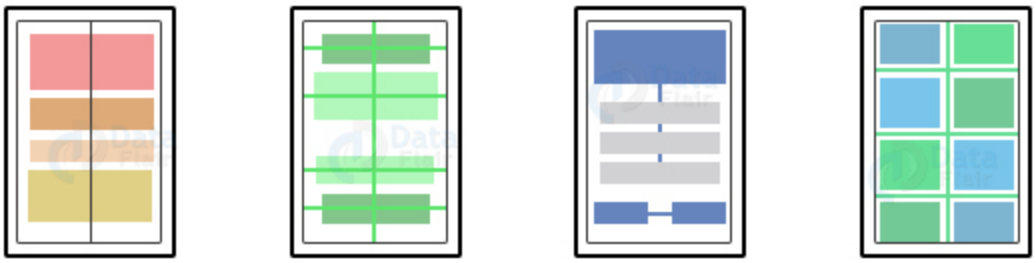
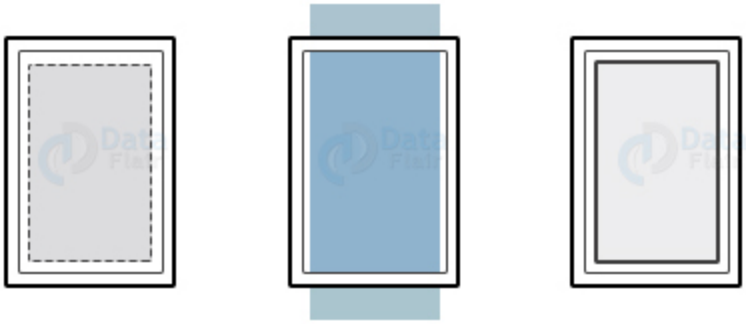
Types of Android Layouts



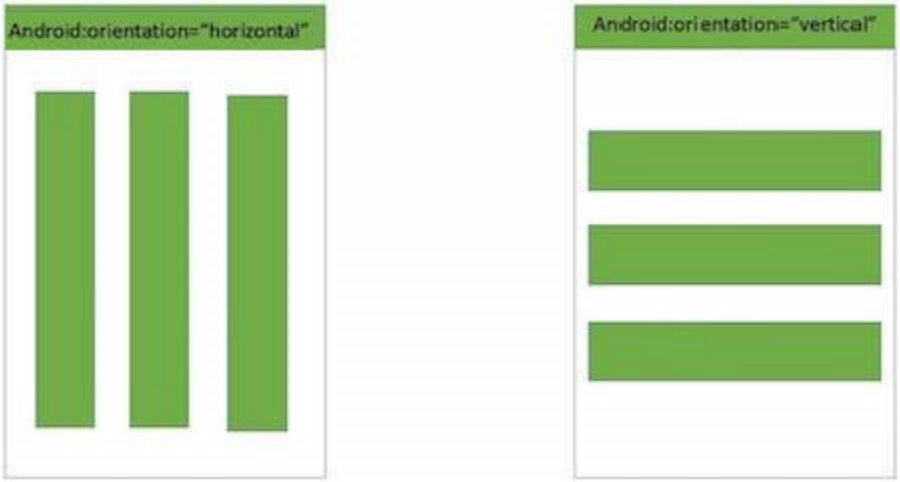
StackLayout AbsoluteLayout RelativeLayout GridLayout



ContentView ScrollView Frame

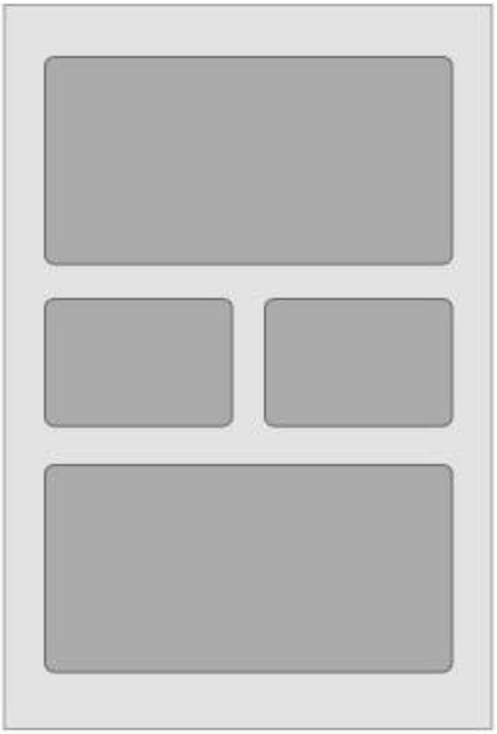
Android Linear Layout

Android LinearLayout is a view group that aligns all child ren in either vertically or horizontally.



```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" or “horizontal” >
<TextView
android:text= "Hijau"
android:gravity="center_horizontal" ( if horizontal )
android:background="#00aa00"
android:layout_width= "fill_parent" "wrap_content"
android:layout_height= "wrap_content" "fill_parent"
android:layout_weight="1"/>
<TextView
android:text= "Merah"
android:gravity="center_horizontal" ( if horizontal )
android:background="#aa0000"
android:layout_width= "fill_parent" "wrap_content"
android:layout_height= "wrap_content" "fill_parent"
android:layout_weight="1"/>
<TextView
android:text= "Biru"
android:gravity="center_horizontal" ( if horizontal )
android:background="#0000aa"
android:layout_width= "fill_parent" "wrap_content"
android:layout_height= "wrap_content" "fill_parent"
android:layout_weight="1"/>
</LinearLayout>
```

Android Relative Layout

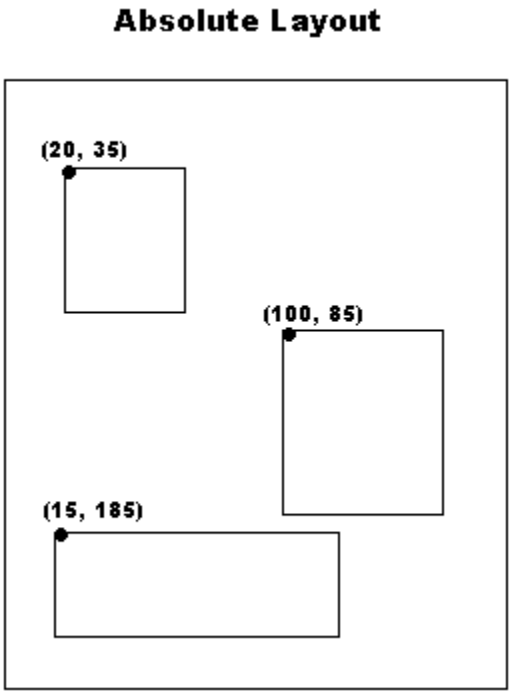


Android RelativeLayout enables you to specify how child views are positioned relative to each other. The position of each view can be specified as relative to sibling elements or relative to the parent.

```xml
<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>
```

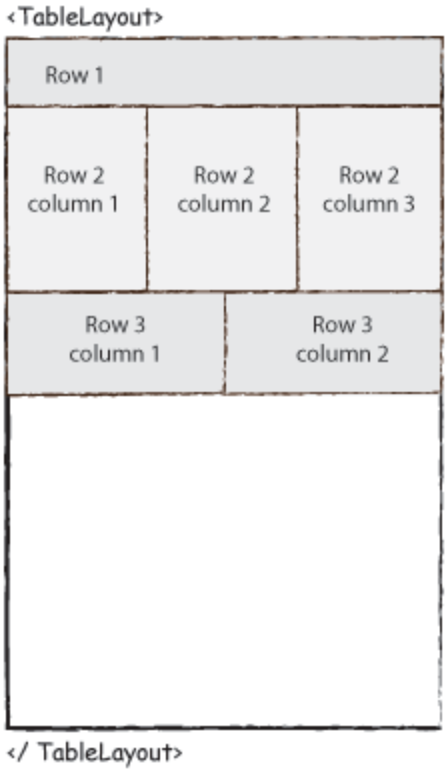
Android Absolute Layout

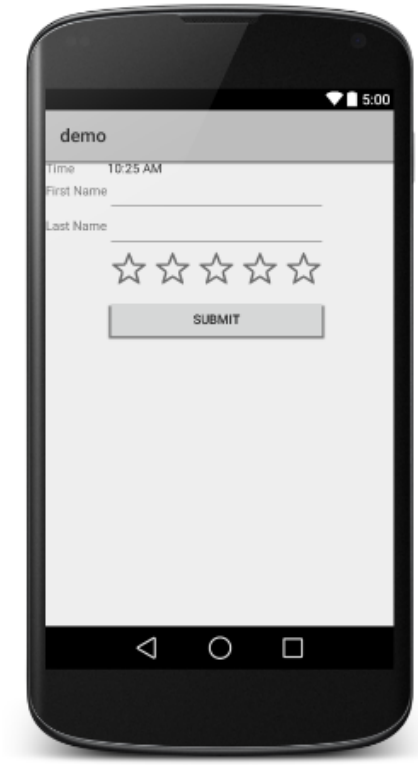
An Absolute Layout lets you specify exact locations (x/y coordinates) of its children. Absolute layouts are less flexible and harder to maintain than other types of layouts without absolute positioning



```
<AbsoluteLayout 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>
```

Android Table Layout





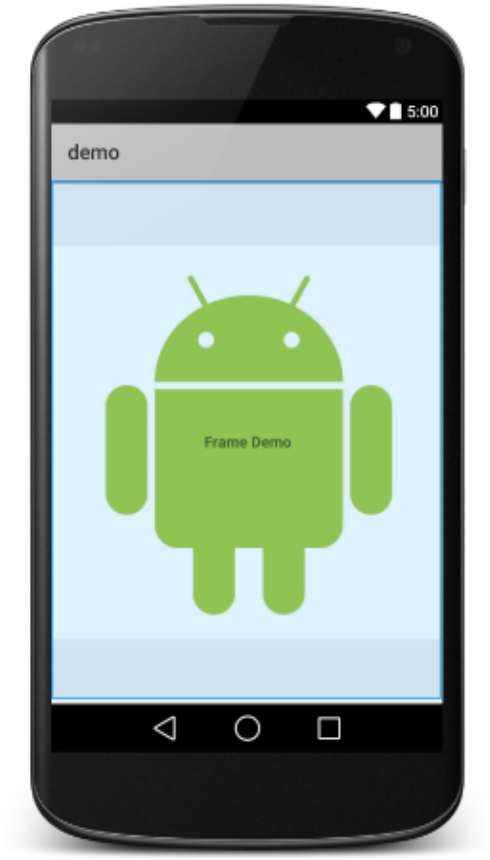
Android TableLayout going to be arranged groups of views into rows and columns. You will use the <TableRow> element to build a row in the table. Each row has zero or more cells; each cell can hold one View object.

```java
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);
  }
```

```xml
<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>
```

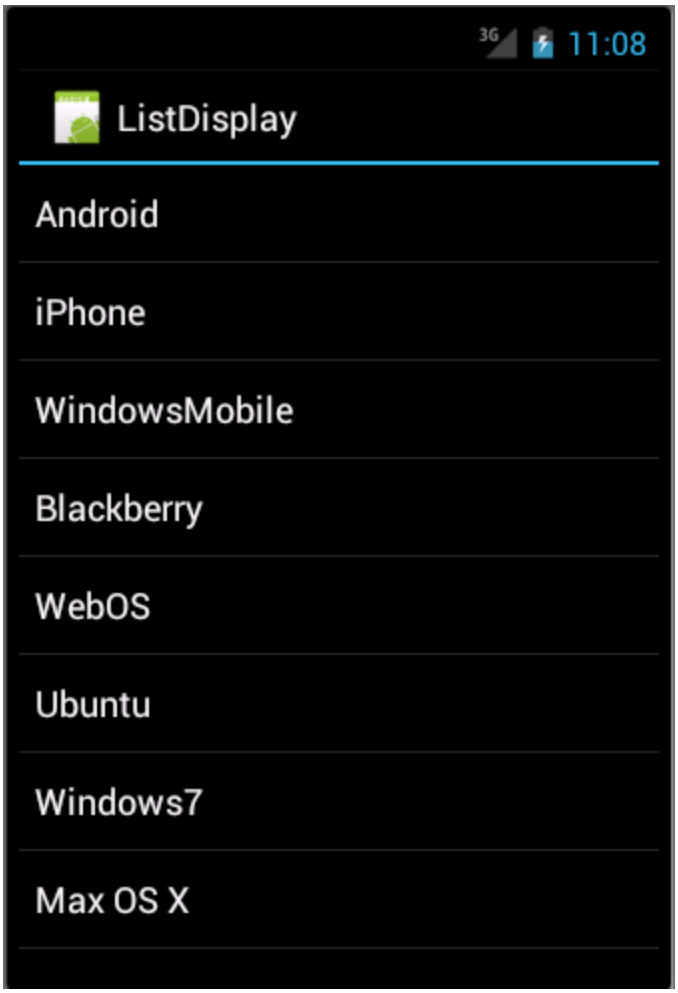
Android Frame Layout

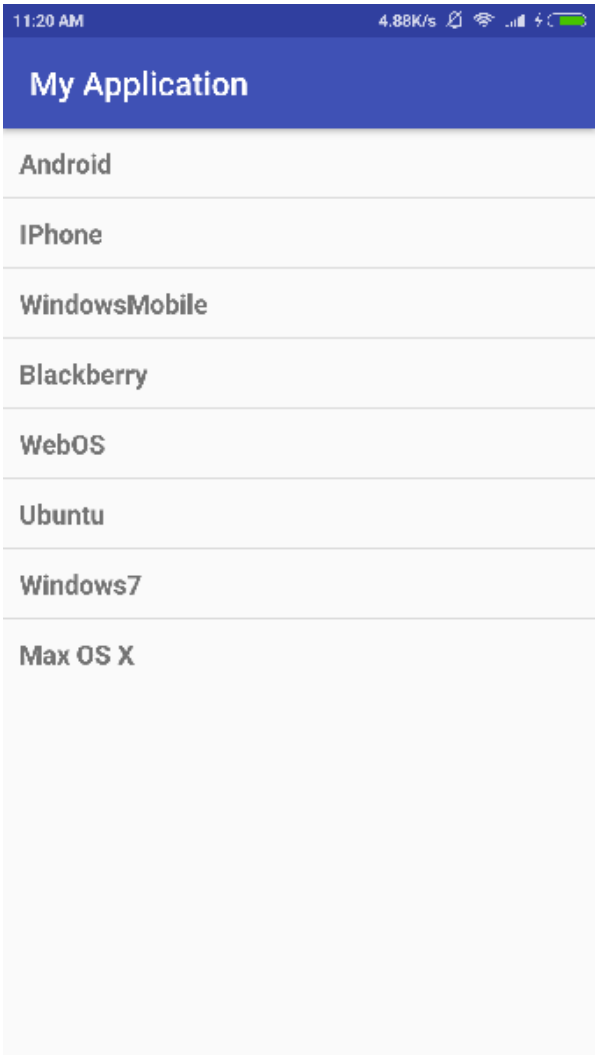


Frame Layout is designed to block out an area on the screen to display a single item. Generally, FrameLayout should be used to hold a single child view, because it can be difficult to organize child views in a way that's scalable to different screen sizes without the children overlapping each other.

```xml
<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>
```

Android List View



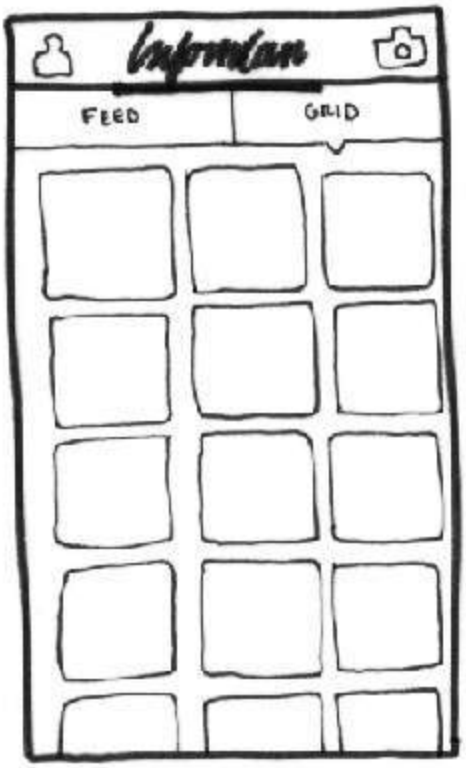


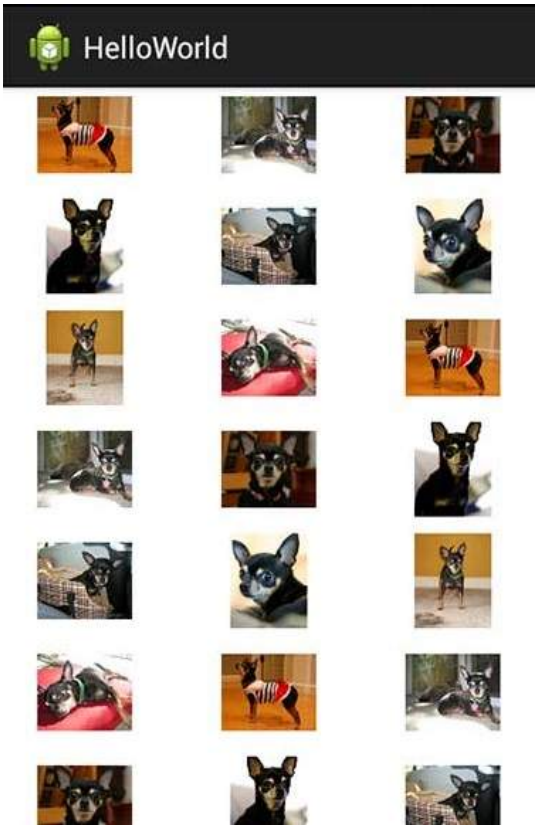
Android ListView is a view which groups several items and display them in vertical scrollable list. The list items are automatically inserted to the list using an Adapter that pulls content from a source such as an array or database.

```java
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);
  }
}
```

```xml
<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>
```

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.

src/com.example.helloworld/MainActivity.java.

```java
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));
  }
}
```

res/layout/activity_main.xml

```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="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"
/>
```

src/com.example.helloworld/ImageAdapter.java

```
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 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"
```

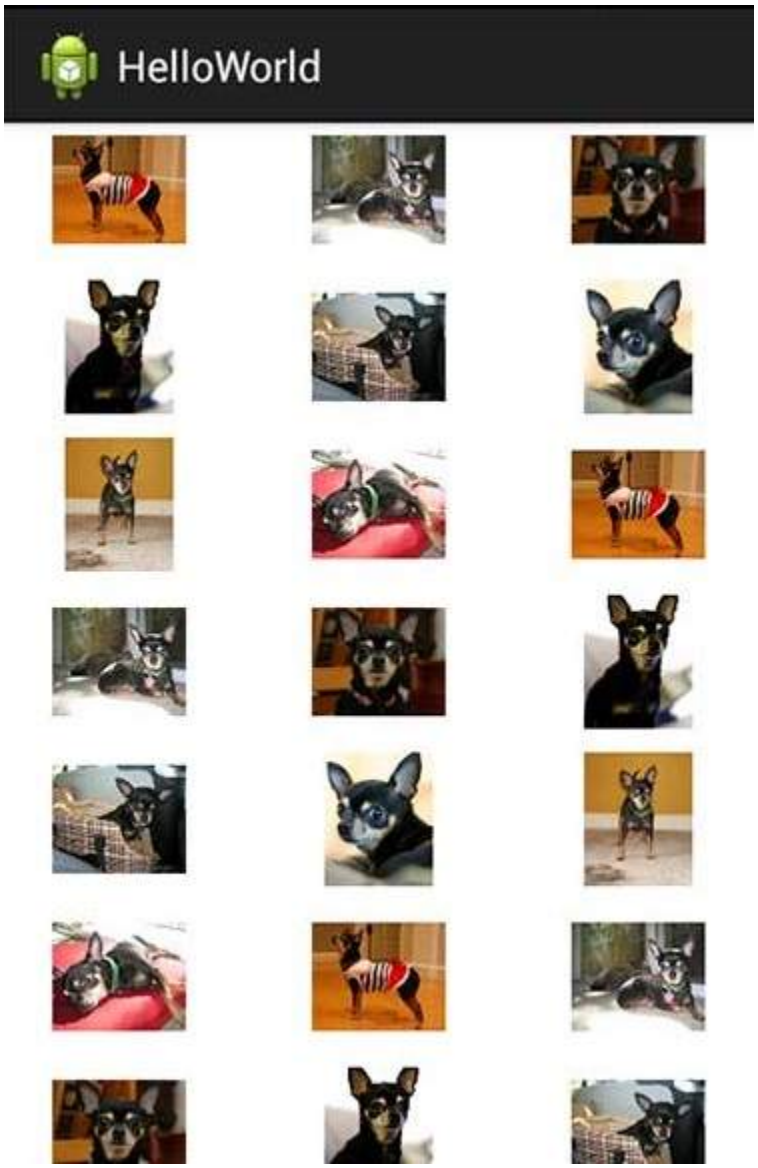
```
  android:orientation="vertical" >

<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 -

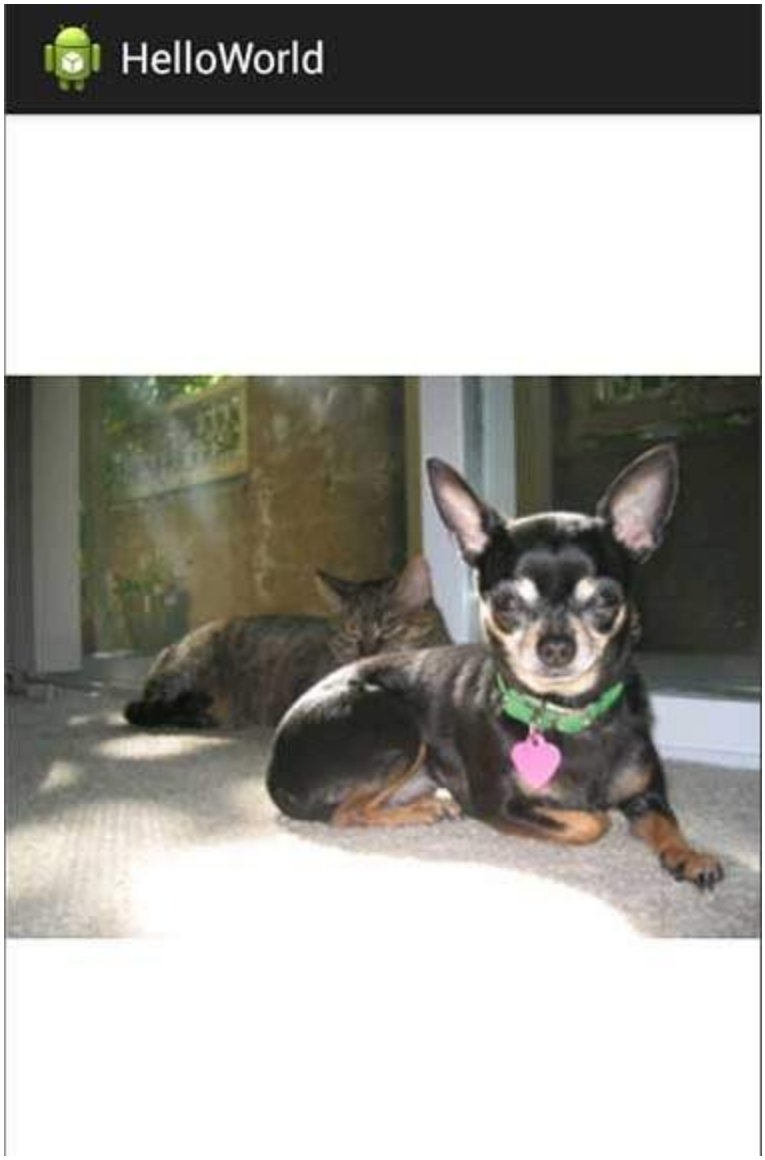
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.helloworld">

  <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

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.