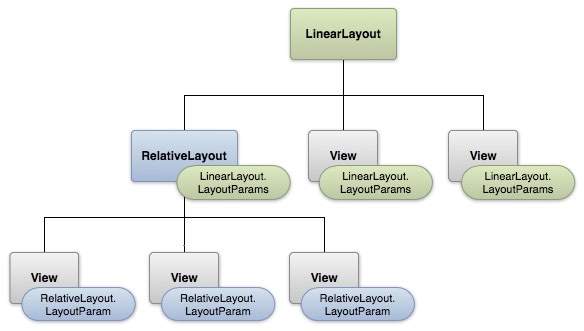
# 4. Android - UI Layouts

The basic building block for user interface is a **View** object which is created from the View class and occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for widgets, which are used to create interactive UI components like buttons, text fields, etc.

The **ViewGroup** is a subclass of **View** and provides invisible container that hold other Views or other ViewGroups and define their layout properties.

At third level we have different layouts which are subclasses of ViewGroup class and a typical layout defines the visual structure for an Android user interface and can be created either at run time using **View/ViewGroup**objects or you can declare your layout using simple XML file **main\_layout.xml** which is located in the res/layout folder of your project.



#### LAYOUT PARAMS

A layout may contain any type of widgets such as buttons, labels, textboxes, and so on. Following is a simple example of XML file having LinearLayout −

<?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" >

<TextView android:id="@+id/text"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="This is a TextView" />

<Button android:id="@+id/button"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="This is a Button" />

<!-- More GUI components go here -->

</LinearLayout>

Once your layout has created, you can load the layout resource from your application code, in your *Activity.onCreate()* callback implementation as shown below −

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

## Android Layout Types

There are number of Layouts provided by Android which you will use in almost all the Android applications to provide different view, look and feel.

|  |  |
| --- | --- |
| **Sr.No** | **Layout & Description** |
| 1 | [**Linear Layout**](https://www.tutorialspoint.com/android/android_linear_layout.htm)  LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally. |
| 2 | [**Relative Layout**](https://www.tutorialspoint.com/android/android_relative_layout.htm)  RelativeLayout is a view group that displays child views in relative positions. |
| 3 | [**Table Layout**](https://www.tutorialspoint.com/android/android_table_layout.htm)  TableLayout is a view that groups views into rows and columns. |
| 4 | [**Absolute Layout**](https://www.tutorialspoint.com/android/android_absolute_layout.htm)  AbsoluteLayout enables you to specify the exact location of its children. |
| 5 | [**Frame Layout**](https://www.tutorialspoint.com/android/android_frame_layout.htm)  The FrameLayout is a placeholder on screen that you can use to display a single view. |
| 6 | [**List View**](https://www.tutorialspoint.com/android/android_list_view.htm)  ListView is a view group that displays a list of scrollable items. |
| 7 | [**Grid View**](https://www.tutorialspoint.com/android/android_grid_view.htm)  GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid. |

## Layout Attributes

Each layout has a set of attributes which define the visual properties of that layout. There are few common attributes among all the layouts and their are other attributes which are specific to that layout. Following are common attributes and will be applied to all the layouts:

|  |  |
| --- | --- |
| **Sr.No** | **Attribute & Description** |
| 1 | **android:id**  This is the ID which uniquely identifies the view. |
| 2 | **android:layout\_width**  This is the width of the layout. |
| 3 | **android:layout\_height**  This is the height of the layout |
| 4 | **android:layout\_marginTop**  This is the extra space on the top side of the layout. |
| 5 | **android:layout\_marginBottom**  This is the extra space on the bottom side of the layout. |
| 6 | **android:layout\_marginLeft**  This is the extra space on the left side of the layout. |
| 7 | **android:layout\_marginRight**  This is the extra space on the right side of the layout. |
| 8 | **android:layout\_gravity**  This specifies how child Views are positioned. |
| 9 | **android:layout\_weight**  This specifies how much of the extra space in the layout should be allocated to the View. |
| 10 | **android:layout\_x**  This specifies the x-coordinate of the layout. |
| 11 | **android:layout\_y**  This specifies the y-coordinate of the layout. |
| 12 | **android:layout\_width**  This is the width of the layout. |
| 13 | **android:layout\_width**  This is the width of the layout. |
| 14 | **android:paddingLeft**  This is the left padding filled for the layout. |
| 15 | **android:paddingRight**  This is the right padding filled for the layout. |
| 16 | **android:paddingTop**  This is the top padding filled for the layout. |
| 17 | **android:paddingBottom**  This is the bottom padding filled for the layout. |

Here width and height are the dimension of the layout/view which can be specified in terms of dp (Density-independent Pixels), sp ( Scale-independent Pixels), pt ( Points which is 1/72 of an inch), px( Pixels), mm ( Millimeters) and finally in (inches).

You can specify width and height with exact measurements but more often, you will use one of these constants to set the width or height −

* **android:layout\_width=wrap\_content** tells your view to size itself to the dimensions required by its content.
* **android:layout\_width=fill\_parent** tells your view to become as big as its parent view.

Gravity attribute plays important role in positioning the view object and it can take one or more (separated by '|') of the following constant values.

|  |  |  |
| --- | --- | --- |
| **Constant** | **Value** | **Description** |
| top | 0x30 | Push object to the top of its container, not changing its size. |
| bottom | 0x50 | Push object to the bottom of its container, not changing its size. |
| left | 0x03 | Push object to the left of its container, not changing its size. |
| right | 0x05 | Push object to the right of its container, not changing its size. |
| center\_vertical | 0x10 | Place object in the vertical center of its container, not changing its size. |
| fill\_vertical | 0x70 | Grow the vertical size of the object if needed so it completely fills its container. |
| center\_horizontal | 0x01 | Place object in the horizontal center of its container, not changing its size. |
| fill\_horizontal | 0x07 | Grow the horizontal size of the object if needed so it completely fills its container. |
| center | 0x11 | Place the object in the center of its container in both the vertical and horizontal axis, not changing its size. |
| fill | 0x77 | Grow the horizontal and vertical size of the object if needed so it completely fills its container. |
| clip\_vertical | 0x80 | Additional option that can be set to have the top and/or bottom edges of the child clipped to its container's bounds. The clip will be based on the vertical gravity: a top gravity will clip the bottom edge, a bottom gravity will clip the top edge, and neither will clip both edges. |
| clip\_horizontal | 0x08 | Additional option that can be set to have the left and/or right edges of the child clipped to its container's bounds. The clip will be based on the horizontal gravity: a left gravity will clip the right edge, a right gravity will clip the left edge, and neither will clip both edges. |
| start | 0x00800003 | Push object to the beginning of its container, not changing its size. |
| end | 0x00800005 | Push object to the end of its container, not changing its size. |

## View Identification

A view object may have a unique ID assigned to it which will identify the View uniquely within the tree. The syntax for an ID, inside an XML tag is −

android:id="@+id/my\_button"

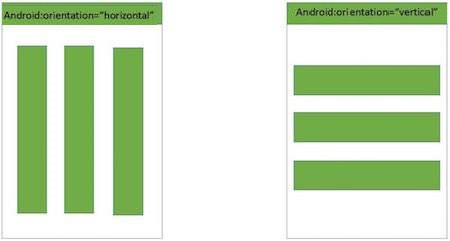
Following is a brief description of @ and + signs −

* The at-symbol (@) at the beginning of the string indicates that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource.
* The plus-symbol (+) means that this is a new resource name that must be created and added to our resources. To create an instance of the view object and capture it from the layout, use the following −

Button B1 = (Button) findViewById(R.id.my\_button);

# Android 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 Eclipse Run Icon 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 −

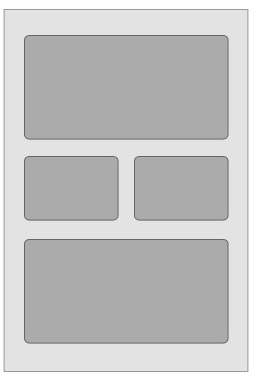


Now let's change the orientation of Layout as **android:orientation="horizontal"** and try to run the same application, it will give following screen −



# Android 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. |

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 −

|  |  |
| --- | --- |
| **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". |

## 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 −

|  |  |
| --- | --- |
| **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. |

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>

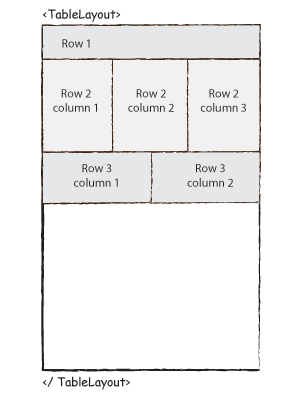
Let's try to run our modified **Hello World!** application we just modified. I assume you had created your **AVD** while doing environment setup. To run the app from Android Studio, open one of your project's activity files and click Run Eclipse Run Icon icon from the toolbar. Android Studio installs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



# Android 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 Eclipse Run Icon 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 −



# Android 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.



## 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 Eclipse Run Icon 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 −



# Android 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.



## 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"?>

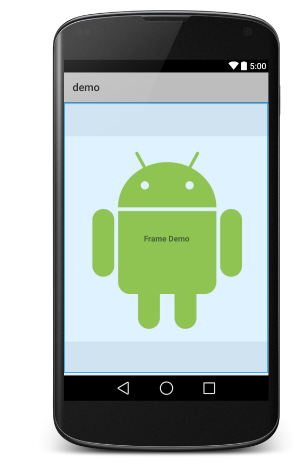
<resources>

<string name="app\_name">demo</string>

<string name="action\_settings">Settings</string>

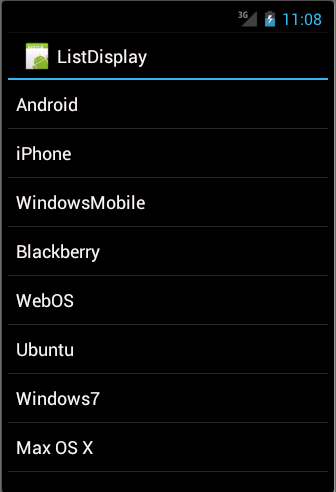
</resources>

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 Eclipse Run Icon 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 −



# Android List View

Android **ListView** is a view which groups several items and display them in vertical scrollable list. The list items are automatically inserted to the list using an **Adapter** that pulls content from a source such as an array or database.



#### LIST VIEW

An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter holds the data and send the data to adapter view, the view can takes the data from adapter view and shows the data on different views like as spinner, list view, grid view etc.

The **ListView** and **GridView** are subclasses of **AdapterView** and they can be populated by binding them to an **Adapter**, which retrieves data from an external source and creates a View that represents each data entry.

Android provides several subclasses of Adapter that are useful for retrieving different kinds of data and building views for an AdapterView ( i.e. ListView or GridView). The common adapters are **ArrayAdapter**,**Base Adapter**,**CursorAdapter**,**SimpleCursorAdapter**,**SpinnerAdapter** and **WrapperListAdapter**. We will see separate examples for both the adapters.

## ListView Attributes

Following are the important attributes specific to GridView −

|  |  |
| --- | --- |
| **Sr.No** | **Attribute & Description** |
| 1 | **android:id**  This is the ID which uniquely identifies the layout. |
| 2 | **android:divider**  This is drawable or color to draw between list items. |
| 3 | **android:dividerHeight**  This specifies height of the divider. This could be in px, dp, sp, in, or mm. |
| 4 | **android:entries**  Specifies the reference to an array resource that will populate the ListView. |
| 5 | **android:footerDividersEnabled**  When set to false, the ListView will not draw the divider before each footer view. The default value is true. |
| 6 | **android:headerDividersEnabled**  When set to false, the ListView will not draw the divider after each header view. The default value is true. |

## ArrayAdapter

You can use this adapter when your data source is an array. By default, ArrayAdapter creates a view for each array item by calling toString() on each item and placing the contents in a **TextView**. Consider you have an array of strings you want to display in a ListView, initialize a new **ArrayAdapter** using a constructor to specify the layout for each string and the string array −

ArrayAdapter adapter = new ArrayAdapter<String>(this,R.layout.ListView,StringArray);

Here are arguments for this constructor −

* First argument **this** is the application context. Most of the case, keep it **this**.
* Second argument will be layout defined in XML file and having **TextView** for each string in the array.
* Final argument is an array of strings which will be populated in the text view.

Once you have array adapter created, then simply call **setAdapter()** on your **ListView** object as follows −

ListView listView = (ListView) findViewById(R.id.listview);

listView.setAdapter(adapter);

You will define your list view under res/layout directory in an XML file. For our example we are going to using activity\_main.xml file.

### Example

Following is the example which will take you through simple steps to show how to create your own Android application using ListView. 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 *ListDisplay* under a package *com.example.ListDisplay* as explained in the *Hello World Example* chapter. |
| 2 | Modify the default content of *res/layout/activity\_main.xml* file to include ListView content with the self explanatory attributes. |
| 3 | No need to change string.xml, Android studio takes care of default string constants. |
| 4 | Create a Text View file *res/layout/activity\_listview.xml*. This file will have setting to display all the list items. So you can customize its fonts, padding, color etc. using this file. |
| 6 | 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.ListDisplay/ListDisplay.java**. This file can include each of the fundamental life cycle methods.

package com.example.ListDisplay;

import android.os.Bundle;

import android.app.Activity;

import android.view.Menu;

import android.widget.ArrayAdapter;

import android.widget.ListView;

public class ListDisplay extends Activity {

// Array of strings...

String[] mobileArray = {"Android","IPhone","WindowsMobile","Blackberry",

"WebOS","Ubuntu","Windows7","Max OS X"};

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

ArrayAdapter adapter = new ArrayAdapter<String>(this,

R.layout.activity\_listview, mobileArray);

ListView listView = (ListView) findViewById(R.id.mobile\_list);

listView.setAdapter(adapter);

}

}

Following will be the content of **res/layout/activity\_main.xml** file −

<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"

tools:context=".ListActivity" >

<ListView

android:id="@+id/mobile\_list"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content" >

</ListView>

</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">ListDisplay</string>

<string name="action\_settings">Settings</string>

</resources>

Following will be the content of **res/layout/activity\_listview.xml** file −

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

<!-- Single List Item Design -->

<TextView xmlns:android="http://schemas.android.com/apk/res/android"

android:id="@+id/label"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

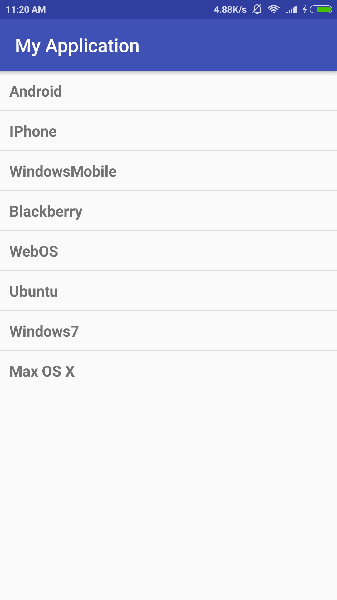
android:padding="10dip"

android:textSize="16dip"

android:textStyle="bold" >

</TextView>

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



## SimpleCursorAdapter

You can use this adapter when your data source is a database Cursor. When using *SimpleCursorAdapter*, you must specify a layout to use for each row in the **Cursor** and which columns in the Cursor should be inserted into which views of the layout.

For example, if you want to create a list of people's names and phone numbers, you can perform a query that returns a Cursor containing a row for each person and columns for the names and numbers. You then create a string array specifying which columns from the Cursor you want in the layout for each result and an integer array specifying the corresponding views that each column should be placed −

String[] fromColumns = {ContactsContract.Data.DISPLAY\_NAME,

ContactsContract.CommonDataKinds.Phone.NUMBER};

int[] toViews = {R.id.display\_name, R.id.phone\_number};

When you instantiate the SimpleCursorAdapter, pass the layout to use for each result, the Cursor containing the results, and these two arrays −

SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,

R.layout.person\_name\_and\_number, cursor, fromColumns, toViews, 0);

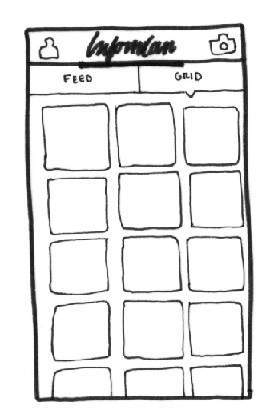
ListView listView = getListView();

listView.setAdapter(adapter);

The SimpleCursorAdapter then creates a view for each row in the Cursor using the provided layout by inserting each from Columns item into the corresponding **toViews** view.

# Android Grid View

Android **GridView** shows items in two-dimensional scrolling grid (rows & columns) and the grid items are not necessarily predetermined but they automatically inserted to the layout using a **ListAdapter**



#### GRID VIEW

An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter can be used to supply the data to like spinner, list view, grid view etc.

The **ListView** and **GridView** are subclasses of **AdapterView** and they can be populated by binding them to an **Adapter**, which retrieves data from an external source and creates a View that represents each data entry.

## GridView Attributes

Following are the important attributes specific to GridView −

|  |  |
| --- | --- |
| **Sr.No** | **Attribute & Description** |
| 1 | **android:id**  This is the ID which uniquely identifies the layout. |
| 2 | **android:columnWidth**  This specifies the fixed width for each column. This could be in px, dp, sp, in, or mm. |
| 3 | **android:gravity**  Specifies the gravity within each cell. Possible values are top, bottom, left, right, center, center\_vertical, center\_horizontal etc. |
| 4 | **android:horizontalSpacing**  Defines the default horizontal spacing between columns. This could be in px, dp, sp, in, or mm. |
| 5 | **android:numColumns**  Defines how many columns to show. May be an integer value, such as "100" or auto\_fit which means display as many columns as possible to fill the available space. |
| 6 | **android:stretchMode**  Defines how columns should stretch to fill the available empty space, if any. This must be either of the values −   * none − Stretching is disabled. * spacingWidth − The spacing between each column is stretched. * columnWidth − Each column is stretched equally. * spacingWidthUniform − The spacing between each column is uniformly stretched.. |
| 7 | **android:verticalSpacing**  Defines the default vertical spacing between rows. This could be in px, dp, sp, in, or mm. |

## Example

This example will take you through simple steps to show how to create your own Android application using GridView. 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 *HelloWorld* under a package *com.example.helloworld* as explained in the *Hello World Example* chapter. |
| 2 | Modify the detault content of *res/layout/activity\_main.xml* file to include GridView content with the self explanatory attributes. |
| 3 | No need to change string.xml, Android studio takes care of defaults strings which are placed at string.xml |
| 4 | Let's put few pictures in *res/drawable-hdpi* folder. I have put sample0.jpg, sample1.jpg, sample2.jpg, sample3.jpg, sample4.jpg, sample5.jpg, sample6.jpg and sample7.jpg. |
| 5 | Create a new class called **ImageAdapter** under a package com.example.helloworld that extends BaseAdapter. This class will implement functionality of an adapter to be used to fill the view. |
| 6 | Run the application to launch Android emulator and verify the result of the changes done in the application. |

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

package com.example.helloworld;

import android.os.Bundle;

import android.app.Activity;

import android.view.Menu;

import android.widget.GridView;

public class MainActivity extends Activity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

GridView gridview = (GridView) findViewById(R.id.gridview);

gridview.setAdapter(new ImageAdapter(this));

}

}

Following will be the content of **res/layout/activity\_main.xml** file −

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

<GridView xmlns:android="http://schemas.android.com/apk/res/android"

android:id="@+id/gridview"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:columnWidth="90dp"

android:numColumns="auto\_fit"

android:verticalSpacing="10dp"

android:horizontalSpacing="10dp"

android:stretchMode="columnWidth"

android:gravity="center"

/>

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>

Following will be the content of **src/com.example.helloworld/ImageAdapter.java** file −

package com.example.helloworld;

import android.content.Context;

import android.view.View;

import android.view.ViewGroup;

import android.widget.BaseAdapter;

import android.widget.GridView;

import android.widget.ImageView;

public class ImageAdapter extends BaseAdapter {

private Context mContext;

// Constructor

public ImageAdapter(Context c) {

mContext = c;

}

public int getCount() {

return mThumbIds.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;

if (convertView == null) {

imageView = new ImageView(mContext);

imageView.setLayoutParams(new GridView.LayoutParams(85, 85));

imageView.setScaleType(ImageView.ScaleType.CENTER\_CROP);

imageView.setPadding(8, 8, 8, 8);

}

else

{

imageView = (ImageView) convertView;

}

imageView.setImageResource(mThumbIds[position]);

return imageView;

}

// Keep all Images in array

public Integer[] mThumbIds = {

R.drawable.sample\_2, R.drawable.sample\_3,

R.drawable.sample\_4, R.drawable.sample\_5,

R.drawable.sample\_6, R.drawable.sample\_7,

R.drawable.sample\_0, R.drawable.sample\_1,

R.drawable.sample\_2, R.drawable.sample\_3,

R.drawable.sample\_4, R.drawable.sample\_5,

R.drawable.sample\_6, R.drawable.sample\_7,

R.drawable.sample\_0, R.drawable.sample\_1,

R.drawable.sample\_2, R.drawable.sample\_3,

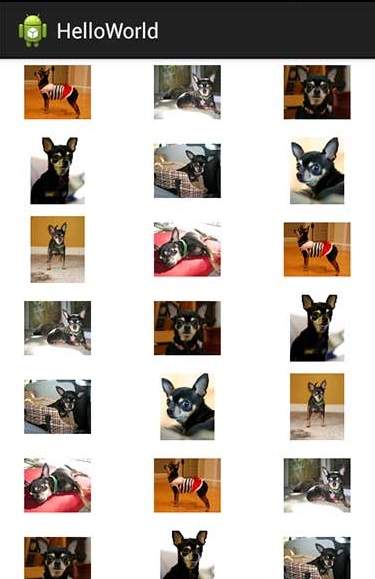
R.drawable.sample\_4, R.drawable.sample\_5,

R.drawable.sample\_6, R.drawable.sample\_7

};

}

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 Eclipse Run Icon 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 −



## Sub-Activity Example

Let's extend the functionality of above example where we will show selected grid image in full screen. To achieve this we need to introduce a new activity. Just keep in mind for any activity we need perform all the steps like we have to implement an activity class, define that activity in AndroidManifest.xml file, define related layout and finally link that sub-activity with the main activity by it in the main activity class. So let's follow the steps to modify above example −

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | You will use Android studio IDE to create an Android application and name it as *HelloWorld* under a package *com.example.helloworld* as explained in the *Hello World Example* chapter. |
| 2 | Create a new Activity class as *SingleViewActivity.java* under a package *com.example.helloworld* as shown below. |
| 3 | Create new layout file for the new activity under **res/layout/** folder. Let's name this XML file as single\_view.xml. |
| 4 | Define your new activity in *AndroidManifest.xml* file using <activity.../> tag. An application can have one or more activities without any restrictions. |
| 5 | Run the application to launch Android emulator and verify the result of the changes done in the application. |

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

package com.example.helloworld;

import android.app.Activity;

import android.content.Intent;

import android.os.Bundle;

import android.view.Menu;

import android.view.View;

import android.widget.AdapterView;

import android.widget.AdapterView.OnItemClickListener;

import android.widget.GridView;

public class MainActivity extends Activity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

GridView gridview = (GridView) findViewById(R.id.gridview);

gridview.setAdapter(new ImageAdapter(this));

gridview.setOnItemClickListener(new OnItemClickListener() {

public void onItemClick(AdapterView<?> parent,

View v, int position, long id){

// Send intent to SingleViewActivity

Intent i = new Intent(getApplicationContext(), SingleViewActivity.class);

// Pass image index

i.putExtra("id", position);

startActivity(i);

}

});

}

}

Following will be the content of new activity file **src/com.example.helloworld/SingleViewActivity.java** file −

package com.example.helloworld;

import android.app.Activity;

import android.content.Intent;

import android.os.Bundle;

import android.widget.ImageView;

public class SingleViewActivity extends Activity {

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.single\_view);

// Get intent data

Intent i = getIntent();

// Selected image id

int position = i.getExtras().getInt("id");

ImageAdapter imageAdapter = new ImageAdapter(this);

ImageView imageView = (ImageView) findViewById(R.id.SingleView);

imageView.setImageResource(imageAdapter.mThumbIds[position]);

}

}

Following will be the content of **res/layout/single\_view.xml** file −

<?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" >

<ImageView android:id="@+id/SingleView"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"/>

</LinearLayout>

Following will be the content of **AndroidManifest.xml** to define two new constants −

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

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.example.helloworld">

<application

android:allowBackup="true"

android:icon="@drawable/ic\_launcher"

android:label="@string/app\_name"

android:theme="@style/AppTheme" >

<activity

android:name="com.example.helloworld.MainActivity"

android:label="@string/app\_name" >

<intent-filter>

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

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

</intent-filter>

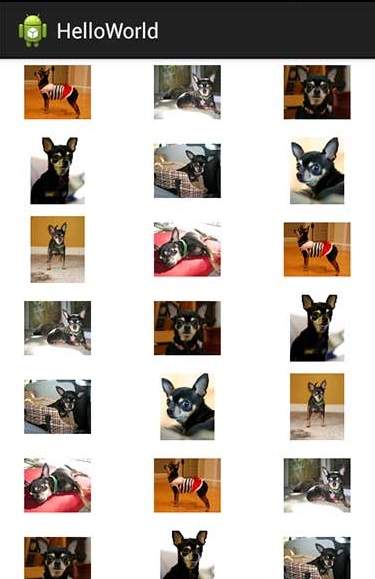
</activity>

<activity android:name=".SingleViewActivity"></activity>

</application>

</manifest>

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 Eclipse Run Icon 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 if you click on either of the images it will be displayed as a single image, for example−



Kindly note above mentioned images have been taken from Android official website.