**1.6**

**Android Application Components**

(Activities, Intents, Views, Layouts, Services)

In android, **application components** are the basic building blocks of an application and these components will act as an entry point to allow system or user to access our app.

Following are the basic core application components that can be used in Android application.

* [Activities](https://www.tutlane.com/tutorial/android/android-activity-lifecycle)
* [Intents](https://www.tutlane.com/tutorial/android/android-intents-implicit-explicit)
* [Content Providers](https://www.tutlane.com/tutorial/android/android-content-providers-with-examples)
* [Broadcast Receivers](https://www.tutlane.com/tutorial/android/android-broadcastreceivers-with-example)
* [Services](https://www.tutlane.com/tutorial/android/android-services-with-examples)

All these application components are defined in android app description file (**AndroidMainfest.xml**) like as shown below.

<?xml version="1.0" encoding="utf-8"?>  
<manifest …..>  
    <application android:allowBackup="true" android:icon="@mipmap/ic\_launcher" ……>  
        <activity android:name=".MainActivity" >  
            <intent-filter>  
                <action android:name="android.intent.action.MAIN" />  
                <category android:name="android.intent.category.LAUNCHER" />  
            </intent-filter>  
        </activity>

       …….

    </application>  
</manifest>

This is how we can define an android application components in **AndroidManiFest.xml** file.

**Android Activities**

In android, [Activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle) represents a single screen with a user interface (UI) and it will acts an entry point for user’s to interact with app.

For example, a contacts app which is having a multiple activities like showing a list of contacts, add new contact, and another activity to search for the contacts. All these activities in contact app are independent of each other but will work together to provide a better user experience.

**Android Intents**

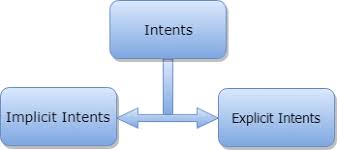
In android, [Intent](https://www.tutlane.com/tutorial/android/android-intents-implicit-explicit) is a messaging object which is used to request an action from another component.

In android, [intents](https://www.tutlane.com/tutorial/android/android-intents-implicit-explicit) are mainly used to perform following things.

* Starting an [Activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle)
* Starting a [Service](https://www.tutlane.com/tutorial/android/android-services-with-examples)
* Delivering a [Broadcast](https://www.tutlane.com/tutorial/android/android-broadcastreceivers-with-example)

There are two types of intents available in android, those are

1. [Implicit Intents](https://www.tutlane.com/tutorial/android/android-implicit-intents-with-examples)
2. [Explicit Intents](https://www.tutlane.com/tutorial/android/android-explicit-intents-with-examples)

**Android Services**

In android, [Service](https://www.tutlane.com/tutorial/android/android-services-with-examples) is a component which keep an app running in the background to perform long running operations based on our requirements. For [Service](https://www.tutlane.com/tutorial/android/android-services-with-examples), we don’t have any user interface and it will run the apps in background like play music in background when the user in different app.

We have two types of [services](https://www.tutlane.com/tutorial/android/android-services-with-examples) available in android, those are

* Local Services
* Remote Services

**Android Broadcast Receivers**

In android, [Broadcast Receiver](https://www.tutlane.com/tutorial/android/android-broadcastreceivers-with-example) is a component which will allow a system to deliver events to the app like sending a low battery 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](https://www.tutlane.com/tutorial/android/android-intents-implicit-explicit) to deliver broadcast events to other apps and Broadcast Receivers use status bar notifications to let user know that broadcast event occurs.

**Android Content Providers**

In android, [Content Providers](https://www.tutlane.com/tutorial/android/android-content-providers-with-examples) are used to exchange the data between the apps based on the requests. The Content Providers can share the app data that store in the file system, [SQLite database](https://www.tutlane.com/tutorial/sqlite), on the web or any other storage location that our app can access.

By using [Content Providers](https://www.tutlane.com/tutorial/android/android-content-providers-with-examples), other apps can query or modify the data of our app based on the permissions provided by content provider. For example, android provides a Content Provider (**ContactsContract.Data**) to manage **contacts** information, by using proper permissions any app can query the content provider to perform read and write operations on contacts information.

**Additional Components**

In android, we have an additional components which are used to build the relation between above components ([Activities](https://www.tutlane.com/tutorial/android/android-activity-lifecycle), [Intents](https://www.tutlane.com/tutorial/android/android-intents-implicit-explicit), [Content Providers](https://www.tutlane.com/tutorial/android/android-content-providers-with-examples), [Services](https://www.tutlane.com/tutorial/android/android-services-with-examples) and [Broadcast Receivers](https://www.tutlane.com/tutorial/android/android-broadcastreceivers-with-example)) to implement our application logic, those are

| **Component** | **Description** |
| --- | --- |
| Fragments | These are used to represent the portion of user interface in an activity |
| Layouts | These are used to define the user interface (UI) for an activity or app |
| Views | These are used to build user interface for an app using UI elements like , TextView, Buttons, lists, etc. |
| Resources | To build android app we required external elements like images, audio files, etc. other than coding |
| Manifest File | It’s a configuration file (**AndroidManifest.xml**) for the application and it will contain the information about [Activities](https://www.tutlane.com/tutorial/android/android-activity-lifecycle), [Intents](https://www.tutlane.com/tutorial/android/android-intents-implicit-explicit), [Content Providers](https://www.tutlane.com/tutorial/android/android-content-providers-with-examples), [Services](https://www.tutlane.com/tutorial/android/android-services-with-examples), [Broadcast Receivers](https://www.tutlane.com/tutorial/android/android-broadcastreceivers-with-example), permissions, etc. |

These are the main application components which are required to build any android application based on our requirements.

**Questions**

* 1. What are the basic core components used in Android application?

Following are the basic core application components that can be used in Android application.

* [Activities](https://www.tutlane.com/tutorial/android/android-activity-lifecycle)
* [Intents](https://www.tutlane.com/tutorial/android/android-intents-implicit-explicit)
* [Content Providers](https://www.tutlane.com/tutorial/android/android-content-providers-with-examples)
* [Broadcast Receivers](https://www.tutlane.com/tutorial/android/android-broadcastreceivers-with-example)
* [Services](https://www.tutlane.com/tutorial/android/android-services-with-examples)
  1. What is Android Intent for, and how many types of Intent are?

In android, [Intent](https://www.tutlane.com/tutorial/android/android-intents-implicit-explicit) is a messaging object which is used to request an action from another component.

3.What is Android Services for, and how many types of Services are?  
  
In android, [Service](https://www.tutlane.com/tutorial/android/android-services-with-examples) is a component which keep an app running in the background to perform long running operations based on our requirements

We have two types of [services](https://www.tutlane.com/tutorial/android/android-services-with-examples) available in android, those are

* Local Services
* Remote Services

4. What is Android Broadcast Receivers for?   
  
In android, [Broadcast Receiver](https://www.tutlane.com/tutorial/android/android-broadcastreceivers-with-example) is a component which will allow a system to deliver events to the app like sending a low battery 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.

5. What is Android Content Providers for? Explain its Additional Components.

In android, [Content Providers](https://www.tutlane.com/tutorial/android/android-content-providers-with-examples) are used to exchange the data between the apps based on the requests. The Content Providers can share the app data that store in the file system, [SQLite database](https://www.tutlane.com/tutorial/sqlite), on the web or any other storage location that our app can access.

**1.7**

**Android Activity Lifecycle**

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

Generally, the android apps contains multiple screens and each screen of our application will be an extension of Activity class. By using activities, we can place all our android application UI components in a single screen.

From the multiple activities in android app, one activity can be marked as a **main activity** and that is the first screen to appear when we launch the application. In android app each activity can start another activity to perform different actions based on our requirements.

For example, a contacts app which is having a multiple activities, in that the main activity screen will show a list of contacts and from the main activity screen we can launch other activities that provides a screens to perform a tasks like add a new contact and search for the contacts. All these activities in contact app are loosely bound to other activities but will work together to provide a better user experience.

Generally, in android there is a minimal dependencies between the activities in an app. To use activities in application we need to register those activities information in our app’s manifest file (**AndroidMainfest.xml**) and need to manage activity life cycle properly.

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

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

          …….

        </activity>

     …….

</application>  
</manifest>

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

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

public class MainActivity extends Activity {

}

This is how we can activities in android application based on our requirements.

**Android Activity Lifecycle**

Generally, the activities in our android application will go through a different stages in their life cycle. In android, **Activity** class have a 7 callback methods to describe how the activity will behave at different stages. These are as below:

 onCreate(), onStart(), onPause(), onRestart(), onResume(), onStop() and onDestroy()

By using activity callback methods we can define how our activity can behave when the user enter or leaves our application.

**Android Activity Lifecycle Callback Methods**

In android, an activity goes through a series of states during its lifetime. By using callback methods we can get the activity transitions between the states.

Android system initiates its program within an **Activity** starting with a call on onCreate() callback method. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity.

This section will give you a detailed information about callback methods to handle activity transitions between states during lifecycle.

**onCreate()**

This is the first callback method and it fires when the system creates an activity for the first time. During the activity creation, activity entered into a **Created** state.

If we have an application start-up logic that needs to perform only once during the life cycle of an activity, then we can write that logic in onCreate() method.

Following is the example of defining a onCreate() method in android activity.

@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity\_main);  
}

Once **onCreate()** method execution is finished, the activity will enter into Started state and system calls the onStart() method.

**onStart()**

The **onStart()** callback method will invoke when an activity entered into **Started** State by completing **onCreate()**method. The **onStart()** method will make an activity visible to the user and this method execution will finish very quickly.

Following is the example of defining a **onStart()** method in android activity.

@Override  
protected void onStart()  
{  
    super.onStart()  
}

After completion of **onStart()** method execution, the activity enters into **Resumed** state and system invoke the **onResume()** method.

**onResume()**

When an activity entered into **Resumed** state, the system invoke onResume() call back method. In this state activity start interacting with user that means user can see the functionality and designing part of an application on the single screen.

Mostly the core functionality of an app is implemented in onResume() method.

The app will stays in this **Resumed** state until an another activity happens to take focus away from the app like getting a phone call or screen turned off, etc.

In case if any interruption events happen in **Resumed** state, the activity will entered into **Paused** state and the system will invoke onPause() method.

After an activity returned from **Paused** state to **Resumed** state, the system again will call onResume() method due this we need to implement onResume() method to initialize the components that we release during onPause() method

Following is the example of defining a onResume() method in android activity.

@Override  
public void onResume() {  
    super.onResume();    
    if (mCamera == null) {  
        initializeCamera();   
    }  
}

If any interruption happen in **Resumed** state, the activity will entered into **Paused** state and the system will invoke onPause() method.

**onPause()**

Whenever the user leaves an activity or the current activity is being Paused then the system invoke onPause() method. The onPause() method is used to pause an operations like stop playing the music when the activity is in paused state or pass an activity while switching from one app to another app because every time only one app can be focused.

Following is the example of defining a onPause() method in android activity.

@Override  
public void onPause() {  
    super.onPause();   
  if (mCamera != null) {  
        mCamera.release();  
        mCamera = null;  
    }  
}

After completion of onPause() method execution, the next method is either onStop()

or onResume() depending on what happens after an activity entered into **Paused** state.

**onStop()**

The system will invoke onStop() callback method when an activity no longer visible to the user, the activity will enter into Stopped state. This happens due to current activity entered into **Resumed** state or newly launched activity covers complete screen or it’s been destroyed.

The onStop() method is useful to release all the app resources which are no longer needed to the user.

Following is the example of defining a onStop() method in android activity.

@Override  
protected void onStop()  
{  
    super.onStop();  
}

The next callback method which raised by system is either onRestart(), in case if the activity coming back to interact with the user or onDestroy(), in case if the activity finished running.

**onRestart()**

The system will invoke onRestart() method when an activity restarting itself after stopping it. The onRestart() method will restore the state of activity from the time that is being stopped.

The onRestart() callback method in android activity will always followed by onStart() method.

**onDestroy()**

The system will invoke onDestroy() method before an activity is destroyed and this is the final callback method which received by the android activity.

The system will invoke this onDestory() callback method either the activity is finishing or system destroying the activity to save space.

Following is the example of defining a onDestroy() method in android activity.

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

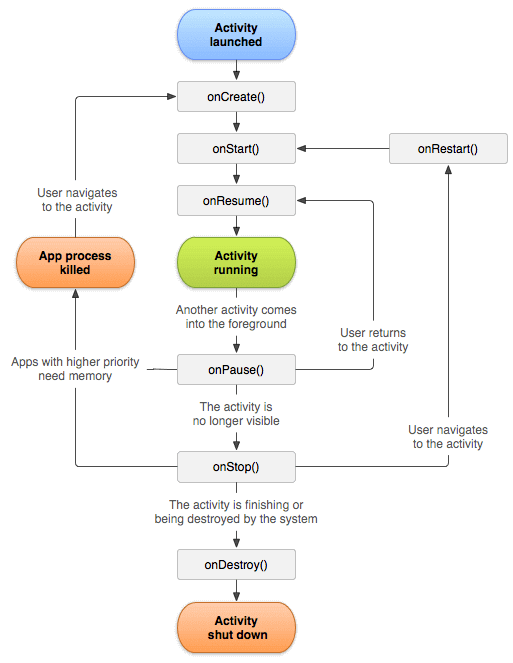
The onDestroy() method will release all the resources which are not released by previous callback onStop() method.

**Android Activity Lifecycle Diagram**

Generally, in android activity class uses different callback methods like

onCreate(), onStart(), onPause(), onRestart(), onResume(), onStop() and onDestroy() to go through a different stages of activity life cycle.

Following is the pictorial representation of Android Activity Life cycle which shows how Activity will behave in different stages using callback methods.



Whenever the user trying to leave an activity like switching from one app to another app, the system will use callback methods to dismantle the activity completely or partially to resume the activity from where the user left off.

Based on our requirements we can implement the activity in android app using callback method and it’s not necessary to use all callback methods in each android application.

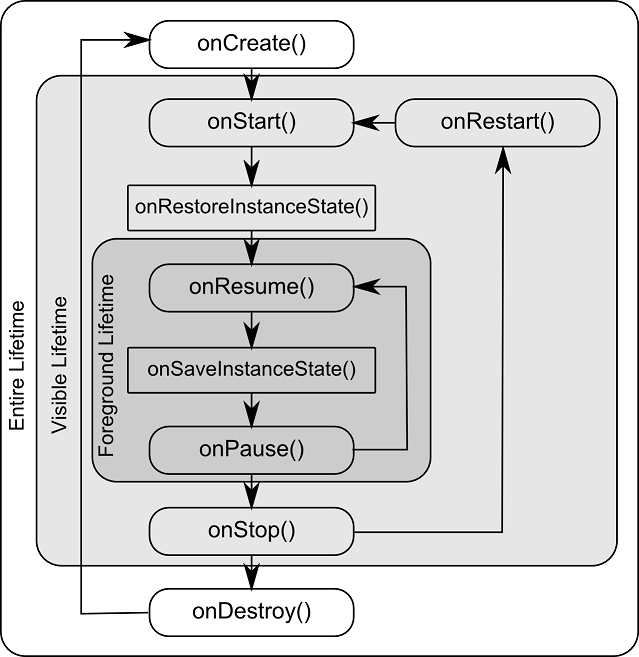
**Lifetimes**

The outline of the entire, visible and foreground lifetimes through which an activity or fragment transition during execution are as below:

* **Entire Lifetime﻿** –The term “entire lifetime” is used to describe everything that takes place between the initial call to the onCreate() method and the call to onDestroy() prior to the object terminating.
* **Visible Lifetime﻿** – Covers the periods of execution between the call to onStart() and onStop(). During this period the activity or fragment is visible to the user though may not be the object with which the user is currently interacting.
* **Foreground Lifetime﻿** – Refers to the periods of execution between calls to the onResume() and onPause() methods.

It is important to note that an activity or fragment may pass through the foreground and visible lifetimes multiple times during the course of the entire lifetime.

The concepts of lifetimes and lifecycle methods are illustrated in Figure:

[](https://www.techotopia.com/index.php/File:Android_activity_lifecycle_methods.png)

**Android Activity Lifecycle Example**

Now we will see, how android activity lifecycle will work with example. Following is the example of invoking an activity callback methods to see the life cycle process of an activity in android application.

Here we are going to use previously created [Android Hello World App example](https://www.tutlane.com/tutorial/android/android-hello-world-app-example) and making some modifications to **MainActivity.java** file like as shown below to capture Android Activity Life Cycle process.

**MainActivity.java File Code**

Following are the code modifications which made to include all life cycle callback methods in **MainActivity.java** file which is in **\java\com.tutlane.helloworld** directory.

package com.sarker.helloworld;  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
import android.util.Log;  
  
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity\_main);  
        Log.d("Activity Lifecycle","onCreate invoked");  
    }  
    @Override  
    protected void onStart() {  
        super.onStart();  
        Log.d("Activity Lifecycle","onStart invoked");  
    }  
    @Override  
    protected void onResume() {  
        super.onResume();  
        Log.d("Activity Lifecycle","onResume invoked");  
    }  
    @Override  
    protected void onPause() {  
        super.onPause();  
        Log.d("Activity Lifecycle","onPause invoked");  
    }  
    @Override  
    protected void onStop() {  
        super.onStop();  
        Log.d("Activity Lifecycle","onStop invoked");  
    }  
    @Override  
    protected void onRestart() {  
        super.onRestart();  
        Log.d("Activity Lifecycle","onRestart invoked");  
    }  
    @Override  
    protected void onDestroy() {  
        super.onDestroy();  
        Log.d("Activity Lifecycle","onDestroy invoked");  
    }  
}

If we observe above code we defined a statement like “setContentView(R.layout.activity\_main);” which will help to load all UI components defined in **activity\_main.xml** file and used **Log.d()** method to generate log messages.

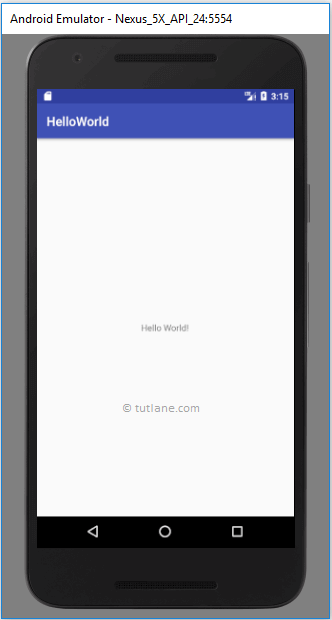
In our application we can have more than one activity file and we need to declare all the activities in **AndroidManifest.xml** file. In manifest xml file by using **MAIN** action and **LAUNCHER** category attributes in intent filters (**<intent-filter>**) we can mention the main activity that opens when the user initially launch our app with the launcher icon. In case if we didn’t mention **MAIN** action, the system will decide which activity need to start and if we didn’t add **LAUNCHER** category for main activity, our app icon will not appear in the home screen’s list of apps.

The code of **AndroidManifest.xml** file will be like as shown below.

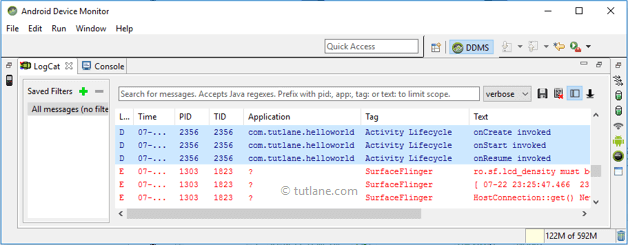
<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
    package="com.tutlane.helloworld" >  
  
    <application  
        android:allowBackup="true"  
        android:icon="@mipmap/ic\_launcher"  
        android:label="@string/app\_name"  
        android:roundIcon="@mipmap/ic\_launcher\_round"  
        android:supportsRtl="true"  
        android:theme="@style/AppTheme" >  
        <activity android:name=".MainActivity" >  
            <intent-filter>  
                <action android:name="android.intent.action.MAIN" />  
                <category android:name="android.intent.category.LAUNCHER" />  
            </intent-filter>  
        </activity>  
    </application>  
</manifest>

**Output of Android Activity Lifecycle Example**

Now run the application using Android Virtual Device (**AVD**) in android studio by clicking on Play icon in toolbar or press Shift + F10. Our application result will be like as shown below.

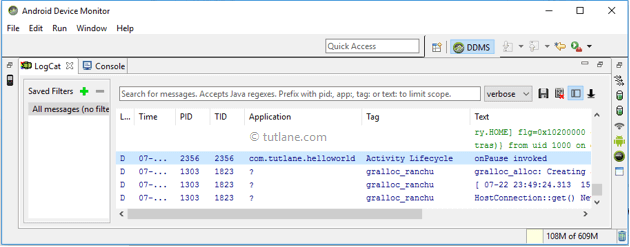


Now open **Android Device Monitor** (Tools à Android à Android Device Monitor) to see our log messages in LogCat window in android studio like as shown below.

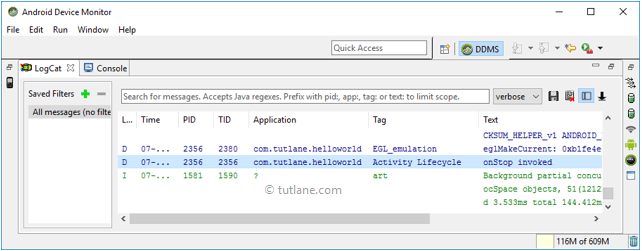


If we observe log messages in **LogCat** window **onCreate**, **onStart** and **onResume** methods are invoked by system.

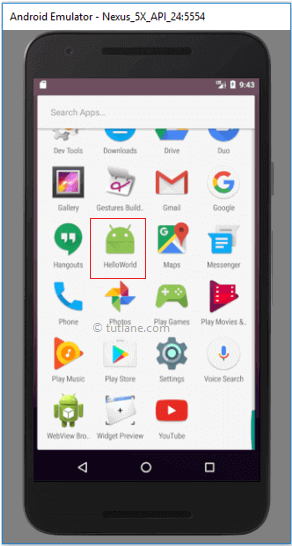
Now click on **Home** button in Android Emulator, immediately activity entered into **Paused** state and system will invoke onPause() method like as shown below.



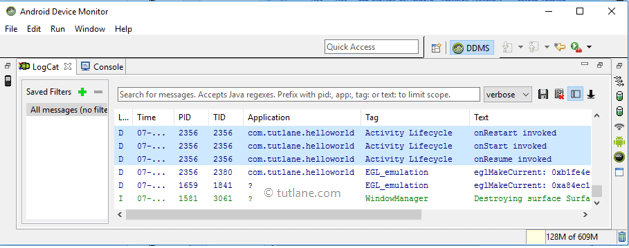
After a while, the activity will enter into **Stopped** state and system will invoke onStop() method like as shown below.



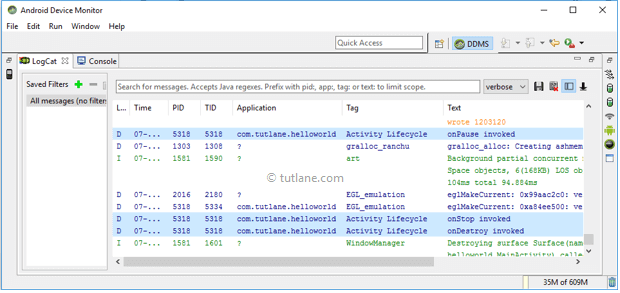
Now again launch our app from the Home screen list of apps like as shown below.



If we observe log messages in **LogCat** window again **onRestart**, **onStart** and **onResume** methods are invoked by system like as shown below.



Now click on Back button in the android emulator, the system will invoke **onPause** method and after a while **onStop**, **onDestroy** methods will be invoked like as shown below.



Here we need to remember that **onCreate** and **onDestroy** methods will invoke only once throughout the activity life cycle.

This is how android activity life cycle process will invoke different methods while transition from one stage to another stage.

**Questions:**

* 1. How many callback methods are in android? Explain their behavior at different stages in Android.
* **Android Activity Lifecycle Callback Methods**
* In android, an activity goes through a series of states during its lifetime. By using callback methods we can get the activity transitions between the states.
* Android system initiates its program within an **Activity** starting with a call on onCreate() callback method. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity.
* This section will give you a detailed information about callback methods to handle activity transitions between states during lifecycle.
* **onCreate()**
* This is the first callback method and it fires when the system creates an activity for the first time. During the activity creation, activity entered into a **Created** state.
* If we have an application start-up logic that needs to perform only once during the life cycle of an activity, then we can write that logic in onCreate() method.
* Following is the example of defining a onCreate() method in android activity.
* @Override  
  protected void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      setContentView(R.layout.activity\_main);  
  }
* Once **onCreate()** method execution is finished, the activity will enter into Started state and system calls the onStart() method.
* **onStart()**
* The **onStart()** callback method will invoke when an activity entered into **Started** State by completing **onCreate()**method. The **onStart()** method will make an activity visible to the user and this method execution will finish very quickly.
* Following is the example of defining a **onStart()** method in android activity.
* @Override  
  protected void onStart()  
  {  
      super.onStart()  
  }
* After completion of **onStart()** method execution, the activity enters into **Resumed** state and system invoke the **onResume()** method.
* **onResume()**
* When an activity entered into **Resumed** state, the system invoke onResume() call back method. In this state activity start interacting with user that means user can see the functionality and designing part of an application on the single screen.
* Mostly the core functionality of an app is implemented in onResume() method.
* The app will stays in this **Resumed** state until an another activity happens to take focus away from the app like getting a phone call or screen turned off, etc.
* In case if any interruption events happen in **Resumed** state, the activity will entered into **Paused** state and the system will invoke onPause() method.
* After an activity returned from **Paused** state to **Resumed** state, the system again will call onResume() method due this we need to implement onResume() method to initialize the components that we release during onPause() method
* Following is the example of defining a onResume() method in android activity.
* @Override  
  public void onResume() {  
      super.onResume();    
      if (mCamera == null) {  
          initializeCamera();   
      }  
  }
* If any interruption happen in **Resumed** state, the activity will entered into **Paused** state and the system will invoke onPause() method.
* **onPause()**
* Whenever the user leaves an activity or the current activity is being Paused then the system invoke onPause() method. The onPause() method is used to pause an operations like stop playing the music when the activity is in paused state or pass an activity while switching from one app to another app because every time only one app can be focused.
* Following is the example of defining a onPause() method in android activity.
* @Override  
  public void onPause() {  
      super.onPause();   
    if (mCamera != null) {  
          mCamera.release();  
          mCamera = null;  
      }  
  }
* After completion of onPause() method execution, the next method is either onStop()
* or onResume() depending on what happens after an activity entered into **Paused** state.
* **onStop()**
* The system will invoke onStop() callback method when an activity no longer visible to the user, the activity will enter into Stopped state. This happens due to current activity entered into **Resumed** state or newly launched activity covers complete screen or it’s been destroyed.
* The onStop() method is useful to release all the app resources which are no longer needed to the user.
* Following is the example of defining a onStop() method in android activity.
* @Override  
  protected void onStop()  
  {  
      super.onStop();  
  }
* The next callback method which raised by system is either onRestart(), in case if the activity coming back to interact with the user or onDestroy(), in case if the activity finished running.
* **onRestart()**
* The system will invoke onRestart() method when an activity restarting itself after stopping it. The onRestart() method will restore the state of activity from the time that is being stopped.
* The onRestart() callback method in android activity will always followed by onStart() method.
* **onDestroy()**
* The system will invoke onDestroy() method before an activity is destroyed and this is the final callback method which received by the android activity.
* The system will invoke this onDestory() callback method either the activity is finishing or system destroying the activity to save space.
* Following is the example of defining a onDestroy() method in android activity.
* @Override  
  public void onDestroy()  
  {  
      super.onDestroy();  
  }
* The onDestroy() method will release all the resources which are not released by previous callback onStop() method.

**2. What is the first callback method in Android? When it is used.**

This is the first callback method and it fires when the system creates an activity for the first time. During the activity creation, activity entered into a **Created** state.

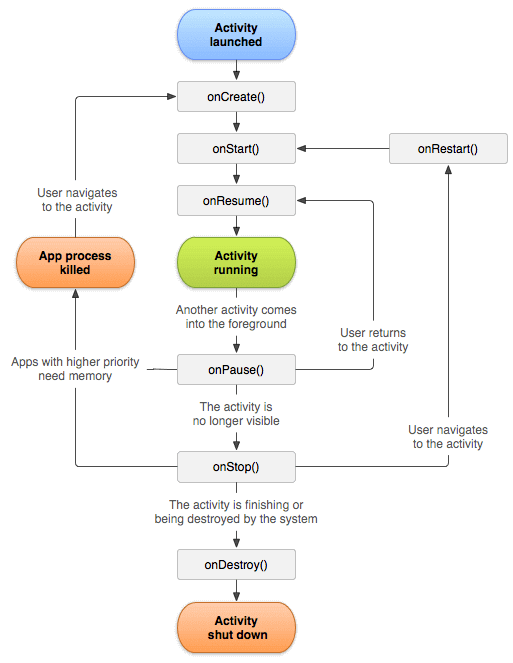
If we have an application start-up logic that needs to perform only once during the life cycle of an activity, then we can write that logic in onCreate() method.

Following is the example of defining a onCreate() method in android activity.

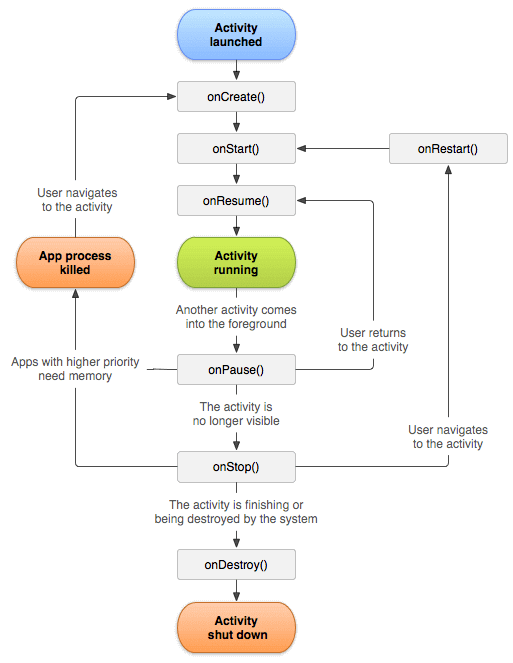
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity\_main);  
}

Once **onCreate()** method execution is finished, the activity will enter into Started state and system calls the onStart() method.

**3. Draw and explain Activity Lifecycle of Android (use necessary code and diagram).**



**4. Sketch the entire, visible and foreground lifetimes of android activity or fragment transition during execution.**



**3.1**

**Android View and ViewGroup with Examples**

In android, **Layout** is used to define the user interface for an app or [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle) and it will hold the UI elements that will appear to the user.

The user interface in android app is made with a collection of View and ViewGroup objects. Generally, the android apps contain one or more activities and each activity is a one screen of app. The activities contain a multiple UI components and those UI components are the instances of View and ViewGroup subclasses.

The user interface in android app is made with a collection of View and ViewGroup objects. Generally, the android apps will contain one or more activities and each activity is a one screen of app. The activities contain a multiple UI components and those UI components are the instances of View and ViewGroup subclasses.

**Android View**

The View is a base class for all UI components in android. For example, the **EditText** class is used to accept the input from users in android apps, which is a sub class of View.

Following are the some of common View subclasses which will be used in android applications.

* [TextView](https://www.tutlane.com/tutorial/android/android-textview-with-examples)
* [EditText](https://www.tutlane.com/tutorial/android/android-edittext-with-examples)
* [Button](https://www.tutlane.com/tutorial/android/android-button-with-examples)
* [CheckBox](https://www.tutlane.com/tutorial/android/android-checkbox-with-examples)
* [RadioButton](https://www.tutlane.com/tutorial/android/android-radiobutton-with-examples)
* [ImageButton](https://www.tutlane.com/tutorial/android/android-imagebutton-with-examples)
* [Progress Bar](https://www.tutlane.com/tutorial/android/android-progressbar-with-examples)
* [Spinner](https://www.tutlane.com/tutorial/android/android-spinner-dropdown-list-with-examples)

Like these we have many View subclasses available in android.

**Android ViewGroup**

The ViewGroup is a subclass of View and it acts as a base class for **layouts** and **layouts parameters**. The ViewGroup provides an invisible containers to hold other **Views** or **ViewGroups** and to define the layout properties.

For example, [Linear Layout](https://www.tutlane.com/tutorial/android/android-linearlayout-with-examples) is the ViewGroup that contains a UI controls like button, textview, etc. and other layouts also.

Following are the commonly used ViewGroup subclasses in android applications.

* [Linear Layout](https://www.tutlane.com/tutorial/android/android-linearlayout-with-examples)
* [Relative Layout](https://www.tutlane.com/tutorial/android/android-relativelayout-with-examples)
* [Table Layout](https://www.tutlane.com/tutorial/android/android-tablelayout-with-examples)
* [Frame Layout](https://www.tutlane.com/tutorial/android/android-framelayout-with-examples)
* [Web View](https://www.tutlane.com/tutorial/android/android-webview-with-examples)
* [List View](https://www.tutlane.com/tutorial/android/android-listview-with-examples)
* [Grid View](https://www.tutlane.com/tutorial/android/android-gridview-with-examples)

Both View and ViewGroup subclasses together will play a key role to create a layouts in android applications.

**Questions**

**What are the Android View and ViewGroup?**The View is a base class for all UI components in android. For example, the **EditText** class is used to accept the input from users in android apps, which is a sub class of View.

The ViewGroup is a subclass of View and it acts as a base class for **layouts** and **layouts parameters**.

1. **How many types of Android Layout?**

* [Linear Layout](https://www.tutlane.com/tutorial/android/android-linearlayout-with-examples)
* [Relative Layout](https://www.tutlane.com/tutorial/android/android-relativelayout-with-examples)
* [Table Layout](https://www.tutlane.com/tutorial/android/android-tablelayout-with-examples)
* [Frame Layout](https://www.tutlane.com/tutorial/android/android-framelayout-with-examples)
* [Web View](https://www.tutlane.com/tutorial/android/android-webview-with-examples)
* [List View](https://www.tutlane.com/tutorial/android/android-listview-with-examples)
* [Grid View](https://www.tutlane.com/tutorial/android/android-gridview-with-examples)

**2. Android UI Layouts**

**(Linear, Relative, Frame, Table, ListView, GridView, WebView)**

In android, **Layout** is used to define the user interface for an app or [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle) and it will hold the UI elements that will appear to the user.

The user interface in android app is made with a collection of View and ViewGroup objects. Generally, the android apps will contain one or more activities and each activity is a one screen of app. The activities will contain a multiple UI components and those UI components are the instances of View and ViewGroup subclasses.

The View is a base class for all UI components in android and it is used to create an interactive UI components such as [TextView](https://www.tutlane.com/tutorial/android/android-textview-with-examples), [EditText](https://www.tutlane.com/tutorial/android/android-edittext-with-examples), [Checkbox](https://www.tutlane.com/tutorial/android/android-checkbox-with-examples), [Radio Button](https://www.tutlane.com/tutorial/android/android-radiobutton-with-examples), etc. and it responsible for event handling and drawing.

The ViewGroup is a subclass of View and it acts as a base class for **layouts** and **layouts parameters**. The ViewGroupwill provide an invisible containers to hold other Views or ViewGroups and to define the layout properties.

In android, we can define a layouts in two ways, those are

* Declare UI elements in XML
* Instantiate layout elements at runtime

The android framework allows us to use either or both of these methods to define our application’s UI.

**Declare UI Elements in XML**

In android, we can create a layouts same like web pages in HTML by using default [Views and ViewGroups](https://www.tutlane.com/tutorial/android/android-view-and-viewgroup-with-examples) in XML file. The layout file must contain only one root element, which must be a View or ViewGroup object. Once we define root element, then we can add additional layout objects or widgets as a child elements to build View hierarchy that defines our layout.

Following is the example of defining a layout in XML file (**activity\_main.xml**) using [LinearLayout](https://www.tutlane.com/tutorial/android/android-linearlayout-with-examples) to hold a [TextView](https://www.tutlane.com/tutorial/android/android-textview-with-examples), [EditText](https://www.tutlane.com/tutorial/android/android-edittext-with-examples) and [Button](https://www.tutlane.com/tutorial/android/android-button-with-examples).

<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical"

    android:layout\_width="match\_parent"  
    android:layout\_height="match\_parent">  
    <TextView  
        android:id="@+id/fstTxt"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:text="Enter Name"  
         />  
    <EditText  
        android:id="@+id/name"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:ems="10">  
    </EditText>  
    <Button  
        android:id="@+id/getName"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:text="Get Name" />  
</LinearLayout>

We need to create a layout files in **/res/layout** project directory, then only the layout files will compile properly.

**Load XML Layout File from an Activity**

Once we are done with creation of layout, we need to load the XML layout resource from our [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle) **onCreate()** callback method like as shown below

protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity\_main);    
}

If we observe above code we are calling our layout using **setContentView** method in the form of **R.layout.layout\_file\_name**. Here our xml file name is **activity\_main.xml** so we used file name **activity\_main**.

Generally, during the launch of our [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle), **onCreate()** callback method will be called by android framework to get the required layout for an [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle).

**Instantiate Layout Elements at Runtime**

If we want to instantiate layout elements at runtime, we need to create own custom View and ViewGroup objects programmatically with required layouts.

Following is the example of creating a layout using [LinearLayout](https://www.tutlane.com/tutorial/android/android-linearlayout-with-examples) to hold a [TextView](https://www.tutlane.com/tutorial/android/android-textview-with-examples), [EditText](https://www.tutlane.com/tutorial/android/android-edittext-with-examples) and [Button](https://www.tutlane.com/tutorial/android/android-button-with-examples) in an [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle) using custom View and ViewGroup objects programmatically.

public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        TextView textView1 = new TextView(this);  
        textView1.setText("Name:");  
        EditText editText1 = new EditText(this);  
        editText1.setText("Enter Name");  
        Button button1 = new Button(this);  
        button1.setText("Add Name");  
        LinearLayout linearLayout = new LinearLayout(this);  
        linearLayout.addView(textView1);  
        linearLayout.addView(editText1);  
        linearLayout.addView(button1);  
        setContentView(linearLayout);  
    }  
}

By creating a custom View and ViewGroup programmatically, we can define a layouts based on our requirements in android applications.

**Width and Height**

When we define a layout using XML file we need to set width and height for every View and ViewGroup element using **layout\_width** and **layout\_height** attributes.

Following is the example of setting width and height for View and ViewGroup elements in XML layout file.

<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical"

    android:layout\_width="match\_parent"  
    android:layout\_height="match\_parent">  
    <TextView  
        android:id="@+id/fstTxt"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:text="Enter Name" />  
</LinearLayout>

If we observe above example, we used different values to set **layout\_width** and **layout\_height**, those are

* match\_parent
* wrap\_content

If we set value match\_parent, then the View or ViewGroup will try to match with parent width or height.

If we set value wrap\_content, then the View or ViewGroup will try to adjust its width or height based on the content.

**Android Layout Attributes**

In android, like **layout\_width** and **layout\_height** we have a different type of attributes available for View and ViewGroupobjects to define the appearance of layouts based on our requirements.

| **Attribute** | **Description** |
| --- | --- |
| android:id | It is used to uniquely identify the view and ViewGroups |
| android:layout\_width | It is used to define the width for View and ViewGroup elements in layout |
| android:layout\_height | It is used to define the height for View and ViewGroup elements in layout |
| android:layout\_marginLeft | It is used to define the extra space in left side for View and ViewGroup elements in layout |
| android:layout\_marginRight | It is used to define the extra space in right side for View and ViewGroup elements in layout |
| android:layout\_marginTop | It is used to define the extra space on top for View and ViewGroup elements in layout |
| android:layout\_marginBottom | It is used to define the extra space in bottom side for View and ViewGroup elements in layout |
| android:paddingLeft | It is used to define the left side padding for View and ViewGroup elements in layout files |
| android:paddingRight | It is used to define the right side padding for View and ViewGroup elements in layout files |
| android:paddingTop | It is used to define padding for View and ViewGroup elements in layout files on top side |
| android:paddingBottom | It is used to define the bottom side padding for View and ViewGroup elements in layout files |
| android:layout\_gravity | It is used to define how child Views are positioned |
| **ems** is a typography term,[is the art and technique of arranging type to make written language legible, readable, and appealing when displayed. Point is the smallest unit of measure in typography (controls text size, etc).](https://en.wikipedia.org/wiki/Typography)The name em was originally a reference to the width of the **capital M**. It sets the width of a TextView/EditText to fit a text of n 'M' letters regardless of the actual text extension and text size. | |

Following are the some of common layout attributes used in android application.

**Android Layout Types**

We have a different type of layouts available in android to implement user interface for our android applications with different designs based on our requirements.

Following are the commonly used layouts in android applications to implement required designs.

* [Linear Layout](https://www.tutlane.com/tutorial/android/android-linearlayout-with-examples)
* [Relative Layout](https://www.tutlane.com/tutorial/android/android-relativelayout-with-examples)
* [Frame Layout](https://www.tutlane.com/tutorial/android/android-framelayout-with-examples)
* [Table Layout](https://www.tutlane.com/tutorial/android/android-tablelayout-with-examples)
* [Web View](https://www.tutlane.com/tutorial/android/android-webview-with-examples)
* [List View](https://www.tutlane.com/tutorial/android/android-listview-with-examples)
* [Grid View](https://www.tutlane.com/tutorial/android/android-gridview-with-examples)

**Android Linear Layout**

In android, [LinearLayout](https://www.tutlane.com/tutorial/android/android-linearlayout-with-examples) is a ViewGroup subclass which is used to render all child View instances one by one either in horizontal direction or vertical direction based on the orientation property.

**Android Relative Layout**

In android, [RelativeLayout](https://www.tutlane.com/tutorial/android/android-relativelayout-with-examples) is a ViewGroup which is used to specify the position of child View instances relative to each other (Child A to the left of Child B) or relative to the parent (Aligned to the top of parent).

**Android Frame Layout**

In android, [FrameLayout](https://www.tutlane.com/tutorial/android/android-framelayout-with-examples) is a ViewGroup subclass which is used to specify the position of View instances it contains on the top of each other to display only single View inside the FrameLayout.

**Android Table Layout**

In android, [TableLayout](https://www.tutlane.com/tutorial/android/android-tablelayout-with-examples) is a ViewGroup subclass which is used to display the child View elements in rows and columns.

**Android Web View**

In android, [WebView](https://www.tutlane.com/tutorial/android/android-webview-with-examples) is a browser which is used to display the web pages as a part of our activity layout.

**Android List View**

In android, [ListView](https://www.tutlane.com/tutorial/android/android-listview-with-examples) is a ViewGroup which is used to display scrollable single column list of items.



**Android Grid View**

In android, [GridView](https://www.tutlane.com/tutorial/android/android-gridview-with-examples) is a ViewGroup which is used to display items in a scrollable grid of columns and rows.

**Questions**

1. **What are the different types of android layouts – explain their features and activities**.

**Android Linear Layout**

In android, [LinearLayout](https://www.tutlane.com/tutorial/android/android-linearlayout-with-examples) is a ViewGroup subclass which is used to render all child View instances one by one either in horizontal direction or vertical direction based on the orientation property.

**Android Relative Layout**

In android, [RelativeLayout](https://www.tutlane.com/tutorial/android/android-relativelayout-with-examples) is a ViewGroup which is used to specify the position of child View instances relative to each other (Child A to the left of Child B) or relative to the parent (Aligned to the top of parent).

**Android Frame Layout**

In android, [FrameLayout](https://www.tutlane.com/tutorial/android/android-framelayout-with-examples) is a ViewGroup subclass which is used to specify the position of View instances it contains on the top of each other to display only single View inside the FrameLayout.

**Android Table Layout**

In android, [TableLayout](https://www.tutlane.com/tutorial/android/android-tablelayout-with-examples) is a ViewGroup subclass which is used to display the child View elements in rows and columns.

**Android Web View**

In android, [WebView](https://www.tutlane.com/tutorial/android/android-webview-with-examples) is a browser which is used to display the web pages as a part of our activity layout.

**Android List View**

In android, [ListView](https://www.tutlane.com/tutorial/android/android-listview-with-examples) is a ViewGroup which is used to display scrollable single column list of items.

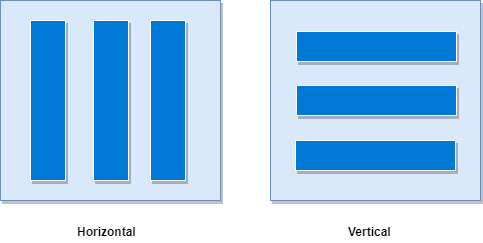


**3. Android Linear Layout with Examples**

In android, **LinearLayout** is a **ViewGroup** subclass which is used to render all child **View** instances one by one either in **Horizontal** direction or **Vertical** direction based on the **orientation** property.

In android, we can specify the linear layout orientation using **android:orientation** attribute.

Following is the pictorial representation of linear layout in android applications.



In **LinearLayout**, the child **View** instances arranged one by one, so the horizontal list will have only one row of multiple columns and vertical list will have one column of multiple rows.

**Android LinearLayout Declaration**

Following is the way we need to define the LinearLayout in android applications.

<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout\_width="match\_parent"  
    android:layout\_height="match\_parent"  
    android:orientation="vertical" >  
    <!-- Add Child Views Here -->

</LinearLayout>

If we observe above code snippet, here we defined orientation as vertical, so this aligns all its child layout / views vertically.

**Android LinearLayout Example**

Following is the example of creating a **LinearLayout** with different controls in android application.

Create a new android application using android studio and give names as **LinearLayout**. In case if you are not aware of creating an app in android studio check this article [Android Hello World App](https://www.tutlane.com/tutorial/android/android-hello-world-app-example).

Now open an **activity\_main.xml** file from **\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:layout\_width="match\_parent"  
    android:layout\_height="match\_parent"  
    android:paddingLeft="20dp"  
    android:paddingRight="20dp"  
    android:orientation="vertical" >  
    <EditText  
        android:id="@+id/txtTo"  
        android:layout\_width="match\_parent"  
        android:layout\_height="wrap\_content"  
        android:hint="To"/>  
    <EditText  
        android:id="@+id/txtSub"  
        android:layout\_width="match\_parent"  
        android:layout\_height="wrap\_content"  
        android:hint="Subject"/>  
    <EditText  
        android:id="@+id/txtMsg"  
        android:layout\_width="match\_parent"  
        android:layout\_height="0dp"  
        android:layout\_weight="1"  
        android:gravity="top"  
        android:hint="Message"/>  
    <Button  
        android:layout\_width="100dp"  
        android:layout\_height="wrap\_content"  
        android:layout\_gravity="right"  
        android:text="Send"/>  
</LinearLayout>

Once we are done with creation of layout, we need to load the XML layout resource from our [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle) **onCreate()** callback method, for that open main activity file **MainActivity.java** from **\java\com.sarker.linearlayout** path and write the code like as shown below.

**MainActivity.java**

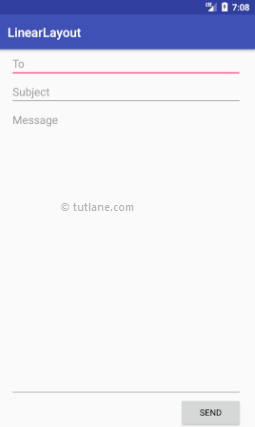
package com.sarker.com.linearlayout;  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
  
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity\_main);  
    }  
}

If we observe above code we are calling our layout using **setContentView** method in the form of **R.layout.layout\_file\_name**. Here our xml file name is **activity\_main.xml** so we used file name **activity\_main**.

Generally, during the launch of our [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle), **onCreate()** callback method will be called by android framework to get the required layout for an activity.

Output

When we run above example using android virtual device (AVD) we will get a result like as shown below.



**Layout Weight Attribute**

If we observe above example we used layout **weight** attribute (android:layout\_weight) in child view. Actually, this attribute is used by child views to specify how much space the **View** should occupy on the screen. If we assign larger weight value to child view, then it will expand to fill any remaining space in the parent view.

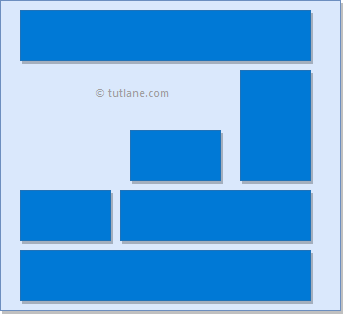
If we observe above example, we used three text fields and we assigned weight value to only one text field. The two text fields without weight will occupy only the area required for its content and the other text field with weight value will expand to fill the remaining space after all three fields measured.

This is how we can use **LinearLayout** in android applications to render all **View** instances one by one either in **Horizontal** direction or **Vertical** direction based on the **orientation** property.

**4. Android RelativeLayout with Examples**

In android, **RelativeLayout** is a **ViewGroup** which is used to specify the position of child **View** instances relative to each other (Child **A** to the left of Child **B**) or relative to the parent (Aligned to the top of parent).

Following is the pictorial representation of relative layout in android applications.



In android, **RelativeLayout** is very useful to design user interface because by using relative layout we can eliminate the nested view groups and keep our layout hierarchy flat, which improves performance of application.

## Android Positioning Views in Relative Layout

As we discussed, in **RelativeLayout** we need to specify the position of child views relative to each other or relative to the parent. In case if we didn’t specify the position of child views, by default all child views are positioned to top-left of the layout.

Following are the some of most useful layout properties available to views in RelativeLayout.

| **Attribute** | **Description** |
| --- | --- |
| layout\_alignParentTop | If it specified “true”, the top edge of view will match the top edge of parent. |
| layout\_alignParentBottom | If it specified “true”, the bottom edge of view will match the bottom edge of parent. |
| layout\_alignParentLeft | If it specified “true”, the left edge of view will match the left edge of parent. |
| layout\_alignParentRight | If it specified “true”, the right edge of view will match the right edge of parent. |
| layout\_centerInParent | If it specified “true”, the view will be aligned to centre of parent. |
| layout\_centerHorizontal | If it specified “true”, the view will be horizontally centre aligned within its parent. |
| layout\_centerVertical | If it specified “true”, the view will be vertically centre aligned within its parent. |
| layout\_above | It accepts another sibling view id and places the view above the specified view id. |
| layout\_below | It accepts another sibling view id and places the view below the specified view id. |
| layout\_toLeftOf | It accepts another sibling view id and places the view left of the specified view id. |
| layout\_toRightOf | It accepts another sibling view id and places the view right of the specified view id. |
| layout\_toStartOf | It accepts another sibling view id and places the view to start of the specified view id. |
| layout\_toEndOf | It accepts another sibling view id and places the view to end of the specified view id. |

**Android RelativeLayout Example**

Following is the example of creating a **RelativeLayout** with different controls in android application.

Create a new android application using android studio and give names as **RelativeLayout**. In case if you are not aware of creating an app in android studio check this article [Android Hello World App](https://www.tutlane.com/tutorial/android/android-hello-world-app-example).

Now open an **activity\_main.xml** file from **\res\layout** path and write the code like as shown below

**activity\_main.xml**

<?xml version="1.0" encoding="utf-8"?>  
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout\_width="match\_parent"  
    android:layout\_height="match\_parent"  
    android:paddingLeft="10dp"  
    android:paddingRight="10dp">  
    <Button  
        android:id="@+id/btn1"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:layout\_alignParentLeft="true"  
        android:text="Button1" />  
    <Button  
        android:id="@+id/btn2"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:layout\_alignParentRight="true"  
        android:layout\_centerVertical="true"  
        android:text="Button2" />  
    <Button  
        android:id="@+id/btn3"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:layout\_alignParentLeft="true"  
        android:layout\_centerVertical="true"  
        android:text="Button3" />  
  
    <Button  
        android:id="@+id/btn4"  
        android:layout\_width="match\_parent"  
        android:layout\_height="wrap\_content"  
        android:layout\_alignParentBottom="true"  
        android:text="Button4" />  
    <Button  
        android:id="@+id/btn5"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:layout\_alignBottom="@+id/btn2"  
        android:layout\_centerHorizontal="true"  
        android:text="Button5" />  
    <Button  
        android:id="@+id/btn6"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:layout\_above="@+id/btn4"  
        android:layout\_centerHorizontal="true"  
        android:text="Button6" />  
    <Button  
        android:id="@+id/btn7"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:layout\_toEndOf="@+id/btn1"  
        android:layout\_toRightOf="@+id/btn1"  
        android:layout\_alignParentRight="true"  
        android:text="Button7" />  
</RelativeLayout>

Once we are done with creation of layout, we need to load the XML layout resource from our [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle) **onCreate()** callback method, for that open main activity file **MainActivity.java** from **\java\com.tutlane.relativelayout** path and write the code like as shown below.

**MainActivity.java**

package com.sarker.linearlayout;  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
  
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity\_main);  
    }  
}

If we observe above code we are calling our layout using **setContentView** method in the form of **R.layout.layout\_file\_name**. Here our xml file name is **activity\_main.xml** so we used file name **activity\_main**.

Generally, during the launch of our [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle), **onCreate()** callback method will be called by android framework to get the required layout for an [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle).

**Output**

When we run above example using android virtual device (AVD) we will get a result like as shown below.

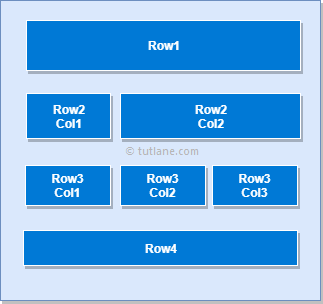


This is how we can use RelativeLayout in android applications based on our requirements.

**5. Android Table Layout with Examples**

In android, **TableLayout** is a **ViewGroup** subclass which is used to display the child View elements in rows and columns.

Following is the pictorial representation of table layout in android applications.



In android, TableLayout will position its children elements into rows and columns and it won’t display any border lines for rows, columns or cells.

The TableLayout in android will work same as HTML table and table will have as many columns as the row with the most cells. The TableLayout can be explained as **<table>** and TableRow is like **<tr>** element.

**Android TableLayout Example**

Following is the example of creating a **TableLayout** with different controls in android application.

Create a new android application using android studio and give names as **TableLayout**. In case if you are not aware of creating an app in android studio check this article [Android Hello World App](https://www.tutlane.com/tutorial/android/android-hello-world-app-example).

Now open an **activity\_main.xml** file from **\res\layout** path and write the code like as shown below

**activity\_main.xml**

<?xml version="1.0" encoding="utf-8"?>  
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout\_width="match\_parent"  
    android:layout\_height="match\_parent"  
    android:layout\_marginTop="100dp"  
    android:paddingLeft="10dp"  
    android:paddingRight="10dp" >  
    <TableRow android:background="#0079D6" android:padding="5dp">  
        <TextView  
            android:layout\_width="wrap\_content"  
            android:layout\_height="wrap\_content"  
            android:layout\_weight="1"  
            android:text="UserId" />  
        <TextView  
            android:layout\_width="wrap\_content"  
            android:layout\_height="wrap\_content"  
            android:layout\_weight="1"  
            android:text="User Name" />  
        <TextView  
            android:layout\_width="wrap\_content"  
            android:layout\_height="wrap\_content"  
            android:layout\_weight="1"  
            android:text="Location" />  
    </TableRow>  
    <TableRow android:background="#DAE8FC" android:padding="5dp">  
        <TextView  
            android:layout\_width="wrap\_content"  
            android:layout\_height="wrap\_content"  
            android:layout\_weight="1"  
            android:text="1" />  
        <TextView  
            android:layout\_width="wrap\_content"  
            android:layout\_height="wrap\_content"  
            android:layout\_weight="1"  
            android:text="Suresh Dasari" />  
        <TextView  
            android:layout\_width="wrap\_content"  
            android:layout\_height="wrap\_content"  
            android:layout\_weight="1"  
            android:text="Hyderabad" />  
    </TableRow>  
    <TableRow android:background="#DAE8FC" android:padding="5dp">  
        <TextView  
            android:layout\_width="wrap\_content"  
            android:layout\_height="wrap\_content"  
            android:layout\_weight="1"  
            android:text="2" />  
        <TextView  
            android:layout\_width="wrap\_content"  
            android:layout\_height="wrap\_content"  
            android:layout\_weight="1"  
            android:text="Rohini Alavala" />  
        <TextView  
            android:layout\_width="wrap\_content"  
            android:layout\_height="wrap\_content"  
            android:layout\_weight="1"  
            android:text="Guntur" />  
    </TableRow>  
    <TableRow android:background="#DAE8FC" android:padding="5dp">  
        <TextView  
            android:layout\_width="wrap\_content"  
            android:layout\_height="wrap\_content"  
            android:layout\_weight="1"  
            android:text="3" />  
        <TextView  
            android:layout\_width="wrap\_content"  
            android:layout\_height="wrap\_content"  
            android:layout\_weight="1"  
            android:text="Trishika Dasari" />  
        <TextView  
            android:layout\_width="wrap\_content"  
            android:layout\_height="wrap\_content"  
            android:layout\_weight="1"  
            android:text="Guntur" />  
    </TableRow>  
</TableLayout>

Once we are done with creation of layout, we need to load the XML layout resource from our [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle) **onCreate()** callback method, for that open main activity file **MainActivity.java** from **\java\com.tutlane.tablelayout** path and write the code like as shown below.

**MainActivity.java**

package com.tutlane.linearlayout;  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
  
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity\_main);  
    }  
}

If observe above code, we are calling our layout using **setContentView** method in the form of **R.layout.layout\_file\_name**. Here our xml file name is **activity\_main.xml** so we used file name **activity\_main**.

Generally, during the launch of our [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle), **onCreate()** callback method will be called by android framework to get the required layout for an [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle).

**Output**

When we run above example using android virtual device (AVD) we will get a result like as shown below.



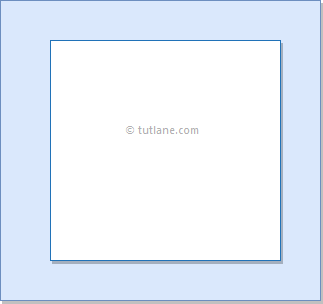
This is how we can use table layout in android applications based on our requirements.

**6. Android FrameLayout with Examples**

In android, **Framelayout** is a **ViewGroup** subclass which is used to specify the position of **View** instances it contains on the top of each other to display only single **View** inside the FrameLayout.

In simple manner, we can say FrameLayout is designed to block out an area on the screen to display a single item.

Following is the pictorial representation of frame layout in android applications.



In android, **FrameLayout** will act as a placeholder on the screen and it is used to hold a single child view.

In FrameLayout, the child views are added in a stack and the most recently added child will show on the top. We can add multiple children views to FrameLayout and control their position by using gravity attributes in FrameLayout.

**Android FrameLayout Example**

Following is the example of creating a **FrameLayout** with different controls in android application.

Create a new android application using android studio and give names as **FrameLayout**. In case if you are not aware of creating an app in android studio check this article [Android Hello World App](https://www.tutlane.com/tutorial/android/android-hello-world-app-example).

Now open an **activity\_main.xml** file from **\res\layout** path and write the code like as shown below

**activity\_main.xml**

<?xml version="1.0" encoding="utf-8"?>  
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout\_width="match\_parent"  
    android:layout\_height="match\_parent"  
    android:orientation="vertical">  
    <ImageView  
        android:id="@+id/imgvw1"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:scaleType="centerCrop"  
        android:src="@drawable/flimg" />  
    <TextView  
        android:id="@+id/txtvw1"  
        android:layout\_width="match\_parent"  
        android:layout\_height="wrap\_content"  
        android:layout\_marginTop="40dp"  
        android:background="#4C374A"  
        android:padding="10dp"  
        android:text="Grand Palace, Bangkok"  
        android:textColor="#FFFFFF"  
        android:textSize="20sp" />  
    <TextView  
        android:id="@+id/txtvw2"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:layout\_gravity="right|bottom"  
        android:background="#AA000000"  
        android:padding="10dp"  
        android:text="21/Aug/2017"  
        android:textColor="#FFFFFF"  
        android:textSize="18sp" />  
</FrameLayout>

If we observe above code we used **ImageView** to show the image (**flimg**) from drawable folder in framelayout. So add your image to **drawable** folder and replace **@drawable/flimg** path with your image path.

Once we are done with creation of layout, we need to load the XML layout resource from our [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle) **onCreate()** callback method, for that open main activity file **MainActivity.java** from **\java\com.tutlane.framelayout** path and write the code like as shown below.

**MainActivity.java**

package com.tutlane.linearlayout;  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
  
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity\_main);  
    }  
}

If we observe above code, we are calling our layout using **setContentView** method in the form of **R.layout.layout\_file\_name**. Here our xml file name is **activity\_main.xml** so we used file name **activity\_main**.

Generally, during the launch of our [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle), **onCreate()** callback method will be called by android framework to get the required layout for an [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle).

**Output**

When we run above example using android virtual device (AVD) we will get a result like as shown below.



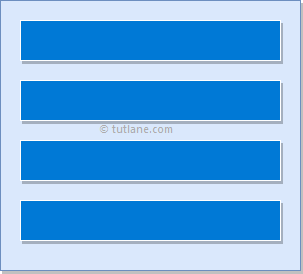
 This is how we can use frame layout in android applications based on our requirements.

**7. Android ListView with Examples**

In android, **ListView** is a **ViewGroup** which is used to display the list of scrollable of items in multiple rows and the list items are automatically inserted to the list using an **adapter**.

Generally, the adapter pulls data from a sources such as an array or database and converts each item into a result view and that’s placed into the list.

Following is the pictorial representation of listview in android applications.



**Android Adapter**

In android, **Adapter** will act as an intermediate between the data sources and adapter views such as [ListView](https://www.tutlane.com/tutorial/android/android-listview-with-examples), [Gridview](https://www.tutlane.com/tutorial/android/android-gridview-with-examples) to fill the data into adapter views. The adapter will hold the data and iterates through an items in data set and generate the views for each item in the list.

Generally, in android we have a different types of adapters available to fetch the data from different data sources to fill the data into adapter views, those are

| **Adapter** | **Description** |
| --- | --- |
| ArrayAdapter | It will expects an Array or List as input. |
| CurosrAdapter | It will accepts an instance of cursor as an input. |
| SimpleAdapter | It will accepts a static data defined in the resources. |
| BaseAdapter | It is a generic implementation for all three adapter types and it can be used for [ListView](https://www.tutlane.com/tutorial/android/android-listview-with-examples), [Gridview](https://www.tutlane.com/tutorial/android/android-gridview-with-examples) or Spinners based on our requirements. |

**Android ListView Example**

Following is the example of creating a **ListView** using **arrayadapter** in android application.

Create a new android application using android studio and give names as **ListView**. In case if you are not aware of creating an app in android studio check this article [Android Hello World App](https://www.tutlane.com/tutorial/android/android-hello-world-app-example).

Now open an **activity\_main.xml** file from **\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"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout\_width="match\_parent"  
    android:layout\_height="match\_parent"  
    android:orientation="vertical">  
    <ListView  
        android:id="@+id/userlist"  
        android:layout\_width="match\_parent"  
        android:layout\_height="wrap\_content" >  
    </ListView>  
</LinearLayout>

Once we are done with creation of layout, now we will bind data to our **ListView**  using

**ArrayAdapter**, for that open main activity file **MainActivity.java** from **\java\com.tutlane.listview**

path and write the code like as shown below.

**MainActivity.java**

package com.tutlane.listview;  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
import android.widget.ArrayAdapter;  
import android.widget.ListView;  
  
public class MainActivity extends AppCompatActivity {  
    private ListView mListView;  
    private ArrayAdapter aAdapter;  
    private String[] users = { "Suresh Dasari", "Rohini Alavala", "Trishika Dasari", "Praveen Alavala", "Madav Sai", "Hamsika Yemineni"};  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity\_main);  
        mListView = (ListView) findViewById(R.id.userlist);  
        aAdapter = new ArrayAdapter(this, android.R.layout.simple\_list\_item\_1, users);  
        mListView.setAdapter(aAdapter);  
    }  
}

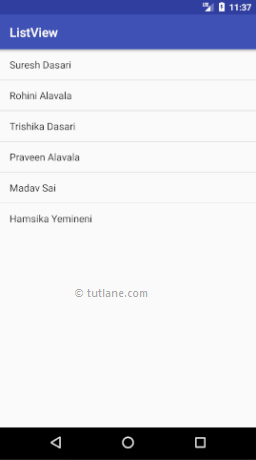
If we observe above code, we are binding static array (**users**) details to **ListView** using

**ArrayAdapter** and calling our layout using **setContentView** method in the form of **R.layout.layout\_file\_name**. Here our xml file name is **activity\_main.xml** so we used file name **activity\_main**.

Generally, during the launch of our [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle), onCreate() callback method will be called by android framework to get the required layout for an [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle).

**Output**

When we run above example using android virtual device (AVD) we will get a result like as shown below.



This is how we can bind data to ListView using ArrayAdapter in android applications based on our requirements.

**Android ListView with Custom Adapter Example**

In previous example, we learned simple way to bind data to **ListView** using **ArrayAdapter** in android application. Now we will see how to create our own custom adapter and bind data to ListView with example.

For this, we need to create our own custom adapter class by extending with **BaseAdapter** class and create a class which will contain a parameters for list row items.

Now create a new android application using android studio and give names as **ListView**. In case if you are not aware of creating an app in android studio check this article [Android Hello World App](https://www.tutlane.com/tutorial/android/android-hello-world-app-example).

Open an **activity\_main.xml** file from \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:layout\_width="fill\_parent"  
    android:layout\_height="fill\_parent"  
    android:orientation="vertical" >  
    <ListView  
        android:id="@+id/user\_list"  
        android:layout\_width="fill\_parent"  
        android:layout\_height="wrap\_content"  
        android:dividerHeight="1dp" />  
</LinearLayout>

Now we need to create a layout for **listview** row items, for that right click on **layouts** folder 🡪 select New 🡪 Layout resource file 🡪 Give name as **list\_row.xml** and click **OK**. Now open newly created file (**list\_row.xml**) and write the code like as shown below

list\_row.xml

<?xml version="1.0" encoding="utf-8"?>  
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout\_width="fill\_parent"  
    android:layout\_height="wrap\_content"  
    android:orientation="horizontal"  
    android:padding="5dip" >  
    <TextView  
        android:id="@+id/name"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:textStyle="bold"  
        android:textSize="17dp" />  
    <TextView  
        android:id="@+id/designation"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:layout\_below="@id/name"  
        android:layout\_marginTop="7dp"  
        android:textColor="#343434"  
        android:textSize="14dp" />  
    <TextView  
        android:id="@+id/location"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:layout\_alignBaseline="@+id/designation"  
        android:layout\_alignBottom="@+id/designation"  
        android:layout\_alignParentRight="true"  
        android:textColor="#343434"  
        android:textSize="14dp" />  
</RelativeLayout>

Now we need to create a custom class (**ListItem.java**) to represent each row in the list, for that right click on **java**folder 🡪 select New 🡪 Java Class 🡪 Give name as **ListItem.java** and click **OK**. Open **ListItem.java** file and write the code like as shown below

**ListItem.java**

package com.tutlane.listview;  
/\*\*  
 \* Created by tutlane on 23-08-2017.  
 \*/  
public class ListItem {  
    private String name;  
    private String designation;  
    private String location;  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public String getDesignation() {  
        return designation;  
    }  
    public void setDesignation(String designation) {  
        this.designation = designation;  
    }  
    public String getLocation() {  
        return location;  
    }  
    public void setLocation(String location) {  
        this.location = location;  
    }  
}

Now we need to create a custom adapter (**CustomListAdapter.java**) and extend it by using **BaseAdapter**. In case if we are extending our class by using **BaseAdapter**, we need to override following methods from **BaseAdapter** class.

| **Method** | **Description** |
| --- | --- |
| getCount() | It will return total number of rows count in listview |
| getItem() | It is used to specify the object data of each row |
| getItemId() | It return the id of each row item |
| getView() | It is used to return a view instance that represents a single row in ListView item. |

To create custom adapter right click on **java** folder 🡪 select New 🡪 Java Class 🡪 Give name as **CustomListAdapter.java**and click **OK**.

Open **CustomListAdapter.java** file and write the code like as shown below

**CustomListAdapter.java**

package com.tutlane.listview;  
import android.content.Context;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.BaseAdapter;  
import android.widget.TextView;  
  
import java.util.ArrayList;  
/\*\*  
 \* Created by tutlane on 23-08-2017.  
 \*/  
public class CustomListAdapter extends BaseAdapter {  
    private ArrayList<ListItem> listData;  
    private LayoutInflater layoutInflater;  
    public CustomListAdapter(Context aContext, ArrayList<ListItem> listData) {  
        this.listData = listData;  
        layoutInflater = LayoutInflater.from(aContext);  
    }  
    @Override  
    public int getCount() {  
        return listData.size();  
    }  
    @Override  
    public Object getItem(int position) {  
        return listData.get(position);  
    }  
    @Override  
    public long getItemId(int position) {  
        return position;  
    }  
    public View getView(int position, View v, ViewGroup vg) {  
        ViewHolder holder;  
        if (v == null) {  
            v = layoutInflater.inflate(R.layout.list\_row, null);  
            holder = new ViewHolder();  
            holder.uName = (TextView) v.findViewById(R.id.name);  
            holder.uDesignation = (TextView) v.findViewById(R.id.designation);  
            holder.uLocation = (TextView) v.findViewById(R.id.location);  
            v.setTag(holder);  
        } else {  
            holder = (ViewHolder) v.getTag();  
        }  
        holder.uName.setText(listData.get(position).getName());  
        holder.uDesignation.setText(listData.get(position).getDesignation());  
        holder.uLocation.setText(listData.get(position).getLocation());  
        return v;  
    }  
    static class ViewHolder {  
        TextView uName;  
        TextView uDesignation;  
        TextView uLocation;  
    }  
}

If we observe above class we are extending our custom adapter by using **BaseAdapter** and we override all **BaseAdapter**methods in our custom adapter.

Now we need to combine all the custom classes in main activity file (**MainActivity.java**) to bind the data to our listview.

Open main activity file (**MainActivity.java**) and write the code like as shown below.

**MainActivity.java**

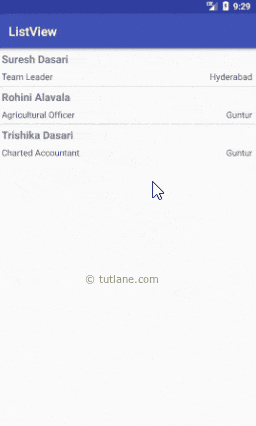
package com.tutlane.listview;  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.AdapterView;  
import android.widget.ListView;  
import android.widget.Toast;  
  
import java.util.ArrayList;  
  
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity\_main);  
        ArrayList userList = getListData();  
        final ListView lv = (ListView) findViewById(R.id.user\_list);  
        lv.setAdapter(new CustomListAdapter(this, userList));  
        lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {  
            @Override  
            public void onItemClick(AdapterView<?> a, View v, int position, long id) {  
                ListItem user = (ListItem) lv.getItemAtPosition(position);  
                Toast.makeText(MainActivity.this, "Selected :" + " " + user.getName()+", "+ user.getLocation(), Toast.LENGTH\_SHORT).show();  
            }  
        });  
    }  
    private ArrayList getListData() {  
        ArrayList<ListItem> results = new ArrayList<>();  
        ListItem user1 = new ListItem();  
        user1.setName("Suresh Dasari");  
        user1.setDesignation("Team Leader");  
        user1.setLocation("Hyderabad");  
        results.add(user1);  
        ListItem user2 = new ListItem();  
        user2.setName("Rohini Alavala");  
        user2.setDesignation("Agricultural Officer");  
        user2.setLocation("Guntur");  
        results.add(user2);  
        ListItem user3 = new ListItem();  
        user3.setName("Trishika Dasari");  
        user3.setDesignation("Charted Accountant");  
        user3.setLocation("Guntur");  
        results.add(user3);  
        return results;  
    }  
}

If we observe above code we are building and binding data to ListView using our custom adapter and calling our layout using **setContentView** method in the form of **R.layout.layout\_file\_name**. Here our xml file name is **activity\_main.xml** so we used file name **activity\_main**.

Generally, during the launch of our [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle), **onCreate()** callback method will be called by android framework to get the required layout for an [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle).

**Output**

When we run above example using android virtual device (AVD) we will get a result like as shown below.



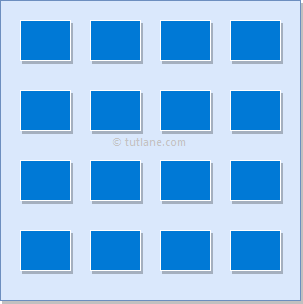
 This is how we can bind data to ListView using custom adapter in android applications based on our requirements.

**8. Android GridView with Examples**

In android, **Grid View** is a **ViewGroup** which is used to display items in a two dimensional, scrollable grid and grid items are automatically inserted to the gridview layout using a list adapter.

Generally, the adapter pulls data from a sources such as an array or database and converts each item into a result view and that’s placed into the list.

Following is the pictorial representation of GridView in android applications.



**Android Adapter**

In android, **Adapter** will act as an intermediate between the data sources and adapter views such as [ListView](https://www.tutlane.com/tutorial/android/android-listview-with-examples), [Gridview](https://www.tutlane.com/tutorial/android/android-gridview-with-examples) to fill the data into adapter views. The adapter will hold the data and iterates through an items in data set and generate the views for each item in the list.

Generally, in android we have a different types of adapters available to fetch the data from different data sources to fill the data into adapter views, those are

| **Adapter** | **Description** |
| --- | --- |
| ArrayAdapter | It will expects an Array or List as input. |
| CurosrAdapter | It will accepts an instance of cursor as an input. |
| SimpleAdapter | It will accepts a static data defined in the resources. |
| BaseAdapter | It is a generic implementation for all three adapter types and it can be used for [ListView](https://www.tutlane.com/tutorial/android/android-listview-with-examples), [Gridview](https://www.tutlane.com/tutorial/android/android-gridview-with-examples) or Spinners based on our requirements |

**Android GridView Example**

Following is the simple example showing user details using **GridView** and showing the position of particular image when clicking on it in android applications.

Create a new android application using android studio and give names as **GridView**. In case if you are not aware of creating an app in android studio check this article [Android Hello World App](https://www.tutlane.com/tutorial/android/android-hello-world-app-example).

Once we create an application, add some sample images to project **/res/drawable** directory to show the images in GridView.

Now open an **activity\_main.xml** file from **/res/layout** path and write the code like as shown below

**activity\_main.xml**

<?xml version="1.0" encoding="utf-8"?>  
<GridView xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/gridview"  
    android:layout\_width="match\_parent"  
    android:layout\_height="match\_parent"  
    android:columnWidth="110dp"  
    android:numColumns="auto\_fit"  
    android:verticalSpacing="10dp"  
    android:horizontalSpacing="10dp"  
    android:stretchMode="columnWidth"  
    android:gravity="center" />

Once we are done with creation of layout, we need to create a custom adapter (**ImageAdapter.java**) by extending it using **BaseExtender** to show all the items in the grid, for that right click on **java** folder 🡪 Give name as **ImageAdapter.java** and click **OK**.

Open **ImageAdapter.java** file and write the code like as shown below

**ImageAdapter.java**

package com.tutlane.gridview;  
import android.content.Context;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.BaseAdapter;  
import android.widget.GridView;  
import android.widget.ImageView;  
/\*\*  
 \* Created by tutlane on 24-08-2017.  
 \*/  
public class ImageAdapter extends BaseAdapter {  
    private Context mContext;  
    public ImageAdapter(Context c) {  
        mContext = c;  
    }  
    public int getCount() {  
        return thumbImages.length;  
    }  
    public Object getItem(int position) {  
        return null;  
    }  
    public long getItemId(int position) {  
        return 0;  
    }  
    // create a new ImageView for each item referenced by the Adapter  
    public View getView(int position, View convertView, ViewGroup parent) {  
            ImageView imageView = new ImageView(mContext);  
            imageView.setLayoutParams(new GridView.LayoutParams(200, 200));  
            imageView.setScaleType(ImageView.ScaleType.CENTER\_CROP);  
            imageView.setPadding(8, 8, 8, 8);  
            imageView.setImageResource(thumbImages[position]);  
            return imageView;  
    }  
    // Add all our images to arraylist  
    public Integer[] thumbImages = {  
            R.drawable.img1, R.drawable.img2,  
            R.drawable.img3, R.drawable.img4,  
            R.drawable.img5, R.drawable.img6,  
            R.drawable.img7, R.drawable.img8,  
            R.drawable.img1, R.drawable.img2,  
            R.drawable.img3, R.drawable.img4,  
            R.drawable.img5, R.drawable.img6,  
            R.drawable.img7, R.drawable.img8,  
            R.drawable.img1, R.drawable.img2,  
            R.drawable.img3, R.drawable.img4,  
            R.drawable.img5  
    };  
}

If we observe above code we referred some images, actually those are the sample images which we added in **/res/drawable** directory.

Now we will bind images to our **GridView** using our custom adapter (**ImageAdapter.java**), for that open main activity file **MainActivity.java** from **\java\com.tutlane.gridview** path and write the code like as shown below.

**MainActivity.java**

package com.tutlane.gridview  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.AdapterView;  
import android.widget.GridView;  
import android.widget.Toast;

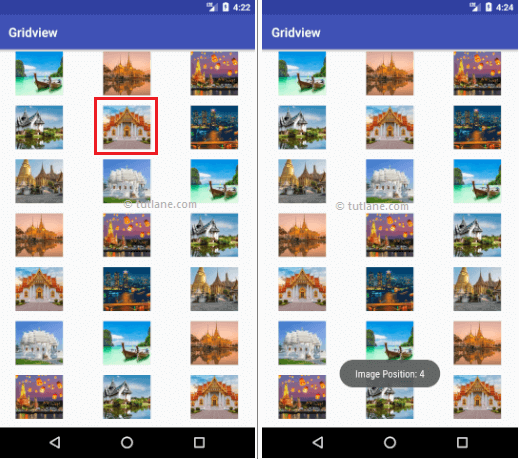
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity\_main);  
        GridView gv = (GridView) findViewById(R.id.gvDetails);  
        gv.setAdapter(new ImageAdapter(this));  
        gv.setOnItemClickListener(new AdapterView.OnItemClickListener() {  
            public void onItemClick(AdapterView<?> parent, View v, int position, long id) {  
                Toast.makeText(MainActivity.this, "Image Position: " + position, Toast.LENGTH\_SHORT).show();  
            }  
        });  
    }  
}

If we observe above code, we are binding image details to **GridView** using our custom adapter (**ImageAdapter.java**) and calling our layout using **setContentView** method in the form of **R.layout.layout\_file\_name**. Here our xml file name is **activity\_main.xml** so we used file name **activity\_main**.

Generally, during the launch of our [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle), **onCreate()** callback method will be called by android framework to get the required layout for an [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle).

**Output**

When we run above example using android virtual device (AVD) we will get a result like as shown below.



This is how we can bind images to GridView using Adapter in android applications based on our requirements.

**Android GridView Details Activity Example**

In above example, we implemented an image gallery using gridview in android application. Now we will extend the functionality of above example to show the selected grid image in full screen.

Now we need to create a new layout (**image\_details.xml**) file in project **/res/layout** directory to show the image details, for that right click on layouts folder 🡪 select New 🡪 Layout resource file 🡪 Give name as **image\_details.xml** and click **OK**. Now open newly created file (**image\_details.xml**) and write the code like as shown below.

**image\_details.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">  
    <ImageView android:id="@+id/full\_image\_view"  
        android:layout\_width="match\_parent"  
        android:layout\_height="match\_parent" />  
</LinearLayout>

Now we need to create a custom activity (**FullImageActivity.java**) to show the image details in our newly created layout (**image\_details.xml**) file, for that right click on java folder 🡪 select New 🡪 Java Class 🡪 Give name as **FullImageActivity.java** and click **OK**.

Open **FullImageActivity.java** file and write the code like as shown below

**FullImageActivity.java**

package com.tutlane.gridview;  
import android.app.Activity;  
import android.content.Intent;  
import android.os.Bundle;  
import android.widget.ImageView;

/\*\*  
 \* Created by tutlane on 24-08-2017.  
 \*/  
public class FullImageActivity extends Activity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.image\_details);  
        // Get intent data  
        Intent i = getIntent();  
        // Get Selected Image Id  
        int position = i.getExtras().getInt("id");  
        ImageAdapter imageAdapter = new ImageAdapter(this);  
        ImageView imageView = (ImageView) findViewById(R.id.full\_image\_view);  
        imageView.setImageResource(imageAdapter.thumbImages[position]);  
    }  
}

Now we need to include our newly created activity file (**FullImageActivity.java**) in **AndroidManifest.xml** file like as shown below. For that, open **AndroidManifest.xml** file and write the code 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.gridview">  
  
    <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>  
        <!-- FullImageActivity -->  
        <activity android:name=".FullImageActivity"></activity>  
    </application>  
</manifest>

Now we need to modify gridview image click function in main activity file (**MainActivity.java**) to get image details and show it in new activity.

Open main activity file (**MainActivity.java**) and write the code like as shown below.

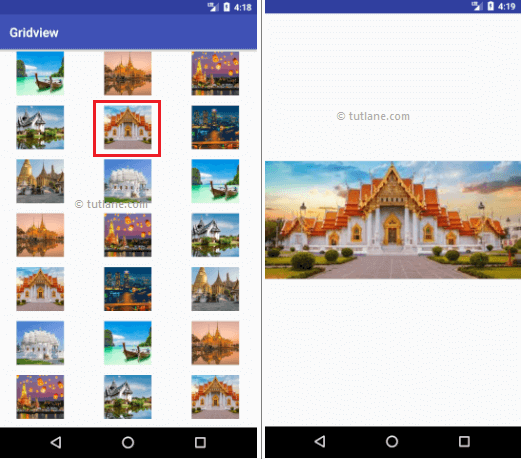
**MainActivity.java**

package com.tutlane.gridview;  
import android.content.Intent;  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.AdapterView;  
import android.widget.GridView;  
  
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity\_main);  
        GridView gv = (GridView) findViewById(R.id.gvDetails);  
        gv.setAdapter(new ImageAdapter(this));  
        gv.setOnItemClickListener(new AdapterView.OnItemClickListener() {  
            public void onItemClick(AdapterView<?> parent, View v, int position, long id) {  
                // Sending image id to FullScreenActivity  
                Intent i = new Intent(getApplicationContext(), FullImageActivity.class);  
                // passing array index  
                i.putExtra("id", position);  
                startActivity(i);  
            }  
        });  
    }  
}

If we observe above code, we are getting the selected image details on image click and sending those details to our newly created activity file to show the image in full screen.

**Output**

When we run above example using android virtual device (AVD) we will get a result like as shown below.



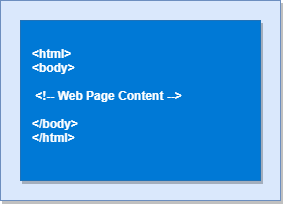
This is how we can build image gallery in gridview and show the selected image in android applications based on our requirements.

**9. Android WebView with Examples**

In android, **WebView** is an extension of **View** class and it is used to show the static HTML web pages content or remote web pages content with URL in android applications as a part of our activity layout.

Generally, in android the **WebView** will act as an embedded browser to include the web pages content in our activity layout and it won’t contain any features of normal browser, such as address bar, navigation controls, etc.

Following is the pictorial representation of WebView in android applications.



Generally, in android **WebView** is useful to include and show the content of other web pages or application content in our required pages, such as an end-user agreements, etc.

**Android WebView Example**

Following is the example of showing a static HTML content in WebView in android applications.

Create a new android application using android studio and give names as **WebView**. In case if you are not aware of creating an app in android studio check this article [Android Hello World App](https://www.tutlane.com/tutorial/android/android-hello-world-app-example).

Now open an **activity\_main.xml** file from **/res/layout** path and write the code like as shown below

**activity\_main.xml**

<?xml version="1.0" encoding="utf-8"?>  
<WebView  xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/webview"  
    android:layout\_width="fill\_parent"  
    android:layout\_height="fill\_parent" />

Once we are done with adding a **WebView** to the layout, now we will show the static HTML content in **WebView**, for that open main activity file **MainActivity.java** from **\java\com.tutlane.webview**

path and write the code like as shown below.

**MainActivity.java**

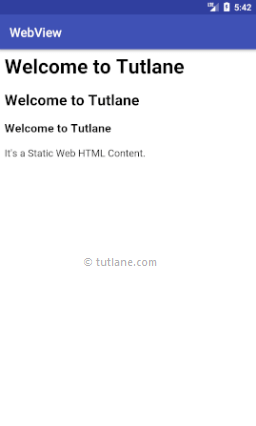
package com.tutlane.webview;  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
import android.webkit.WebView;  
  
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity\_main);  
        WebView wv = (WebView) findViewById(R.id.webview);  
        String customHtml = "<html><body><h1>Welcome to Tutlane</h1>" +  
                "<h2>Welcome to Tutlane</h2><h3>Welcome to Tutlane</h3>" +  
                "<p>It's a Static Web HTML Content.</p>" +  
                "</body></html>";  
        wv.loadData(customHtml, "text/html", "UTF-8");  
    }  
}

If we observe above code, we are calling our layout using **setContentView** method in the form of **R.layout.layout\_file\_name**. Here our xml file name is **activity\_main.xml** so we used file name **activity\_main** and trying to show the static HTML content in **WebView**.

Generally, during the launch of our [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle), onCreate() callback method will be called by android framework to get the required layout for an [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle).

**Output**

When we run above example using android virtual device (AVD) we will get a result like as shown below.



This is how we can show the static HTML content using WebView in android applications based on our requirements.

**Android Show Web URL Content in WebView Example**

Generally, in android **WebView** will act as an embedded browser to show the static or remote web page content in our android applications.

Now we will see how to load remote URL content in WebView with example in android application.

By using WebView **LoadURL** property we can load remote URL content in our android applications. To show the remote URL content in webview modify **MainActivity.java** file code like as shown below.

**ManiActivity.java**

package com.tutlane.webview;  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
import android.webkit.WebView;  
  
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity\_main);  
        WebView webView = (WebView) findViewById(R.id.webview);  
        webView.getSettings().setJavaScriptEnabled(true);  
        webView.loadUrl("http://www.juniv.edu");  
    }  
}

If we observe above example, we are trying to load the remote URL (**http://sarker.com**) content in our android application using **WebView** and we set a property **setJavaScriptEnabled()** to enable JavaScript because by default the JavaScript is disabled in WebView.

Generally, the web page which we are loading in **WebView** might use JavaScript. In case if we won’t enable the JavaScript, the functionality which related to JavaScript in web page won’t work that’s the reason we are enabling the JavaScript using setJavaScriptEnabled()

To load the remote web URL content, our application must have an access to the internet. We need to set the internet access permissions like as shown below.

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

……  
</manifest>

Now open our application **AndroidManifest.xml** file in **/manifests** directory and write the code like as shown below

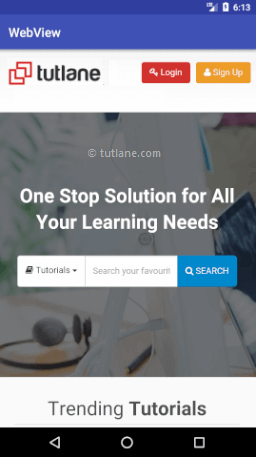
**AndroidManifest.xml**

<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
    package="com.tutlane.webview">  
    <application  
        android:allowBackup="true"  
        android:icon="@mipmap/ic\_launcher"  
        android:label="@string/app\_name"  
        android:roundIcon="@mipmap/ic\_launcher\_round"  
        android:supportsRtl="true"  
        android:theme="@style/AppTheme">  
        <activity android:name=".MainActivity">  
            <intent-filter>  
                <action android:name="android.intent.action.MAIN" />  
                <category android:name="android.intent.category.LAUNCHER" />  
            </intent-filter>  
        </activity>  
    </application>  
    <uses-permission android:name="android.permission.INTERNET" />  
</manifest>

Once we are done with required settings, now we will run and see the result of our application.

**Output**

When we run above example using android virtual device (AVD) we will get a result like as shown below.



This is how we can show the remote URL web pages content using **WebView** in android applications based on our requirements.