayout>
```

```
</LinearLayout>
```

It's similar to the previous layout except that modify and delete buttons are added.

ModifyCountryActivity.java

```
package com.journaldev.sqlite;
```

```
import android.app.Activity;
```

```
import android.content.Intent;
```

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

```
import android.view.View;
```

```
import android.view.View.OnClickListener;
```

```
import android.widget.Button;
```

```
import android.widget.EditText;
```

```
public class ModifyCountryActivity extends Activity implements OnClickListener {
```

```
    private EditText titleText;
```

```
    private Button updateBtn, deleteBtn;
```

```
    private EditText descText;
```

```
    private long _id;
```

```
private DBManager dbManager;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setTitle("Modify Record");

    setContentView(R.layout.activity_modify_record);

    dbManager = new DBManager(this);
    dbManager.open();

    titleText = (EditText) findViewById(R.id.subject_edittext);
    descText = (EditText) findViewById(R.id.description_edittext);

    updateBtn = (Button) findViewById(R.id.btn_update);
    deleteBtn = (Button) findViewById(R.id.btn_delete);

    Intent intent = getIntent();
    String id = intent.getStringExtra("id");
    String name = intent.getStringExtra("title");
    String desc = intent.getStringExtra("desc");

    _id = Long.parseLong(id);
```

```
titleText.setText(name);  
descText.setText(desc);  
  
updateBtn.setOnClickListener(this);  
deleteBtn.setOnClickListener(this);  
}
```

```
@Override
```

```
public void onClick(View v) {  
    switch (v.getId()) {  
        case R.id.btn_update:  
            String title = titleText.getText().toString();  
            String desc = descText.getText().toString();  
  
            dbManager.update(_id, title, desc);  
            this.returnHome();  
            break;  
  
        case R.id.btn_delete:  
            dbManager.delete(_id);  
            this.returnHome();  
            break;  
    }  
}
```

```
public void returnHome() {  
    Intent home_intent = new Intent(getApplicationContext(), CountryListActivity.class)
```

```
.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);  
  
startActivity(home_intent);  
  
}  
  
}
```

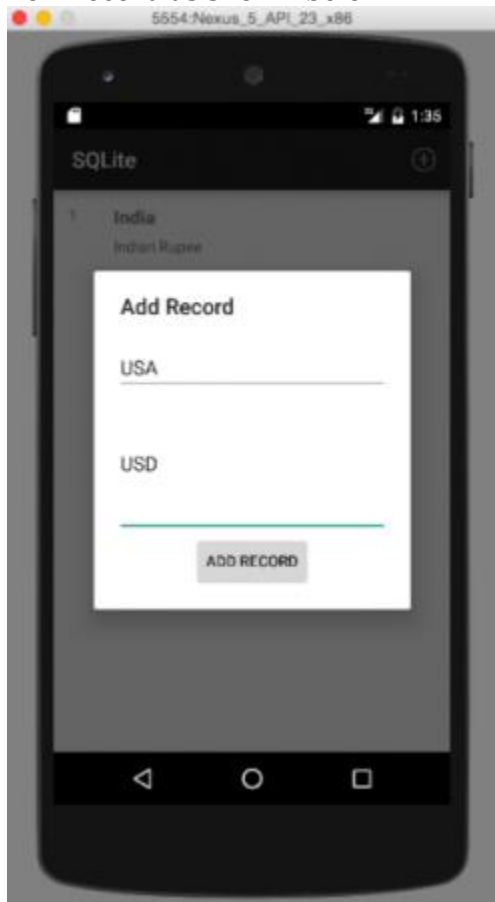
The CRUD operations performed here are updating and deleting a record.

The below images are the screenshots of the final output of our project.

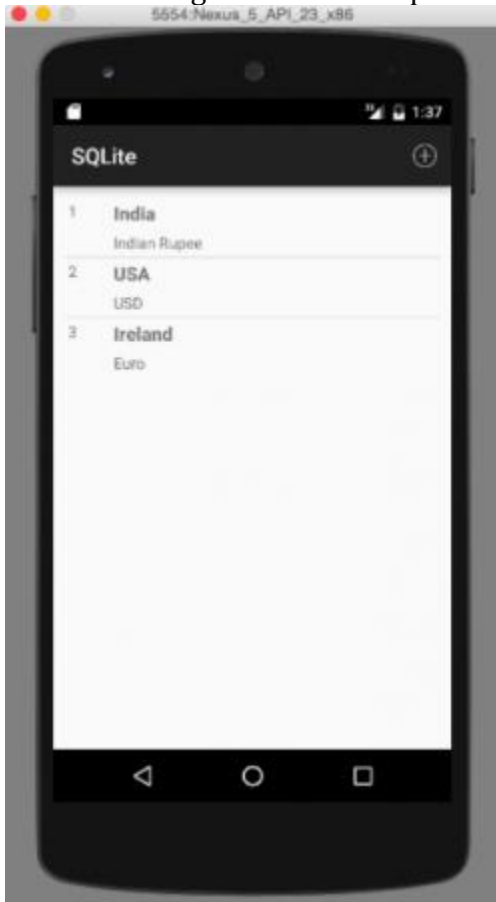
The first image is the output seen when the application is launched for the first time.



The second image is the result of clicking the menu option from the ActionBar to add a new record as shown below.



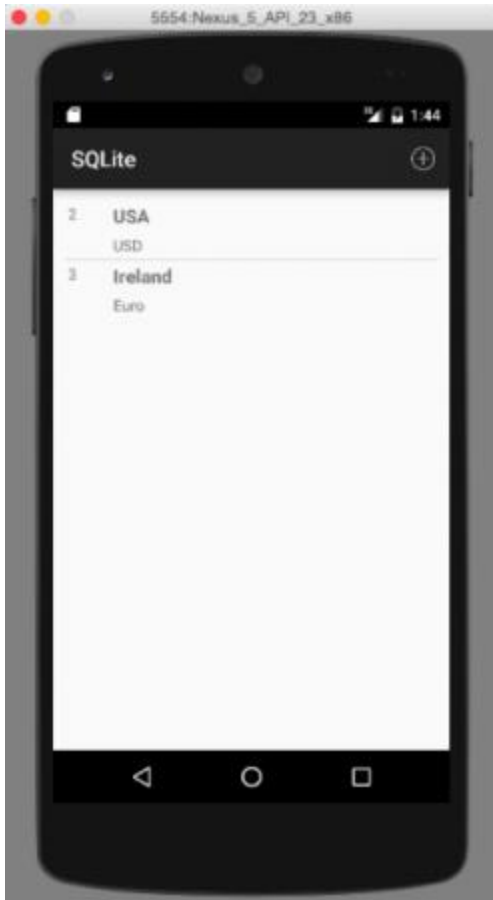
The third image shows an output when 3 records are added :



The fourth image shows the output when any list item is clicked to modify or delete a record :



The final image is the output when a record is deleted. In this example we delete the first record :

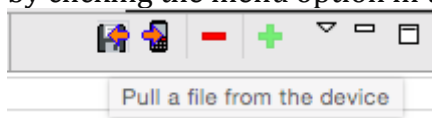


Opening the Android SQLite Database file

As we've discussed earlier in this tutorial, the database file is stored in the internal storage that is accessible from the Android Device Monitor as visible in the pic below.

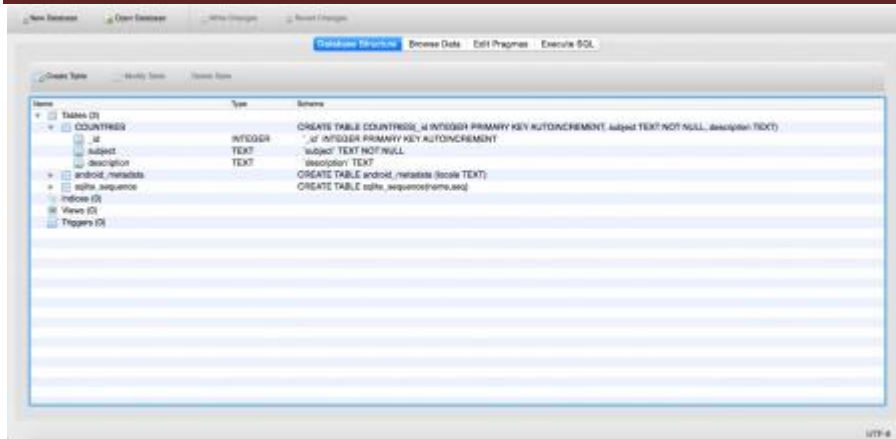


To view this database we need to pull this file from the device to our desktop. This is done by clicking the menu option in the top right as seen in the image below :



To open this file download the SQLiteBrowser from [this](#) link.

The snippets below show the schema and tables in the browser.



To view the table go to the Browse Data tab on top. The following image is seen:

	_id	subject	description
	Filter	Filter	Filter
1	2	USA	USD
2	3	Ireland	Euro

This brings an end to Android SQLite tutorial.

SQLiteDatabase

In Android, the SQLiteDatabase namespace defines the functionality to connect and manage a database. It provides functionality to create, delete, manage and display database content.

Simple steps to create a database and handle are as follows.

1. Create "SQLiteDatabase" object.
2. Open or Create a database and create a connection.
3. Perform insert, update or delete operation.
4. Create a Cursor to display data from the table of the database.
5. Close the database connectivity.

Following tutorial helps you to create a database and insert records in it.

Step 1: Instantiate "SQLiteDatabase" object

```
SQLiteDatabase db;
```

Before you can use the above object, you must import the android.database.sqlite.SQLiteDatabase namespace in your application.

```
db=openOrCreateDatabase(String path, int mode, SQLiteDatabase.CursorFactory factory)
```

This method is used to create/open database. As the name suggests, it will open a database connection if it is already there, otherwise, it will create a new one.

Example,

```
db=openOrCreateDatabase("XYZ_Database",SQLiteDatabase.CREATE_IF_NECESSARY,  
null);
```

Arguments:

String path	Name of the database
Int mode	operating mode. Use 0 or "MODE_PRIVATE" for the default operation, or "CREATE_IF_NECESSARY" if you like to give an option that "if a database is not there, create it"
CursorFactory factory	An optional factory class that is called to instantiate a cursor when a query is called

Step 2: Execute DDL command

```
db.execSQL(String sql) throws SQLException
```

This command is used to execute a single SQL statement that doesn't return any data means other than SELECT or any other.

```
db.execSQL("Create Table Temp (id Integer, name Text)");
```

In the above example, it takes "CREATE TABLE" statement of SQL. This will create a table of "Integer" & "Text" fields.

Try and Catch block is required while performing this operation. An exception that indicates there was an error with SQL parsing or execution.

Step 3: Create an object of "ContentValues" and Initiate it.

```
ContentValues values=new ContentValues();
```

This class is used to store a set of values. We can also say, it will map ColumnName and relevant ColumnValue.

```
values.put("id", eid.getText().toString());  
values.put("name", ename.getText().toString());
```

String Key	Name of the field as in table. Ex. "id", "name"
String Value	Value to be inserted.

Step 4: Perform Insert Statement.

```
insert(String table, String nullColumnHack, ContentValues values)
```

String table	Name of table related to the database.
String nullColumnHack	If not set to null, the nullColumnHack parameter provides the name of nullable column name to explicitly insert a NULL into in the case where your <i>values</i> is empty.
ContentValues values	This map contains the initial column values for the row.

This method returns a long. The row ID of the newly inserted row, or -1 if an error occurred.

Example,

```
db.insert("temp", null, values);
```

Step 5: Create Cursor

This interface provides random read-write access to the result set returned by a database query.

```
Cursor c=db.rawQuery(String sql, String[] selectionArgs)
```

String sql	The SQL query
String []selectionArgs	You may include ?s in where clause in the query, which will be replaced by the values from selectionArgs. The values will be bound as Strings.

Example,

```
Cursor c=db.rawQuery("SELECT * FROM temp",null);
```

Methods

moveToFirst	Moves cursor pointer at a first position of a result set
moveToNext	Moves cursor pointer next to the current position.
isAfterLast	Returns false, if the cursor pointer is not at last position of a result set.

Example,
c.moveToFirst();

```

while(!c.isAfterLast())

{

    //statementâ€™!

c.moveToNext();

}

```

Step 6: Close Cursor and Close Database connectivity

It is very important to release our connections before closing our activity. It is advisable to release the Database connectivity in "onStop" method. And Cursor connectivity after use it.

DatabaseDemoActivity.java

```

package com.DataBaseDemo;
import android.app.Activity;
import android.content.ContentValues;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
public class DataBaseDemoActivity extends Activity {
    /** Called when the activity is first created. */
    SQLiteDatabase db;
    Button btnInsert;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        btnInsert=(Button)findViewById(R.id.button1);
        try{
            db=openOrCreateDatabase("StudentDB",SQLiteDatabase.CREATE_IF_NECESSARY,null
);
            db.execSQL("Create Table Temp(id integer,name text)");
        }catch(SQLException e)
        {
        }
        btnInsert.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                EditText eid=(EditText) findViewById(R.id.editText1);
                EditText ename=(EditText)findViewById(R.id.editText2);
                ContentValues values=new ContentValues();

```

```

        values.put("id", eid.getText().toString());
        values.put("name", ename.getText().toString());
        if((db.insert("temp", null, values))!=-1)
        {
            Toast.makeText(DataBaseDemoActivity.this, "Record Successfully Inserted", 2000).
show();
        }
        else
        {
            Toast.makeText(DataBaseDemoActivity.this, "Insert Error", 2000).show();
        }
        eid.setText("");
        ename.setText("");
        Cursor c=db.rawQuery("SELECT * FROM temp",null);
        c.moveToFirst();
        while(!c.isAfterLast())
        {
            Toast.makeText(DataBaseDemoActivity.this,c.getString(0)+ " "+c.getString(1),1000
).show();
            c.moveToNext();
        }
        c.close();
    }
});
}
@Override
protected void onStop() {
    // TODO Auto-generated method stub
    db.close();
    super.onStop();
}
}

```

Note:

If you want to see where your database stored? Follow below instruction.

1. Start Your Emulator (It is necessary to start Emulator to see File Explorer content)
2. Open "File Explorer"
3. Data -> Data -> find your "package" -> databases -> "database"

SQLite Transaction

Transactions are units or sequences of work accomplished in a logical order, whether in a manual fashion by a user or automatically by some sort of a database program.

A transaction is the propagation of one or more changes to the database. For example, if you are creating, updating, or deleting a record from the table, then you

are performing transaction on the table. It is important to control transactions to ensure data integrity and to handle database errors.

SQLite & ACID

SQLite is a transactional database that all changes and queries are atomic, consistent, isolated, and durable (ACID).

SQLite guarantees all the transactions are ACID compliant even if the transaction is interrupted by a program crash, operation system dump, or power failure to the computer.

- **Atomic:** a transaction should be atomic. It means that a change cannot be broken down into smaller ones. When you commit a transaction, either the entire transaction is applied or not.
- **Consistent:** a transaction must ensure to change the database from one valid state to another. When a transaction starts and executes a statement to modify data, the database becomes inconsistent. However, when the transaction is committed or rolled back, it is important that the transaction must keep the database consistent.
- **Isolation:** a pending transaction performed by a session must be isolated from other sessions. When a session starts a transaction and executes the INSERT or UPDATE statement to change the data, these changes are only visible to the current session, not others. On the other hand, the changes committed by other sessions after the transaction started should not be visible to the current session.
- **Durable:** if a transaction is successfully committed, the changes must be permanent in the database regardless of the condition such as power failure or program crash. On the contrary, if the program crashes before the transaction is committed, the change should not persist.

SQLite transaction statements

By default, SQLite operates in auto-commit mode. It means that for each command, SQLite starts, processes, and commits the transaction automatically.

To start a transaction explicitly, you use the following steps:

First, open a transaction by issuing the `BEGIN TRANSACTION` command.

```
BEGIN TRANSACTION;
```

After executing the statement `BEGIN TRANSACTION`, the transaction is open until it is explicitly committed or rolled back.

Second, issue SQL statements to select or update data in the database. Note that the change is only visible to the current session (or client).

Third, commit the changes to the database by using the COMMIT or COMMIT TRANSACTION statement.

```
COMMIT;
```

If you do not want to save the changes, you can roll back using the ROLLBACK or ROLLBACK TRANSACTION statement:

```
ROLLBACK;
```

SQLite transaction example

We will create two new tables: accounts and account_changes for the demonstration. The accounts table stores data about the account numbers and their balances.

The account_changes table stores the changes of the accounts.

First, create the accounts and account_changes tables by using the following CREATE TABLE statements:

```
CREATE TABLE accounts (  
    account_no INTEGER NOT NULL,  
    balance DECIMAL NOT NULL DEFAULT 0,  
    PRIMARY KEY(account_no),  
    CHECK(balance >= 0)  
);
```

```
CREATE TABLE account_changes (  
    change_no INT NOT NULL PRIMARY KEY,  
    account_no INTEGER NOT NULL,  
    flag TEXT NOT NULL,  
    amount DECIMAL NOT NULL,  
    changed_at TEXT NOT NULL  
);
```

Second, insert some sample data into the accounts table.

```
INSERT INTO accounts (account_no,balance)  
VALUES (100,20100);
```

```
INSERT INTO accounts (account_no,balance)  
VALUES (200,10100);
```

Third, query data from the accounts table:

```
SELECT * FROM accounts;
```

account_no	balance
100	20,100
200	10,100

Fourth, transfer 1000 from account 100 to 200, and log the changes to the table account_changes in a single transaction.

```
BEGIN TRANSACTION;
```

```
UPDATE accounts
```

```
  SET balance = balance - 1000
```

```
WHERE account_no = 100;
```

```
UPDATE accounts
```

```
  SET balance = balance + 1000
```

```
WHERE account_no = 200;
```

```
INSERT INTO account_changes(account_no,flag,amount,changed_at)
```

```
VALUES(100,'-',1000,datetime('now'));
```

```
INSERT INTO account_changes(account_no,flag,amount,changed_at)
```

```
VALUES(200,'+',1000,datetime('now'));
```

```
COMMIT;
```

Fifth, query data from the accounts table:

```
SELECT * FROM accounts;
```

account_no	balance
100	19,100
200	11,100

As you can see, balances have been updated successfully.

Sixth, query the contents of the account_changes table:

```
SELECT * FROM account_changes;
```

change_no	account_no	flag	amount	changed_at
1	100	-	1,000	2019-08-19 10:33:01
2	200	+	1,000	2019-08-19 10:33:04

Let's take another example of rolling back a transaction.

First, attempt to deduct 20,000 from account 100:


```
BEGIN TRANSACTION;
```

```
UPDATE accounts
```

```
  SET balance = balance - 20000
```

```
WHERE account_no = 100;
```

```
INSERT INTO account_changes(account_no,flag,amount,changed_at)
```

```
VALUES(100,'-',20000,datetime('now'));
```

SQLite issued an error due to not enough balance:

[SQLITE_CONSTRAINT] Abort due to constraint violation (CHECK constraint failed: accounts)

However, the log has been saved to the account_changes table:

```
SELECT * FROM account_changes;
```

change_no	account_no	flag	amount	changed_at
1	100	-	1,000	2019-08-19 10:48:38
2	200	+	1,000	2019-08-19 10:48:40
3	100	-	20,000	2019-08-19 10:54:07

Second, roll back the transaction by using the ROLLBACK statement:

```
ROLLBACK;
```

Finally, query data from the account_changes table, you will see that the change no #3 is not there anymore:

```
SELECT * FROM account_changes;
```

change_no	account_no	flag	amount	changed_at
1	100	-	1,000	2019-08-19 10:48:38
2	200	+	1,000	2019-08-19 10:48:40