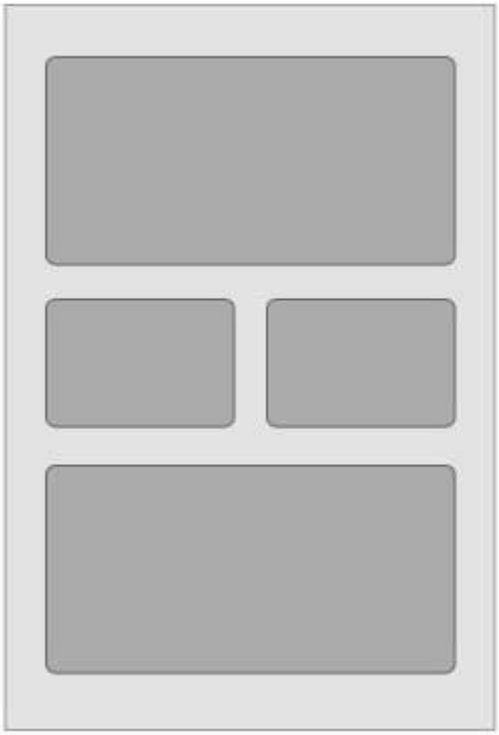
Relative Layout

Android RelativeLayout enables you to specify how child views are positioned relative to each other. The position of each view can be specified as relative to sibling elements or relative to the parent.



Relative Layout

RelativeLayout Attributes

Following are the important attributes specific to RelativeLayout −

Sr.No.	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:gravity** This specifies how an object should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
3	**android:ignoreGravity** This indicates what view should not be affected by gravity.

Sr.No.	Attribute & Description
1	**android:layout_above** Positions the bottom edge of this view above the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name"
2	**android:layout_alignBottom** Makes the bottom edge of this view match the bottom edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
3	**android:layout_alignLeft** Makes the left edge of this view match the left edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
4	**android:layout_alignParentBottom** If true, makes the bottom edge of this view match the bottom edge of the parent. Must be a boolean value, either "true" or "false".
5	**android:layout_alignParentEnd** If true, makes the end edge of this view match the end edge of the parent. Must be a boolean value, either "true" or "false".
6	**android:layout_alignParentLeft** If true, makes the left edge of this view match the left edge of the parent. Must be a boolean value, either "true" or "false".
7	**android:layout_alignParentRight** If true, makes the right edge of this view match the right edge of the parent. Must be a boolean value, either "true" or "false".
8	**android:layout_alignParentStart** If true, makes the start edge of this view match the start edge of the parent. Must be a boolean value, either "true" or "false".
9	**android:layout_alignParentTop** If true, makes the top edge of this view match the top edge of the parent. Must be a boolean value, either "true" or "false".
10	**android:layout_alignRight** Makes the right edge of this view match the right edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
11	**android:layout_alignStart** Makes the start edge of this view match the start edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
12	**android:layout_alignTop** Makes the top edge of this view match the top edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
13	**android:layout_below**

	Positions the top edge of this view below the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
14	**android:layout_centerHorizontal** If true, centers this child horizontally within its parent. Must be a boolean value, either "true" or "false".
15	**android:layout_centerInParent** If true, centers this child horizontally and vertically within its parent. Must be a boolean value, either "true" or "false".
16	**android:layout_centerVertical** If true, centers this child vertically within its parent. Must be a boolean value, either "true" or "false".
17	**android:layout_toEndOf** Positions the start edge of this view to the end of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
18	**android:layout_toLeftOf** Positions the right edge of this view to the left of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
19	**android:layout_toRightOf** Positions the left edge of this view to the right of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
20	**android:layout_toStartOf** Positions the end edge of this view to the start of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".

Using RelativeLayout, you can align two elements by right border, or make one below another, centered in the screen, centered left, and so on. By default, all child views are drawn at the top-left of the layout, so you must define the position of each view using the various layout properties available from **RelativeLayout.LayoutParams** and few of the important attributes are given below −

Step	Description
1	You will use Android Studio IDE to create an Android application and name it as *demo* under a package *com.example.demo* as explained in the *Hello World Example* chapter.
2	Modify the default content of *res/layout/activity_main.xml* file to include few widgets in Relative layout.
3	Define required constants in *res/values/strings.xml* file

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

Example

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

Following is the content of the modified main activity file **src/com.example.demo/MainActivity.java**. This file can include each of the fundamental lifecycle methods.

```
package com.example.demo;

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

public class MainActivity extends Activity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }

}
```

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

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:paddingLeft="16dp"
   android:paddingRight="16dp" >

   <EditText
      android:id="@+id/name"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:hint="@string/reminder" />

   <LinearLayout
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:layout_alignParentStart="true"
      android:layout_below="@+id/name">

      <Button
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="New Button"
         android:id="@+id/button" />
```

```
      <Button
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="New Button"
         android:id="@+id/button2" />

   </LinearLayout>

</RelativeLayout>
```

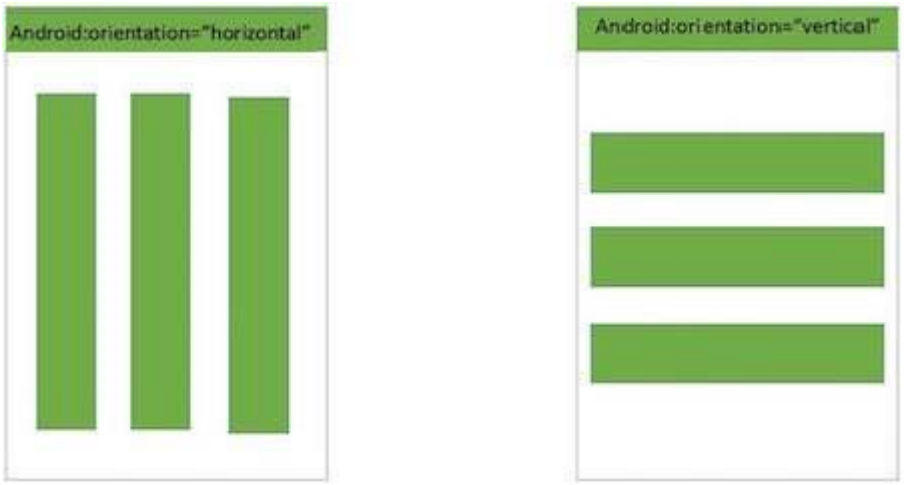
Following will be the content of **res/values/strings.xml** to define two new constants −

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="action_settings">Settings</string>
   <string name="reminder">Enter your name</string>
</resources>
```

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

Linear Layout

Android LinearLayout is a view group that aligns all children in either *vertically* or *horizontally*.



Linear Layout

LinearLayout Attributes

Following are the important attributes specific to LinearLayout −

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:baselineAligned** This must be a boolean value, either "true" or "false" and prevents the layout from aligning its children's baselines.
3	**android:baselineAlignedChildIndex** When a linear layout is part of another layout that is baseline aligned, it can specify which of its children to baseline align.
4	**android:divider** This is drawable to use as a vertical divider between buttons. You use a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
5	**android:gravity** This specifies how an object should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
6	**android:orientation** This specifies the direction of arrangement and you will use "horizontal" for a row, "vertical" for a column. The default is horizontal.
7	**android:weightSum**

Sum up of child weight

Example

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

Step	Description
1	You will use Android Studio to create an Android application and name it as *Demo* under a package *com.example.demo* as explained in the *Hello World Example* chapter.
2	Modify the default content of *res/layout/activity_main.xml* file to include few buttons in linear layout.
3	No need to change string Constants.Android studio takes care of default strings
4	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **src/com.example.demo/MainActivity.java**. This file can include each of the fundamental lifecycle methods.

```
package com.example.demo;

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

public class MainActivity extends Activity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }
}
```

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

```
<?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" >

   <Button android:id="@+id/btnStartService"
      android:layout_width="270dp"
      android:layout_height="wrap_content"
      android:text="start_service"/>
```

```
   <Button android:id="@+id/btnPauseService"
      android:layout_width="270dp"
      android:layout_height="wrap_content"
      android:text="pause_service"/>

   <Button android:id="@+id/btnStopService"
      android:layout_width="270dp"
      android:layout_height="wrap_content"
      android:text="stop_service"/>

</LinearLayout>
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">HelloWorld</string>
   <string name="action_settings">Settings</string>
</resources>
```

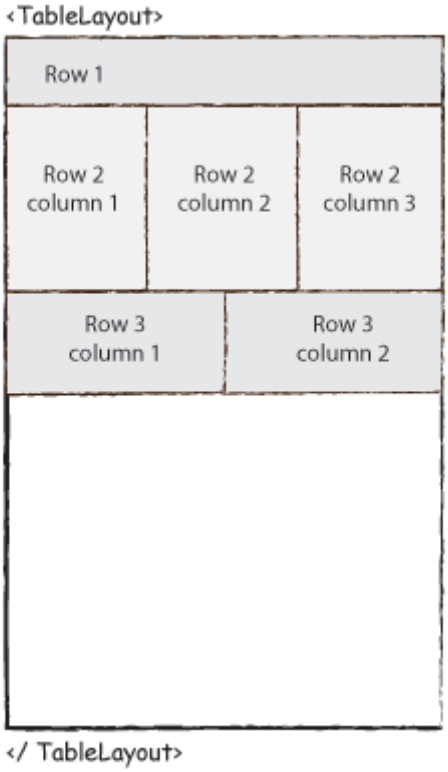
Let's try to run our modified **Hello World!** application we just modified. I assume you had created your **AVD** while doing environment setup. To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar. Android studio installs the app on your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

Now let's change the orientation of Layout as **android:orientation="horizontal"** and try to run the same application, it will give following screen −

Table Layout

Android TableLayout going to be arranged groups of views into rows and columns. You will use the <TableRow> element to build a row in the table. Each row has zero or more cells; each cell can hold one View object.

TableLayout containers do not display border lines for their rows, columns, or cells.



TableLayout Attributes

Following are the important attributes specific to TableLayout −

Sr.No.	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:collapseColumns** This specifies the zero-based index of the columns to collapse. The column indices must be separated by a comma: 1, 2, 5.
3	**android:shrinkColumns** The zero-based index of the columns to shrink. The column indices must be separated by a comma: 1, 2, 5.
4	**android:stretchColumns** The zero-based index of the columns to stretch. The column indices must be separated by a comma: 1, 2, 5.

Example

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

Step	Description
1	You will use Android Studio IDE to create an Android application and name it as *demo* under a package *com.example.demo* as explained in the *Hello World Example* chapter.
2	Modify the default content of *res/layout/activity_main.xml* file to include few widgets in table layout.
3	No need to modify string.xml, Android studio takes care of default constants
4	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **src/com.example.demo/MainActivity.java**. This file can include each of the fundamental lifecycle methods.

```
package com.example.demo;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

public class MainActivity extends Activity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
```

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

}
```

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

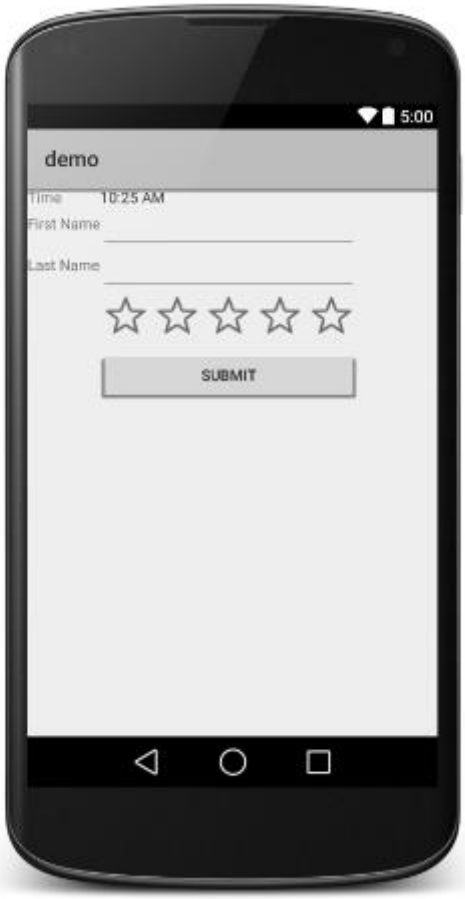
```
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">

   <TableRow
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">

      <TextView
         android:text="Time"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_column="1" />

      <TextClock
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:id="@+id/textClock"
         android:layout_column="2" />

   </TableRow>

   <TableRow>

      <TextView
         android:text="First Name"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_column="1" />

      <EditText
         android:width="200px"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content" />
   </TableRow>

   <TableRow>

      <TextView
         android:text="Last Name"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_column="1" />

      <EditText
```

```
         android:width="100px"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content" />
   </TableRow>

   <TableRow
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">

      <RatingBar
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:id="@+id/ratingBar"
         android:layout_column="2" />
   </TableRow>

   <TableRow
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"/>

   <TableRow
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">

      <Button
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Submit"
         android:id="@+id/button"
         android:layout_column="2" />
   </TableRow>

</TableLayout>
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">HelloWorld</string>
   <string name="action_settings">Settings</string>
</resources>
```

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



Frame Layout

Frame Layout is designed to block out an area on the screen to display a single item. Generally, FrameLayout should be used to hold a single child view, because it can be difficult to organize child views in a way that's scalable to different screen sizes without the children overlapping each other.

You can, however, add multiple children to a FrameLayout and control their position within the FrameLayout by assigning gravity to each child, using the android:layout_gravity attribute.

Frame Layout

FrameLayout Attributes

Following are the important attributes specific to FrameLayout −

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:foreground** This defines the drawable to draw over the content and possible values may be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
3	**android:foregroundGravity** Defines the gravity to apply to the foreground drawable. The gravity defaults to fill. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
4	**android:measureAllChildren** Determines whether to measure all children or just those in the VISIBLE or INVISIBLE state when measuring. Defaults to false.

Example

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

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *demo* under a package *com.example.demo* as explained in the *Hello World Example* chapter.
2	Modify the default content of *res/layout/activity_main.xml* file to include few widgets in frame layout.
3	No need to change string.xml, android takes care default constants
4	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **src/com.example.demo/MainActivity.java**. This file can include each of the fundamental lifecycle methods.

```
package com.example.demo;

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

public class MainActivity extends Activity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }
}
```

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

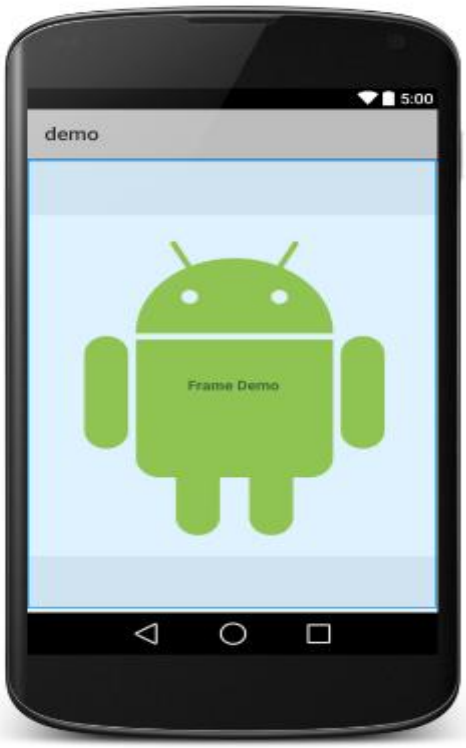
```
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">

   <ImageView
      android:src="@drawable/ic_launcher"
      android:scaleType="fitCenter"
      android:layout_height="250px"
      android:layout_width="250px"/>

   <TextView
      android:text="Frame Demo"
      android:textSize="30px"
      android:textStyle="bold"
```

```
      android:layout_height="fill_parent"
      android:layout_width="fill_parent"
      android:gravity="center"/>
</FrameLayout>
```

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

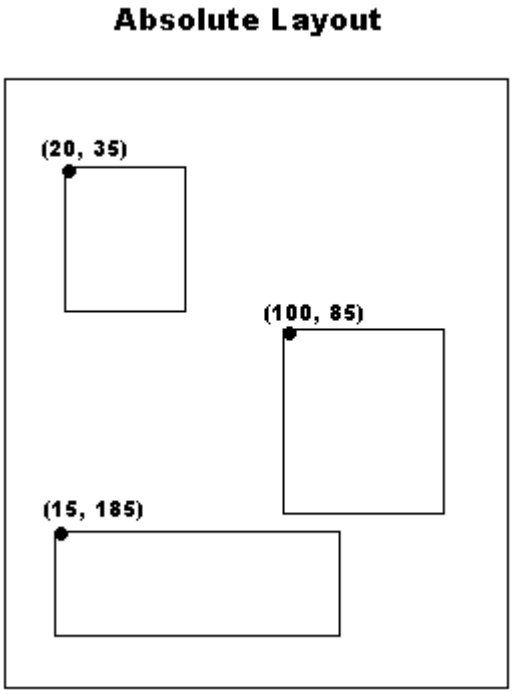
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">demo</string>
   <string name="action_settings">Settings</string>
</resources>
```

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



Absolute Layout

An Absolute Layout lets you specify exact locations (x/y coordinates) of its children. Absolute layouts are less flexible and harder to maintain than other types of layouts without absolute positioning.



Absolute Layout

AbsoluteLayout Attributes

Following are the important attributes specific to AbsoluteLayout −

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:layout_x** This specifies the x-coordinate of the view.
3	**android:layout_y** This specifies the y-coordinate of the view.

Public Constructors

AbsoluteLayout(Context context)
AbsoluteLayout(Context context, AttributeSet attrs)
AbsoluteLayout(Context context, AttributeSet attrs, int defStyleAttr)

AbsoluteLayout(Context context, AttributeSet attrs, int defStyleAttr, int defStyleRes)

Example

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

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *demo* under a package *com.example.demo* as explained in the *Hello World Example* chapter.
2	Modify the default content of *res/layout/activity_main.xml* file to include few widgets in absolute layout.
3	No need to modify string.xml, Android studio takes care of default constants
4	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **src/com.example.demo/MainActivity.java**. This file can include each of the fundamental lifecycle methods.

```
package com.example.demo;

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

public class MainActivity extends Activity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }

}
```

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

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

   <Button
      android:layout_width="100dp"
      android:layout_height="wrap_content"
      android:text="OK"
      android:layout_x="50px"
```

```
      android:layout_y="361px" />
   <Button
      android:layout_width="100dp"
      android:layout_height="wrap_content"
      android:text="Cancel"
      android:layout_x="225px"
      android:layout_y="361px" />

</AbsoluteLayout>
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">demo</string>
   <string name="action_settings">Settings</string>
</resources>
```

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