Chapter 5:
Building UI with Basic Views & ViewGroups

Outline

- Understanding viewgroups
- Using basic views to design UI
- ViewGroups to use to lay out views
- Adapt and manage changes in screen orientation

Introduction

- In Chapter 3, you learned about the Activity class and its life cycle. You learned that an activity is a means by which users interact with the application.
- An *activity* displays the user interface of your application, which may contain widgets such as buttons, labels, textboxes, and so on. Typically, you define your UI using an XML file (e.g., the main.xml file located in the res/layout folder of your project)
- However, an activity by itself does not have a presence on the screen. Instead, it has to draw the screen using *Views* and *ViewGroups.*

Introduction

- During compilation, each element in the XML fi le is compiled into its equivalent Android GUI class, with attributes represented by methods. The Android system then creates the UI of the activity when it is loaded.
- During runtime, you load the XML UI in the **onCreate()** method handler in your Activity class, using the **setContentView()** method of the Activity class

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
}
```

Introduction

- To declare app layout,

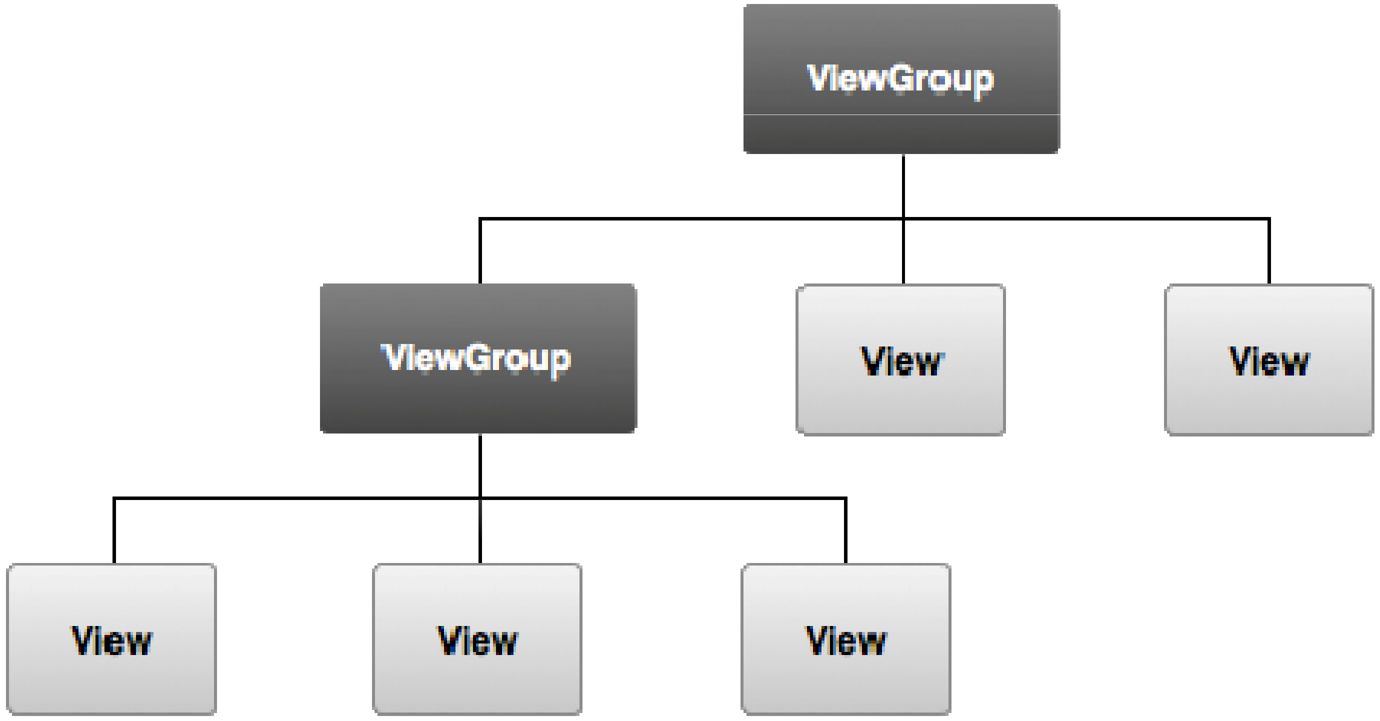
1. Define *layout* with an XML file (e.g., the main.xml file located in the **res/layout** folder of project)
2. Instantiate View objects **dynamically** during runtime entirely using Java code, and

View and ViewGroup

- Android provides a collection of both View and ViewGroup subclasses that offer common input controls (such as buttons and text fields) and various layout models (such as a linear or relative layout).
- **View**:
  - an object that draws something on the screen that the user can interact with.
  - UI widgets such as buttons, labels, and textboxes or text fields
- **ViewGroup**:
  - invisible view containers that *define how the child views are laid out*, such as in a grid or a vertical list.
  - an object that holds other View (and ViewGroup) objects in order to define the layout of the interface.

View and ViewGroup

- The graphical user interface for an Android app is built using a hierarchy of **View** and **ViewGroup** objects.
- View is derived from the base class **android.view.View**



Layout

- A layout defines the visual structure for a user interface, such as the UI for an activity or app widget.
- You can declare a layout in two ways:
  - **Declare UI elements in XML**. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
  - **Instantiate layout elements at runtime**. Your application can create View and ViewGroup objects (and manipulate their properties) programmatically. (Chapter 6)

ViewGroup

- View group is the base class for layouts and views containers
- Derived from the base class **android.view.ViewGroup**
  - LinearLayout
  - AbsoluteLayout
  - TableLayout
  - RelativeLayout
  - FrameLayout
  - ScrollView
- In practice it is common to combine different types of layouts to create the UI desired.

Write the XML

- Open the **main.xml** file from the **res/layout/** directory

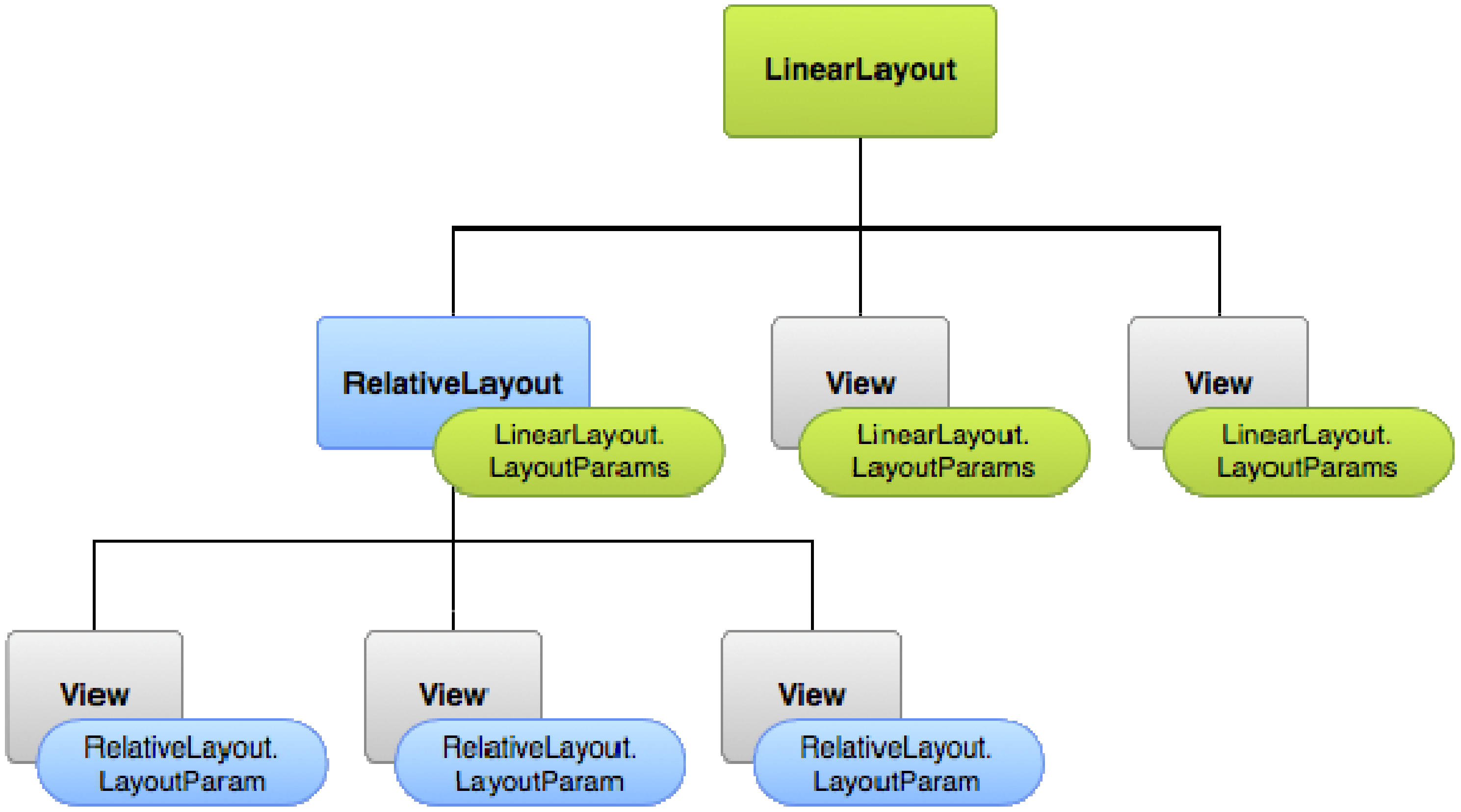
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

Load the XML Resource

- When you compile an application, each XML layout file is compiled into a View resource. You load the layout resource from your application code, in your Activity.onCreate() callback implementation.
- Do so by calling setContentView(), passing it the reference to your layout resource in the form of: R.layout.*layout_file_name*

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

Layout Parameters


LinearLayout
RelativeLayout
LinearLayout. LayoutParams
View
LinearLayout. LayoutParams
View
LinearLayout. LayoutParams
View
RelativeLayout. LayoutParam
View
RelativeLayout. LayoutParam
View
RelativeLayout. LayoutParam

Layout Parameters

- Every ViewGroup class implements a nested class that extends ViewGroup.LayoutParams. This subclass contains *property types* that define the *size* and *position* for each child view, as appropriate for the view group.
- Each child element must define LayoutParams that are appropriate for its parent, though it may also define different LayoutParams for its own children.

ID

- An XML attribute common to all View objects (defined by the View class)
- Uniquely identify the View within the tree.
- When the application is compiled, this ID is referenced as an integer, but the ID is typically assigned in the layout XML file as a string, in the id attribute.

ID

- 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 (in the R.java file).

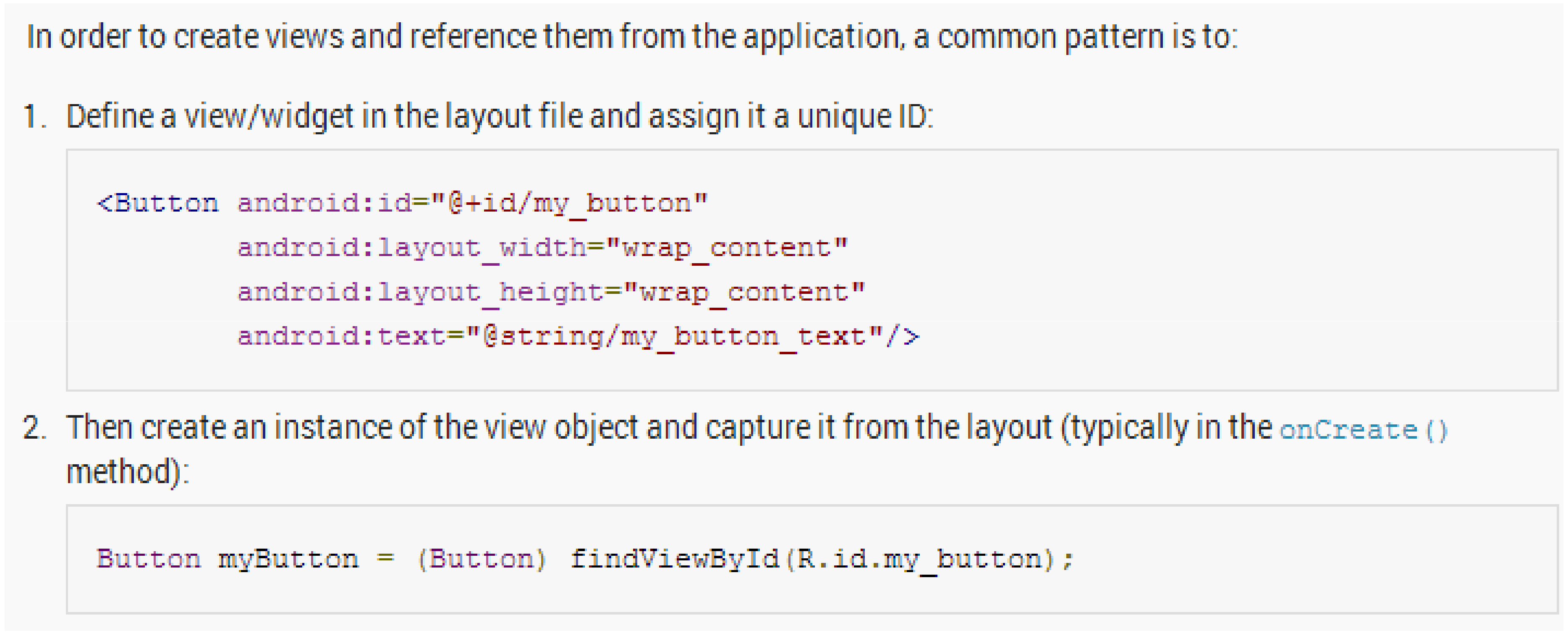
```
android:id="@+id/my_button"
```

- When referencing an Android resource ID, must add the android package namespace

```
android:id="@android:id/empty"
```

- Defining IDs for view objects is important when creating a RelativeLayout. In a relative layout, sibling views can define their layout relative to another sibling view, which is referenced by the unique ID.

ID



In order to create views and reference them from the application, a common pattern is to:

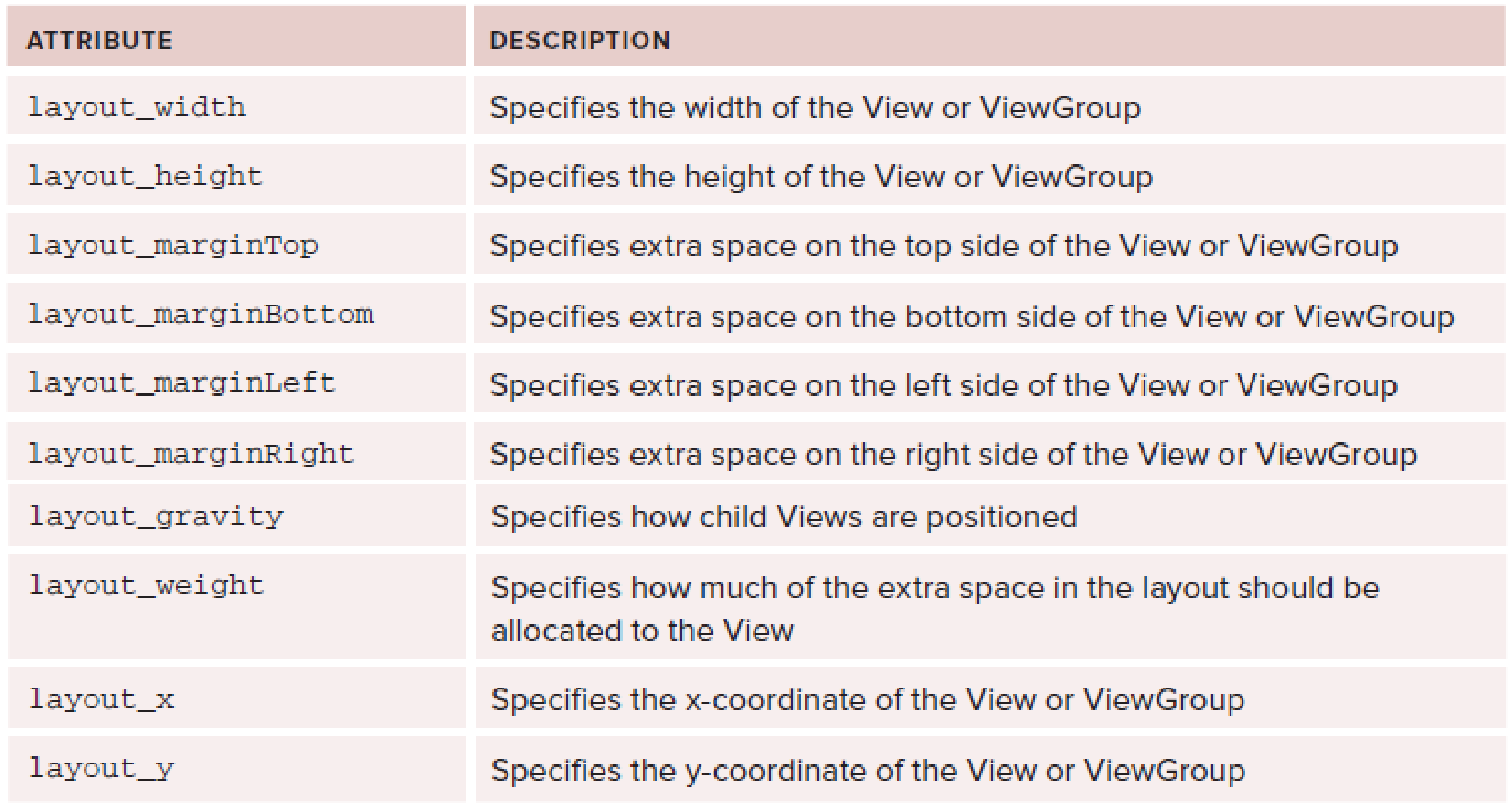
1. Define a view/widget in the layout file and assign it a unique ID:

```
<Button android:id="@+id/my_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/my_button_text"/>
```

2. Then create an instance of the view object and capture it from the layout (typically in the `onCreate()` method):

```
Button myButton = (Button) findViewById(R.id.my_button);
```

Attributes Used in Views and ViewGroups

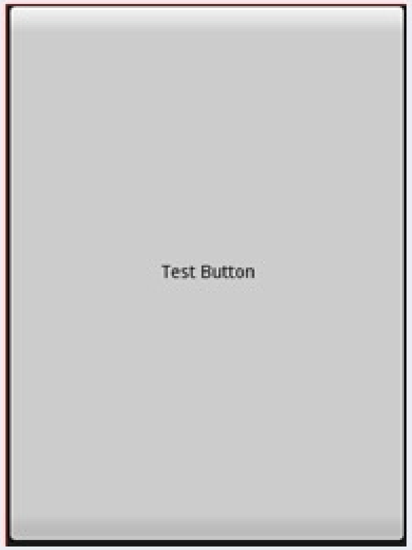
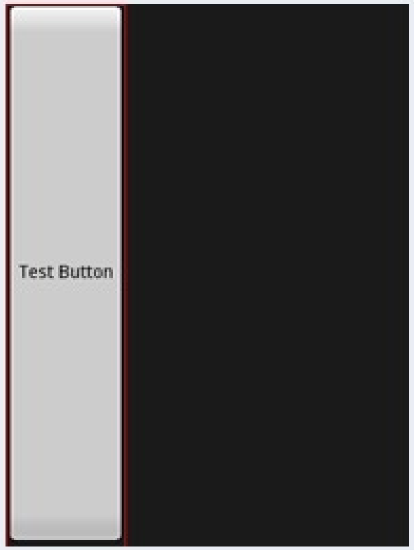
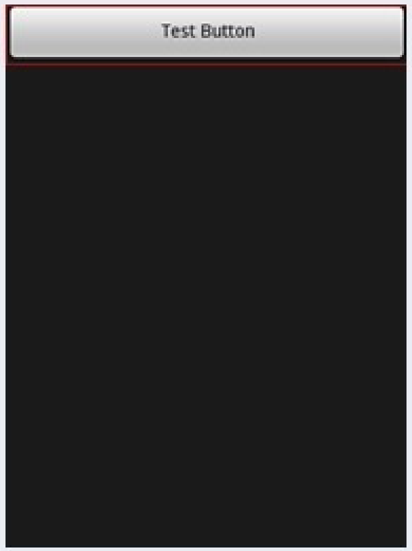
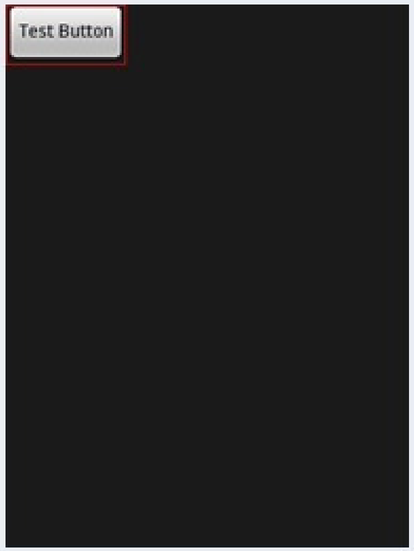


ATTRIBUTE	DESCRIPTION
`layout_width`	Specifies the width of the View or ViewGroup
`layout_height`	Specifies the height of the View or ViewGroup
`layout_marginTop`	Specifies extra space on the top side of the View or ViewGroup
`layout_marginBottom`	Specifies extra space on the bottom side of the View or ViewGroup
`layout_marginLeft`	Specifies extra space on the left side of the View or ViewGroup
`layout_marginRight`	Specifies extra space on the right side of the View or ViewGroup
`layout_gravity`	Specifies how child Views are positioned
`layout_weight`	Specifies how much of the extra space in the layout should be allocated to the View
`layout_x`	Specifies the x-coordinate of the View or ViewGroup
`layout_y`	Specifies the y-coordinate of the View or ViewGroup

- Some of the attributes are applicable only when a View is in a specific ViewGroup

layout_width & layout_height

- You can specify width and height with exact measurements, though you probably won't want to do this often.
- *In general, specifying a layout width and height using absolute units such as pixels is not recommended.*
- Instead, using relative measurements such as **density-independent pixel units (*dp*)**, ***wrap_content***, or ***fill_parent/match_parent*** because it helps ensure that your application will display properly across a variety of device screen sizes.
  - **wrap_content** tells your view to size itself to the dimensions required by its content (plus padding)
  - **fill_parent** (renamed *match_parent* in API Level 8) tells your view to become as big as its parent view group will allow (minus padding)

Height \ width	fill_parent	Wrap_content
fill_parent	 Test Button	 Test Button
Wrap_content	 Test Button	 Test Button

Layout position

- The geometry of a view is that of a **rectangle**.
- A view has a location, expressed as a pair of *left* and *top* coordinates, and two dimensions, expressed as a **width** and a **height**.
- The unit for location and dimensions is the **pixel**.
- Retrieve the location of a view by
  - getLeft, getTop, getRight, getBottom

Size

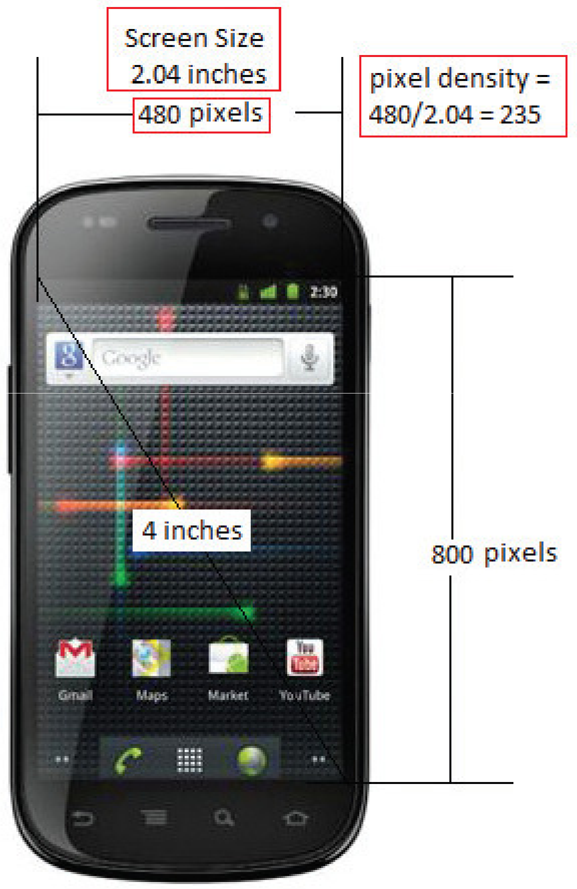
- *measured width* and *measured height* – define how big a view wants to be within its parent
  - getMeasuredWidth() & getMeasuredHeight()
- *width* & *height* (*drawing width* & *drawing height*) - define the actual size of the view on screen, at drawing time and after layout.
  - These values may, but do not have to, be different from the measured width and height.
  - getWidth() and getHeight()

Padding & Margin

- **Padding** can be used to offset the content of the view by a specific amount of pixels.
- For instance, a left padding of 2 will push the view's content by 2 pixels to the right of the left edge.
- set by setPadding(int, int, int, int)
- queried by getPaddingLeft(), getPaddingTop(), getPaddingRight() and getPaddingBottom()
- Support for margin only in ViewGroup, not in Views

Pixel density

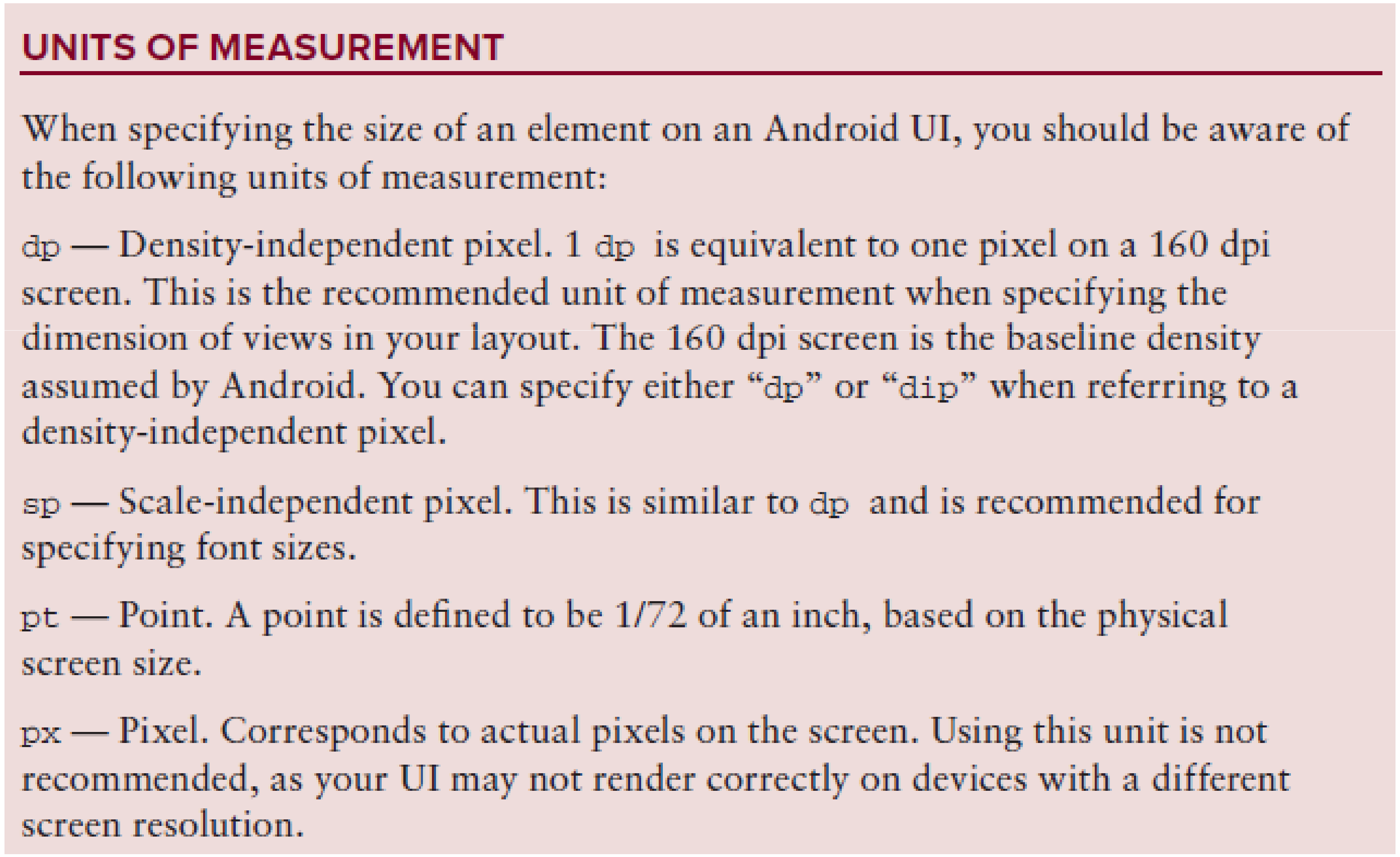
- The screen of the Nexus S. It has a 4-inch screen (diagonally), with a screen width of 2.04 inches.
- Its resolution is 480 (width) 800 (height) pixels. With 480 pixels spread across a width of 2.04 inches, the result is a pixel density of about 235 dots per inch (dpi).



Pixel density

- The pixel density of a screen varies according to **screen size** and **resolution**.
- Android defines and recognizes four **screen densities**:
  - Low density (*ldpi) — 120 dpi*
  - Medium density (*mdpi) — 160 dpi*
  - High density (*hdpi) — 240 dpi*
  - Extra High density (*xhdpi) — 320 dpi*
- Using the *dp unit ensures that your views are always displayed in the right proportion regardless of* the screen density — Android automatically scales the size of the view depending on the density of the screen.

Units of measurement



UNITS OF MEASUREMENT

When specifying the size of an element on an Android UI, you should be aware of the following units of measurement:

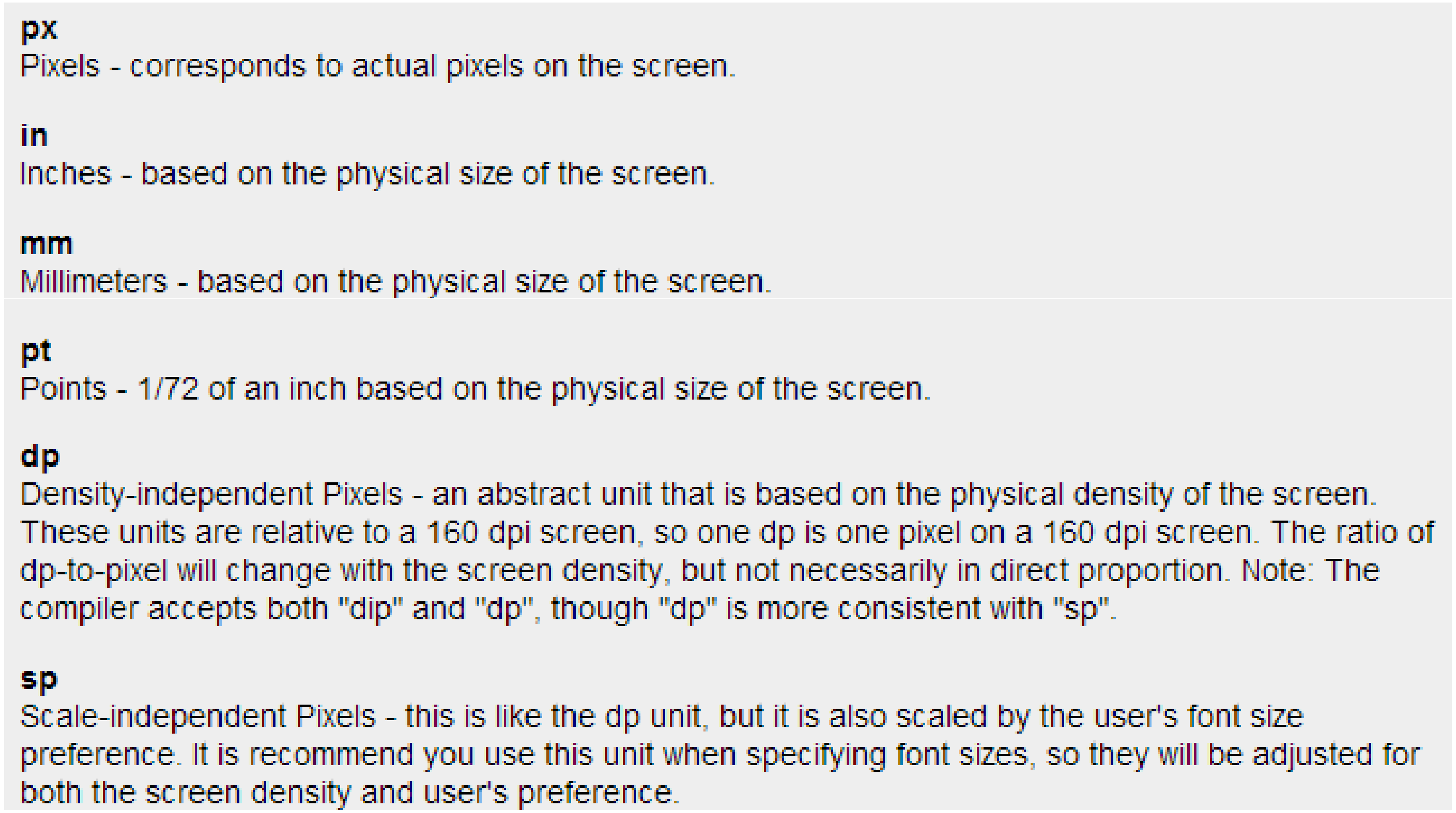
`dp` — Density-independent pixel. 1 `dp` is equivalent to one pixel on a 160 dpi screen. This is the recommended unit of measurement when specifying the dimension of views in your layout. The 160 dpi screen is the baseline density assumed by Android. You can specify either "`dp`" or "`dip`" when referring to a density-independent pixel.

`sp` — Scale-independent pixel. This is similar to `dp` and is recommended for specifying font sizes.

`pt` — Point. A point is defined to be 1/72 of an inch, based on the physical screen size.

`px` — Pixel. Corresponds to actual pixels on the screen. Using this unit is not recommended, as your UI may not render correctly on devices with a different screen resolution.

Units of measurement



px
Pixels - corresponds to actual pixels on the screen.

in
Inches - based on the physical size of the screen.

mm
Millimeters - based on the physical size of the screen.

pt
Points - 1/72 of an inch based on the physical size of the screen.

dp
Density-independent Pixels - an abstract unit that is based on the physical density of the screen. These units are relative to a 160 dpi screen, so one dp is one pixel on a 160 dpi screen. The ratio of dp-to-pixel will change with the screen density, but not necessarily in direct proportion. Note: The compiler accepts both "dip" and "dp", though "dp" is more consistent with "sp".

sp
Scale-independent Pixels - this is like the dp unit, but it is also scaled by the user's font size preference. It is recommend you use this unit when specifying font sizes, so they will be adjusted for both the screen density and user's preference.

HOW TO CONVERT DP TO PX

The formula for converting `dp` to `px` (pixels) is as follows:

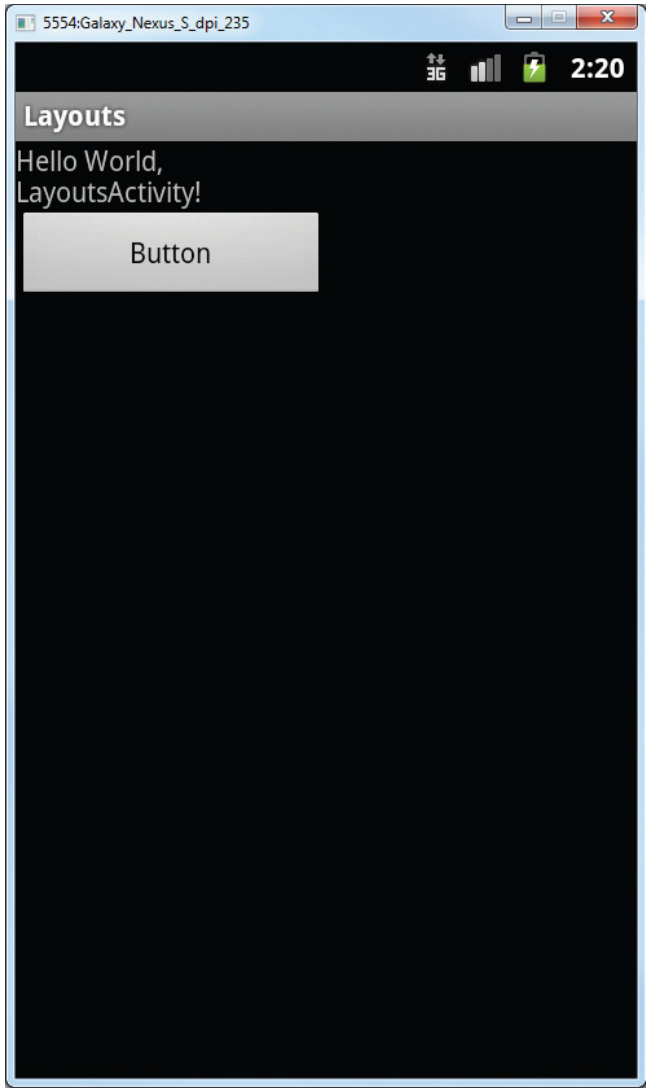
Actual pixels = dp * (dpi / 160), where dpi is either 120, 160, 240, or 320.

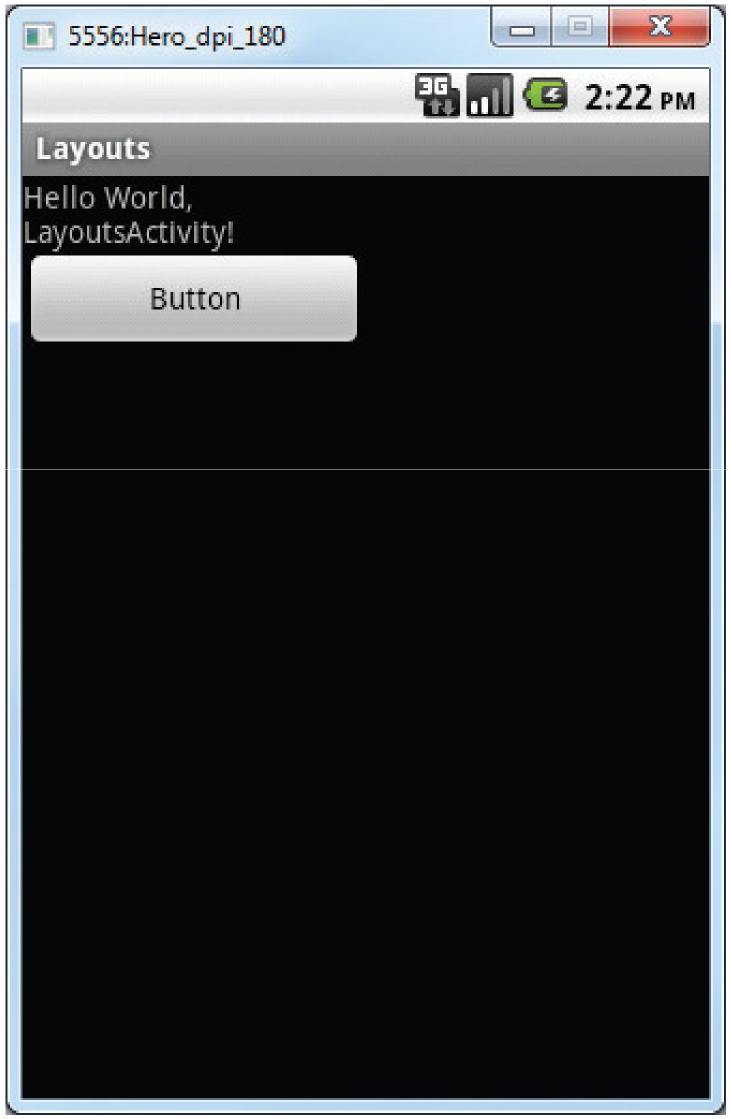
Therefore, in the case of the `Button` on a 235 dpi screen, its actual width is 160 * (240/160) = 240 `px`. When run on the 180 dpi emulator (regarded as a 160 dpi device), its actual pixel is now 160 * (160/160) = 160 `px`. In this case, one `dp` is equivalent to one `px`.

To prove that this is indeed correct, you can use the `getWidth()` method of a `View` object to get its width in pixels:

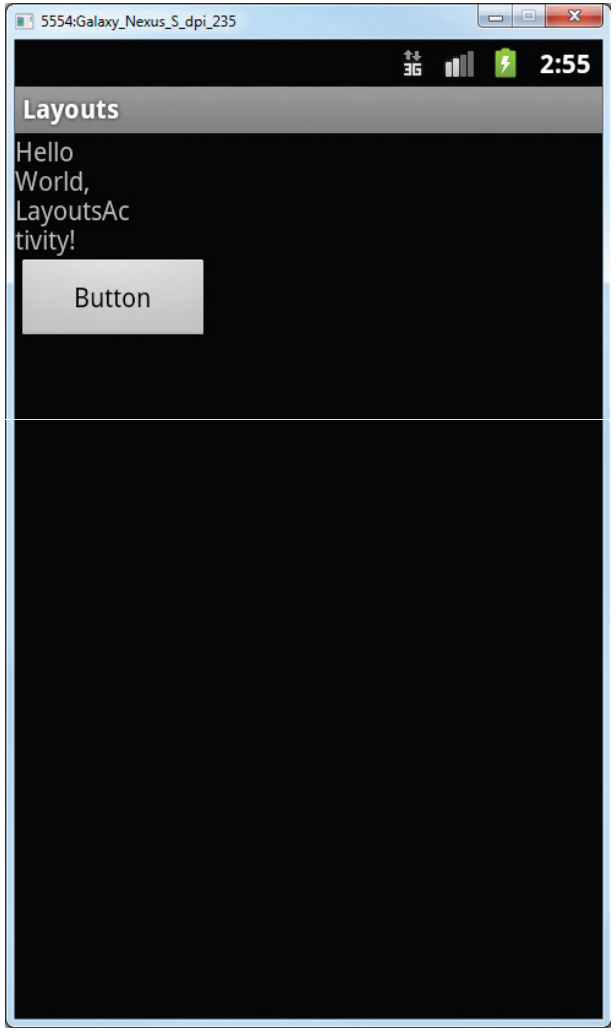
```
public void onClick(View view) {
    Toast.makeText(this,
        String.valueOf(view.getWidth()),
        Toast.LENGTH_LONG).show();
}
```

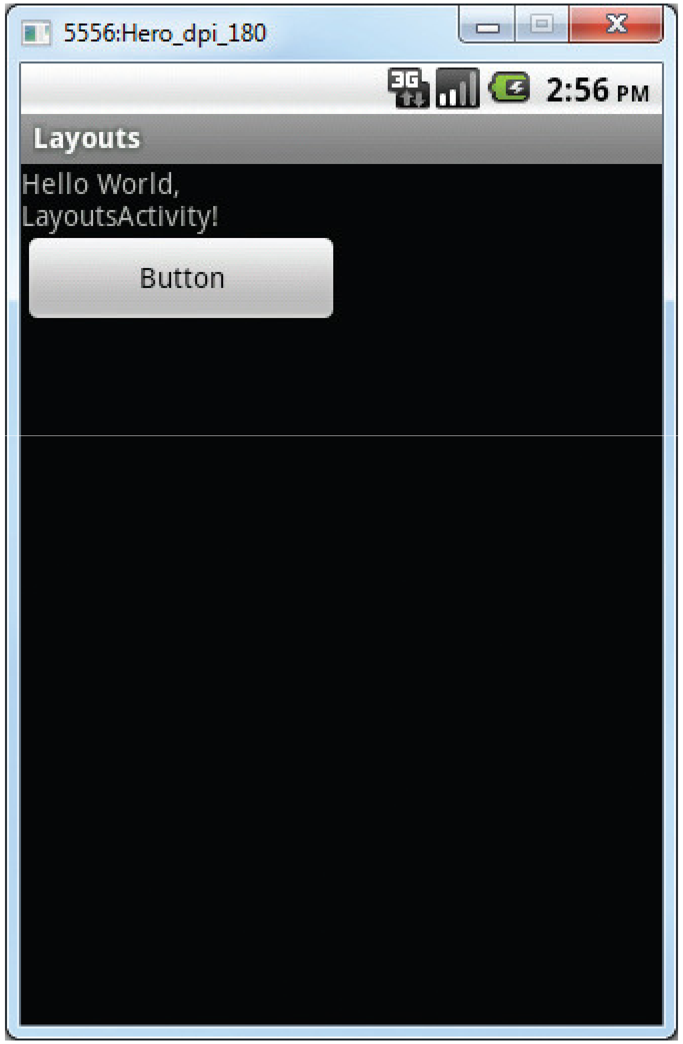
If you use dp on different resolution





If you use px on different resolutions





Extra reading

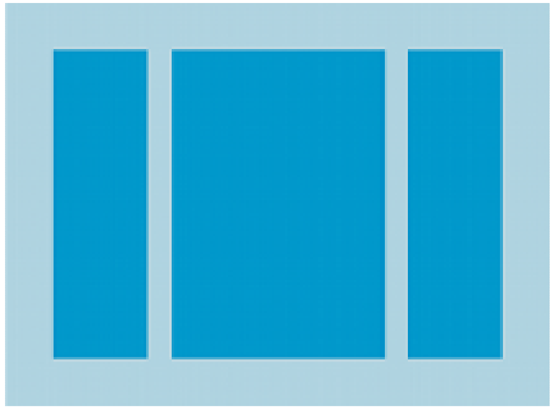
- Supporting multiple screens
  - http://developer.android.com/guide/practices/screens_support.html
- Supporting different densities
  - http://developer.android.com/training/multiscreen/screendensities.html

Orientation

- Use "horizontal“ (0) for a row, "vertical“ (1) for a column.
- The default is **horizontal**

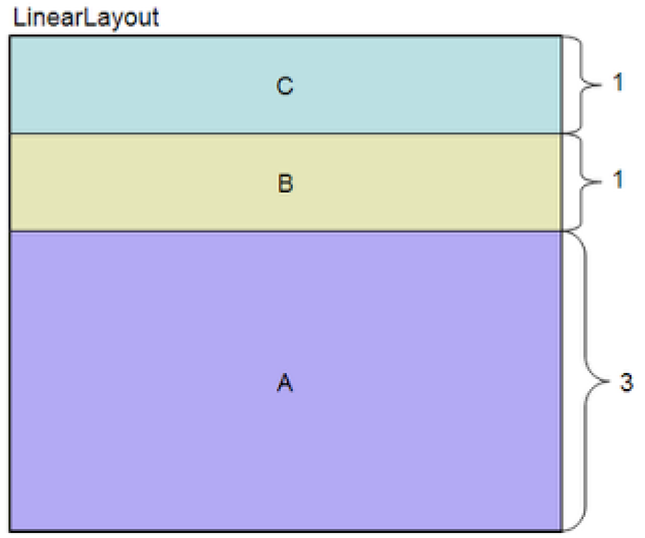
LinearLayout

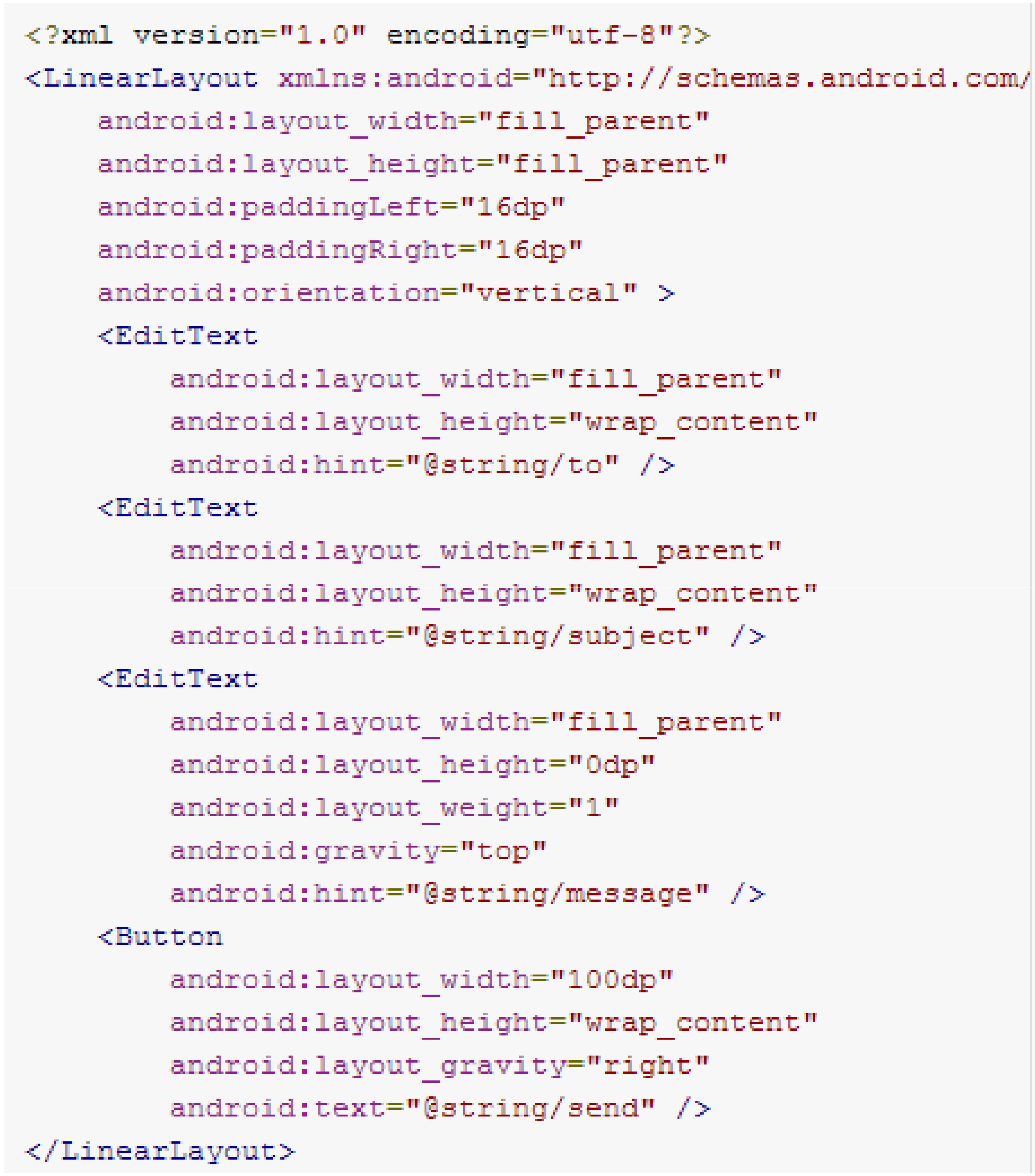
- Organizes its children into a single **horizontal** or **vertical** row (based on android:orientation).
- Child views can be arranged either vertically or horizontally (default).
- It creates a scrollbar if the length of the window exceeds the length of the screen.



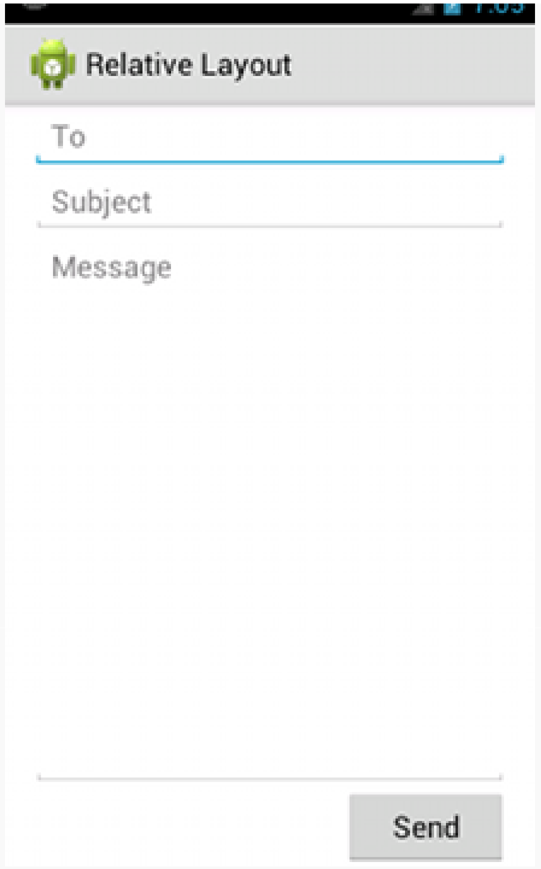
layout_weight

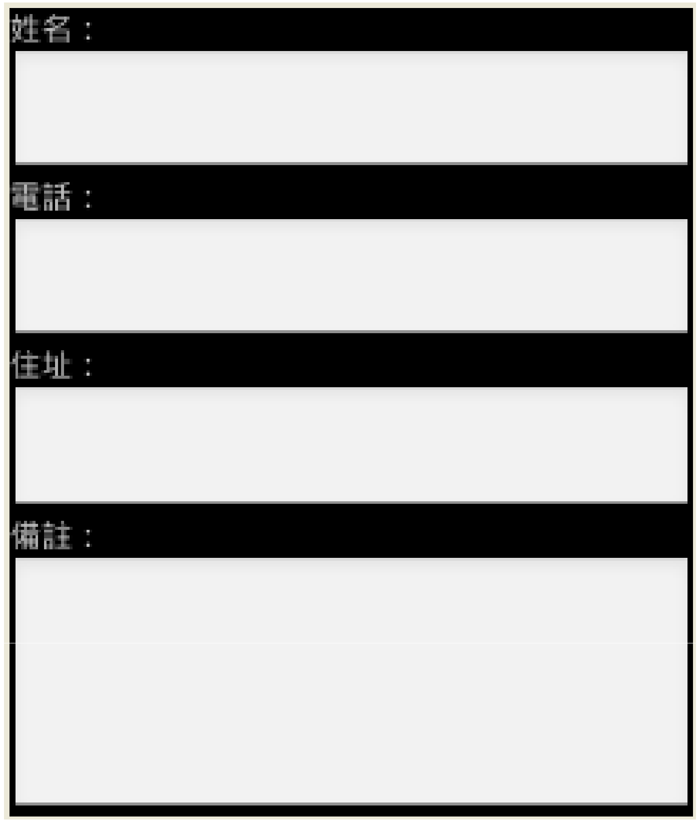
- Assigns an "importance" value to a view in terms of how much space is should occupy on the screen.
- A larger weight value allows it to expand to fill any remaining space in the parent view.
- Child views can specify a weight value, and then any remaining space in the view group is assigned to children in the proportion of their declared weight.
- Default weight is zero.



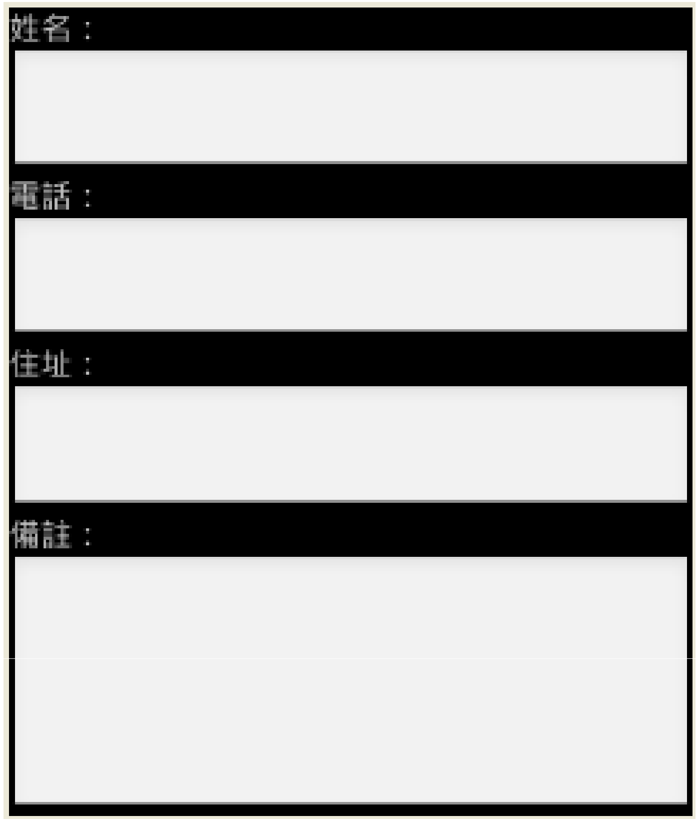


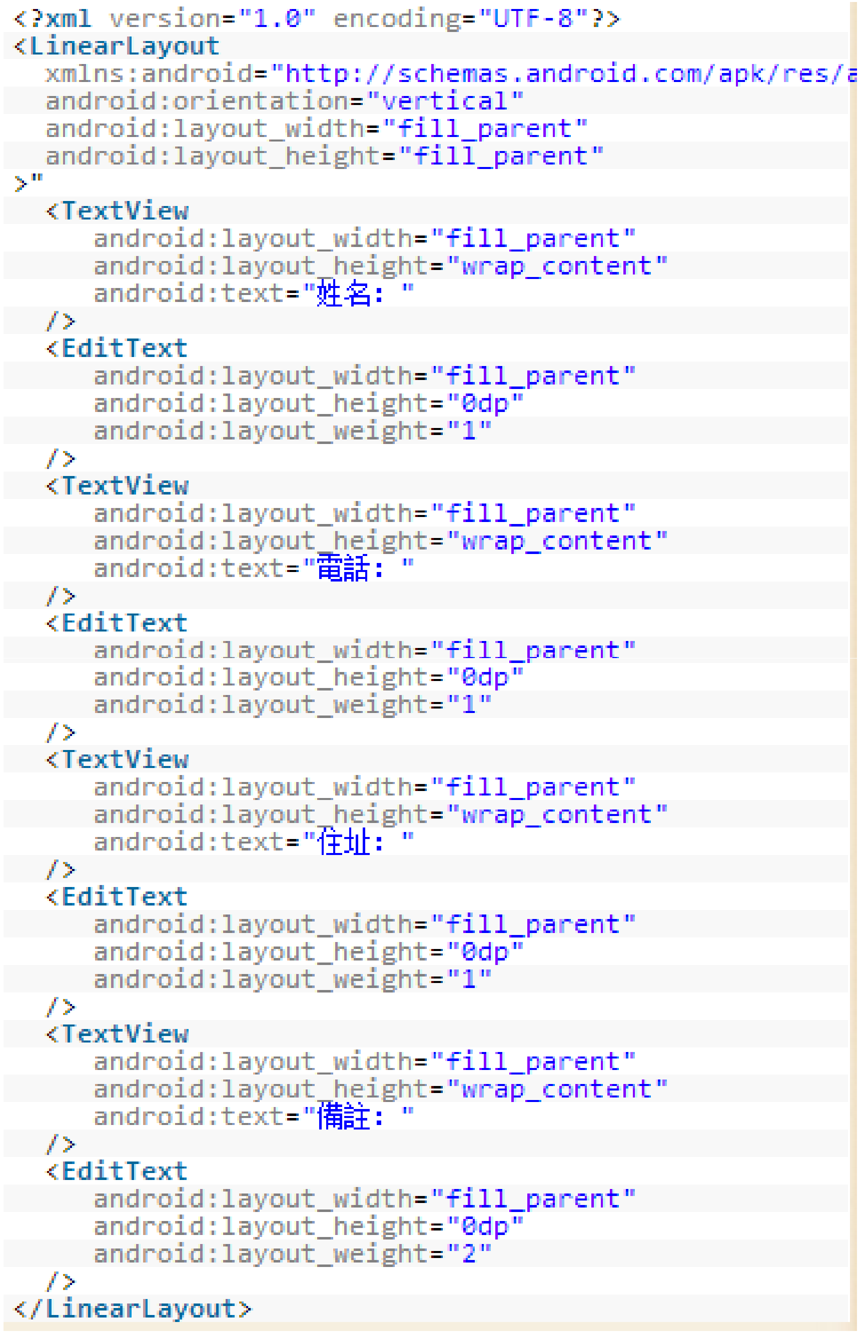
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" >
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/subject" />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="@string/send" />
</LinearLayout>
```





```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res,
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
>"
  <TextView
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:text="姓名: "
  />
  <EditText
     android:layout_width="fill_parent"
     android:layout_height="50dp"
  />
  <TextView
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:text="電話: "
  />
  <EditText
     android:layout_width="fill_parent"
     android:layout_height="50dp"
  />
  <TextView
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:text="住址: "
  />
  <EditText
     android:layout_width="fill_parent"
     android:layout_height="50dp"
  />
  <TextView
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:text="備註: "
  />
  <EditText
     android:layout_width="fill_parent"
     android:layout_height="100dp"
  />
</LinearLayout>
```




```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/a
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
>"
  <TextView
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:text="姓名： "
  />
  <EditText
     android:layout_width="fill_parent"
     android:layout_height="0dp"
     android:layout_weight="1"
  />
  <TextView
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:text="電話： "
  />
  <EditText
     android:layout_width="fill_parent"
     android:layout_height="0dp"
     android:layout_weight="1"
  />
  <TextView
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:text="住址： "
  />
  <EditText
     android:layout_width="fill_parent"
     android:layout_height="0dp"
     android:layout_weight="1"
  />
  <TextView
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:text="備註： "
  />
  <EditText
     android:layout_width="fill_parent"
     android:layout_height="0dp"
     android:layout_weight="2"
  />
</LinearLayout>
```

How to create equally weighted children?

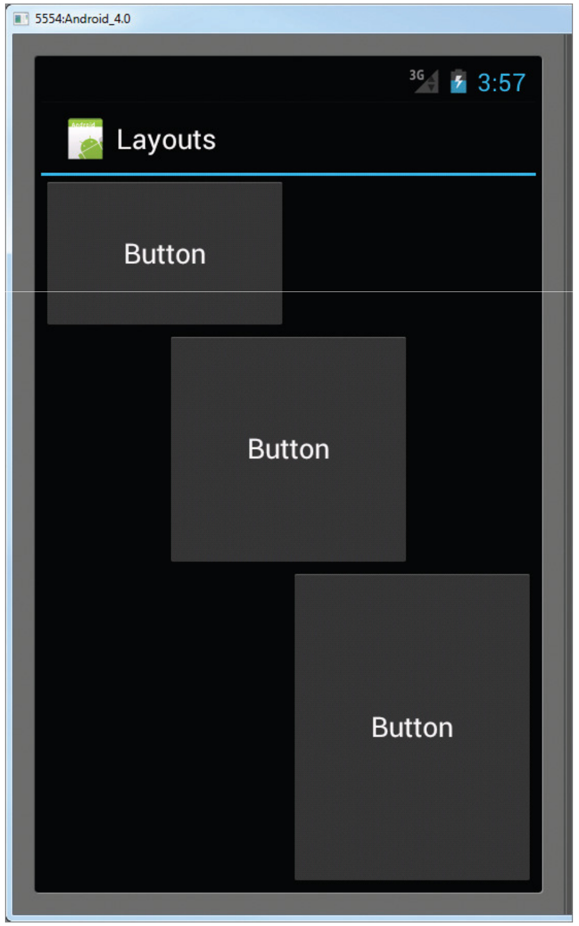
- To create a linear layout in which each child uses the same amount of space on the screen, set the **android:layout_height** of each view to "0dp" (for a vertical layout) or the **android:layout_width** of each view to "0dp" (for a horizontal layout).
- Then set the **android:layout_weight** of each view to "1".

layout_gravity & gravity

- **android:layout_gravity** is the **outside** gravity of the View. That means, to specify the direction in which the View should touch it's parent's border.
- **android:gravity** is the **Inside** gravity of that View. This means, in which direction it's contents should align.

layout_weight & layout_gravity

```
<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:layout_width="160dp"
    android:layout_height="wrap_content"
    android:text="Button"
    android:layout_gravity="left"
    android:layout_weight="1" />

<Button
    android:layout_width="160dp"
    android:layout_height="wrap_content"
    android:text="Button"

    android:layout_gravity="center"
    android:layout_weight="2" />

<Button
    android:layout_width="160dp"
    android:layout_height="wrap_content"
    android:text="Button"
    android:layout_gravity="right"
    android:layout_weight="3" />

</LinearLayout>
```



layout_weight & layout_gravity

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

<Button
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Button"
    android:layout_gravity="left"
    android:layout_weight="1" />

<Button
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Button"
    android:layout_gravity="center_horizontal"
    android:layout_weight="2" />

<Button
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Button"
    android:layout_gravity="right"
    android:layout_weight="3" />

</LinearLayout>
```

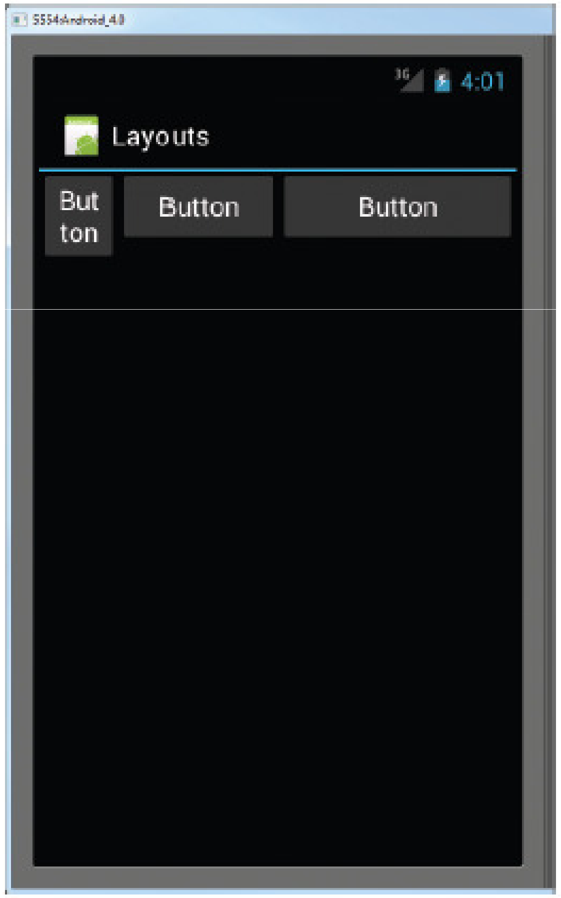


FIGURE 3-10

RelativeLayout

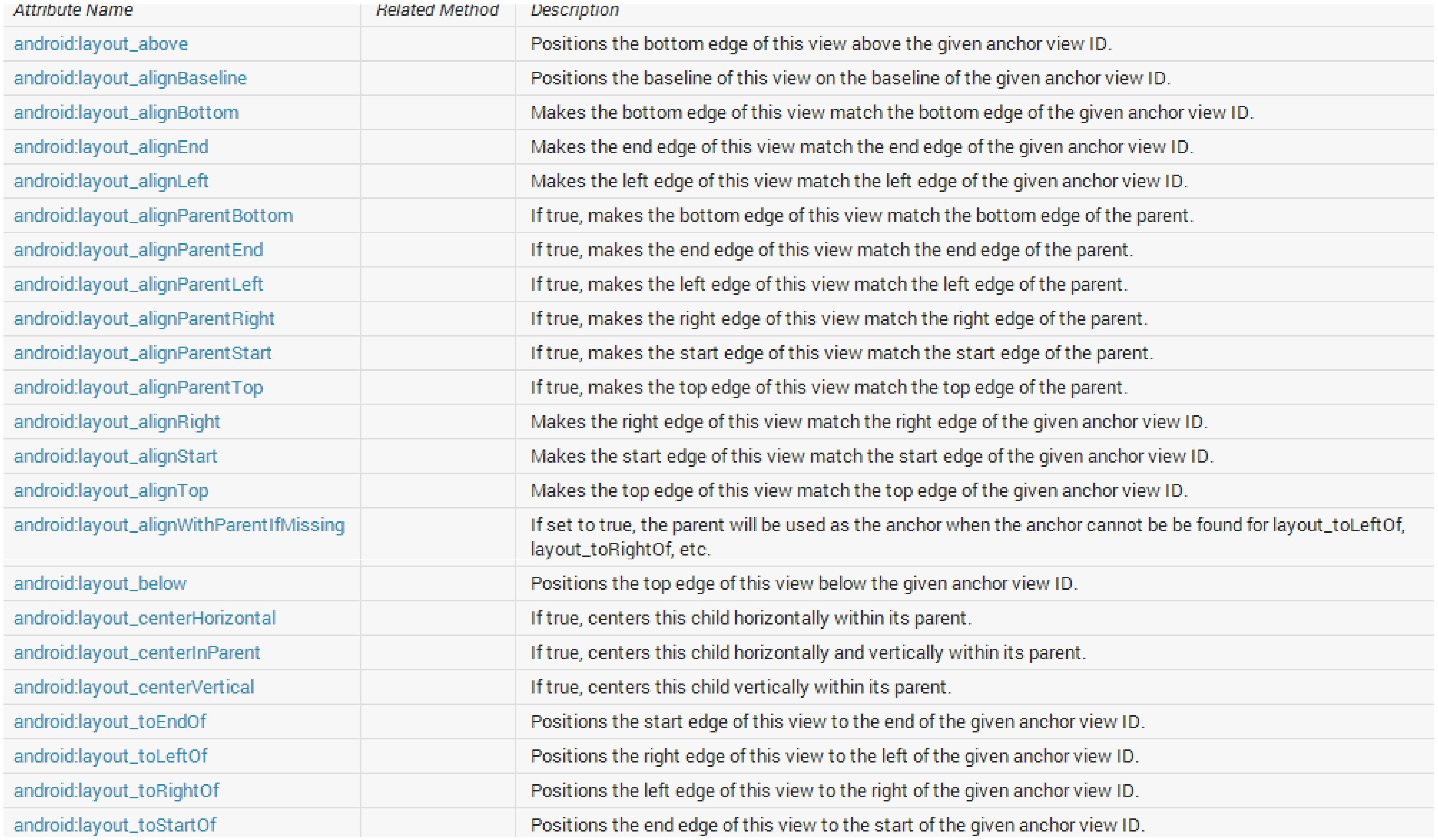
- Enables you to specify the location of child objects **relative to each other** (child A to the left of child B) or to the parent (aligned to the top of the parent).
- Can eliminate nested view groups and keep your layout hierarchy flat, which improves performance.

RelativeLayout

- Each view embedded within the RelativeLayout has attributes that enable it to align with another view.
- These attributes are as follows:
  - layout_alignParentTop
  - layout_alignParentLeft
  - layout_alignLeft
  - layout_alignRight
  - layout_below
  - layout_centerHorizontal

Positioning Views

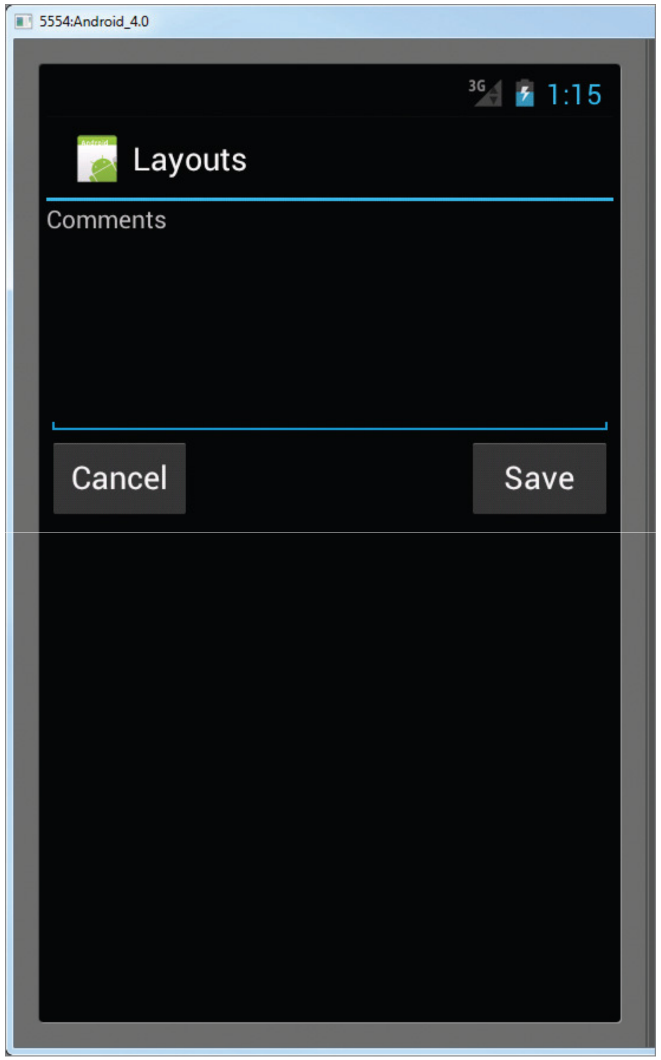
- By default, all child views are drawn at the top-left of the layout, so you must define the position of each view
- android:layout_alignParentTop If "true", makes the top edge of this view match the top edge of the parent.
- android:layout_centerVertical If "true", centers this child vertically within its parent.
- android:layout_below Positions the top edge of this view below the view specified with a resource ID.
- android:layout_toRightOf Positions the left edge of this view to the right of the view specified with a resource ID.



Attribute Name	Related Method	Description
android:layout_above		Positions the bottom edge of this view above the given anchor view ID.
android:layout_alignBaseline		Positions the baseline of this view on the baseline of the given anchor view ID.
android:layout_alignBottom		Makes the bottom edge of this view match the bottom edge of the given anchor view ID.
android:layout_alignEnd		Makes the end edge of this view match the end edge of the given anchor view ID.
android:layout_alignLeft		Makes the left edge of this view match the left edge of the given anchor view ID.
android:layout_alignParentBottom		If true, makes the bottom edge of this view match the bottom edge of the parent.
android:layout_alignParentEnd		If true, makes the end edge of this view match the end edge of the parent.
android:layout_alignParentLeft		If true, makes the left edge of this view match the left edge of the parent.
android:layout_alignParentRight		If true, makes the right edge of this view match the right edge of the parent.
android:layout_alignParentStart		If true, makes the start edge of this view match the start edge of the parent.
android:layout_alignParentTop		If true, makes the top edge of this view match the top edge of the parent.
android:layout_alignRight		Makes the right edge of this view match the right edge of the given anchor view ID.
android:layout_alignStart		Makes the start edge of this view match the start edge of the given anchor view ID.
android:layout_alignTop		Makes the top edge of this view match the top edge of the given anchor view ID.
android:layout_alignWithParentIfMissing		If set to true, the parent will be used as the anchor when the anchor cannot be be found for layout_toLeftOf, layout_toRightOf, etc.
android:layout_below		Positions the top edge of this view below the given anchor view ID.
android:layout_centerHorizontal		If true, centers this child horizontally within its parent.
android:layout_centerInParent		If true, centers this child horizontally and vertically within its parent.
android:layout_centerVertical		If true, centers this child vertically within its parent.
android:layout_toEndOf		Positions the start edge of this view to the end of the given anchor view ID.
android:layout_toLeftOf		Positions the right edge of this view to the left of the given anchor view ID.
android:layout_toRightOf		Positions the left edge of this view to the right of the given anchor view ID.
android:layout_toStartOf		Positions the end edge of this view to the start of the given anchor view ID.

- http://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html

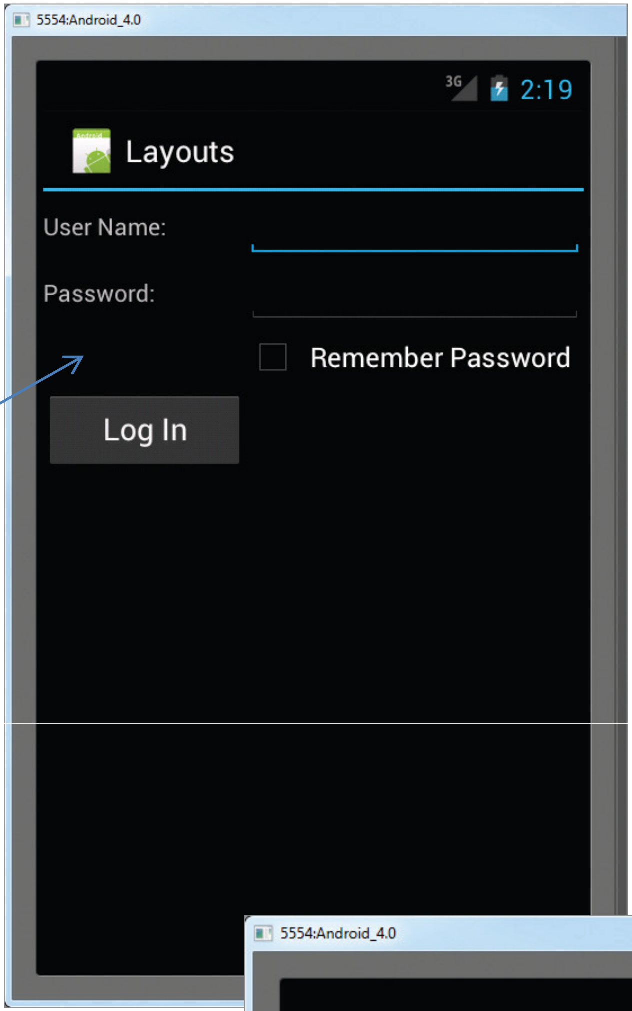
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    android:id="@+id/RLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android" >

    <TextView
        android:id="@+id/lblComments"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Comments"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true" />

    <EditText
        android:id="@+id/txtComments"

        android:layout_width="fill_parent"
        android:layout_height="170px"
        android:textSize="18sp"
        android:layout_alignLeft="@+id/lblComments"
        android:layout_below="@+id/lblComments"
        android:layout_centerHorizontal="true" />

    <Button
        android:id="@+id/btnSave"
        android:layout_width="125px"
        android:layout_height="wrap_content"
        android:text="Save"
        android:layout_below="@+id/txtComments"
        android:layout_alignRight="@+id/txtComments" /

    <Button
        android:id="@+id/btnCancel"
        android:layout_width="124px"
        android:layout_height="wrap_content"
        android:text="Cancel"
        android:layout_below="@+id/txtComments"
        android:layout_alignLeft="@+id/txtComments" />
</RelativeLayout>
```

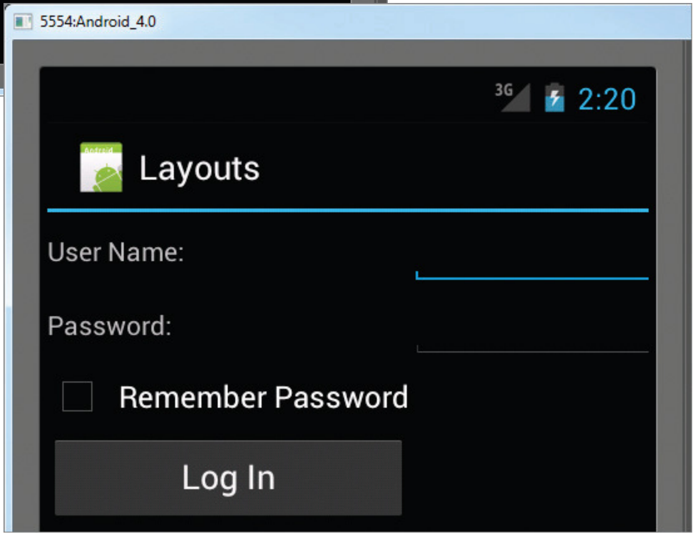


TableLayout

- Groups views into **rows** and **columns**.
- Use <TableRow> element to designate a row in the table. Each row can contain one or more views. Each view placed within a row forms a cell.
- The width of each column is determined by the largest width of each cell in that column.

```
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="fill_parent"
    android:layout_width="fill_parent" >
    <TableRow>
        <TextView
            android:text="User Name:"
            android:width ="120dp"
            />
        <EditText
            android:id="@+id/txtUserName"
            android:width="200dp" />
    </TableRow>
    <TableRow>
        <TextView
            android:text="Password:"
            />
        <EditText
            android:id="@+id/txtPassword"
            android:password="true"
            />
    </TableRow>
    <TableRow>
        <TextView />
        <CheckBox android:id="@+id/chkRememberPassword"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Remember Password"
            />
    </TableRow>
    <TableRow>
        <Button
            android:id="@+id/buttonSignIn"
            android:text="Log In" />
    </TableRow>
</TableLayout>
```

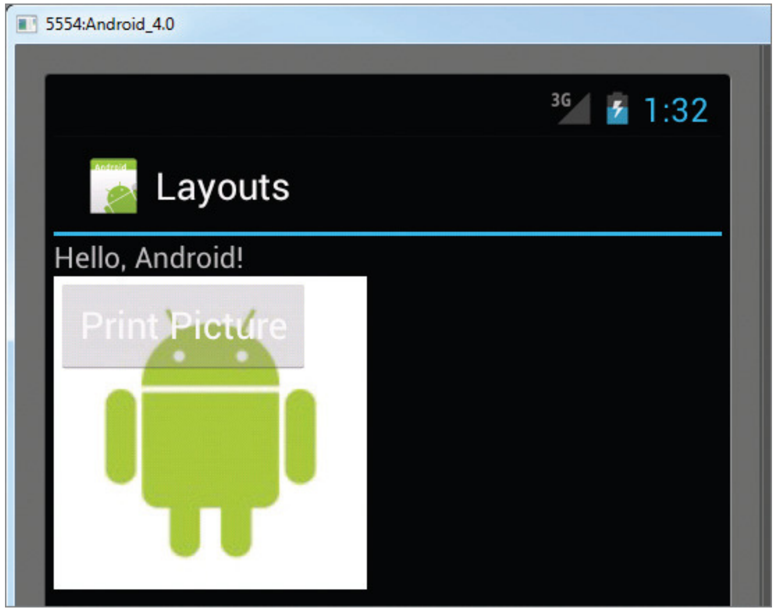




FrameLayout

- Placeholder on screen that you can use to display a **single view**.
- Block out an area on the screen to display a **single item**.
- Views that you add to a FrameLayout are always anchored to the top left of the layout.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    android:id="@+id/RLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android"
    >
    <TextView
        android:id="@+id/lblComments"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, Android!"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        />
    <FrameLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/lblComments"
        android:layout_below="@+id/lblComments"
        android:layout_centerHorizontal="true" >
        <ImageView
            android:src = "@drawable/droid"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

        <Button
            android:layout_width="124dp"
            android:layout_height="wrap_content"
            android:text="Print Picture" />

    </FrameLayout>
</RelativeLayout>
```

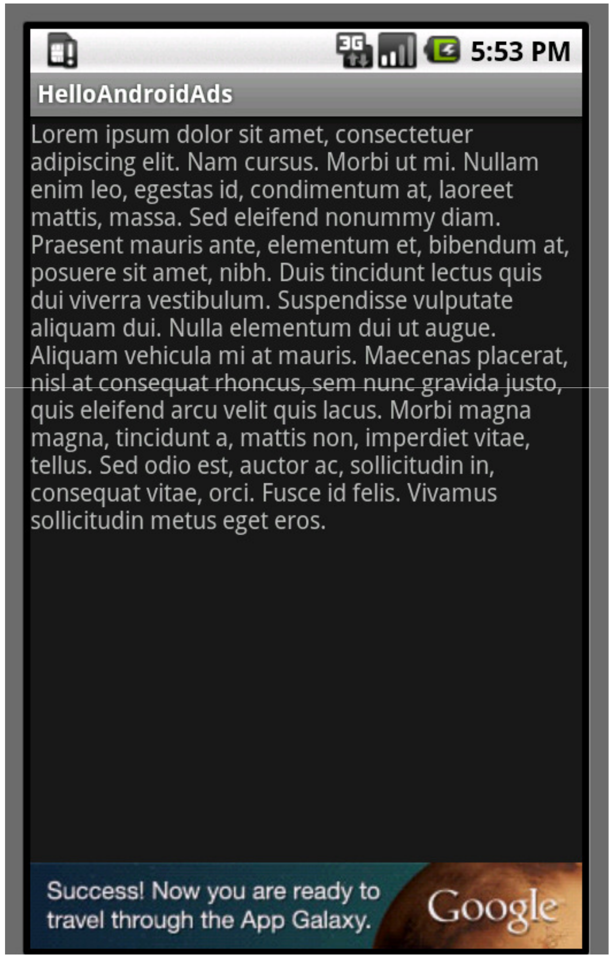


- FrameLayout within a RelativeLayout
- Within the FrameLayout, you embed an ImageView and a button

ScrollView

- Special type of FrameLayout in that it enables users to scroll through a list of views that occupy more space than the physical display.
- The ScrollView can contain **only one child view** or **ViewGroup**, which normally is a LinearLayout.

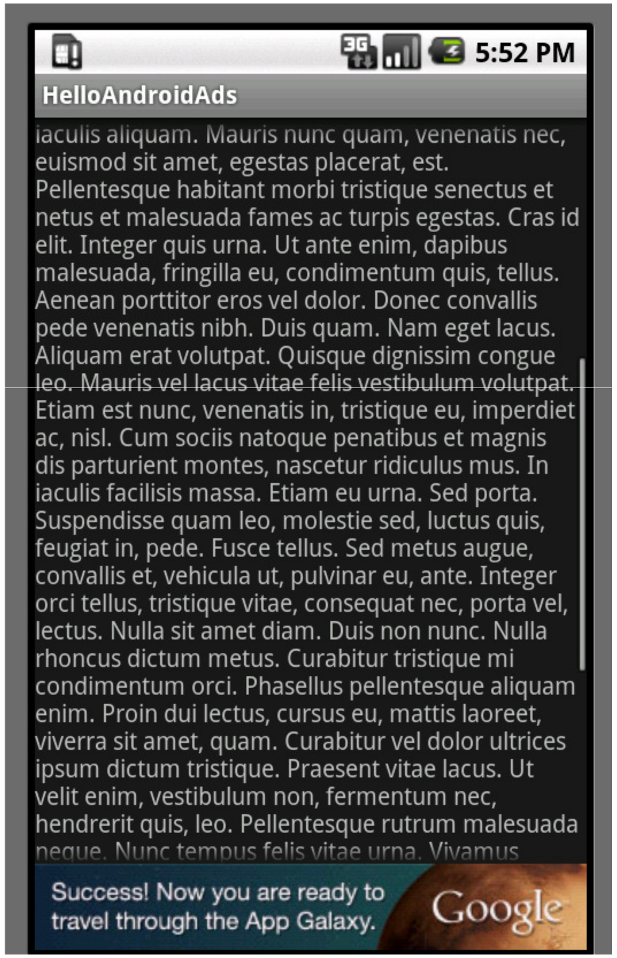
ScrollView


5:53 PM

HelloAndroidAds

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nam cursus. Morbi ut mi. Nullam enim leo, egestas id, condimentum at, laoreet mattis, massa. Sed eleifend nonummy diam. Praesent mauris ante, elementum et, bibendum at, posuere sit amet, nibh. Duis tincidunt lectus quis dui viverra vestibulum. Suspendisse vulputate aliquam dui. Nulla elementum dui ut augue. Aliquam vehicula mi at mauris. Maecenas placerat, nisl at consequat rhoncus, sem nunc gravida justo, quis eleifend arcu velit quis lacus. Morbi magna magna, tincidunt a, mattis non, imperdiet vitae, tellus. Sed odio est, auctor ac, sollicitudin in, consequat vitae, orci. Fusce id felis. Vivamus sollicitudin metus eget eros.

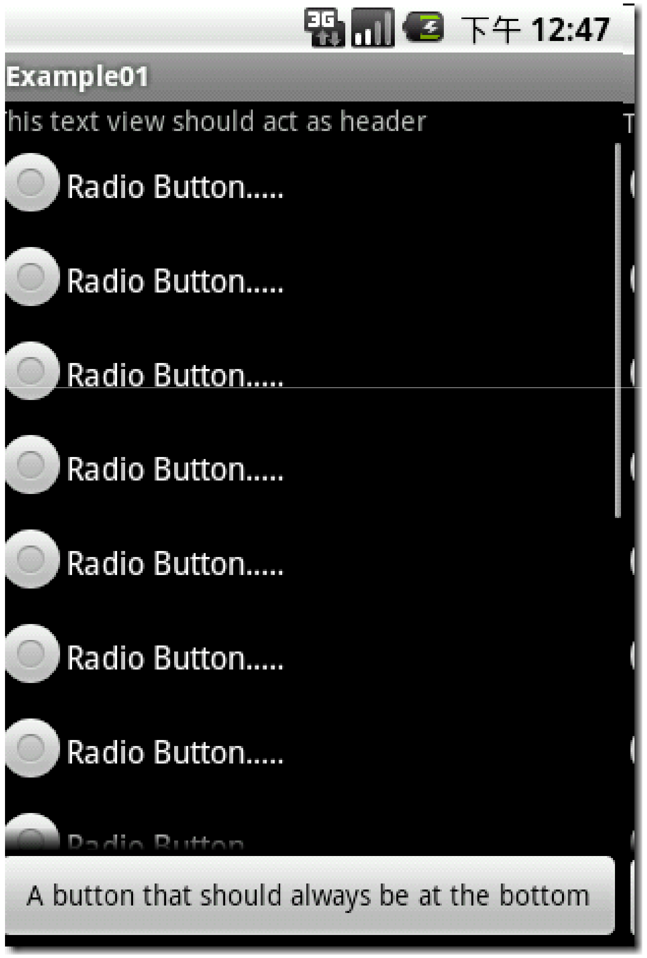
Success! Now you are ready to travel through the App Galaxy.


5:52 PM

HelloAndroidAds

iaculis aliquam. Mauris nunc quam, venenatis nec, euismod sit amet, egestas placerat, est. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Cras id elit. Integer quis urna. Ut ante enim, dapibus malesuada, fringilla eu, condimentum quis, tellus. Aenean porttitor eros vel dolor. Donec convallis pede venenatis nibh. Duis quam. Nam eget lacus. Aliquam erat volutpat. Quisque dignissim congue leo. Mauris vel lacus vitae felis vestibulum volutpat. Etiam est nunc, venenatis in, tristique eu, imperdiet ac, nisl. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. In iaculis facilisis massa. Etiam eu urna. Sed porta. Suspendisse quam leo, molestie sed, luctus quis, feugiat in, pede. Fusce tellus. Sed metus augue, convallis et, vehicula ut, pulvinar eu, ante. Integer orci tellus, tristique vitae, consequat nec, porta vel, lectus. Nulla sit amet diam. Duis non nunc. Nulla rhoncus dictum metus. Curabitur tristique mi condimentum orci. Phasellus pellentesque aliquam enim. Proin dui lectus, cursus eu, mattis laoreet, viverra sit amet, quam. Curabitur vel dolor ultrices ipsum dictum tristique. Praesent vitae lacus. Ut velit enim, vestibulum non, fermentum nec, hendrerit quis, leo. Pellentesque rutrum malesuada neque. Nunc tempus felis vitae urna. Vivamus

Success! Now you are ready to travel through the App Galaxy.


下午 12:47

Example01

his text view should act as header

Radio Button.....

Radio Button.....

Radio Button.....

Radio Button.....

Radio Button.....

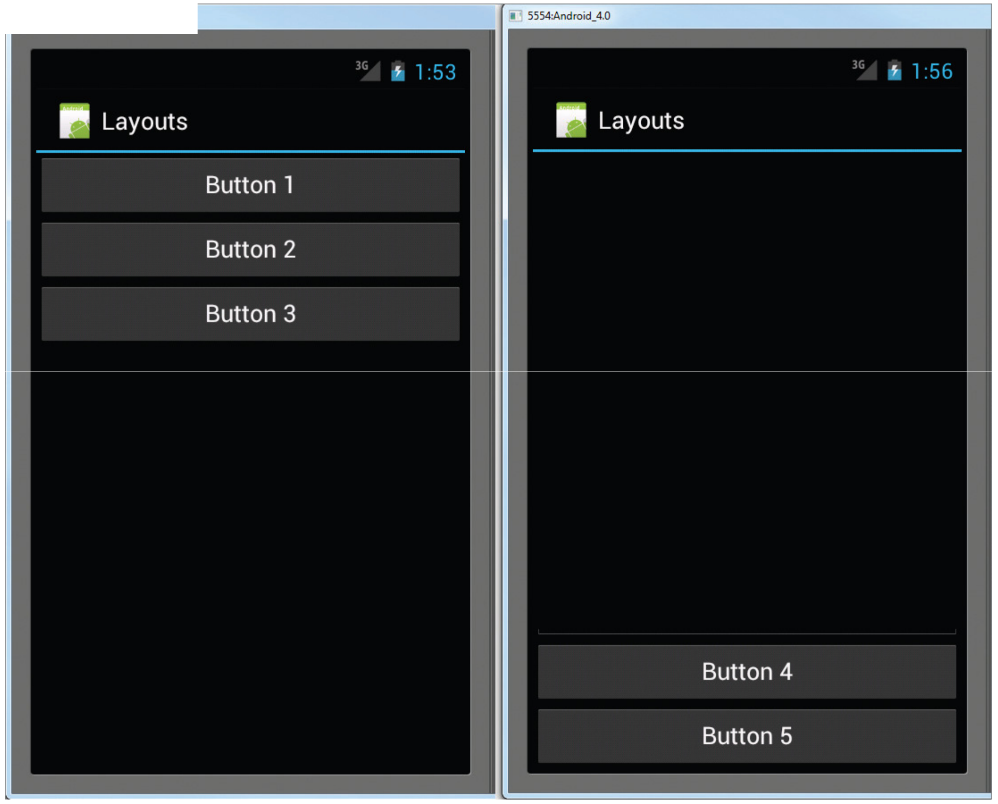
Radio Button.....

Radio Button.....

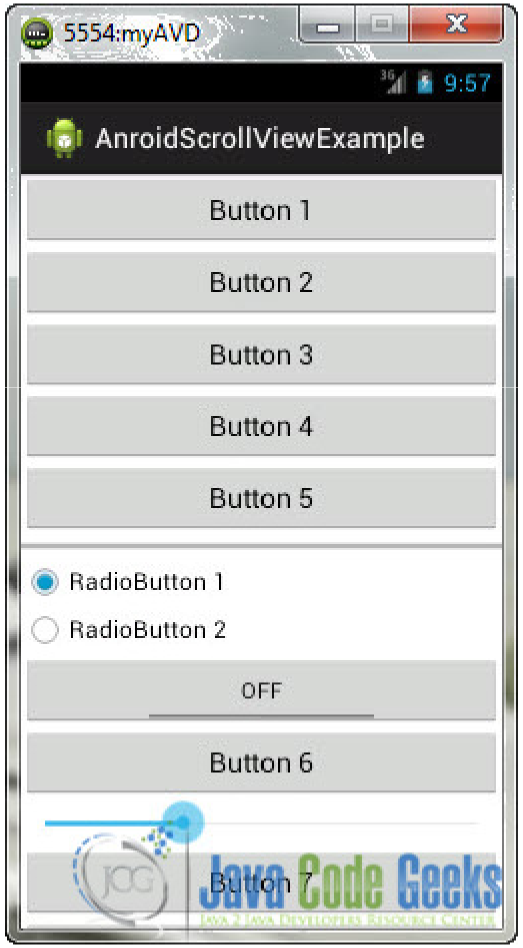
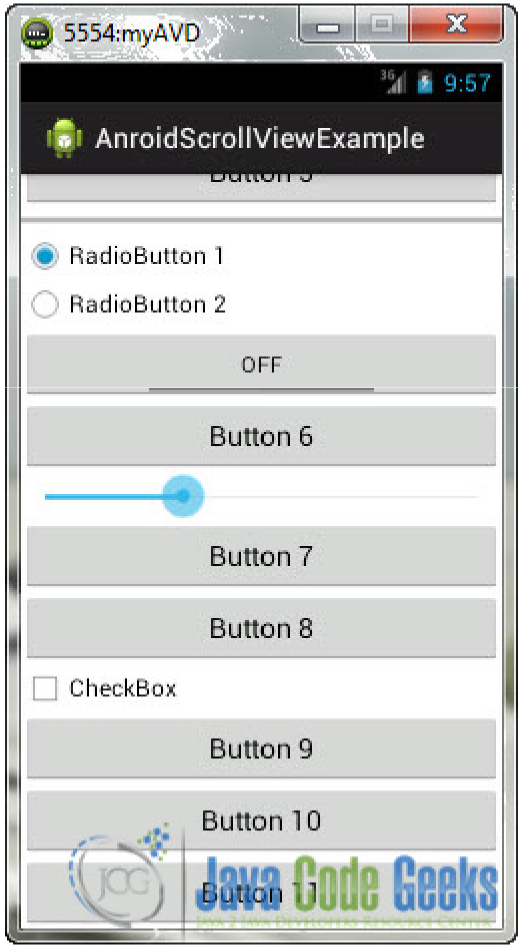
Radio Button

A button that should always be at the bottom

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

  <LinearLayout
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:orientation="vertical"
      android:focusable="true"
      android:focusableInTouchMode="true" >

        <Button
            android:id="@+id/button1"

            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Button 1" />
        <Button
            android:id="@+id/button2"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Button 2" />
        <Button
            android:id="@+id/button3"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Button 3" />
        <EditText
            android:id="@+id/txt"
            android:layout_width="fill_parent"
            android:layout_height="600dp" />
        <Button
            android:id="@+id/button4"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Button 4" />
        <Button
            android:id="@+id/button5"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Button 5" />
    </LinearLayout>
</ScrollView>
```

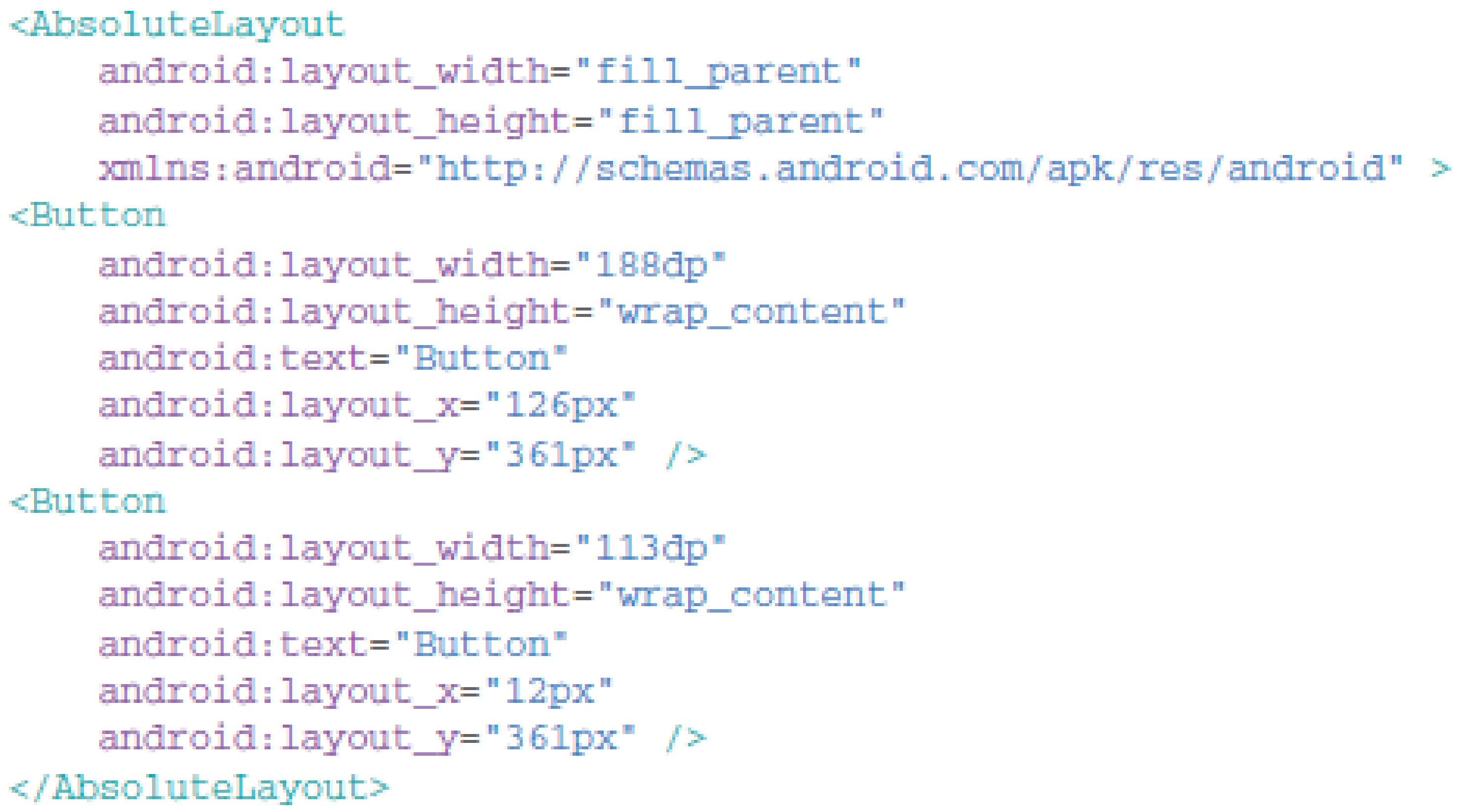


```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" >

    <LinearLayout

        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical" >

        <Button
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Button 1" />

        <Button
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Button 2" />

        <Button
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Button 3" />

        <Button
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Button 4" />

        <Button
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Button 5" />

        <ProgressBar
            android:id="@+id/progressBar1"
            style="?android:attr/progressBarStyleHorizontal"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />

        <RadioButton
            android:id="@+id/radioButton2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="RadioButton 1" />

        <RadioButton
            android:id="@+id/radioButton1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="RadioButton 2" />

        <ToggleButton
            android:id="@+id/toggleButton1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="ToggleButton" />

        <Button
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Button 6" />

        <SeekBar
            android:id="@+id/seekBar1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" /

        <Button
            android:id="@+id/button3"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Button 7" />

        <Button
            android:id="@+id/button4"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Button 8" />

        <CheckBox
            android:id="@+id/checkBox1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="CheckBox" />

        <Button
            android:id="@+id/button2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Button 9" />

        <Button
            android:id="@+id/button1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Button 10" />

        <Button
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Button 11" />
    </LinearLayout>

</ScrollView>
```


5554:myAVD
9:57
AnroidScrollViewExample
Button 1
Button 2
Button 3
Button 4
Button 5
RadioButton 1
RadioButton 2
OFF
Button 6
Button 7

5554:myAVD
9:57
AnroidScrollViewExample
RadioButton 1
RadioButton 2
OFF
Button 6
Button 7
Button 8
CheckBox
Button 9
Button 10
Button 11

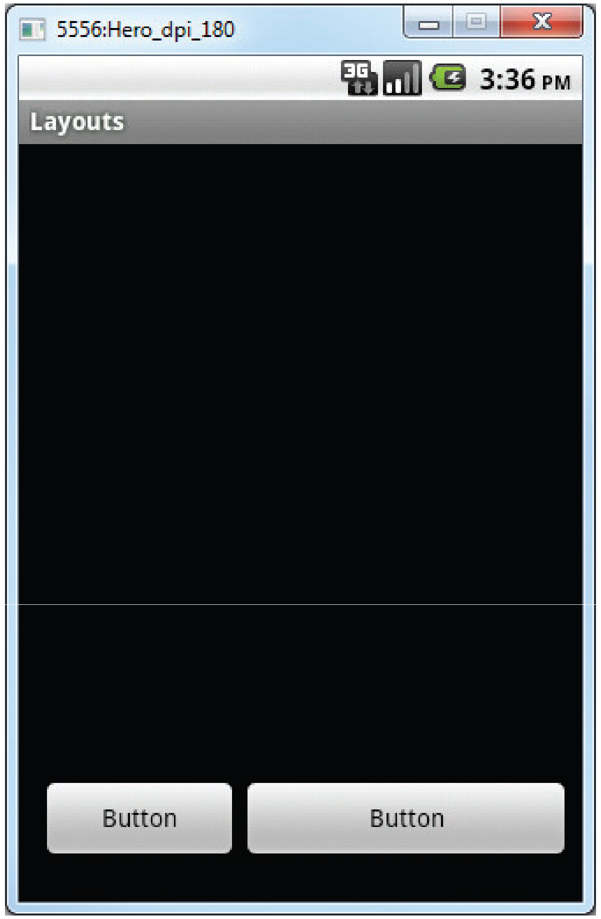
AbsoluteLayout

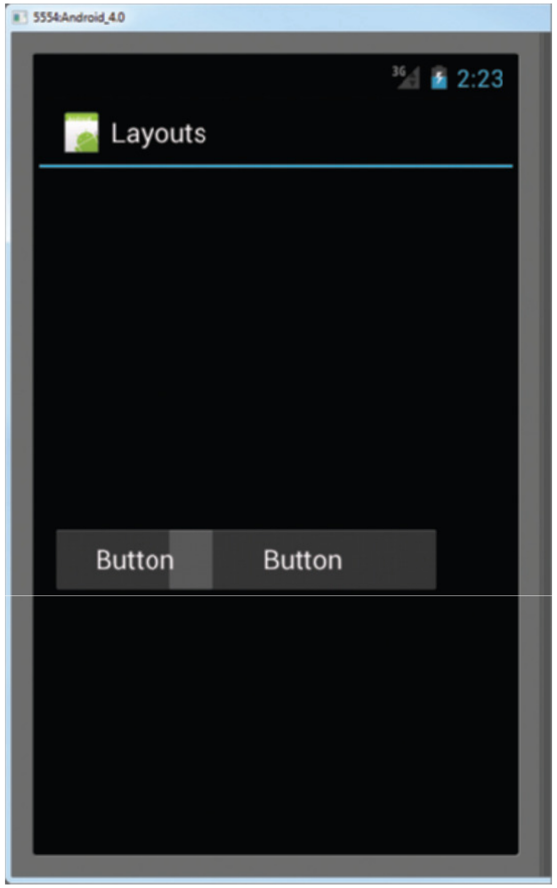
- Specify the exact location ofits children



```
<AbsoluteLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android" >
<Button
    android:layout_width="188dp"
    android:layout_height="wrap_content"
    android:text="Button"
    android:layout_x="126px"
    android:layout_y="361px" />
<Button
    android:layout_width="113dp"
    android:layout_height="wrap_content"
    android:text="Button"
    android:layout_x="12px"
    android:layout_y="361px" />
</AbsoluteLayout>
```

- Problem:

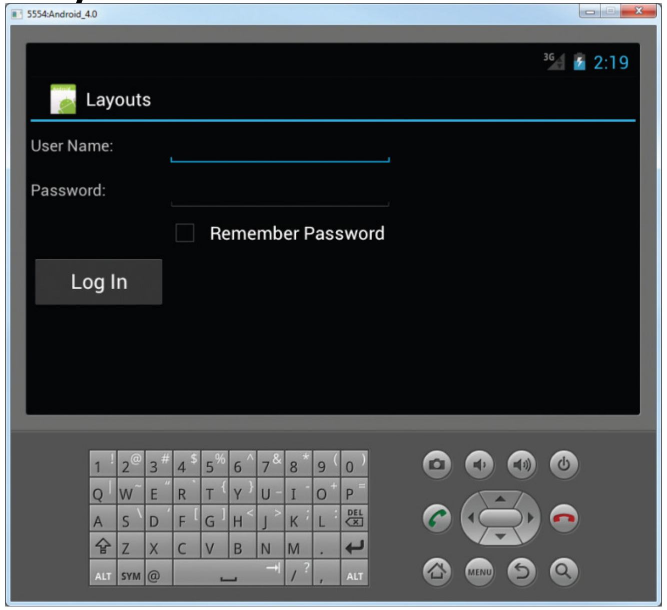




- has been deprecated since Android 1.5 (although it is still supported in the current version).

Screen orientation

- Android supports two screen orientations: *portrait* and *landscape.*
- By default, when you change the display orientation of your Android device, the current activity that is displayed automatically redraws its content in the new orientation.
- The **onCreate()** method of the activity is fired whenever there is a change in display orientation.

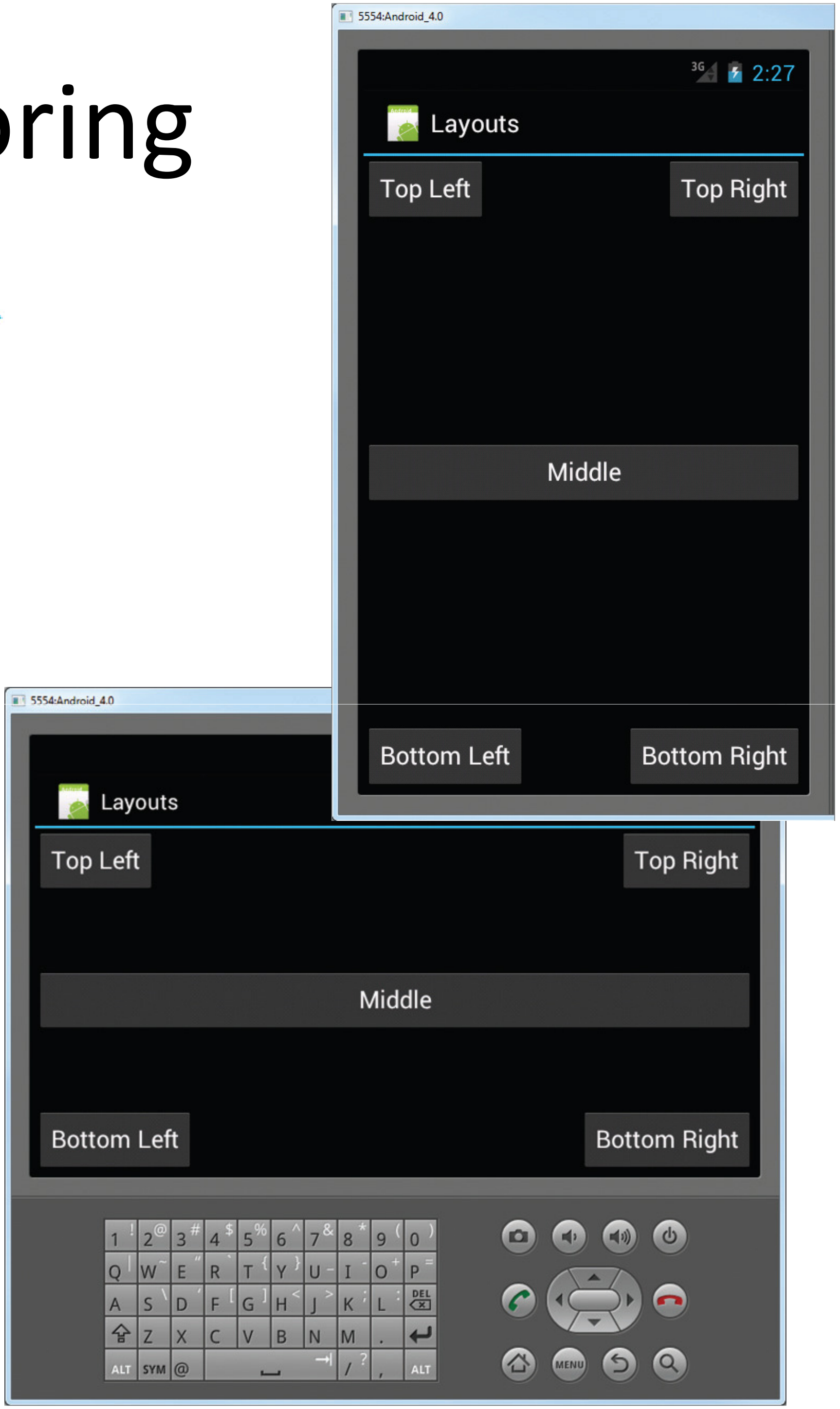


Adapting to screen orientation

- 2 techniques to handle changes in screen orientation
  - **Anchoring** to the four edges of the screen. When the screen orientation changes, the views can anchor neatly to the edges. It is achieved by using RelativeLayout.
  - **Resizing and repositioning** each and every view according to the current screen orientation

Anchoring

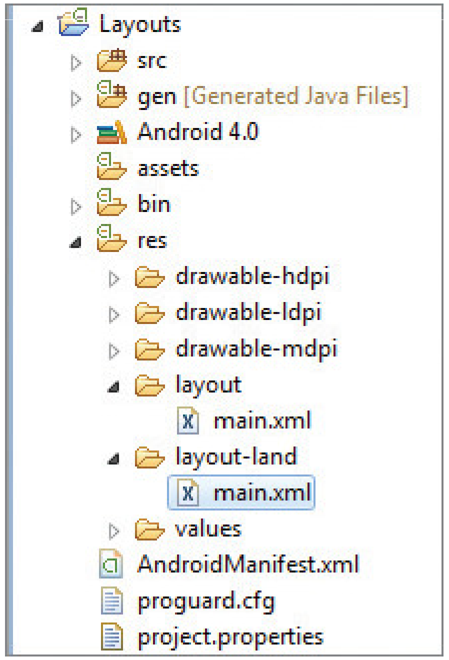
```
<RelativeLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Left"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true" />
    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Right"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true" />
    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom Left"
        android:layout_alignParentLeft="true"
        android:layout_alignParentBottom="true" />
    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom Right"
        android:layout_alignParentRight="true"
        android:layout_alignParentBottom="true" />
    <Button
        android:id="@+id/button5"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Middle"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```



- layout_alignParentLeft — Aligns the view to the left of the parent view
- layout_alignParentRight — Aligns the view to the right of the parent view
- layout_alignParentTop — Aligns the view to the top of the parent view
- layout_alignParentBottom — Aligns the view to the bottom of the parent view
- layout_centerVertical — Centers the view vertically within its parent view
- layout_centerHorizontal — Centers the view horizontally within its parent view

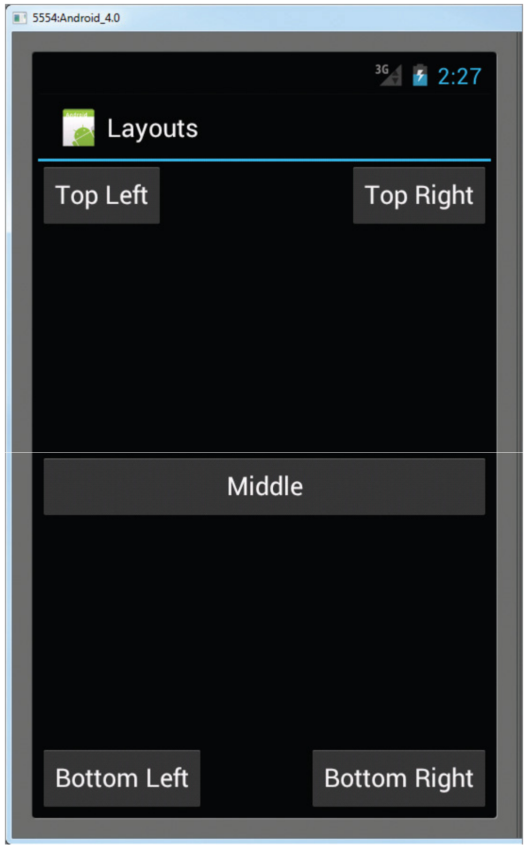
Resizing and Repositioning

- To support landscape mode, you can create a new folder in the res folder and name it as layout-land (representing landscape).



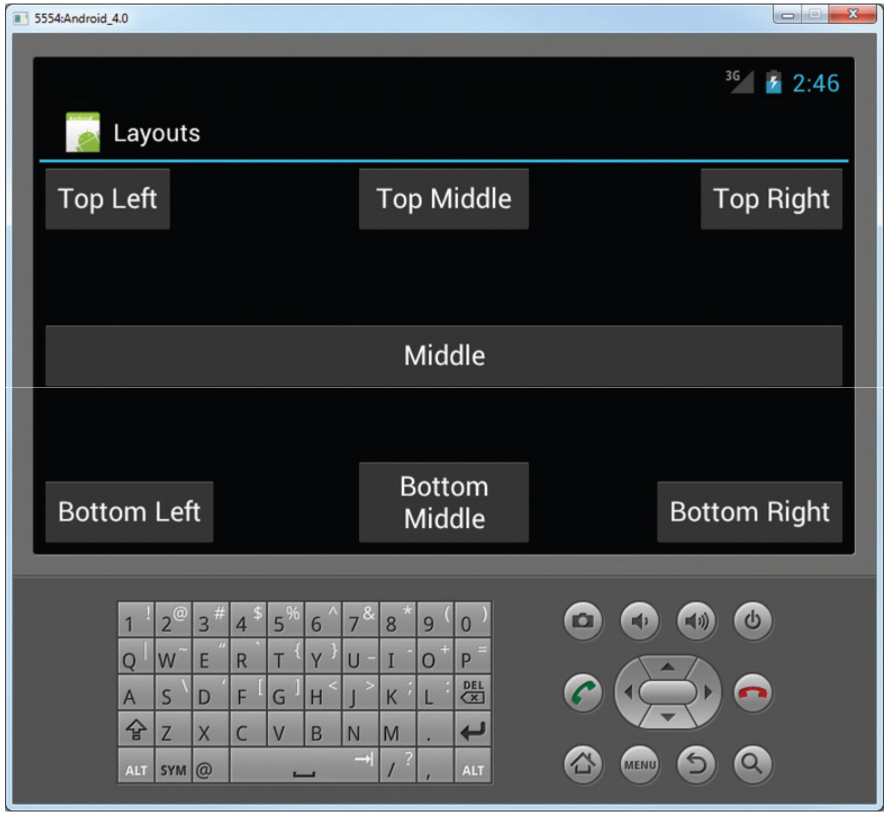
main.xml in layout

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Left"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true" />
    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Right"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true" />
    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom Left"
        android:layout_alignParentLeft="true"
        android:layout_alignParentBottom="true" />
    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom Right"
        android:layout_alignParentRight="true"
        android:layout_alignParentBottom="true" />
    <Button
        android:id="@+id/button5"
        android:layout_width="fill_parent"

        android:layout_height="wrap_content"
        android:text="Middle"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```



main.xml in layout-land

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Left"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true" />
    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Right"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true" />
    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom Left"
        android:layout_alignParentLeft="true"
        android:layout_alignParentBottom="true" />
    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom Right"
        android:layout_alignParentRight="true"
        android:layout_alignParentBottom="true" />
    <Button
        android:id="@+id/button5"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Middle"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />
    <Button
        android:id="@+id/button6"
        android:layout_width="180px"

        android:layout_height="wrap_content"
        android:text="Top Middle"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true"
        android:layout_alignParentTop="true" />
    <Button
        android:id="@+id/button7"
        android:layout_width="180px"
        android:layout_height="wrap_content"
        android:text="Bottom Middle"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true"
        android:layout_alignParentBottom="true" />
</RelativeLayout>
```



Try it out

- Page 130

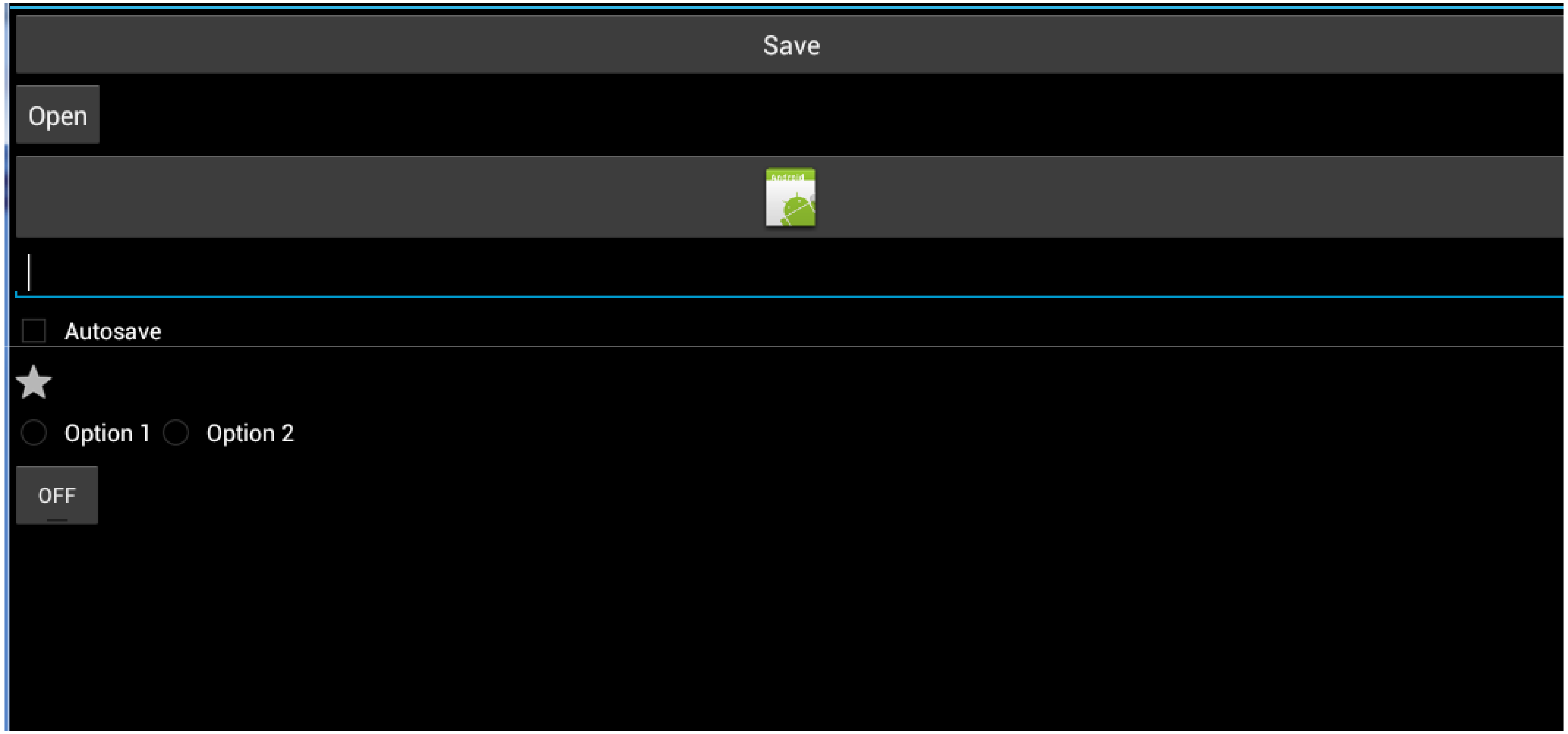
Views

- Import **android.view.View**.
- **Basic views** — Commonly used views such as the *TextView*, *EditText*, and *Button* views
- **Picker views** — Views that enable users to select from a list, such as the *TimePicker* and *DatePicker* views
- **List views** — Views that display a long list of items, such as the ListView and the SpinnerView views
- **Specialized fragments** — Special fragments that perform specific functions

Basic views

- TextView
- EditText
- Button
- ImageButton
- CheckBox
- ToggleButton
- RadioButton
- RadioGroup

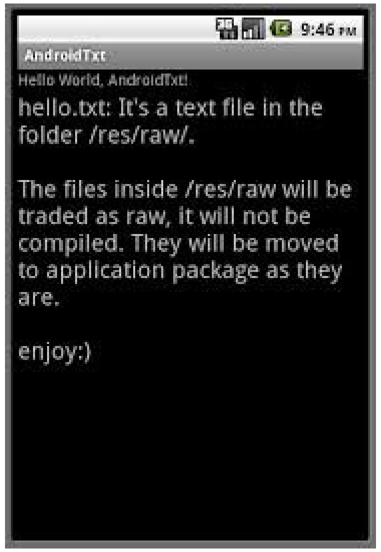
Basic views (Try it out Page 161)



TextView

- Displays text to the user and optionally allows them to edit it.
- A TextView is a complete text editor, however the basic class is configured to not allow editing
- http://developer.android.com/reference/android/widget/TextView.html

```
<?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:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />

</LinearLayout>
```



Hello World, AndroidTxt!
hello.txt: It's a text file in the folder /res/raw/.

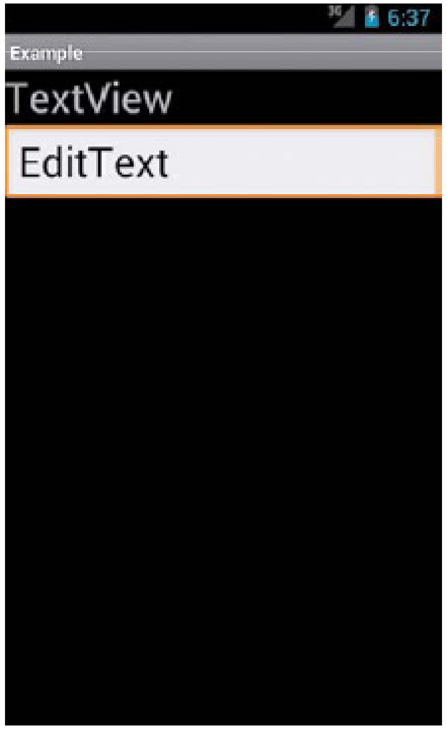
The files inside /res/raw will be traded as raw, it will not be compiled. They will be moved to application package as they are.

enjoy:)

EditText

- Subclass of TextView configured to allow the user to **edit** the text inside
- http://developer.android.com/reference/android/widget/EditText.html

```
<EditText android:id="@+id/txtName"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
```

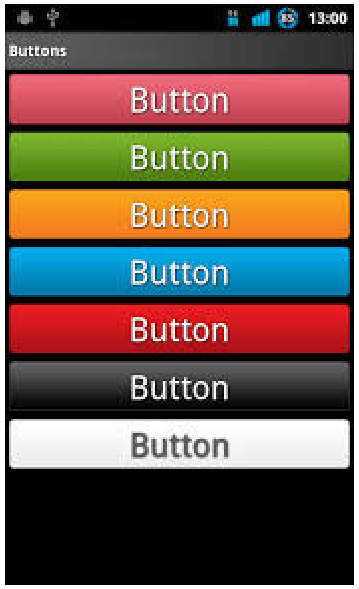


Button

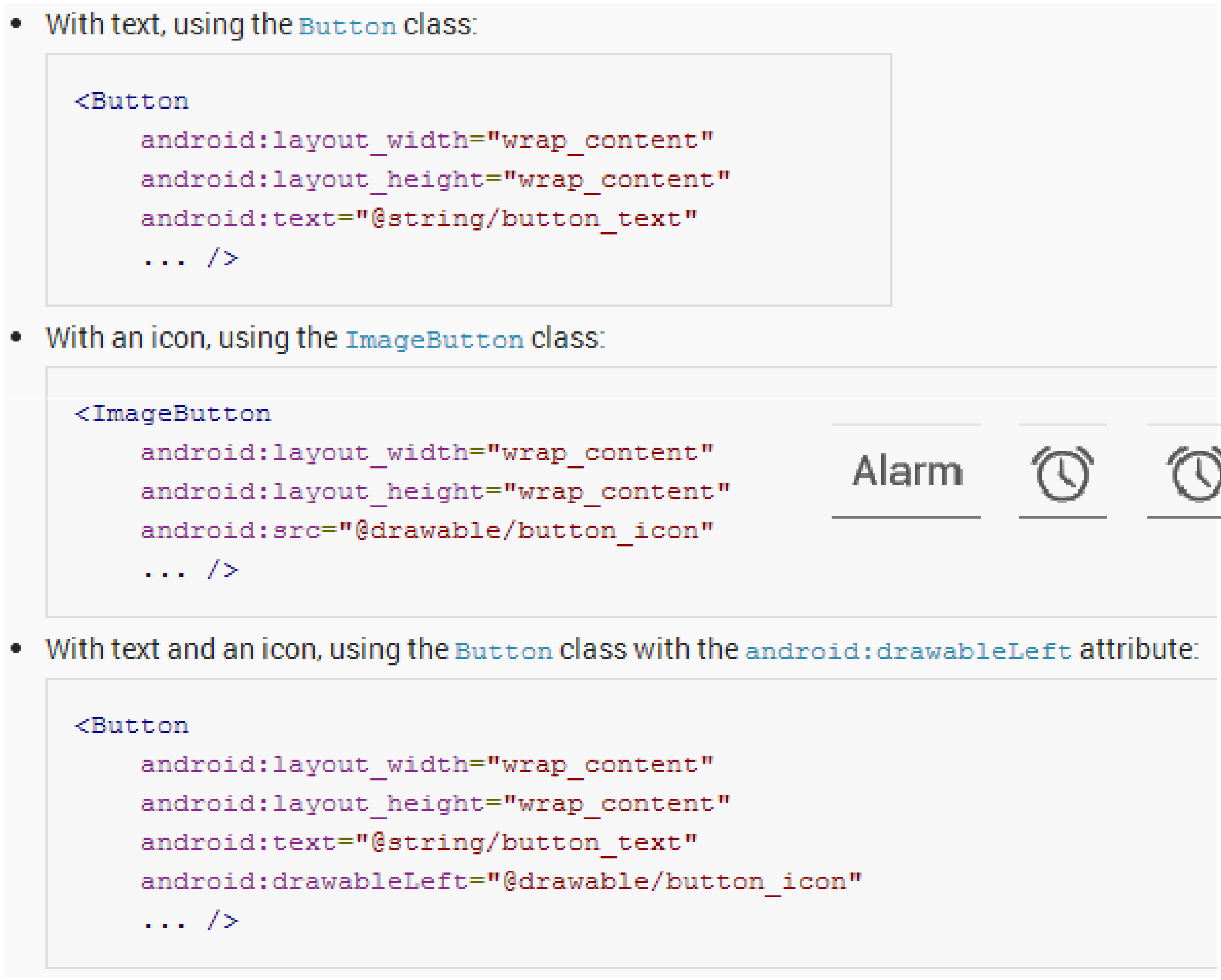
- Button
- Checkbox
- Radio button

Button

- Represent a **push-button** widget
- A button consists of text or an icon (or both text and an icon) that communicates what action occurs when the user touches it.
- Represents a push-button widget. Push-buttons can be **pressed**, or **clicked**, by the user to perform an action.
- http://developer.android.com/reference/android/widget/Button.html



Button



- With text, using the `Button` class:

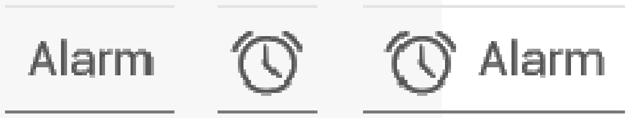
```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    ... />
```

- With an icon, using the `ImageButton` class:

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/button_icon"
    ... />
```

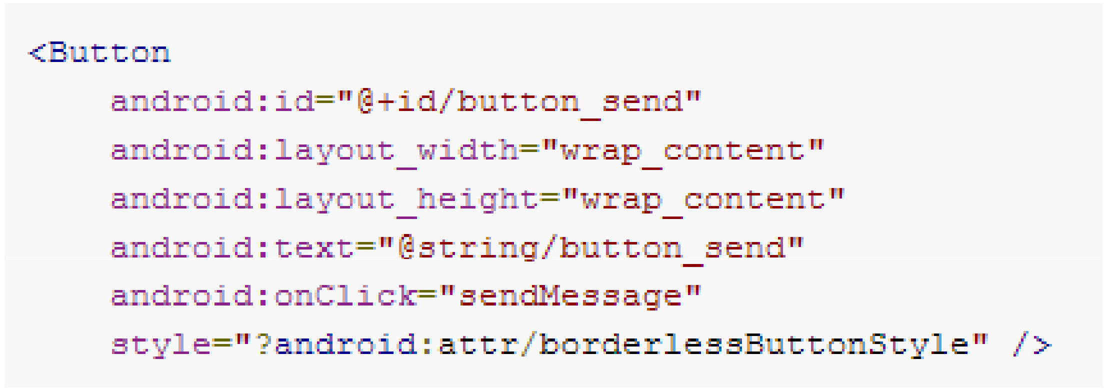
- With text and an icon, using the `Button` class with the `android:drawableLeft` attribute:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    android:drawableLeft="@drawable/button_icon"
    ... />
```



Styling buttons

- Borderless button



```
<Button
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage"
    style="?android:attr/borderlessButtonStyle" />
```

- Custom background: your background should be a state list resource that changes appearance depending on the button's current state.
- http://developer.android.com/guide/topics/ui/controls/button.html

Add a Button to a layout

- Each child of a LinearLayout appears on the screen in the order in which it appears in the XML.
- Define the button view in the layout file and assign it a unique ID:

```
<Button android:id="@+id/my_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/my_button_text"/>
```

Reference a button at the code

- Then create an instance of the view object and capture it from the layout (typically in the onCreate() method):

```
Button myButton = (Button) findViewById(R.id.my_button);
```

Responding to Click Events

- To define the click event handler for a button, add the android:onClick attribute to the <Button> element in your XML layout.

```
<?xml version="1.0" encoding="utf-8"?>
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

- Within the Activity that hosts this layout, the following method handles the click event: (public; Return void; Define a View as its only parameter

```
/** Called when the user touches the button */
public void sendMessage(View view) {
    // Do something in response to button click
}
```

Responding to Click Events

- The method you declare in the android:onClick attribute must:
  - Be **public**
  - Return **void**
  - Define a **View** as its only parameter (this will be the View that was clicked)
- http://developer.android.com/guide/topics/ui/controls/checkbox.html#HandlingEvents

Using an **OnClickListener**

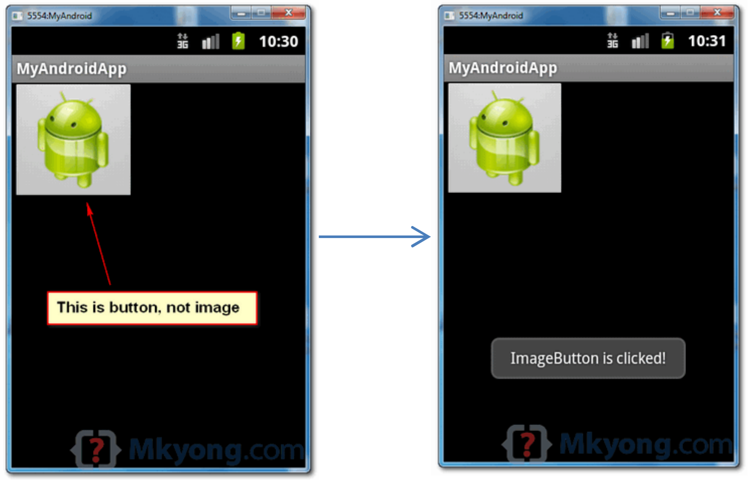
- Declare the click event handler programmatically rather than in an XML layout if you instantiate the Button at runtime or you need to declare the click behavior in a Fragment subclass.

```
public class MyActivity extends Activity {
    protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);

        setContentView(R.layout.content_layout_id);

        final Button button = (Button) findViewById(R.id.button_id);
        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                // Perform action on click
            }
        });
    }
}
```

ImageButton

- Similar to the Button view, except that it also displays an image

1. Add Image to Resources
   – Put image "**android_button.png**" into "*res/drawable-?dpi*" folder. So that Android know where to find your image.
2. Add ImageButton
3. Code the code

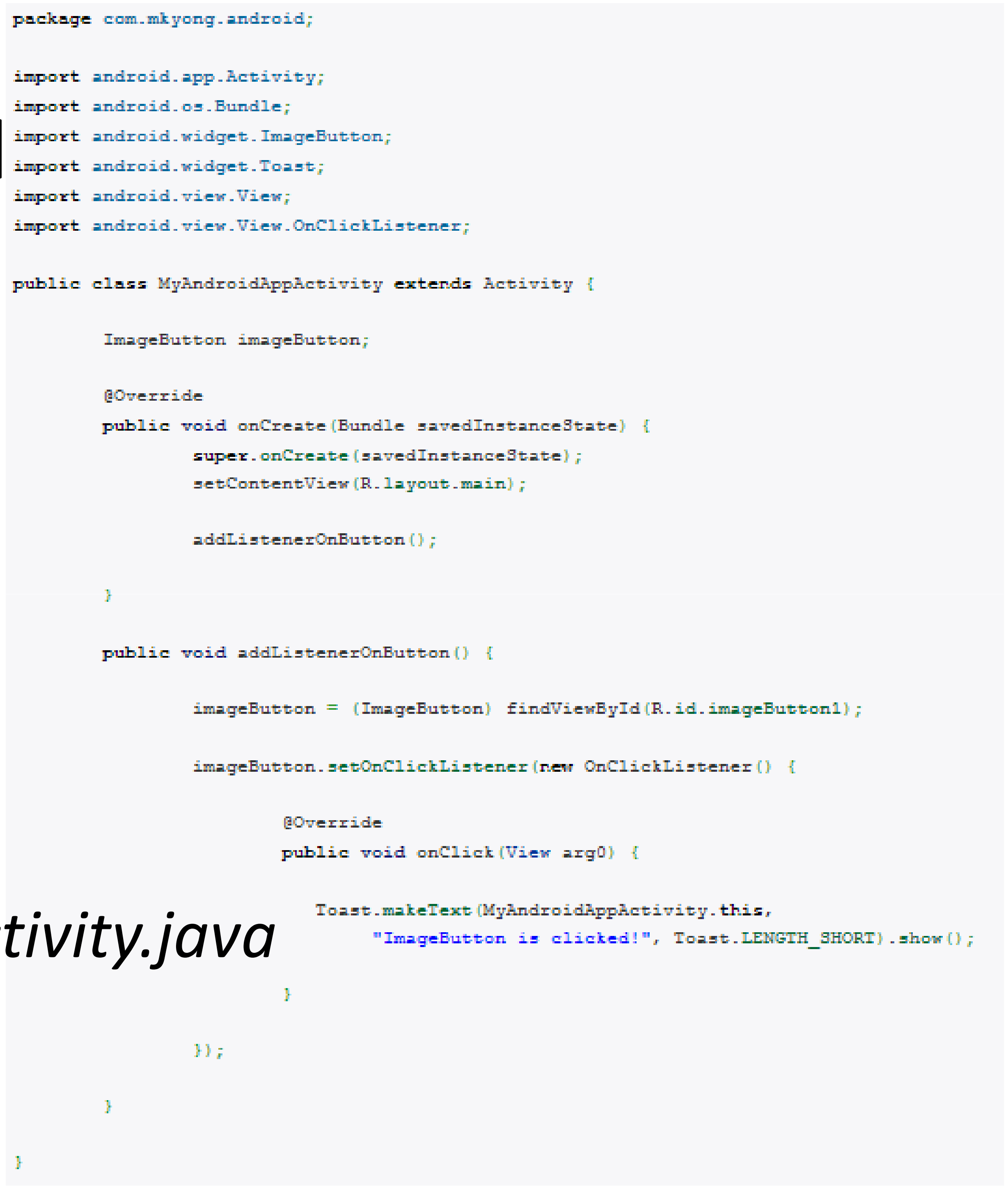


Add ImageButton

- Open “**res/layout/main.xml**” file, add a “ImageButton” tag, and defined the background image via “android:src“.

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

    <ImageButton
        android:id="@+id/imageButton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/android_button" />

</LinearLayout>
```

Code the cod

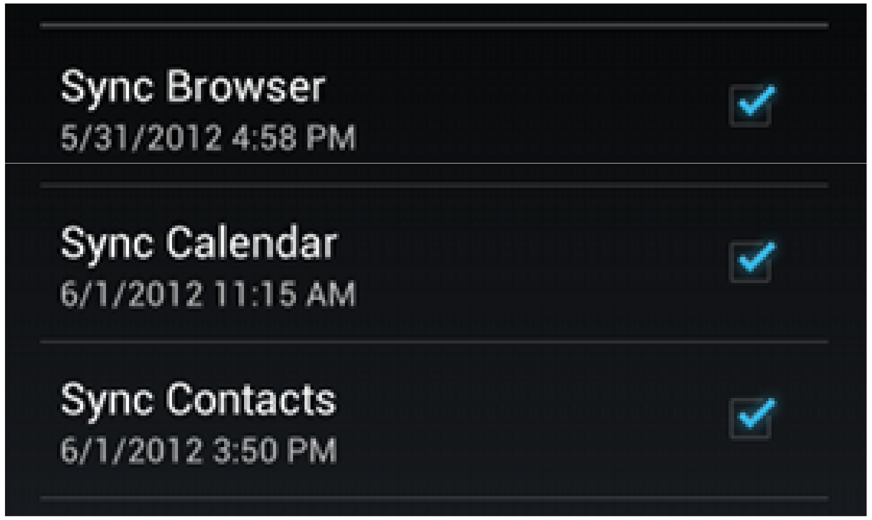


```
package com.mkyong.android;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ImageButton;
import android.widget.Toast;
import android.view.View;
import android.view.View.OnClickListener;

public class MyAndroidAppActivity extends Activity {

        ImageButton imageButton;

        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);

                addListenerOnButton();

        }

        public void addListenerOnButton() {

                imageButton = (ImageButton) findViewById(R.id.imageButton1);

                imageButton.setOnClickListener(new OnClickListener() {

                        @Override
                        public void onClick(View arg0) {

                           Toast.makeText(MyAndroidAppActivity.this,
                                "ImageButton is clicked!", Toast.LENGTH_SHORT).show();

                        }

                });

        }

}
```

- *MyAndroidAppActivity.java*

Checkbox

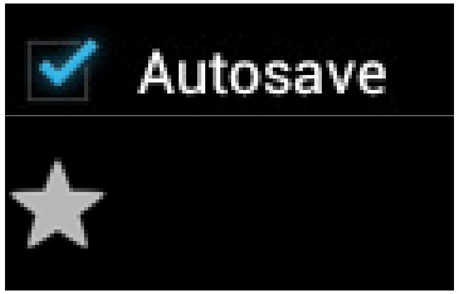
- A special type of button that has two states: checked or unchecked



Sync Browser
5/31/2012 4:58 PM

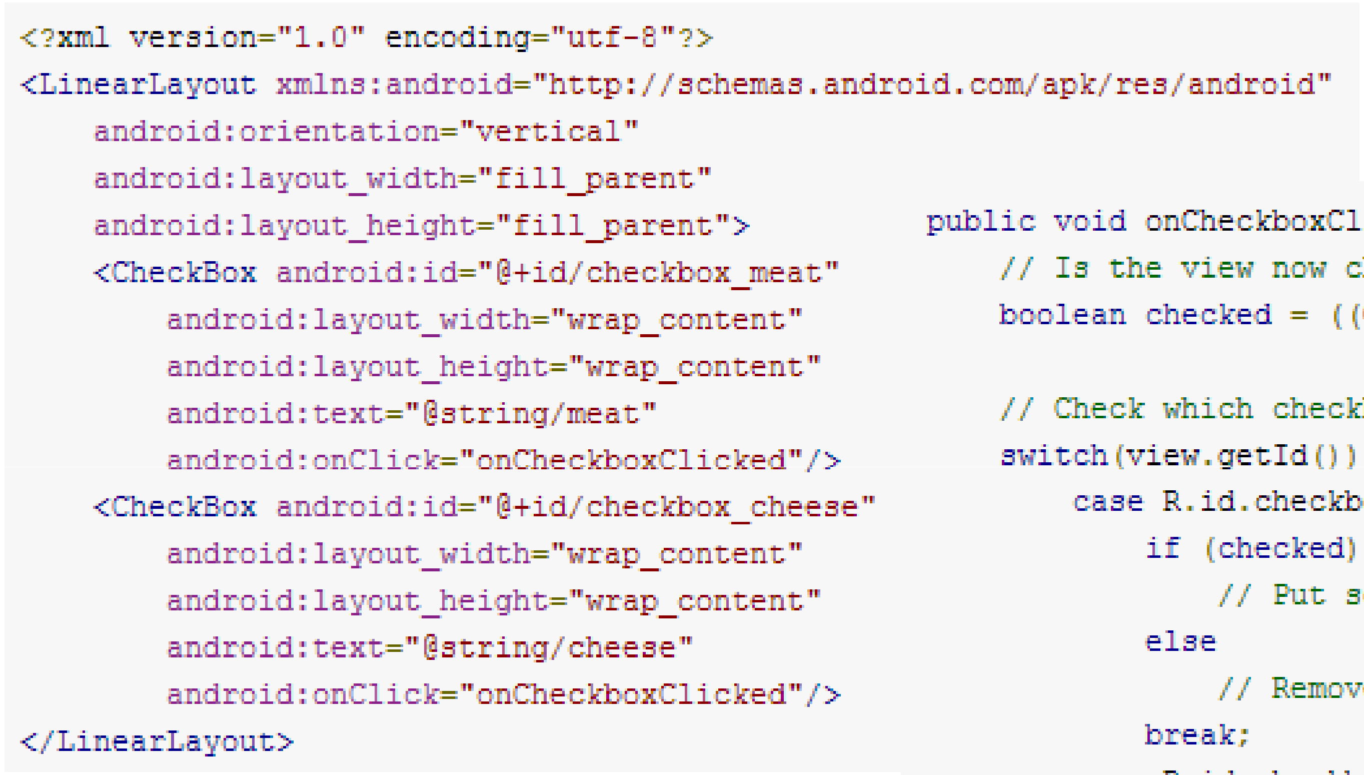
Sync Calendar
6/1/2012 11:15 AM

Sync Contacts
6/1/2012 3:50 PM

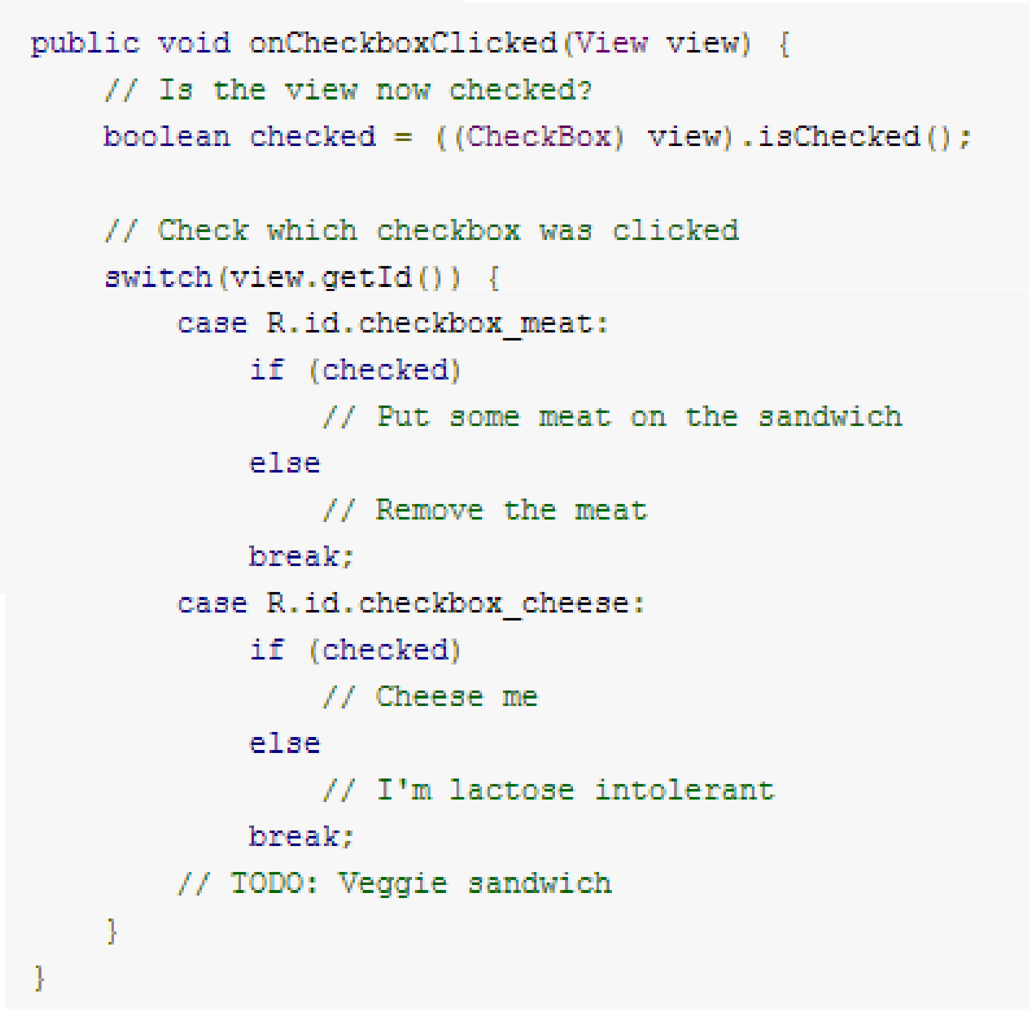


```
<CheckBox android:id="@+id/chkAutosave"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Autosave" />

<CheckBox android:id="@+id/star"
    style="?android:attr/starStyle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

Responding to Click Events



```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <CheckBox android:id="@+id/checkbox_meat"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/meat"
        android:onClick="onCheckboxClicked"/>
    <CheckBox android:id="@+id/checkbox_cheese"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cheese"
        android:onClick="onCheckboxClicked"/>
</LinearLayout>
```

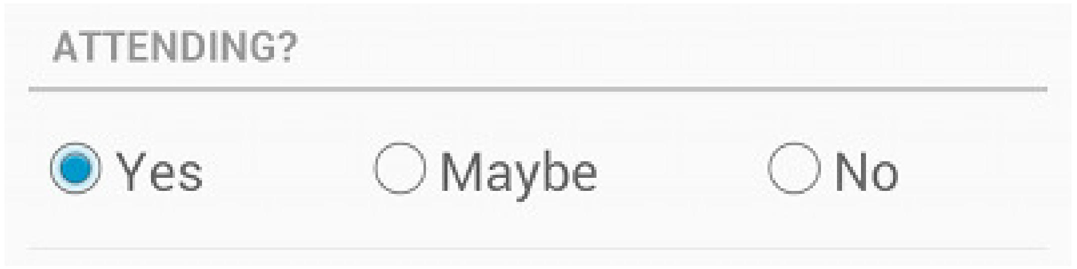


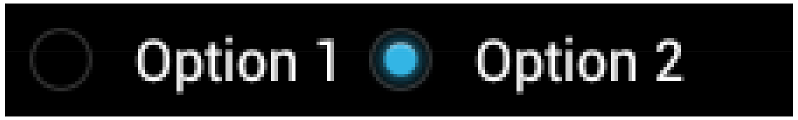
```java
public void onCheckboxClicked(View view) {
    // Is the view now checked?
    boolean checked = ((CheckBox) view).isChecked();

    // Check which checkbox was clicked
    switch(view.getId()) {
        case R.id.checkbox_meat:
            if (checked)
                // Put some meat on the sandwich
            else
                // Remove the meat
            break;
        case R.id.checkbox_cheese:
            if (checked)
                // Cheese me
            else
                // I'm lactose intolerant
            break;
        // TODO: Veggie sandwich
    }
}
```

RadioGroup & RadioButton

- The RadioButton has two states:
  - checked or unchecked.
- A RadioGroup is used to group together one or more RadioButton views, thereby allowing only one RadioButton to be checked within the RadioGroup.
- If not necessary to show all options side-by-side, use a spinner instead.
- http://developer.android.com/guide/topics/ui/controls/radiobutton.html#HandlingEvents

RadioGroup & RadioButton





```
<RadioGroup android:id="@+id/rdbGp1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical" >

    <RadioButton android:id="@+id/rdb1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Option 1" />

    <RadioButton android:id="@+id/rdb2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Option 2" />

</RadioGroup>
```

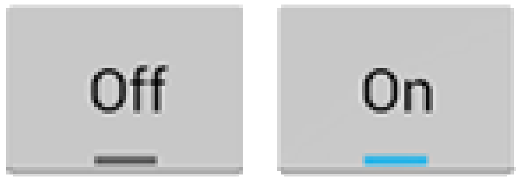
Responding to Click Events

```xml
<?xml version="1.0" encoding="utf-8"?>
<RadioGroup xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton android:id="@+id/radio_pirates"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pirates"
        android:onClick="onRadioButtonClicked"/>
    <RadioButton android:id="@+id/radio_ninjas"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/ninjas"
        android:onClick="onRadioButtonClicked"/>
</RadioGroup>
```

```java
public void onRadioButtonClicked(View view) {
    // Is the button now checked?
    boolean checked = ((RadioButton) view).isChecked();

    // Check which radio button was clicked
    switch(view.getId()) {
        case R.id.radio_pirates:
            if (checked)
                // Pirates are the best
            break;
        case R.id.radio_ninjas:
            if (checked)
                // Ninjas rule
            break;
    }
}
```

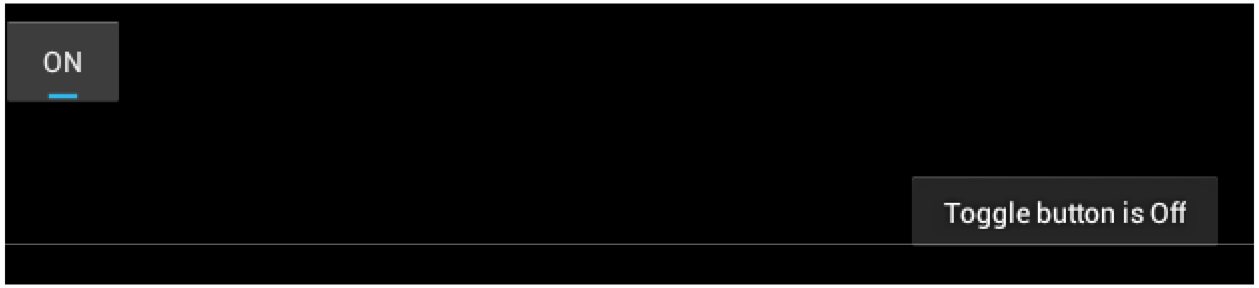
ToggleButton

- Change a setting between two states.
- Displays checked/unchecked states using a light indicator



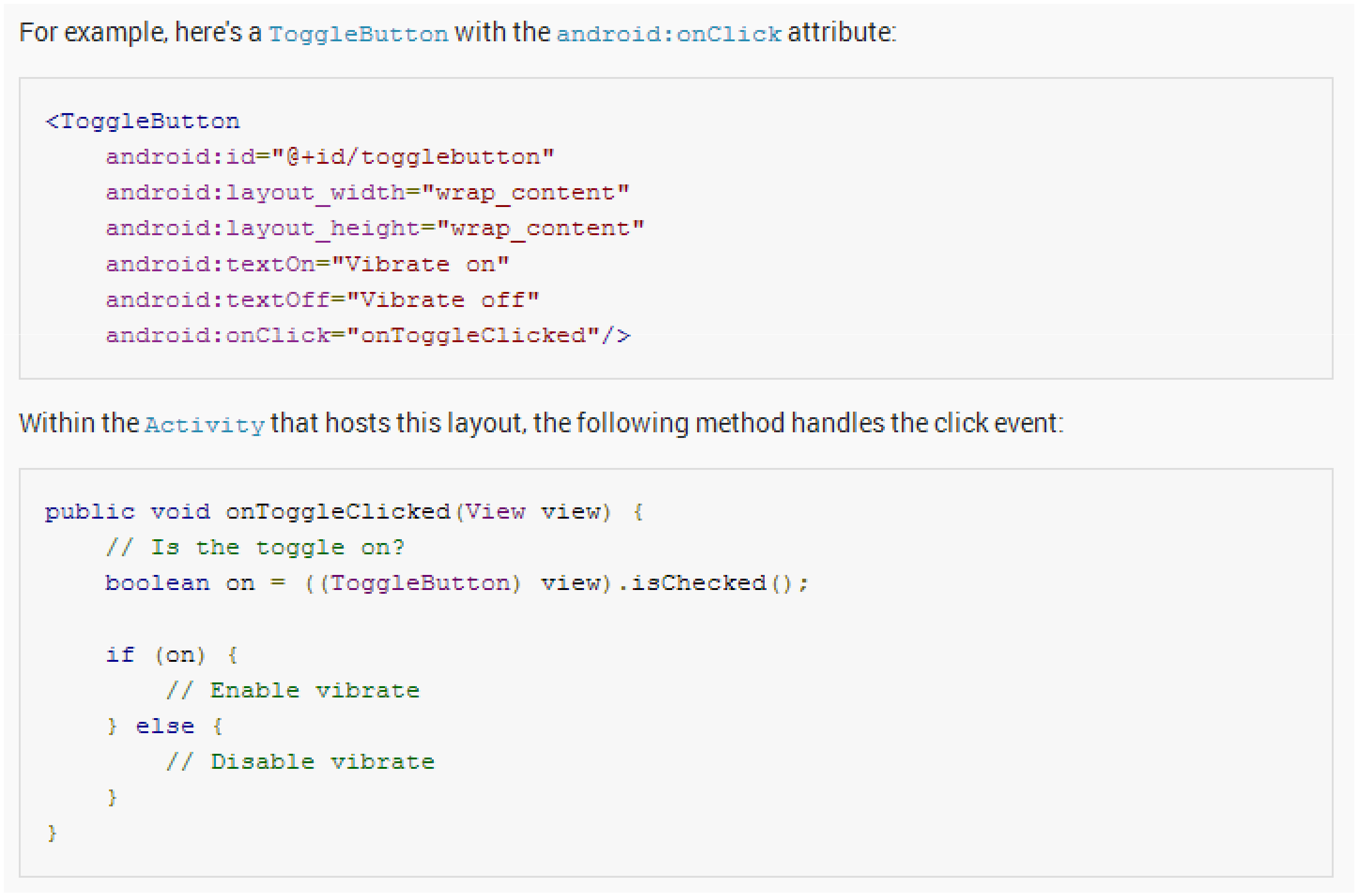
Switches (in Android 4.0+)

ToggleButton



```
<ToggleButton android:id="@+id/toggle1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

Responding to Click Events



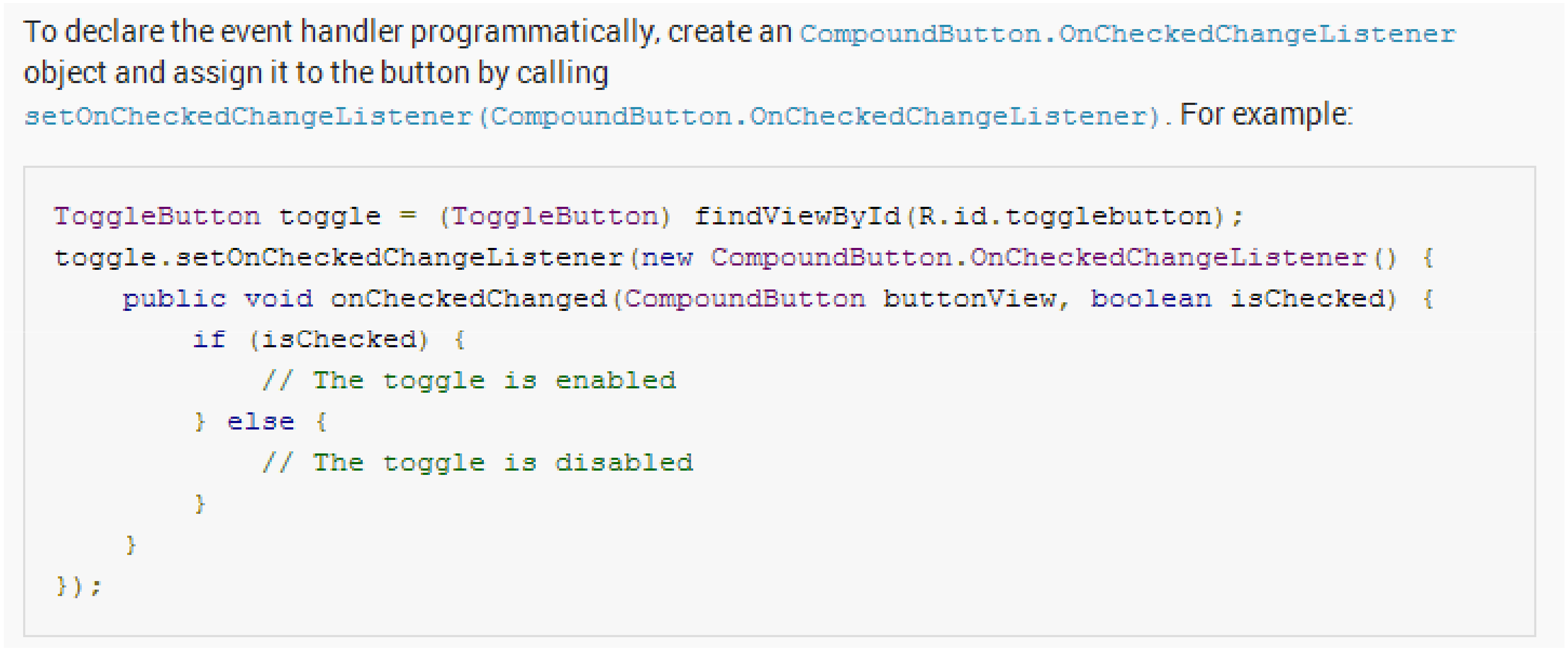
For example, here's a `ToggleButton` with the `android:onClick` attribute:

```
<ToggleButton
    android:id="@+id/togglebutton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="Vibrate on"
    android:textOff="Vibrate off"
    android:onClick="onToggleClicked"/>
```

Within the `Activity` that hosts this layout, the following method handles the click event:

```
public void onToggleClicked(View view) {
    // Is the toggle on?
    boolean on = ((ToggleButton) view).isChecked();

    if (on) {
        // Enable vibrate
    } else {
        // Disable vibrate
    }
}
```

Declare a click event handler programmatically rather than in an XML layout

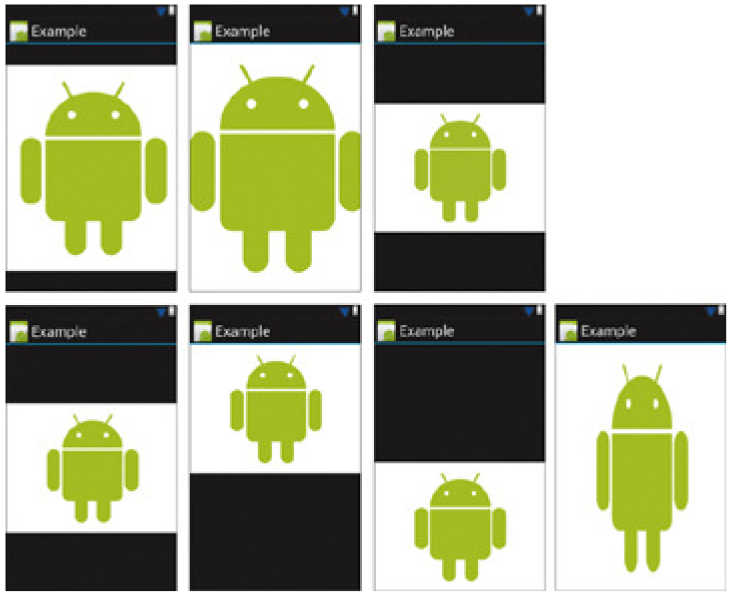


To declare the event handler programmatically, create an `CompoundButton.OnCheckedChangeListener` object and assign it to the button by calling `setOnCheckedChangeListener(CompoundButton.OnCheckedChangeListener)`. For example:

```
ToggleButton toggle = (ToggleButton) findViewById(R.id.togglebutton);
toggle.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
            // The toggle is enabled
        } else {
            // The toggle is disabled
        }
    }
});
```

ImageView

- Displays an **arbitrary image**, such as an icon.
- The ImageView class can load images from various sources (such as resources or content providers), takes care of computing its measurement from the image so that it can be used in any layout manager, and provides various display options such as scaling and tinting.
- More in Chapter 7



Try it out

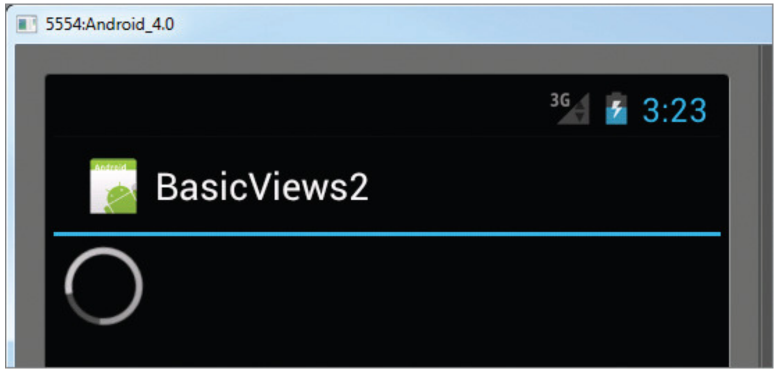
- Page 161 – Basic Views
- Page 168 – Handling View events

ProgressBar

- Provides **visual feedback** about some ongoing tasks, such as when you are performing a task in the background.
- For example, you might be downloading some data from the web and need to update the user about the status of the download. (determinate)
- A progress bar can also be made **indeterminate**. In indeterminate mode, the progress bar shows a cyclic animation without an indication of progress. This mode is used by applications when the length of the task is unknown. The indeterminate progress bar can be either a spinning wheel or a horizontal bar.

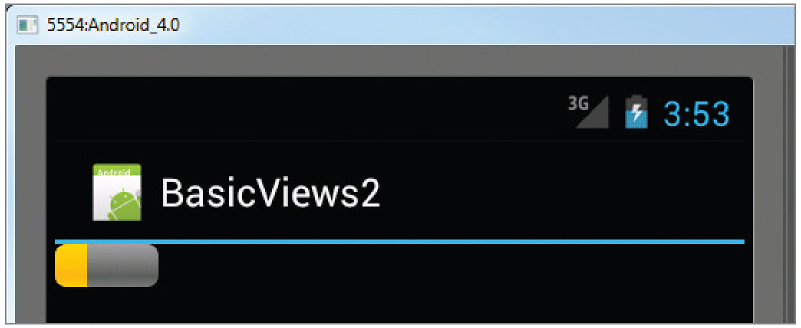
ProgressBar

```
<ProgressBar android:id="@+id/progressbar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```



Interdeterminate (default)

```
<ProgressBar android:id="@+id/progressbar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    style="@android:style/Widget.ProgressBar.Horizontal" />
```



determinate

ProgressBar

- Style constants for ProgressBar:
  - Widget.ProgressBar.Horizontal
  - Widget.ProgressBar.Small
  - Widget.ProgressBar.Large
  - Widget.ProgressBar.Inverse
  - Widget.ProgressBar.Small.Inverse
  - Widget.ProgressBar.Large.Inverse

Indeterminate

```
<ProgressBar android:id="@+id/progressbar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

```
static int progress;
ProgressBar progressBar;
int progressStatus = 0;
Handler handler = new Handler();

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    progress = 0;
    progressBar = (ProgressBar) findViewById(R.id.progressbar);

    //---do some work in background thread---
    new Thread(new Runnable()
    {
        public void run()
        {
            //---do some work here---
            while (progressStatus < 10)
            {
                progressStatus = doSomeWork();
            }

            //---hides the progress bar---

            handler.post(new Runnable()
            {
                public void run()
                {
                    //---0 - VISIBLE; 4 - INVISIBLE; 8 - GONE---
                    progressBar.setVisibility(View.GONE);
                }
            });
        }

        //---do some long running work here---
        private int doSomeWork()
        {
            try {
                //---simulate doing some work---
                Thread.sleep(500);
            } catch (InterruptedException e)
            {
                e.printStackTrace();
            }
            return ++progress;
        }
    }).start();
}
```

determinate

```xml
<ProgressBar android:id="@+id/progressbar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    style="@android:style/Widget.ProgressBar.Horizontal
```

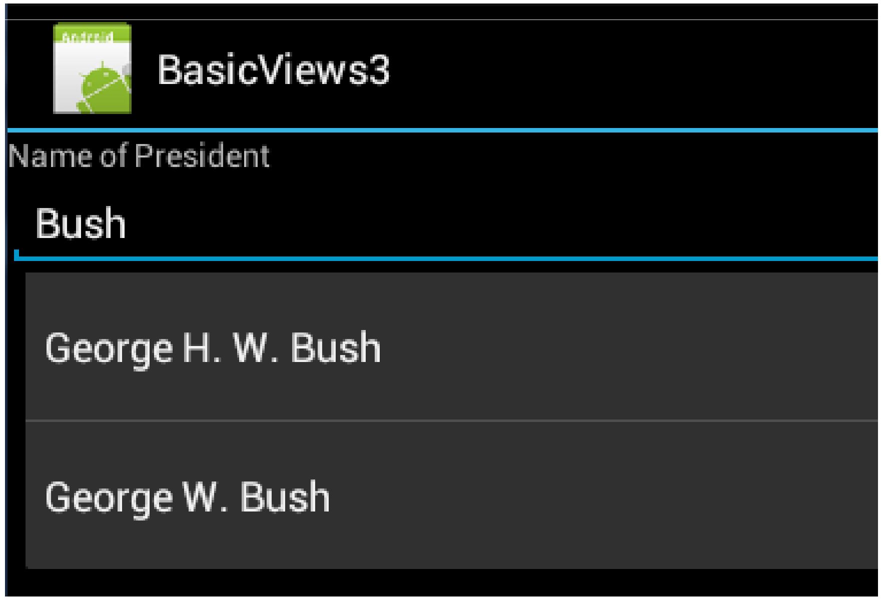
```java
progress = 0;
progressBar = (ProgressBar) findViewById(R.id.progressbar);
progressBar.setMax(200);

//---do some work in background thread---
new Thread(new Runnable()
{
    public void run()
    {
        //—-do some work here—-
        while (progressStatus < 100)
        {
            progressStatus = doSomeWork();

            //—-Update the progress bar—-
            handler.post(new Runnable()
            {
                public void run() {
                    progressBar.setProgress(progressStatus);
                }
            });
        }

        //---hides the progress bar---
        handler.post(new Runnable()
        {
            public void run()
            {
                //---0 - VISIBLE; 4 - INVISIBLE; 8 - GONE---
                progressBar.setVisibility(View.GONE);
            }
        });
    }

    //---do some long running work here---
    private int doSomeWork()
    {
        try {
            //---simulate doing some work---
            Thread.sleep(50);
        } catch (InterruptedException e)
        {
            e.printStackTrace();
        }
        return ++progress;
    }
}).start();
```

Try it out

- Page 172
- Page 174

AutoCompleteTextView

- Subclass of EditText
- It shows a list of completion suggestions automatically while the user is typing.



```xml
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Name of President" />

<AutoCompleteTextView android:id="@+id/txtCountries"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
```

```java
import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;

public class BasicViews3Activity extends Activity {
    String[] presidents = {
            "Dwight D. Eisenhower",
            "John F. Kennedy",
            "Lyndon B. Johnson",
            "Richard Nixon",
            "Gerald Ford",
            "Jimmy Carter",
            "Ronald Reagan",
            "George H. W. Bush",
            "Bill Clinton",
            "George W. Bush",
            "Barack Obama"
        };

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
            android.R.layout.simple_dropdown_item_1line, presidents);

        AutoCompleteTextView textView = (AutoCompleteTextView)
            findViewById(R.id.txtCountries);

        textView.setThreshold(3);
        textView.setAdapter(adapter);
    }
}
```

Try it out

- Page 177

Chapter 6 – more on UI

- Using the picker and list views
- Using specialized fragments
- Listen for UI notifications
- Create UI programmatically