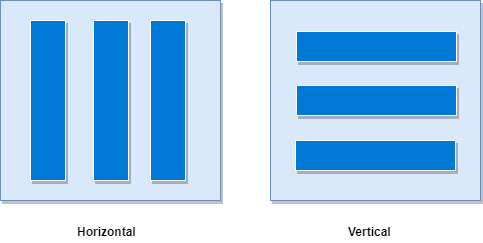
**4Android LinearLayout with Examples**

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 you 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**.

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 **onCreate()** callback method, for that open main activity file **MainActivity.java** from **\java\com.tutlane.linearlayout** path and write the code like as shown below.

**MainActivity.java**

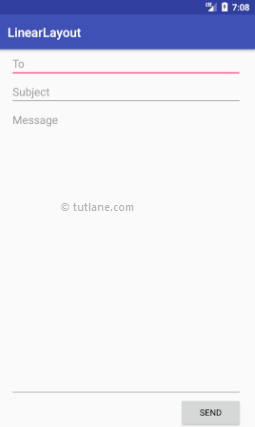
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
  
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity\_main);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

If you 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, the **onCreate()** callback method will be called by the android framework to get the required layout for an activity.

**Output of Android LinearLayout Example**

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



**Layout Weight Attribute**

If you observe the 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 a larger weight value to the child view, then it will expand to fill any remaining space in the parent view.

If you 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.