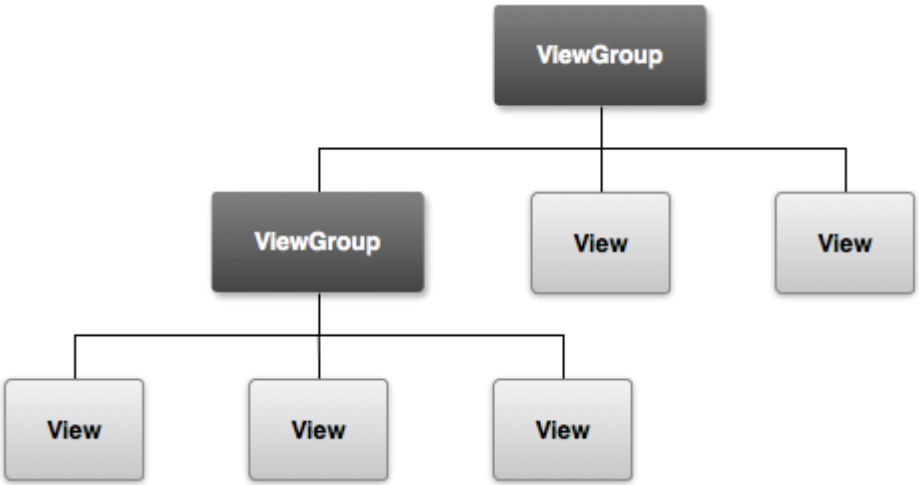
Mobile Application Development

User Interface Basics

The View Class

- All user interface elements in an Android app are built using View and ViewGroup objects.
- A **View** is an object that draws something on the screen that the user can interact with.
  - Examples: buttons and text fields
- A **ViewGroup** is an object that holds other View (and ViewGroup) objects in order to define the layout of the interface.
  - Examples: linear or relative layout



Graphical UI to XML Layout

Actual UI displayed by the app

Text version: *activity_main.xml* file →

```xml
<RelativeLayout
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:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="35dp"
        android:layout_marginTop="35dp"
        android:ems="10"
        android:hint="Enter your NAME here" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/editText1"
        android:layout_below="@+id/editText1"
        android:layout_marginLeft="54dp"
        android:layout_marginTop="26dp"
        android:text="Go" />

</RelativeLayout>
```

UI Via XML

- Each Screen in your app will likely have an xml layout file describes the container and widgets on the screen / UI
- Edit XML file or use drag and drop editor
- Alter container and layout attributes for the set up you want
- We can access and manipulate the container and widgets in our Java code associated with the UI / screen.

Using Views

Dealing with widgets & layouts typically involves the following operations:

1. **Set properties**: For example setting the background color, text, font and size of a TextView.
2. **Set up listeners**: For example, an image could be programmed to respond to various events such as: click, long-tap, mouse-over, etc.
3. **Set focus**: To set focus on a specific view, you call the method requestFocus() or use XML tag <requestFocus />
4. **Set visibility**: You can hide or show views using setVisibility(...).

Common Layouts

Linear Layout

A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.

Relative Layout

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).

Web View

Displays web pages.

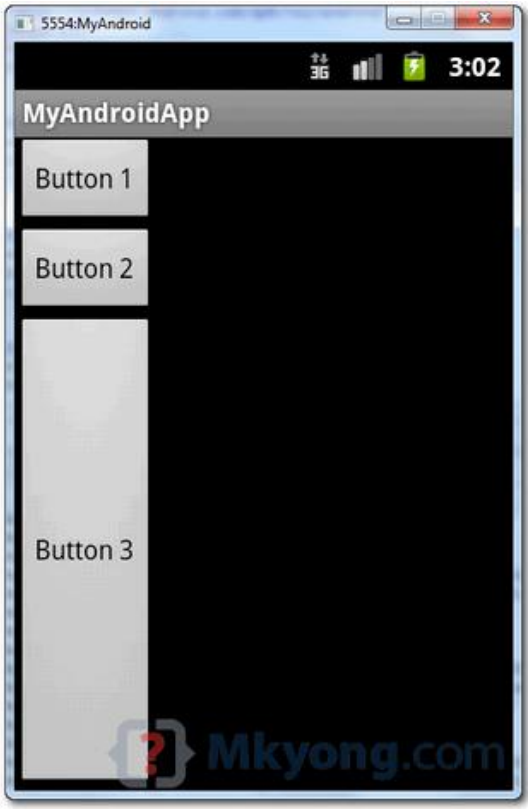
Linear Layout: Horizontal

```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="horizontal" >

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 1" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 2" />

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 3"
        android:layout_weight="1"/>

</LinearLayout>
```

File: res/layout/activity_main1.xml

Linear Layout: Horizontal

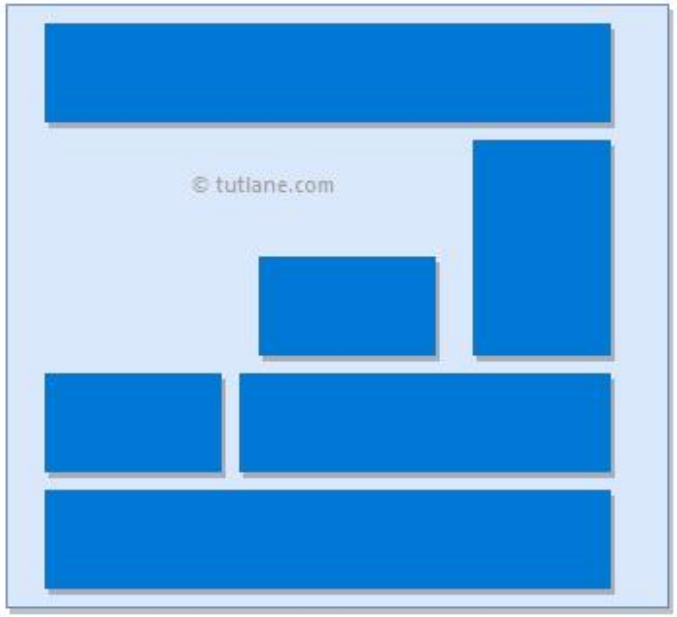
```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 1" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 2" />

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 3"
        android:layout_weight="1"/>

</LinearLayout>
```



res/layout/activity_main2.xml

Relative Layout

In android, **RelativeLayout** is a **ViewGroup** which is used to specify the position of child **View** instances relative to each other (Child **A** to the left of Child **B**) or relative to the parent (Aligned to the top of parent).



res/layout/activity_main2.xml

Relative Layout

Attributes to control the relative position of views within a *Relative Layout*.

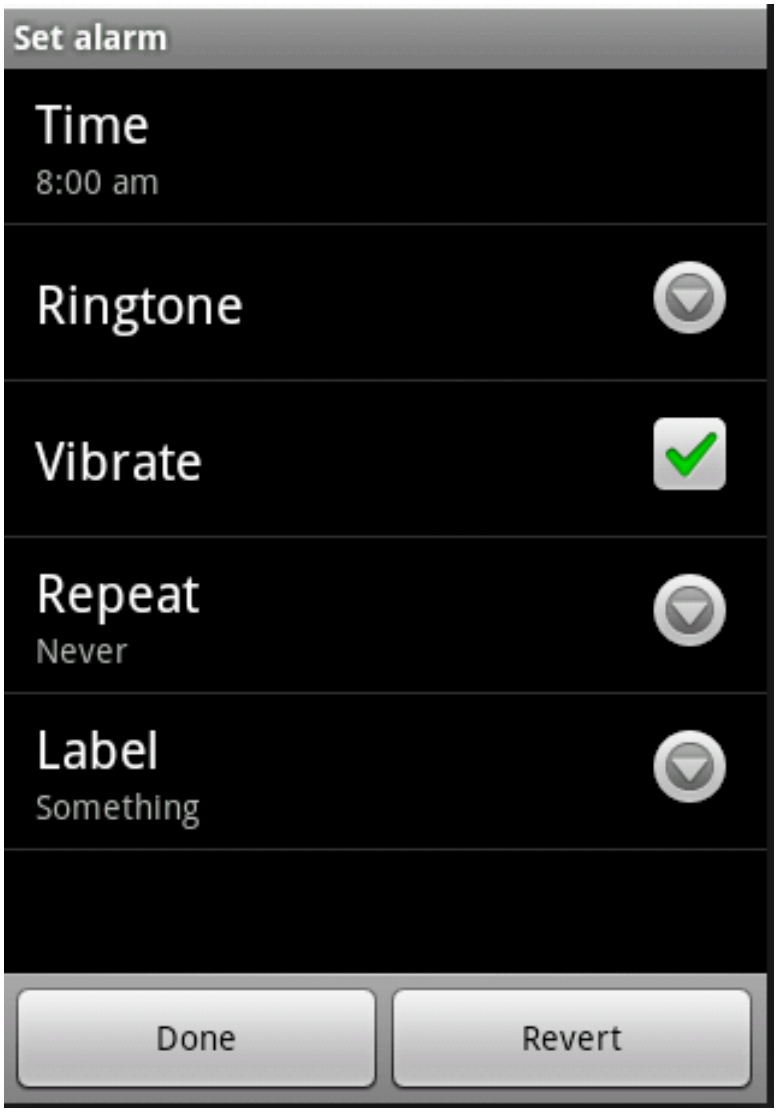
Attribute	Description
layout_alignParentTop	If **true**, the top edge of view will match the top edge of parent.
layout_alignParentBottom	If **true**, the bottom edge of view will match the bottom edge of parent.
layout_alignParentLeft	If **true**, the left edge of view will match the left edge of parent.
layout_alignParentRight	If **true**, the right edge of view will match the right edge of parent.
layout_centerInParent	If **true**, the view will be aligned to centre of parent.
layout_centerHorizontal	If **true**, the view will be horizontally centre aligned within its parent.
layout_centerVertical	If **true**, the view will be vertically centre aligned within its parent.
layout_above	It places the current view above the specified view id.
layout_below	It places the current view below the specified view id.
layout_toLeftOf	It places the current view left of the specified view id.
layout_toRightOf	It places the current view right of the specified view id.
layout_toStartOf	It places the current view to start of the specified view id.
layout_toEndOf	It places the current view to end of the specified view id.

UI Programming with Widgets

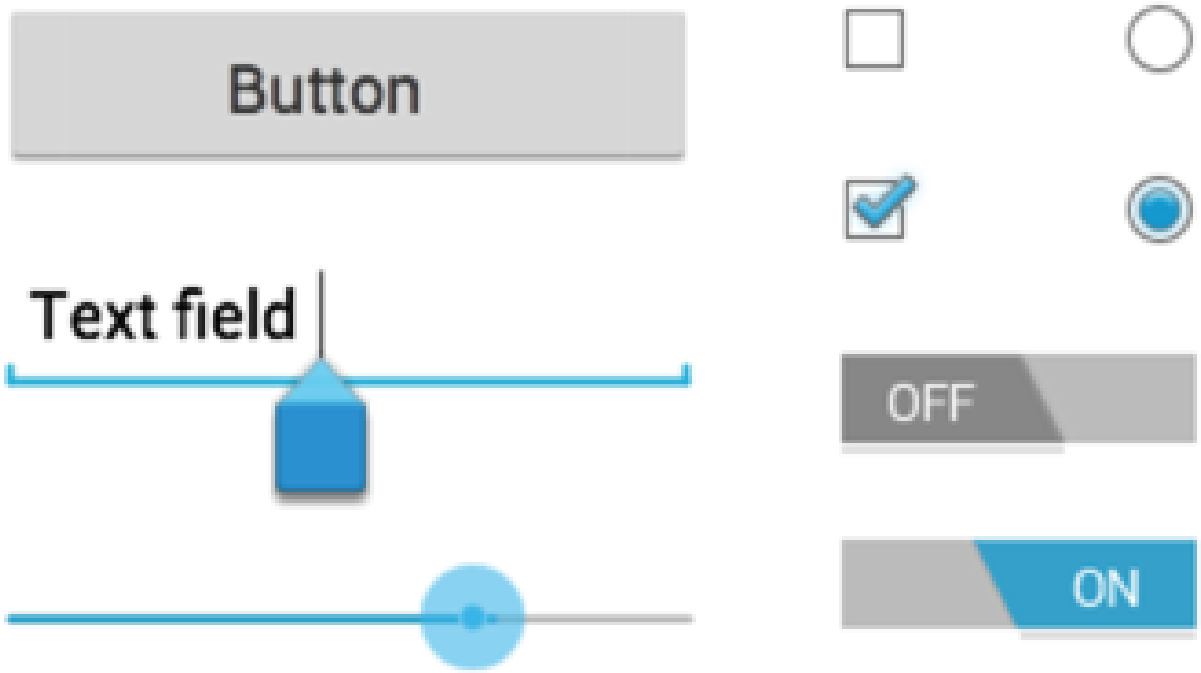
- Widgets are an element in a Graphical User Interface (GUI)
  - not to be confused with app widgets placed on the home screen, mini version of app
- Widgets are building blocks
- User interacts with a given widget
- **Often** use prebuilt widgets
  - Advanced developers create their own (Chris Renke, Square)

Widgets

- Including:
- Text Views
- Buttons
- Check Boxes
- Spinners (drop down menus)
- and many, many more

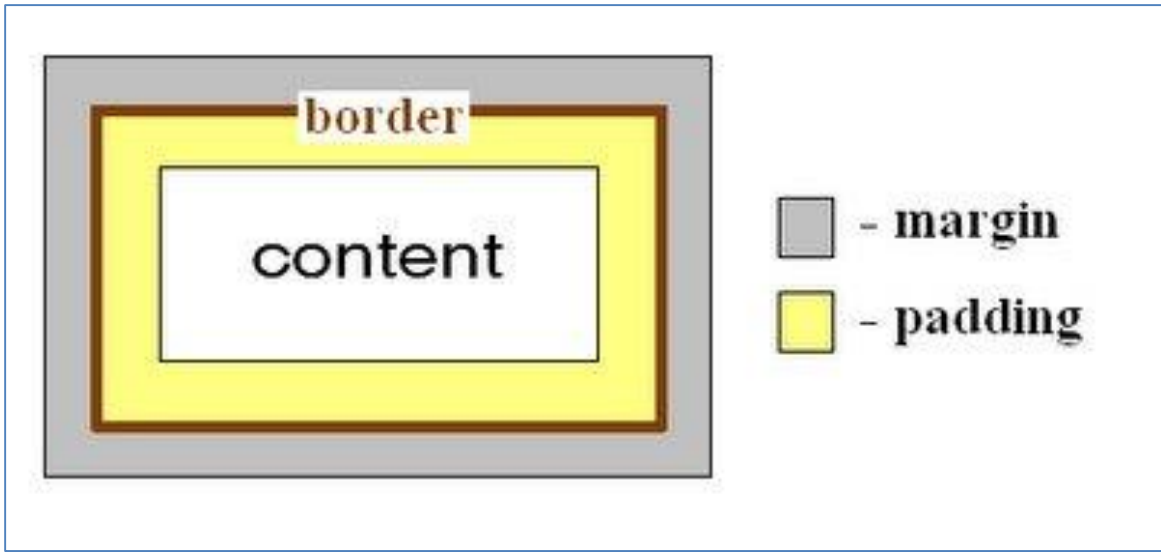


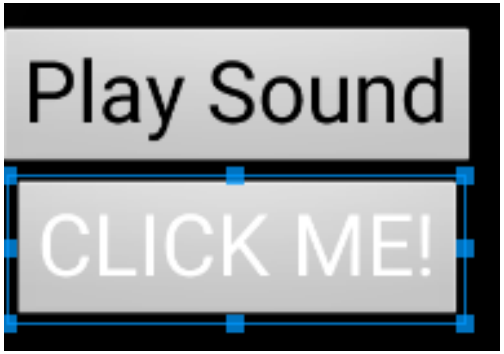
Common Widgets for Input Control



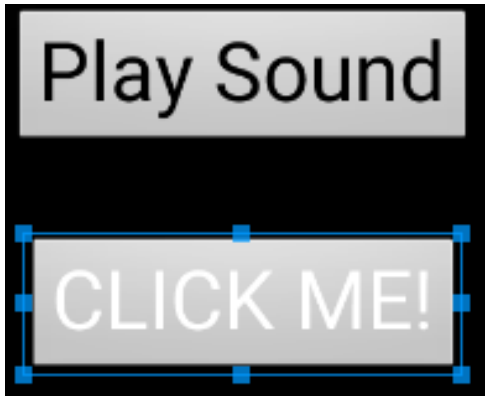
Widget Attributes

- Size
  - layout width
  - layout height
- Margin
- Padding





No specified margin
or padding



Top Margin of 30dp
(density independent pixels)

Top Margin of 30dp,
padding of 20dp

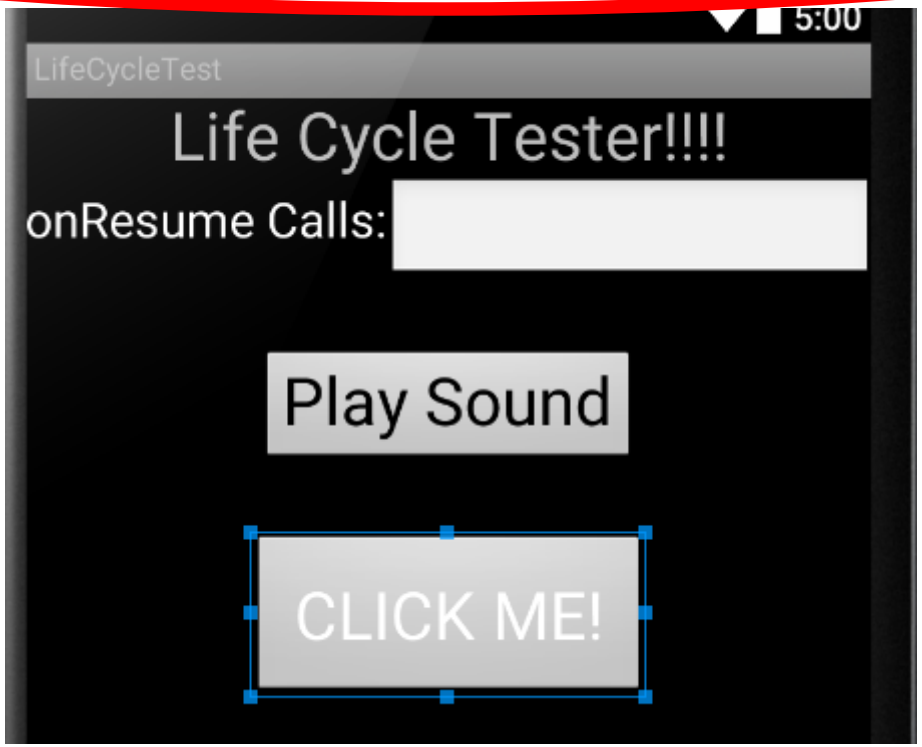
Size

Three options:

- Specified (hard coded) size in dp, density independent pixels
- wrap_content
  - widget is just big enough to show content inside the widget (text, icon)
- match_parent
  - match my parent's size
  - widgets stored in a *container or ViewGroup*

Size - Wrap Content

```
<Button
    android:id="@+id/clickForActivityButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```



Size - Match Parent

```
<Button
    android:id="@+id/clickForActivityButton"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
```

Attributes

a lot of attributes

XML Attributes		
Attribute Name	*Related Method*	*Description*
android:baselineAligned	setBaselineAligned(boolean)	When set to false, prevents the layout from aligning its children's baselines.
android:baselineAlignedChildIndex	setBaselineAlignedChildIndex(int)	When a linear layout is part of another layout that is baseline aligned, it can specify which of its children to baseline align to (that is, which child TextView).
android:divider	setDividerDrawable(Drawable)	Drawable to use as a vertical divider between buttons.
android:gravity	setGravity(int)	Specifies how to place the content of an object, both on the x- and y-axis, within the object itself.
android:measureWithLargestChild	setMeasureWithLargestChildEnabled(boolean)	When set to true, all children with a weight will be considered having the minimum size of the largest child.
android:orientation	setOrientation(int)	Should the layout be a column or a row? Use "horizontal" for a row, "vertical" for a column.
android:weightSum		Defines the maximum weight sum.

attributes can be set in the xml and most can changed programmatically

```
<Button
    android:id="@+id/clickForActivityButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_marginTop="30dp"
    android:onClick="getName"
    android:padding="20dp"
    android:text="@string/clickForActivityButtonTitle"
    android:textColor="#FFF"
    android:textSize="30sp" />
```

in layout xml file

```
private void changeButtonPadding() {
    Button b = (Button) findViewById(R.id.clickForActivityButton)
    b.setPadding(20, 15, 20, 15);
}
```

Programmatically in Activity (Java code)

in program

Dimension of Widgets

A dimension value defined in XML. A dimension is specified with a number followed by a unit of measure. For example: 10px, 2in, 5sp.

The following units of measure are supported by Android:

- dp : density-independent pixels
- sp : scale-independent pixels
- pt : points – 1/72 of an inch
- px : Actual pixels; advice not to use.
- mm : millimeters
- in : inches

Dimensions: 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 (dots per inch) screen, on which 1dp is roughly equal to 1px.

When running on a higher density screen, the number of pixels used to draw 1dp is scaled up by a factor appropriate for the screen's dpi. Likewise, when on a lower density screen, the number of pixels used for 1dp is scaled down.

Using dp units (instead of px units) is a simple solution to making the view dimensions in your layout resize properly for different screen densities. In other words, it provides consistency for the real-world sizes of your UI elements across different devices.

Dimensions: 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 the user's preference.

Dimensions: pt

Points

1/72 of an inch based on the physical size of the screen, assuming a 72dpi density screen.

Dimensions: px

Pixels

Corresponds to actual pixels on the screen. This unit of measure is not recommended because the actual representation can vary across devices; each devices may have a different number of pixels per inch and may have more or fewer total pixels available on the screen.

Dimensions: mm, in

mm : Millimeters - Based on the physical size of the screen.

In : Inches - Based on the physical size of the screen.

TYPES OF WIDGETS

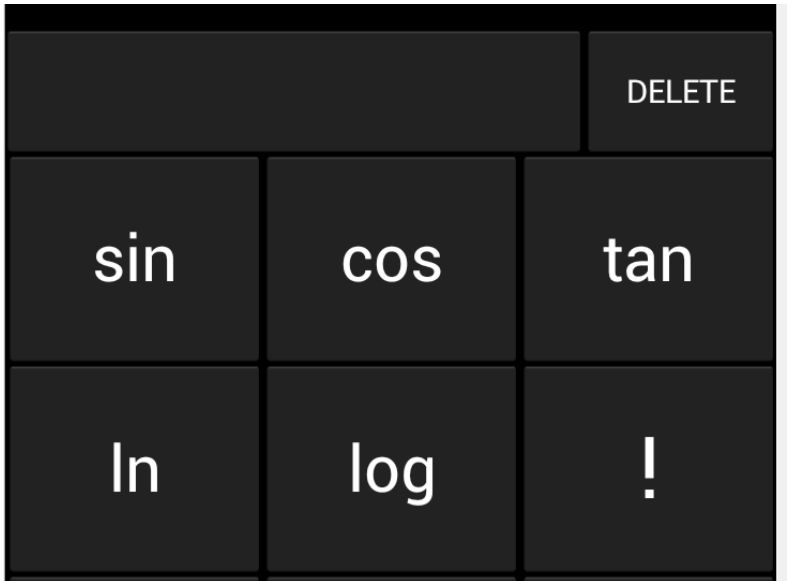
Common Controls - TextView

- a simple label
- display information, not for interaction
- common attributes: width, height, padding, visibility, text size, text color, background color
  - units for width / height: px (pixels), dp or dip (density-independent pixels 160 dpi base), sp (scaled pixels based on preferred font size), in (inches), mm (millimeters)
  - recommended units: sp for font sizes and dp for everything else
  - http://developer.android.com/guide/topics/resources/more-resources.html#Dimension

Common Controls - Button

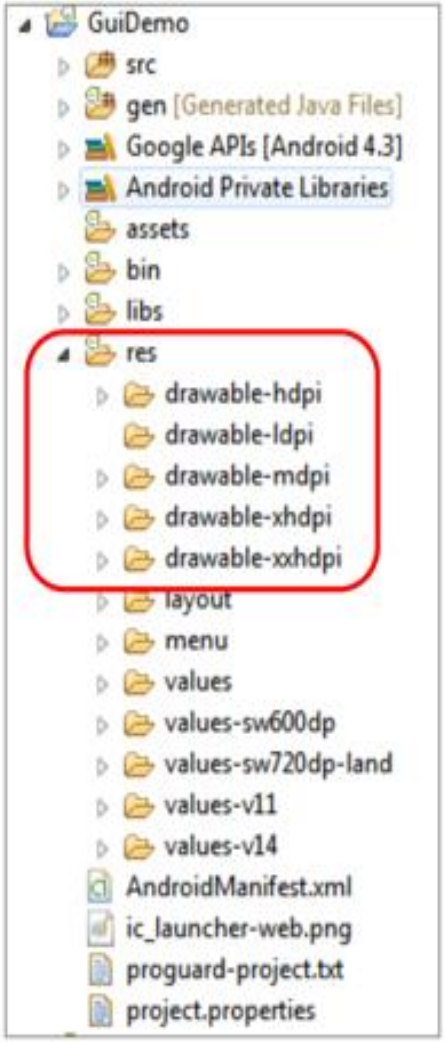
- Text or icon or both on View
- button press triggers some action
  - set android:onClick attribute in XML file
  - OR create a ClickListener object, override onClick method, and register it with the checkbox
    - typically done with anonymous inner class
  - possible to customize appearance of buttons

    http://developer.android.com/guide/topics/ui/controls/button.html#CustomBackground



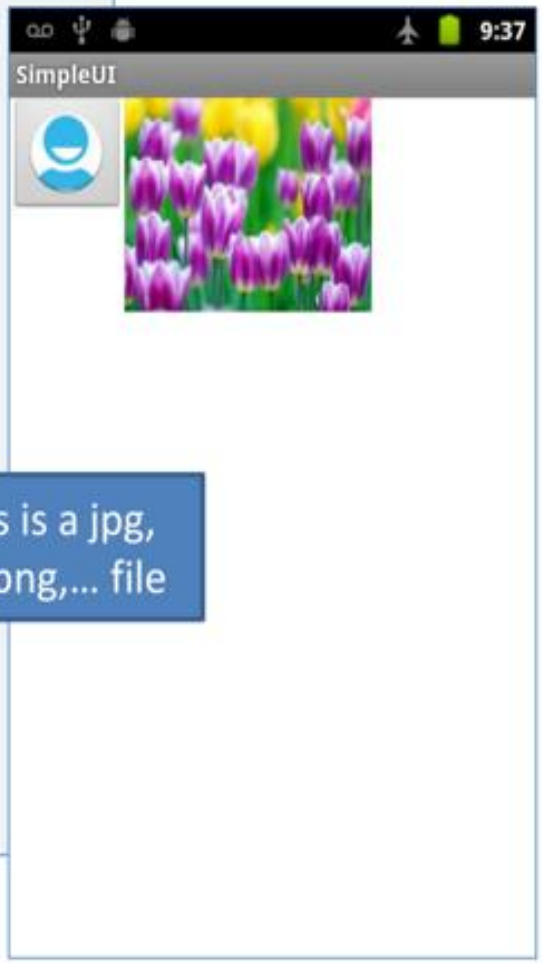
Basic Widgets: Images

- **ImageView** and **ImageButton** are two Android widgets that allow embedding of images in your applications.
- Analogue to *TextView* and *Button* controls (respectively).
- Each widget takes an `android:src` or `android:background` attribute (in an XML layout) to specify what picture to use.
- Pictures are usually stored in the **res/drawable** folder (optionally a low, medium, and high definition version of the same image could be stored to later be used with different types of screens)



Basic Widgets: Images

```
<LinearLayout
    . . .

    <ImageButton
        android:id="@+id/myImageBtn1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_launcher" >
    </ImageButton>

    <ImageView
        android:id="@+id/myImageView1"
        android:layout_width="150dp"
        android:layout_height="120dp"
        android:scaleType="fitXY"
        android:src="@drawable/flower1" >
    </ImageView>

</LinearLayout>
```

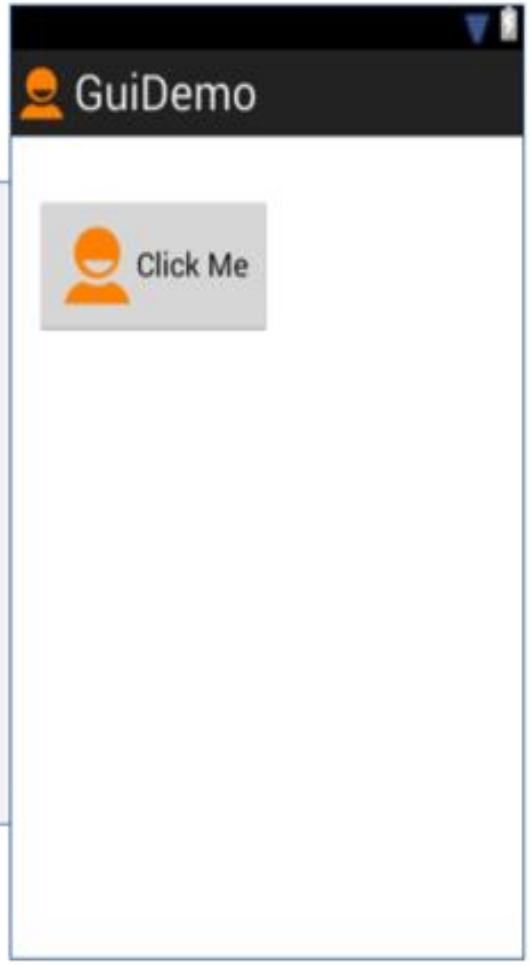


This is a jpg, gif, png,... file

Basic Widgets: Combining Images & Text

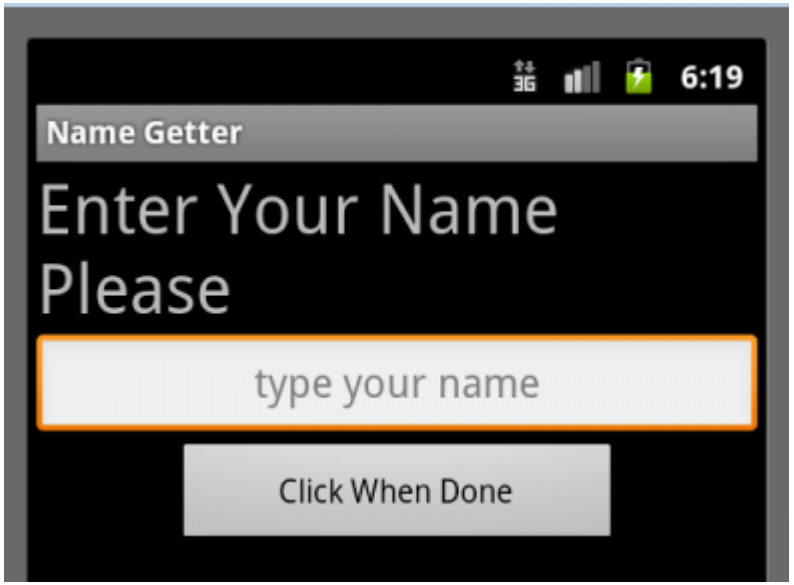
A common **Button** could display text and a simple image as shown below

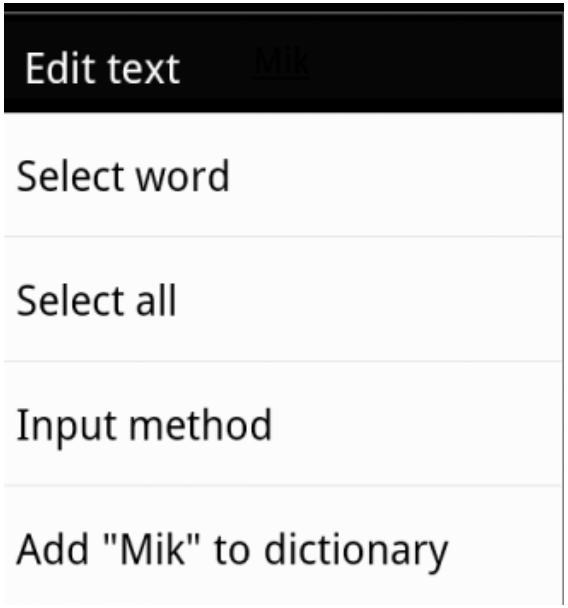
```
<LinearLayout
    . . .

 <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:drawableLeft="@drawable/ic_happy_face"
        android:gravity="left|center_vertical"
        android:padding="15dp"
        android:text="@string/click_me" />


</LinearLayout>
```



Common Controls - EditText

- Common component to get information from the user
- long press brings up context menu





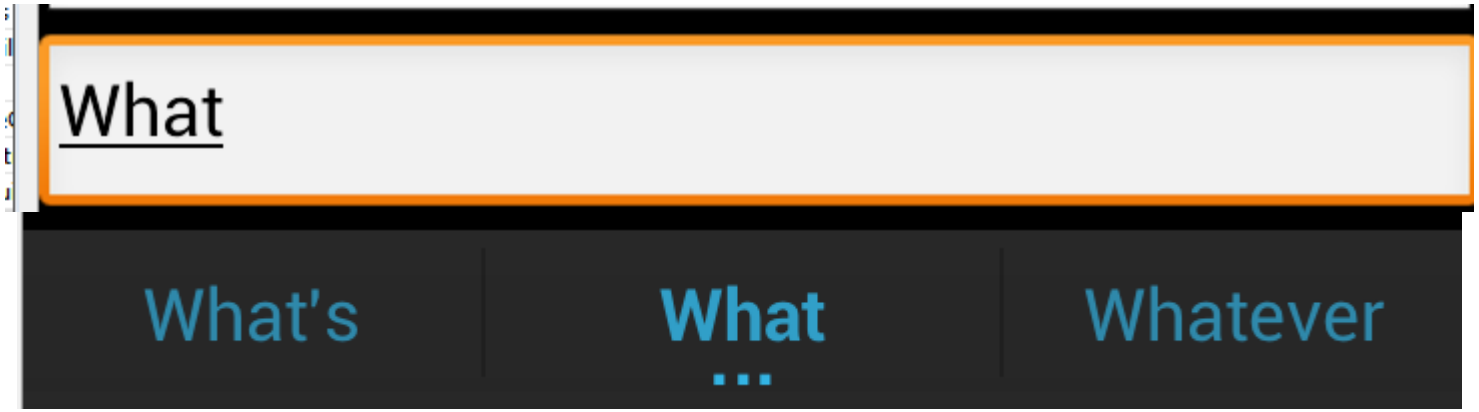

```
<EditText
    android:id="@+id/edittext"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:gravity="center"
    android:inputType="textPersonName"
    android:hint="type your name" />
```

EditText

- can span multiple lines via android:lines attribute
- **Text fields can have different input types, such as number, date, password, or email address**
  - android:inputType attribute
  - affects what type of keyboard pops up for user and behaviors such as is every word capitalized

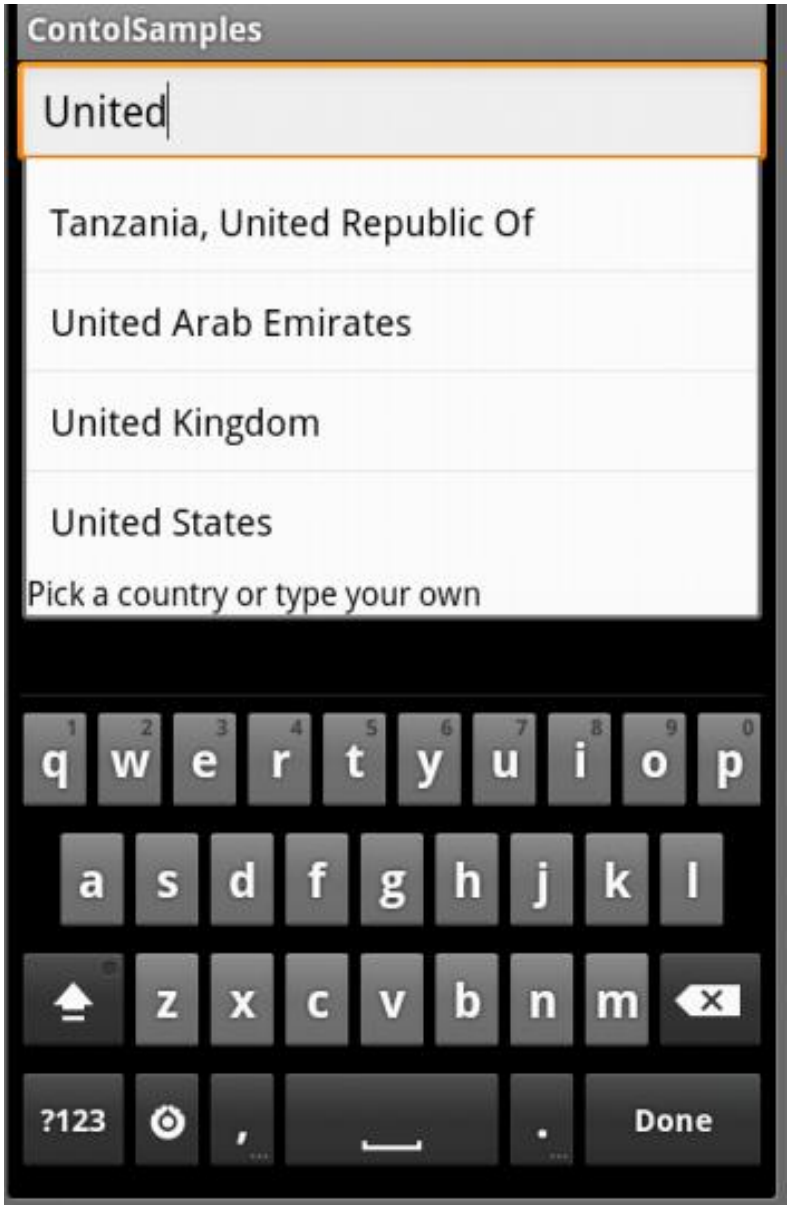
Auto Complete Options

- Depending on EditText inputType suggestions can be displayed
  - works on actual devices



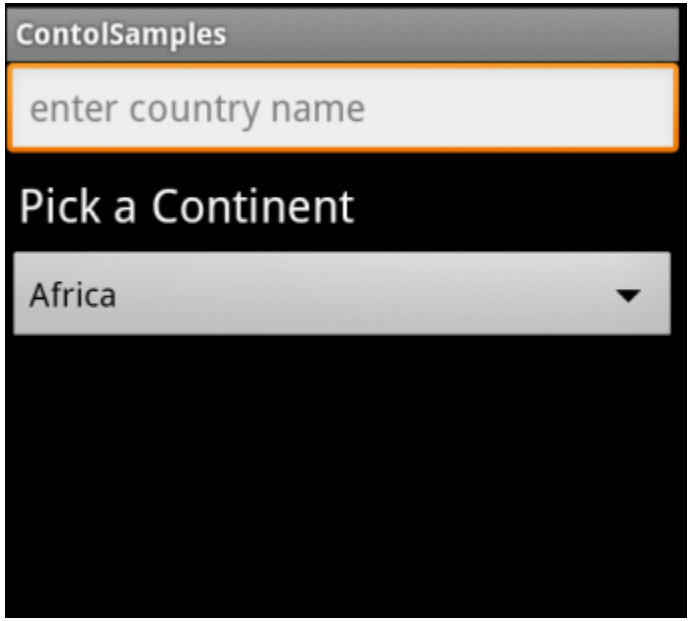
- Developer list
  - use ArrayAdapter connected to array
  - best practice: put array in array.xml file

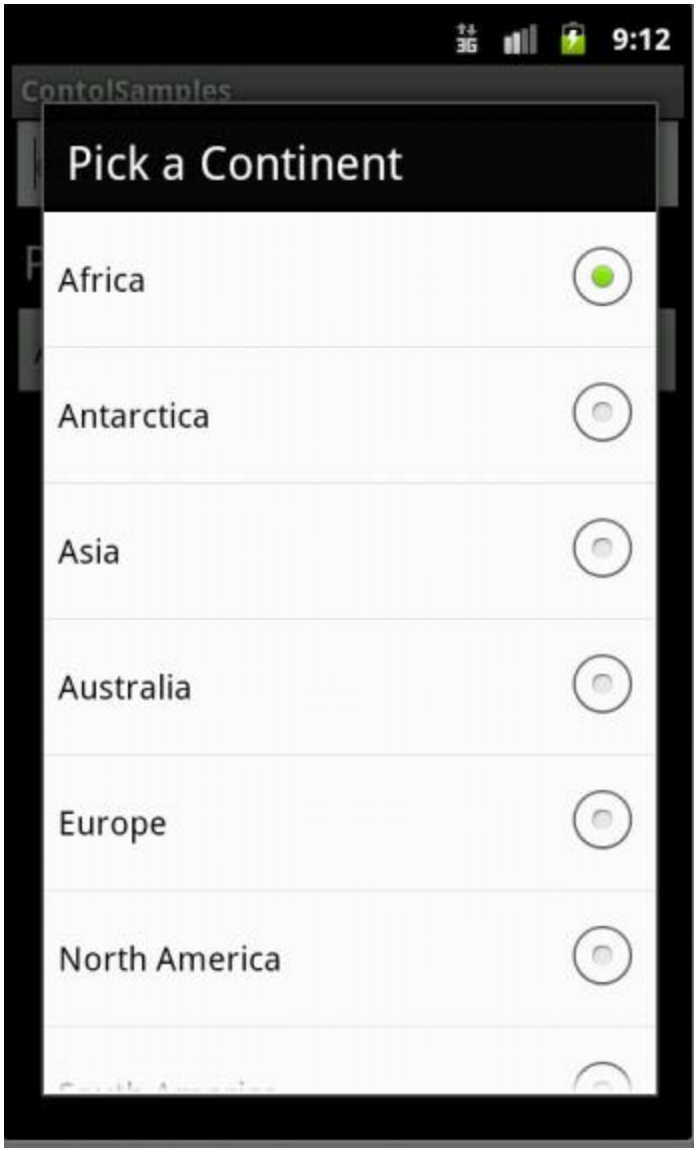
AutoComplete Using Array



Spinner Controls

- Similar to auto complete, but user **must** select from a set of choices





Spinner Control

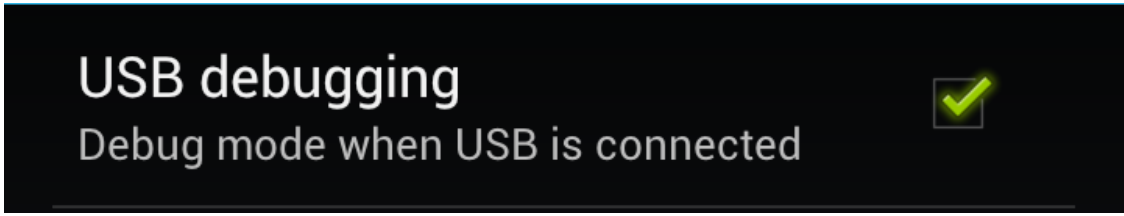
```
<Spinner
    android:id="@+id/spinner1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:entries="@array/continents"
    android:prompt="@string/pickCon"
    />
```

strings.xml in res/values

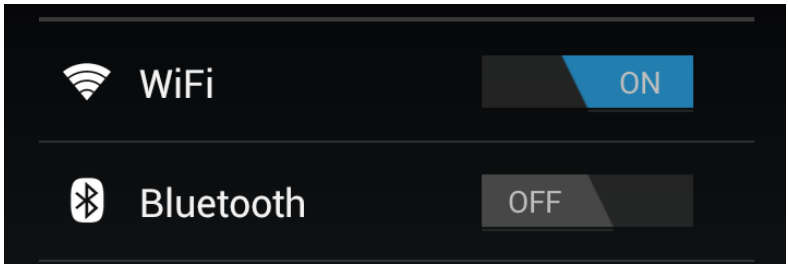
```
<string-array name="continents">
    <item>Africa</item>
    <item>Antarctica</item>
    <item>Asia</item>
    <item>Australia</item>
    <item>Europe</item>
    <item>North America</item>
    <item>South America</item>
</string-array>
```

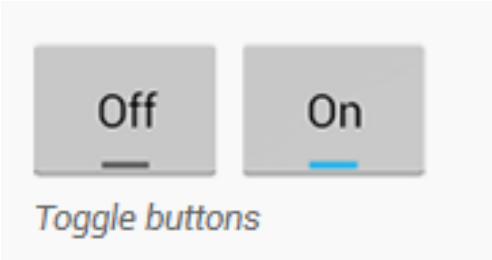
Simple User Selections

- CheckBox
  - set android:onClick attribute or create a ClickListener object, override onClick method, and register it with the checkbox



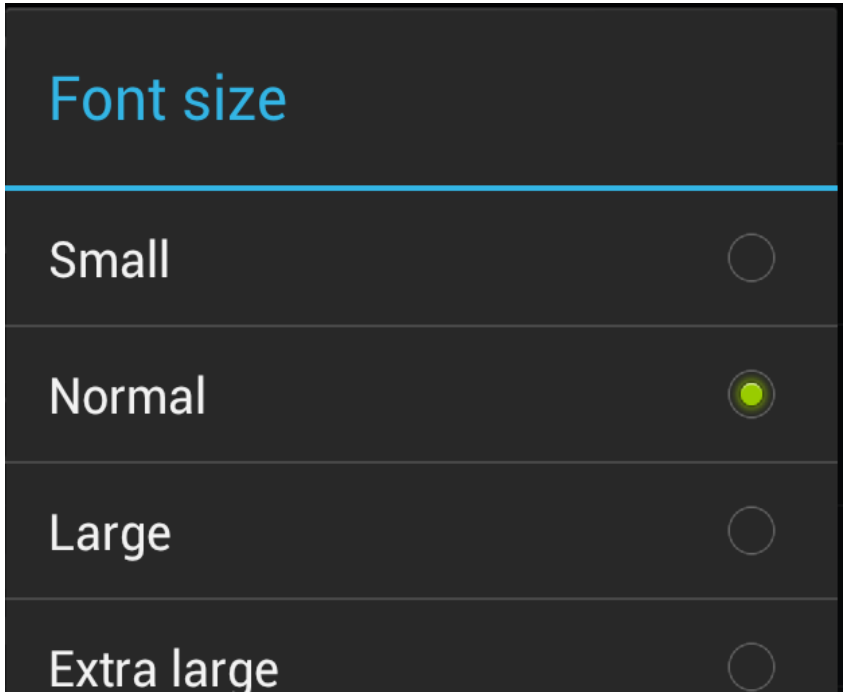
- Switches and ToggleButton
  - similar to CheckBox with two states, but visually shows states
  - on and off text





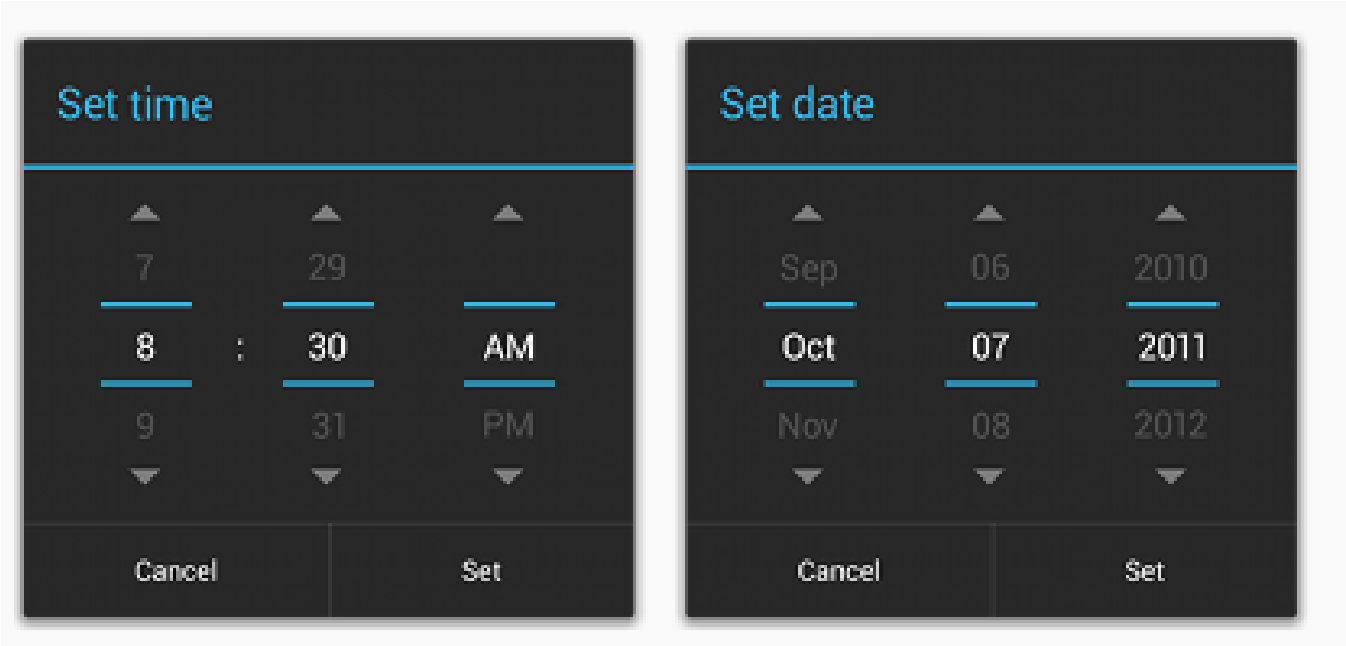
RadioButton

- Select one option from a set
- set onClick method for each button
  - generally same method
- Collected in RadioGroup
  - sub class of LinearLayout
  - vertical or horizontal orientation



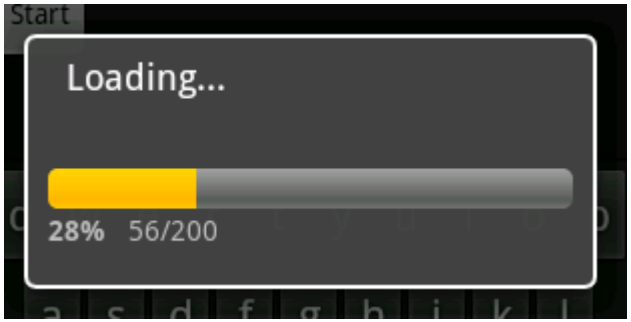
Pickers

- TimePicker and DatePicker
- Typically displayed in a TimePickerDialog or DatePickerDialog
  - dialogs are small windows that appear in front of the current activity

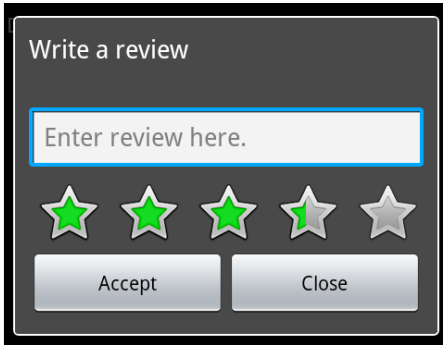


Indicators

- Variety of built in indicators in addition to TextView
- ProgressBar



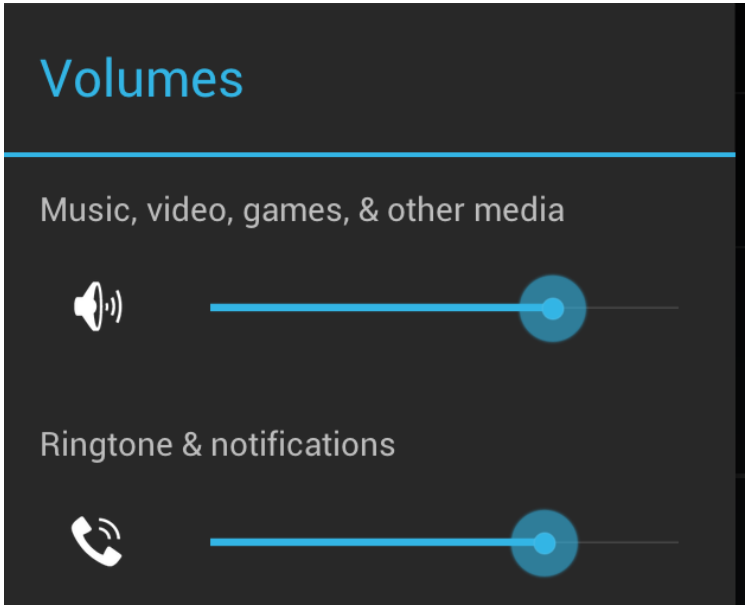
- RatingBar



- DigitalClock
- AnalogClock

SeekBar

- a slider
- Subclass of progress bar
- implement a SeekBar.OnSeekBarChangeListener to respond to changes in setting



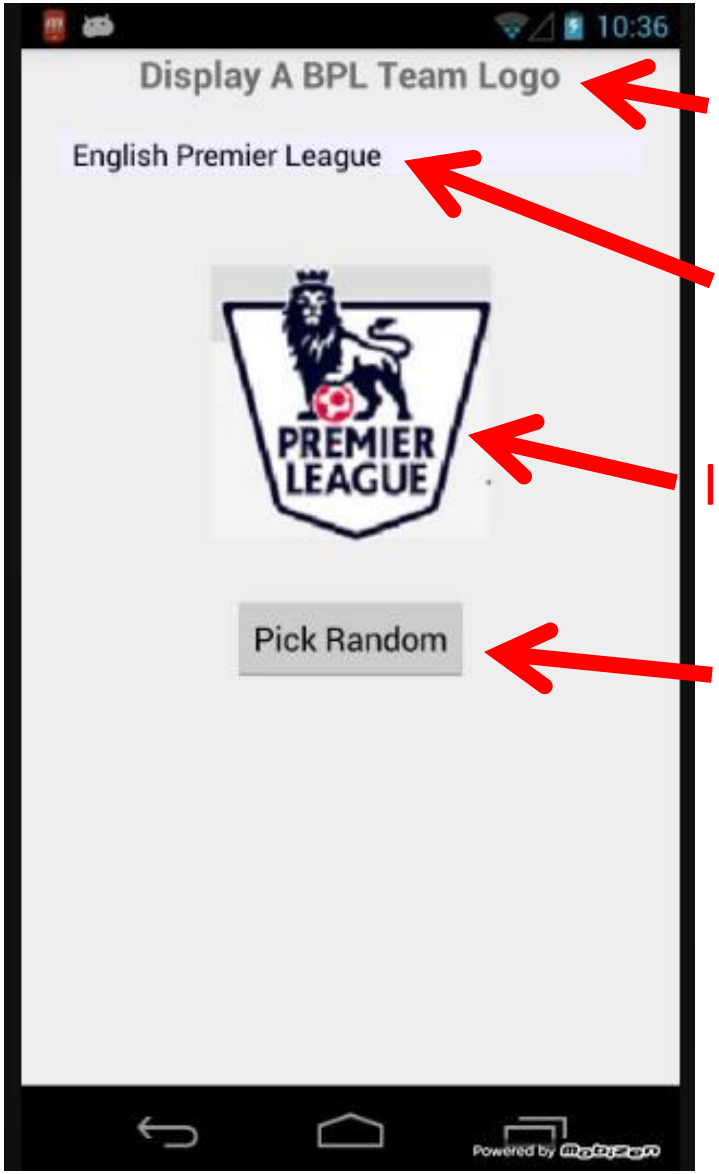
INTERACTING WITH WIDGETS

Interacting with Widgets

- Some widgets simply display information.
  - TextView, ImageView
- Many widgets respond to the user.
- We must implement code to respond to the user action.
- Typically we implement a listener and connect to the widget in code.
  - logic / response in the code

Example - Display Random Image

- App to display crests of British Premier League Football teams
- Allow user to select team from spinner control
- Or, press button to display a random crest



Button in XML layout file

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Pick Random"
    android:id="@+id/random_button"
    android:layout_gravity="center_horizontal"
    android:layout_marginTop="30dp" />
```

- Notice button reacts when pressed, but nothing happens
- Possible to disable button so it does not react

Responding to Button Press

- Two ways:
- Hard way, create a listener and attach to the button
  - shorter way exists for Views, but this approach is typical for many, many other widgets behaviors besides clicking
- Implement an onClickListener and attach to button

Accessing Button in Code

- R.java file automatically generated and creates ids for resources in project folder
  - if id attribute declared

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */

package edu.utexas.scottm.bplteams;

public final class R {

    public static final class id {

        public static final int random_button=0x7f0c0042;
```

Setting Activity Layout / GUI

- Usually the GUI for an *Activity* is set in the onCreate method.
- Typically a layout file is used

```
public class BPL_Activity extends Activity {

 @Override
 protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_bpl_);
```

- set content view will *inflate* runtime objects for all the widgets in the layout file

Accessing Layout Widget

- To attach a listener we need a handle (reference) to the runtime object for the button (or desired widget)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_bpl_);
    getImageIDs();
    setSpinnerListener();
    setRandomButtonListener();
}
```

```
private void setRandomButtonListener() {
    Button randomButton = (Button) findViewById(R.id.random_button);
```

findViewById returns a View object

often necessary to cast to correct type

Creating and attaching a Listener

```
randomButton.setOnClickListener(
        new View.OnClickListener() {
```

- setOnClickerListener is method that attaches the listener
- View.onClickListener is a Java interface with one method onClick
- We are implementing interface with an *anonymous inner class*

onClick Logic

```
@Override
public void onClick(View v) {
    // get the current selection
    Spinner spinner
            = (Spinner) findViewById(R.id.football_club_spinner);
    int oldIndex = spinner.getSelectedItemPosition();
    Log.d(TAG, "old index  = " + oldIndex);
    // don't want to pick the BPL symbol itself, so index 1 - 20
    int newIndex = randNumGen.nextInt(imageIDs.size() - 1) + 1;
    // don't let the new one be the old one
    // are we worried this will result in infinite loop with just
    while (oldIndex == newIndex) {
        newIndex = randNumGen.nextInt(imageIDs.size() - 1) + 1;
    }
    Log.d(TAG, "new index  = " + newIndex);
    ImageView iv = (ImageView) findViewById(R.id.imageView);
    iv.setImageResource(imageIDs.get(newIndex));
    spinner.setSelection(newIndex);
}
```

CONTAINERS FOR WIDGETS
VIEW GROUPS

ViewGroups - Layouts

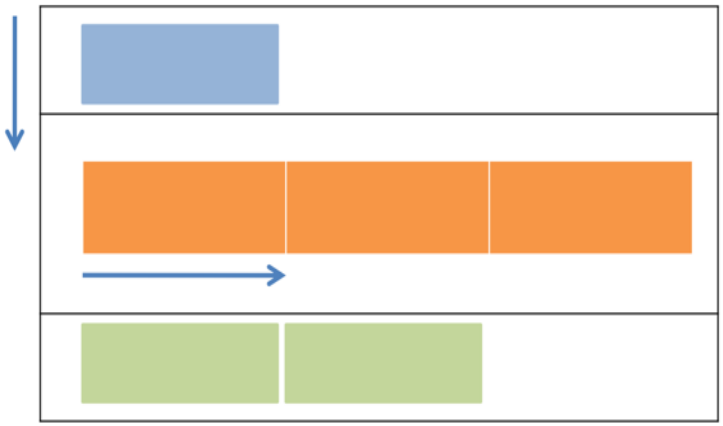
- Layouts are subclasses of ViewGroup
- Still a view but doesn't actually draw anything
- serves as a container for other views
  - similar to Java layout managers
- options on how sub views (and view groups) are arranged
- Useful Layouts: FrameLayout, LinearLayout, TableLayout, GridLayout, RelativeLayout, ListView, GridView, ScrollView, DrawerLayout, ViewPager

FrameLayout

- FrameLayout
  - simplest type of layout object
  - fill with a single object (such as a picture) that can be switched in and out
  - child elements pinned to top left corner of screen and cannot be move
  - adding a new element / child draws over the last one

LinearLayout

- Supports a filling strategy in which new elements are stacked either in a horizontal or vertical fashion.
- If the layout has a vertical orientation new rows are placed one on top of the other.
- A horizontal layout uses a side-by-side column placement policy.



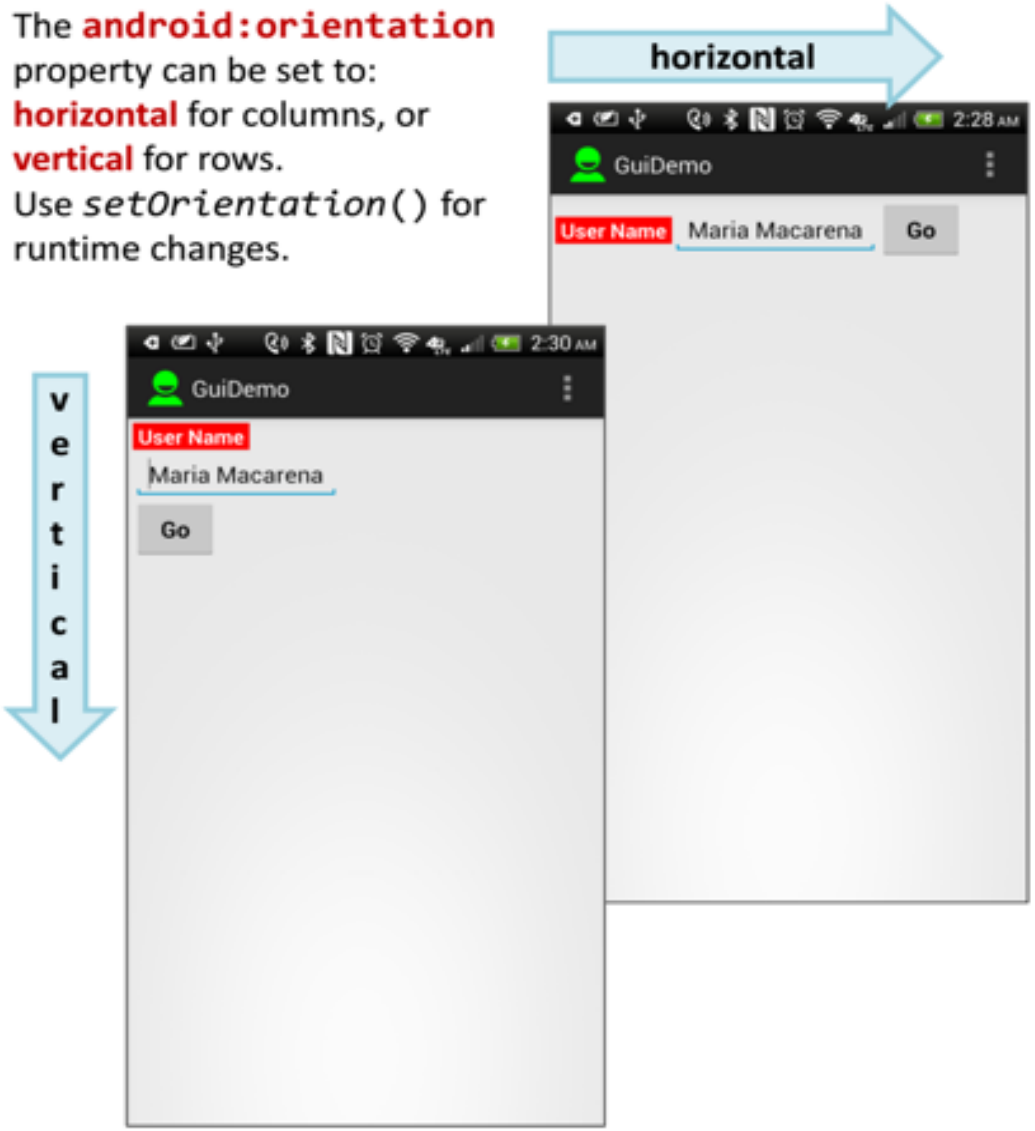
LinearLayout Attributes

Configuring a LinearLayout usually requires you to set the following attributes:

• orientation	*(vertical, horizontal)*
• fill model	*(match_parent, wrap_contents)*
• weight	*(0, 1, 2, …n)*
• gravity	*(top, bottom, center,…)*
• padding	*(dp – dev. independent pixels)*
• margin	*(dp – dev. independent pixels)*

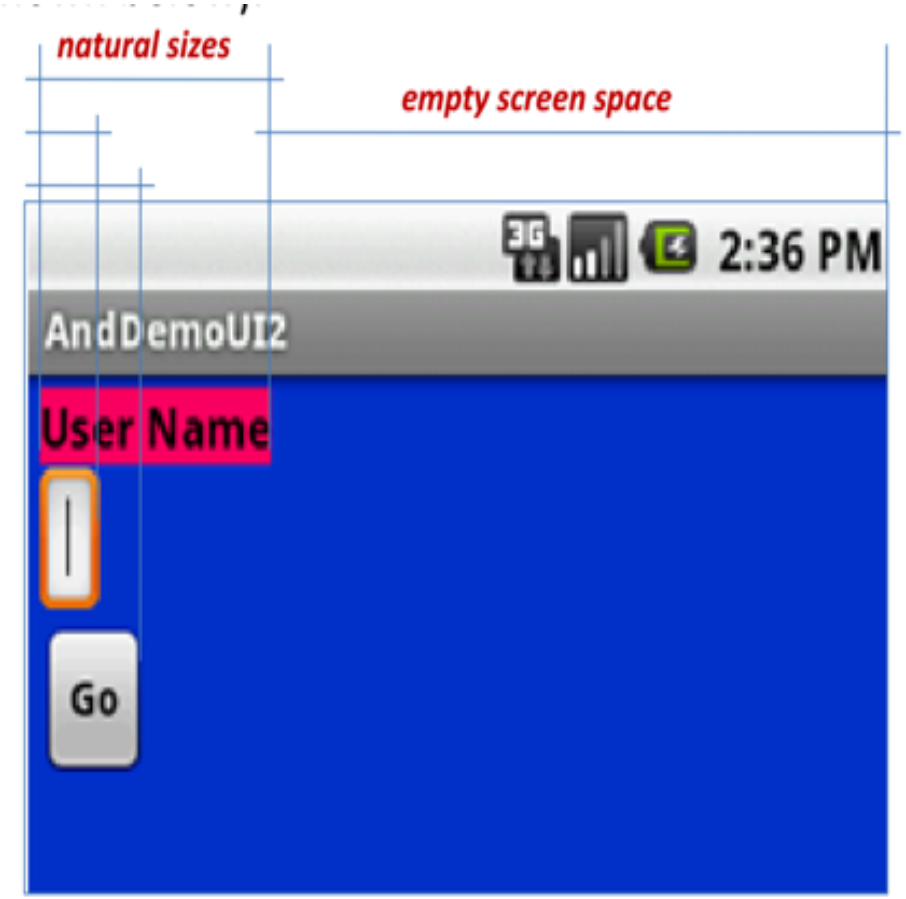
LinearLayout - Orientation

The **`android:orientation`** property can be set to:
horizontal for columns, or
vertical for rows.
Use *`setOrientation()`* for runtime changes.



LinearLayout - Fill Model

- Widgets have a "natural size“ based on their included text (rubber band effect).
- On occasions you may want your widget to have a specific space allocation (height, width) even if no text is initially provided (as is the case of the empty text box shown below).



LinearLayout - Fill Model

All widgets inside a LinearLayout must include ‘width’ and ‘height’ attributes.

`android:layout_width`
`android:layout_height`

Values used in defining height and width can be:

1. A specific dimension such as **125dp** (device independent pixels, a.k.a. dip)
2. **wrap_content** indicates the widget should just fill up its natural space.
3. **match_parent** (previously called **fill_parent**) indicates the widget wants to be as big as the enclosing parent.

LinearLayout - Fill Model

1.2 Fill Model

Medium resolution is: 320 x 480 dpi.

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myLinearLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#ff0033cc"
    android:orientation="vertical"
    android:padding="4dp" >

    <TextView
        android:id="@+id/labelUserName"
        android:layout_width="_parent"
        android:layout_height="wrap_content"
        android:background="#ffff0066"
        android:text="User Name"
        android:textColor="#ff000000"
        android:textSize="16sp"
        android:textStyle="bold" />

    <EditText
        android:id="@+id/ediName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="18sp" />

    <Button
        android:id="@+id/btnGo"
        android:layout_width="125dp"
        android:layout_height="wrap_content"
        android:text="Go"
        android:textStyle="bold" />

</LinearLayout>
```

Row-wise

Use all the row

Specific size: 125dp

LinearLayout - Layout Weight

`android:layout_weight` indicates how much of the extra space in the LinearLayout will be allocated to the view. The bigger the weight the larger the extra space given to that widget.

Use 0 if the view should not be stretched.

Default weights

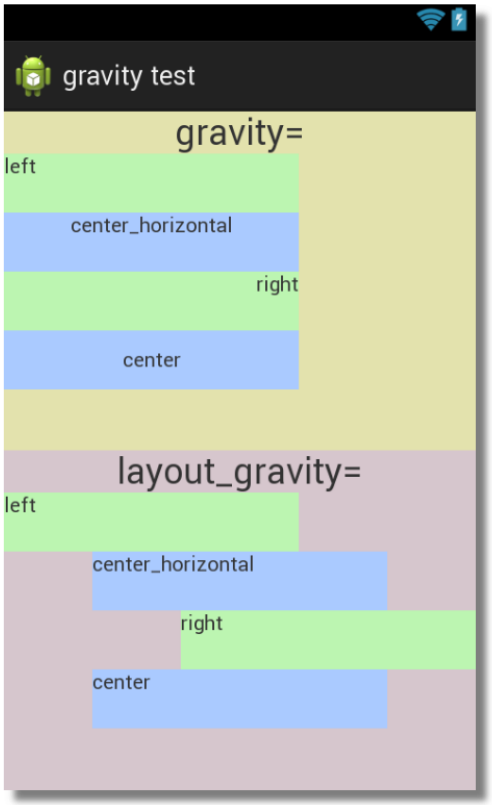
Weights – 1,1,0

Weights – 1,1,2

LinearLayout Layout Gravity

- Indicates how a control will align on the screen.
- By default, widgets are left- and top-aligned.
- Use android:layout_gravity="..." to set other arrangements: left, center, right, top, bottom, etc.

gravity vs. layout_gravity



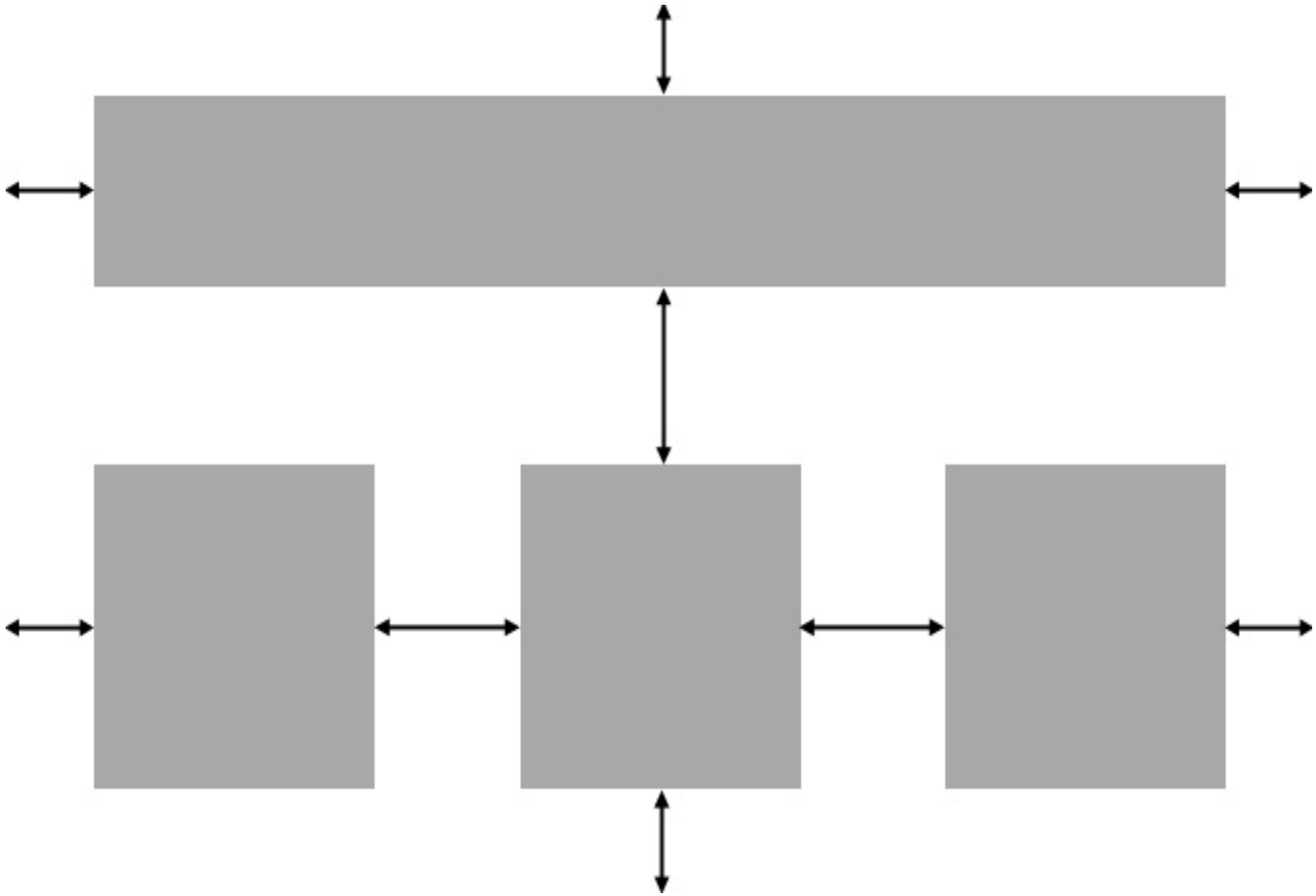
- **`android:gravity`** sets the gravity of the content of the View it's used on
- **`android:layout_gravity`** sets the gravity of the View or Layout in its parent

Margin and Padding

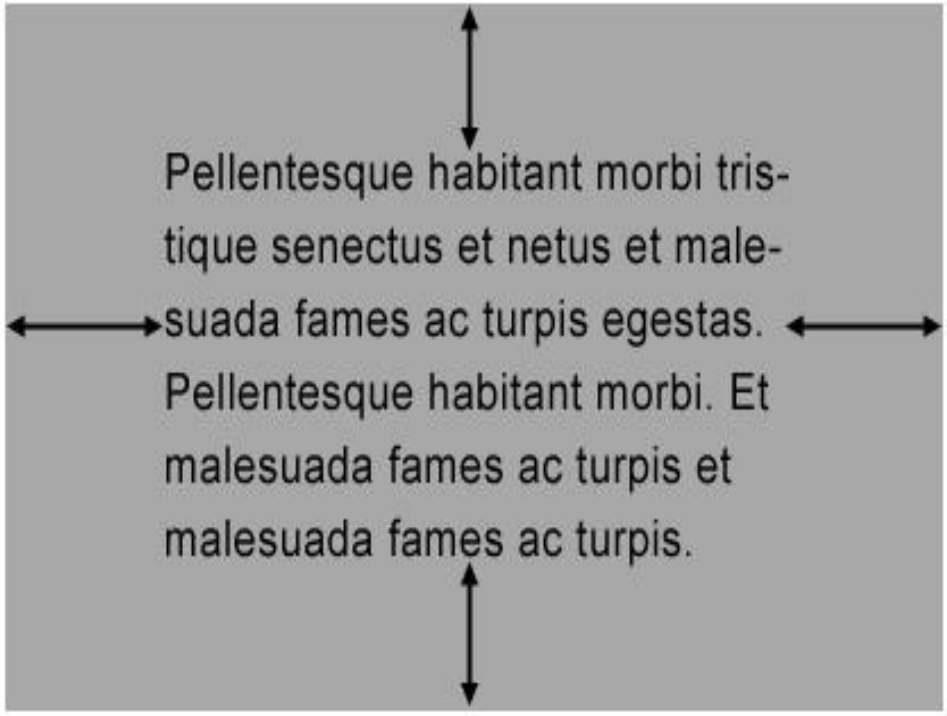
Margins are the spaces **outside** the border, between the border and the other elements next to this view.
Controlled by **`android:layout_margin`** property.

Padding is the space **inside** the border, between the border and the actual view's content.
Controlled by **`android:padding`** property.

Margin



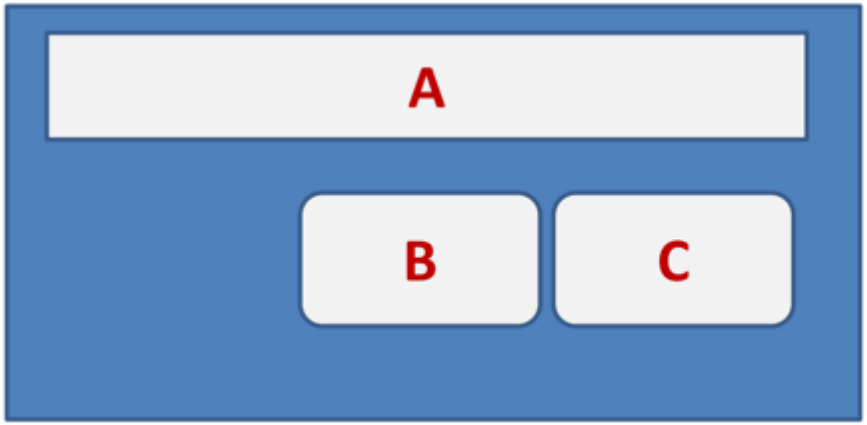
Padding


Pellentesque habitant morbi tris-
tique senectus et netus et male-
suada fames ac turpis egestas.
Pellentesque habitant morbi. Et
malesuada fames ac turpis et
malesuada fames ac turpis.

Margin and Padding

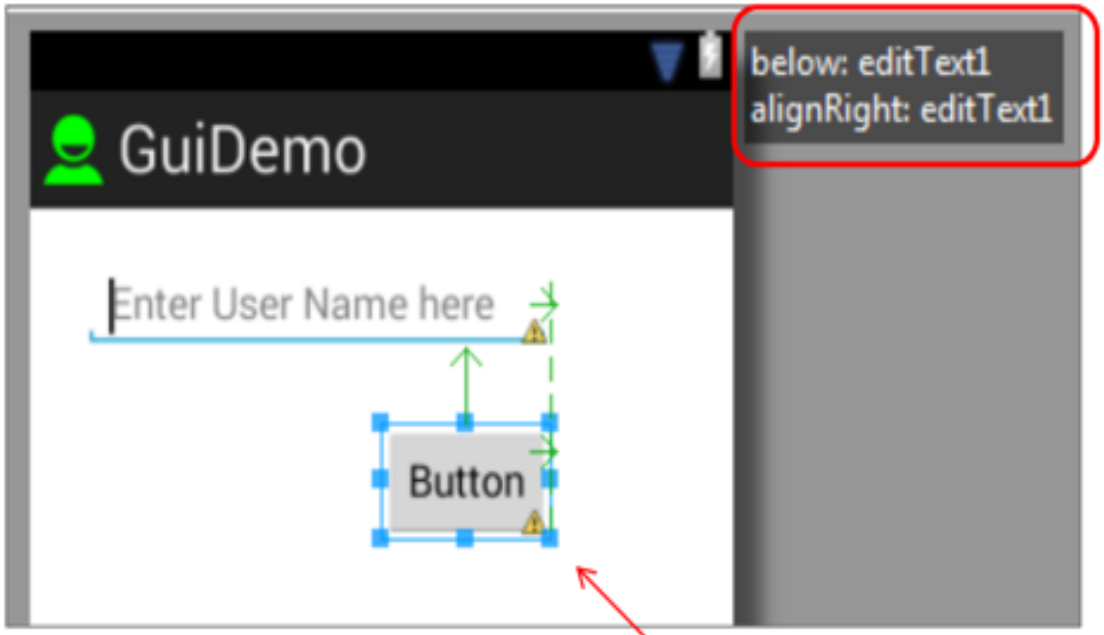
RelativeLayout

The placement of widgets in a RelativeLayout is based on their **positional relationship** to other widgets in the container and the parent container.



Example:
A is by the parent's top
C is below A, to its right
B is below A, to the left of C

RelativeLayout

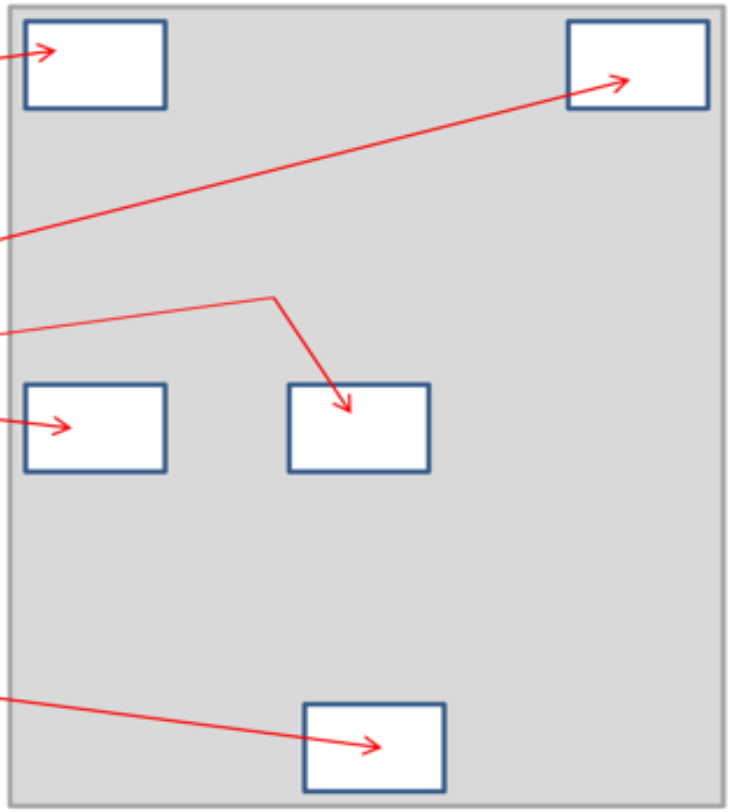


Location of the button is expressed in reference to its relative position with respect to the EditText box.

RelativeLayout - Referring to the container

Below there is a list of some positioning XML boolean properties (="true/false") useful for collocating a widget based on the location of its parent container.

```
android:layout_alignParentTop
android:layout_alignParentBottom

android:layout_alignParentLeft
android:layout_alignParentRight

android:layout_centerInParent
android:layout_centerVertical
android:layout_centerHorizontal
```

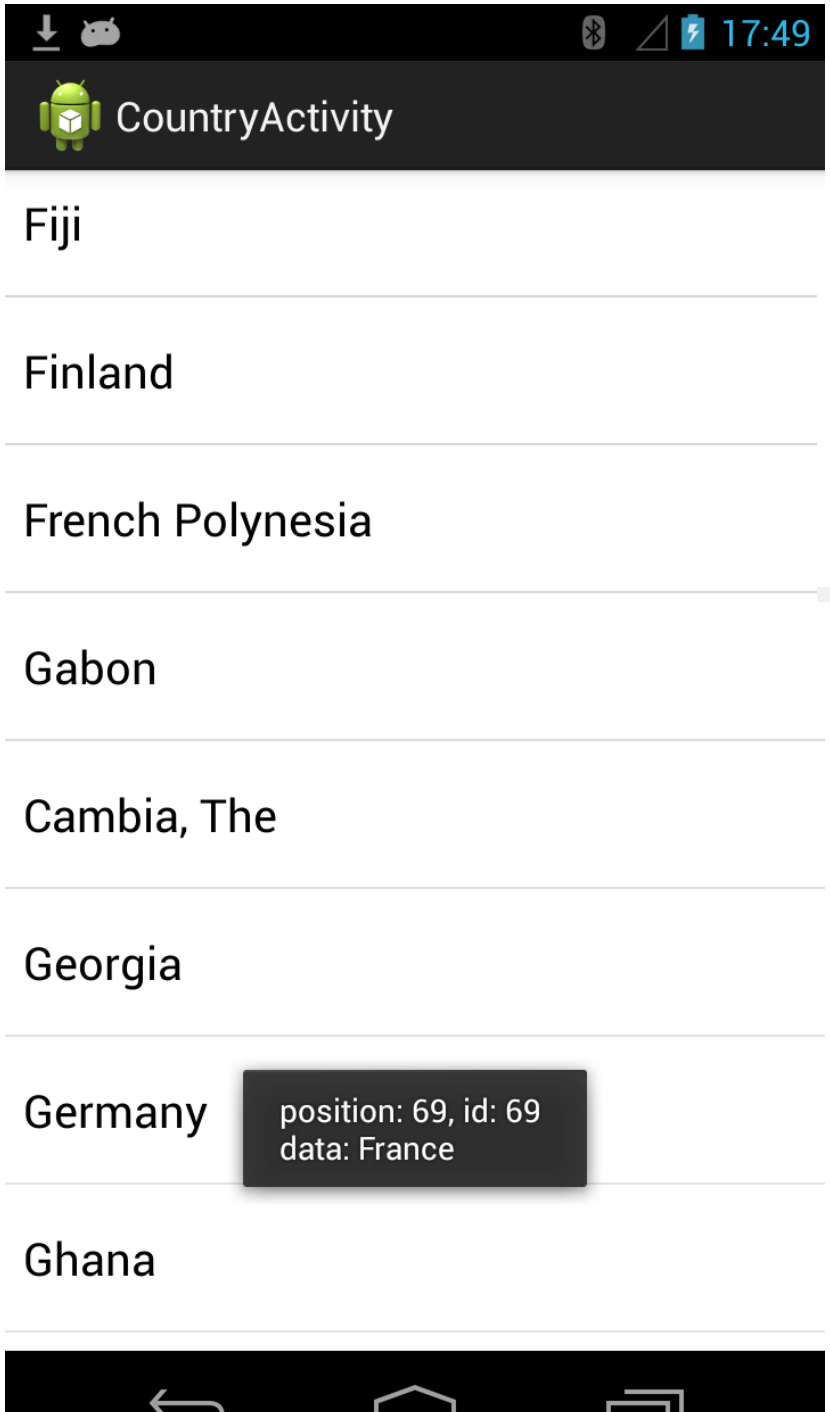


TableLayout

- rows and columns
- rows normally TableRows (subclass of LinearLayout)
- TableRows contain other elements such as buttons, text, etc.

GridLayout

- added in Android 4.0
- child views / controls can span multiple rows and columns
  - different than TableLayout
- child views specify row and column they are in or what rows and columns they span



A Toast

"A toast provides simple feedback about an operation in a small popup."

Creating a Toast

- Inside the OnItemClickListener anonymous inner class

```
Toast.makeText(CountryActivity.this,
        "position: " + position +
        ", id: " + id + "\ndata: "
        + countries.get(position),
        Toast.LENGTH_LONG).show();
```

Other Layouts - Tabbed Layouts

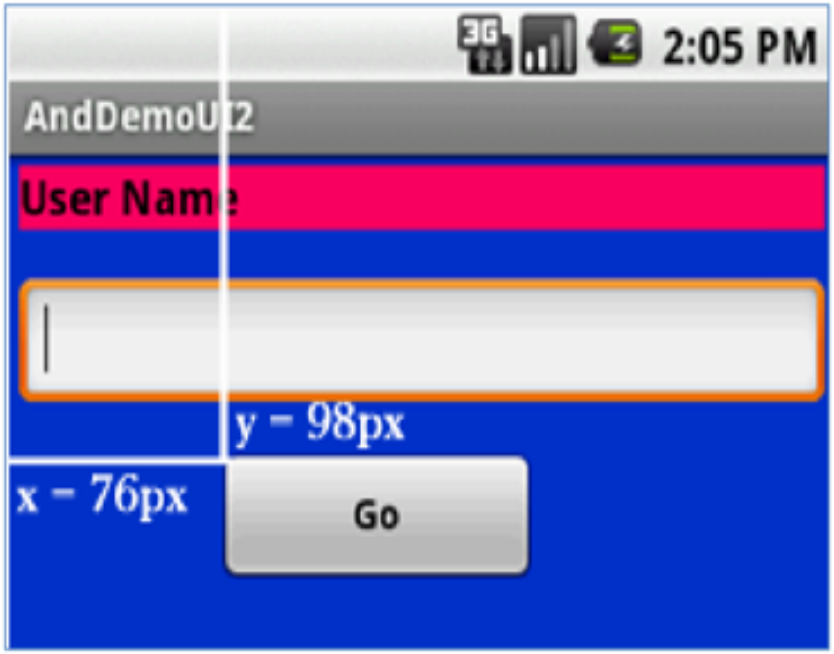
- Uses a TabHost and TabWidget
- TabHost consists of TabSpecs
- can use a TabActivity to simplify some operations
- Tabs can be
  - predefined View
  - Activity launched via Intent
  - generated View from TabContentFactory

Scrolling

- ListView supports vertical scrolling
- Other views for Scrolling:
  - ScrollView for vertical scrolling
  - HorizontalScrollView
- Only one child View
  - but could have children of its own
- examples:
  - scroll through large image
  - Linear Layout with lots of elements

AbsoluteLayout

- A layout that 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.
- Not recommended

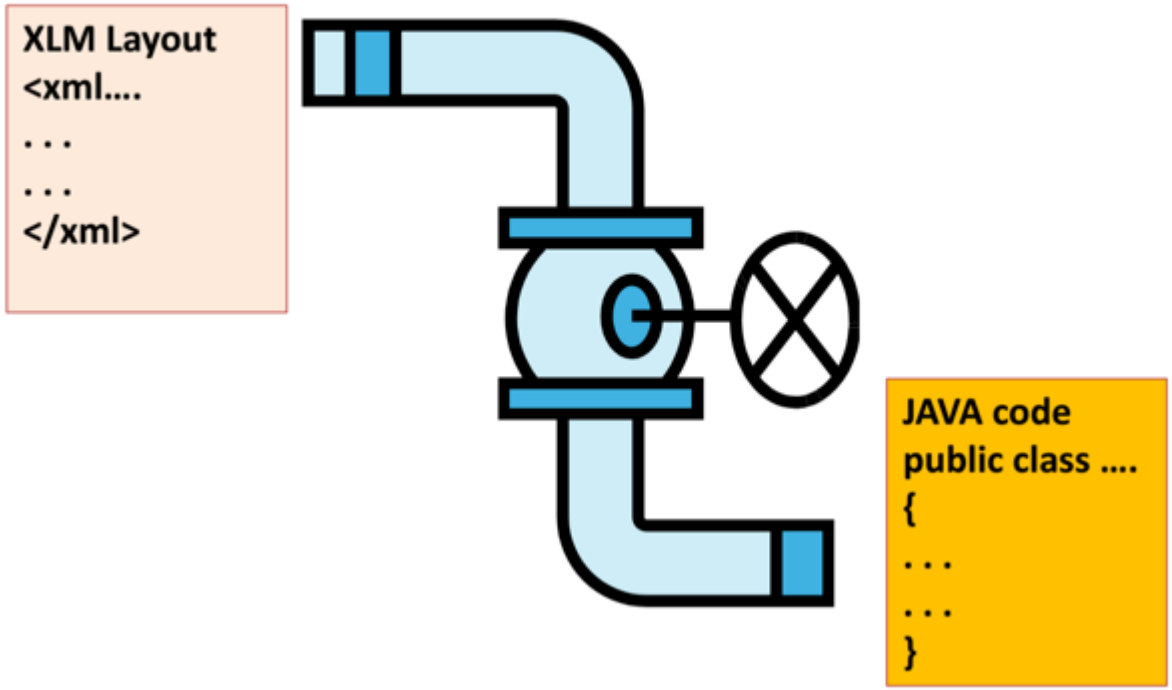


Constraint Layout

- Position and size widgets in a *flexible* way
- Constraints available:
  - Relative positioning
  - Centering positioning
  - Circular positioning
  - Chains of widgets

Attaching Layouts to Java Code

PLUMBING. You must 'connect' the XML elements with equivalent objects in your Java activity. This allows you to manipulate the UI with code.



Attaching Layouts to Java Code

Assume the UI in *res/layout/main.xml* has been created. This layout could be called by an application using the statement

```
setContentView(R.layout.main);
```

Individual widgets, such as *myButton* could be accessed by the application using the statement `findViewByID(...)` as in

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

Where **R** is a class automatically generated to keep track of resources available to the application. In particular **R.id...** is the collection of widgets defined in the XML layout.

Attaching Layouts to Java Code

Attaching Listeners to the Widgets

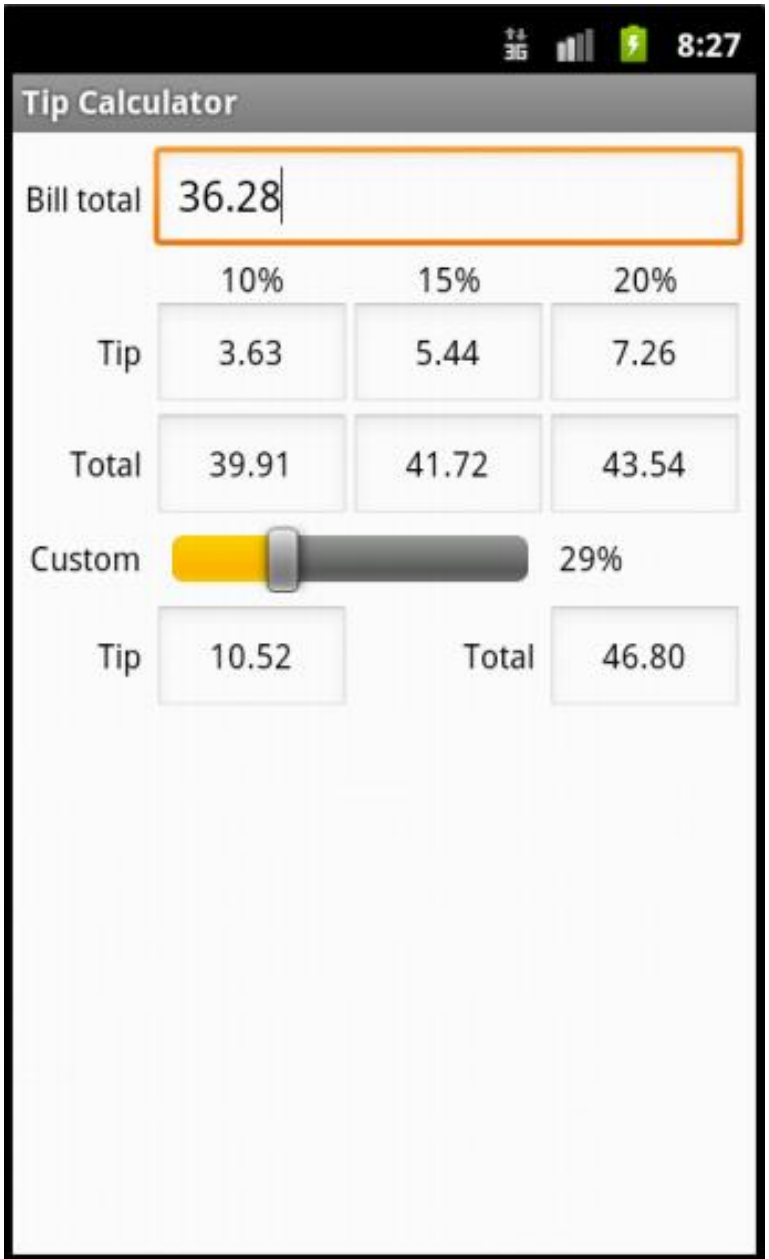
The button of our example could now be used, for instance a listener for the click event could be written as:

```
btn.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        updateTime();
    }
});

private void updateTime() {
    btn.setText(new Date().toString());
}
```

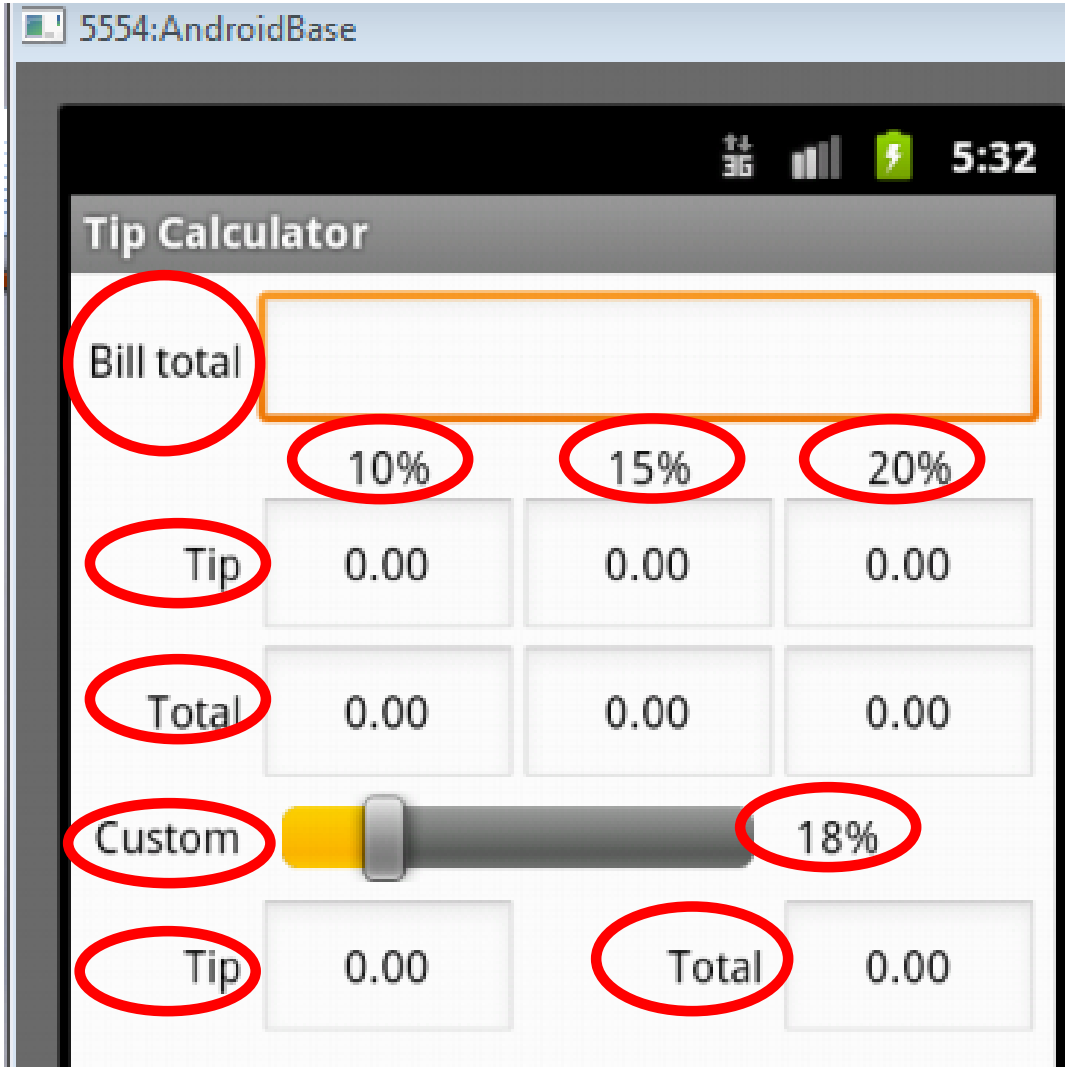
CONCRETE UI EXAMPLE - TIP CALCULATOR

Concrete Example

- Tip Calculator
- What kind of layout to use?
- Widgets:
  - TextView
  - EditText
  - SeekBar



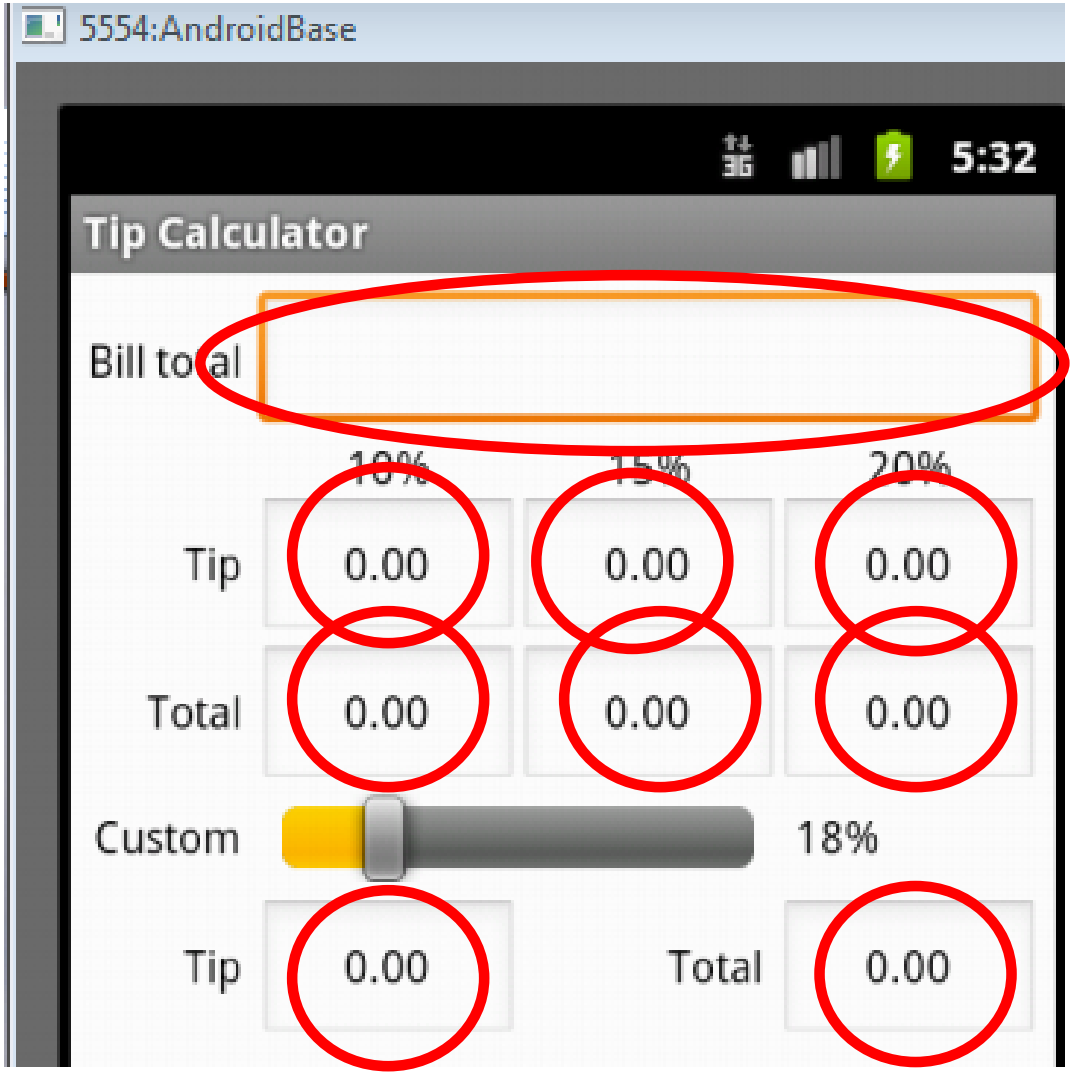
TextViews


5554:AndroidBase
5:32
Tip Calculator
Bill total
10%
15%
20%
Tip
0.00
0.00
0.00
Total
0.00
0.00
0.00
Custom
18%
Tip
0.00
Total
0.00

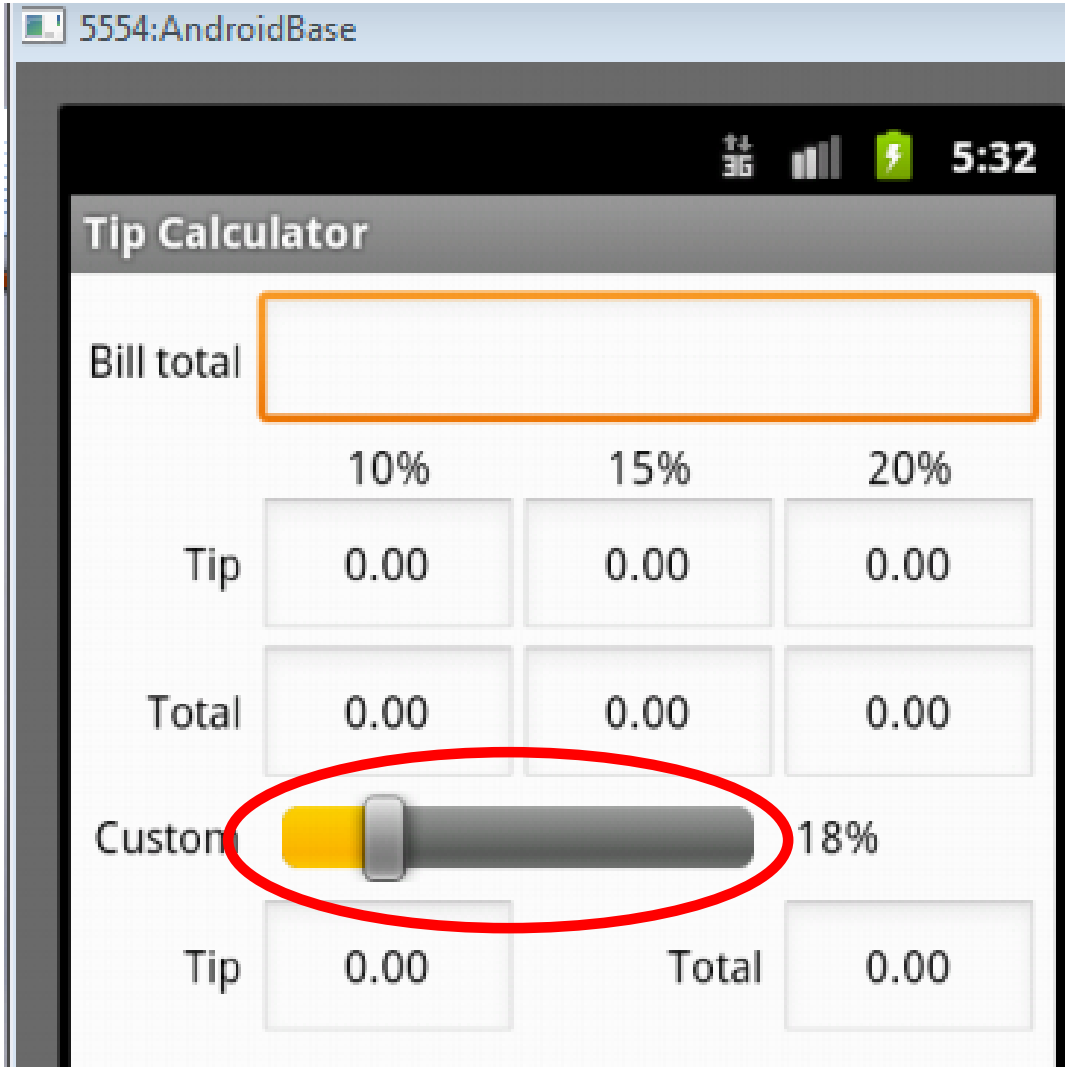
EditText

All but top EditText are uneditable

Alternative? TextViews?

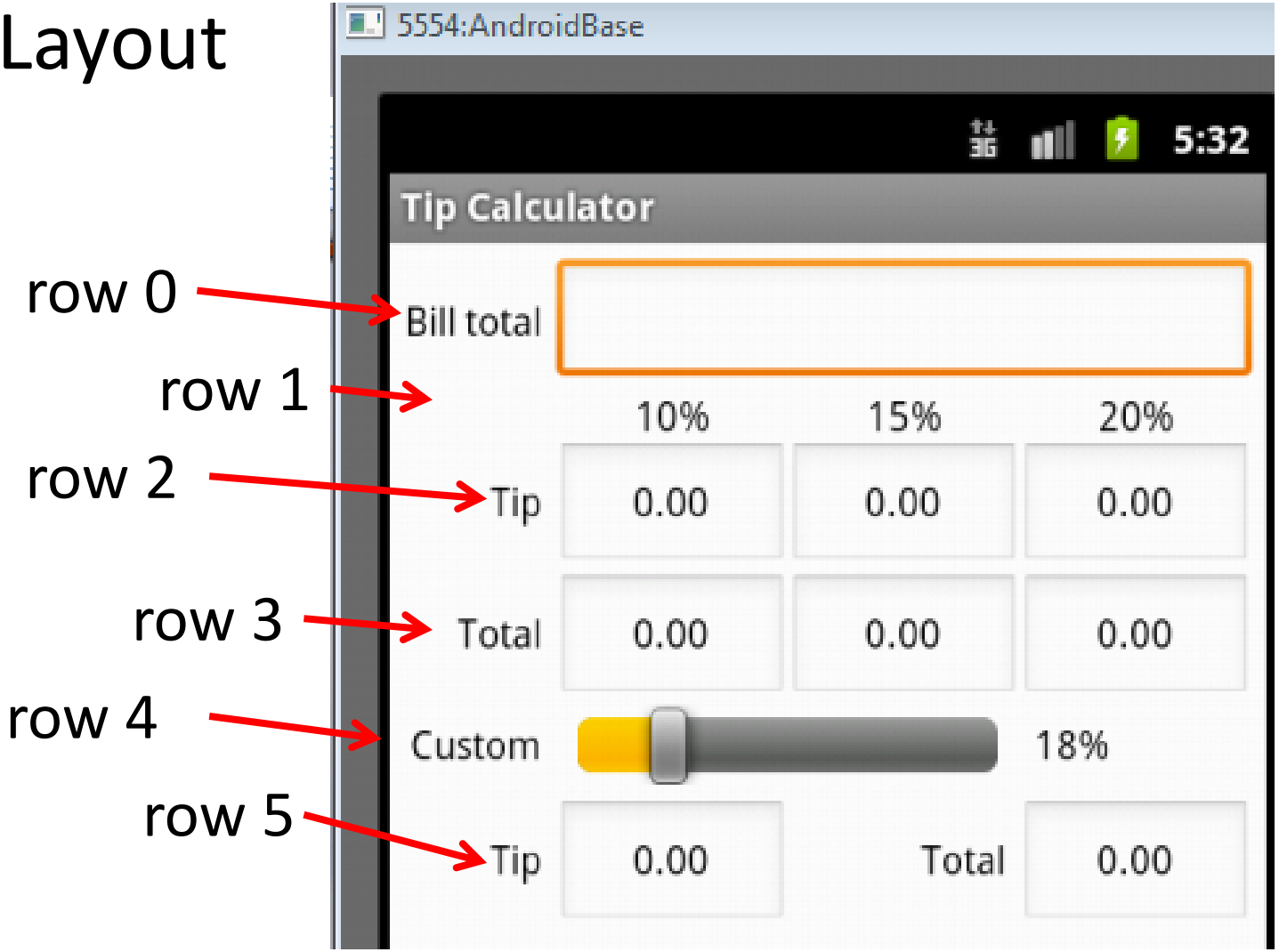


SeekBar


5554:AndroidBase
5:32
Tip Calculator
Bill total
10%
15%
20%
Tip
0.00
0.00
0.00
Total
0.00
0.00
0.00
Custom
18%
Tip
0.00
Total
0.00

Layout

- TableLayout



Color Resources

main.xml | colors.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="Cardinal">#C41E3A</color>
    <color name="White">#FFFFFF</color>
</resources>
```

```xml
android:layout_width="match_parent"
android:layout_height="match_parent"
android:background="@color/White"
android:padding="5dp"
```

- Good Resource / W3C colors
  - http://tinyurl.com/6py9huk

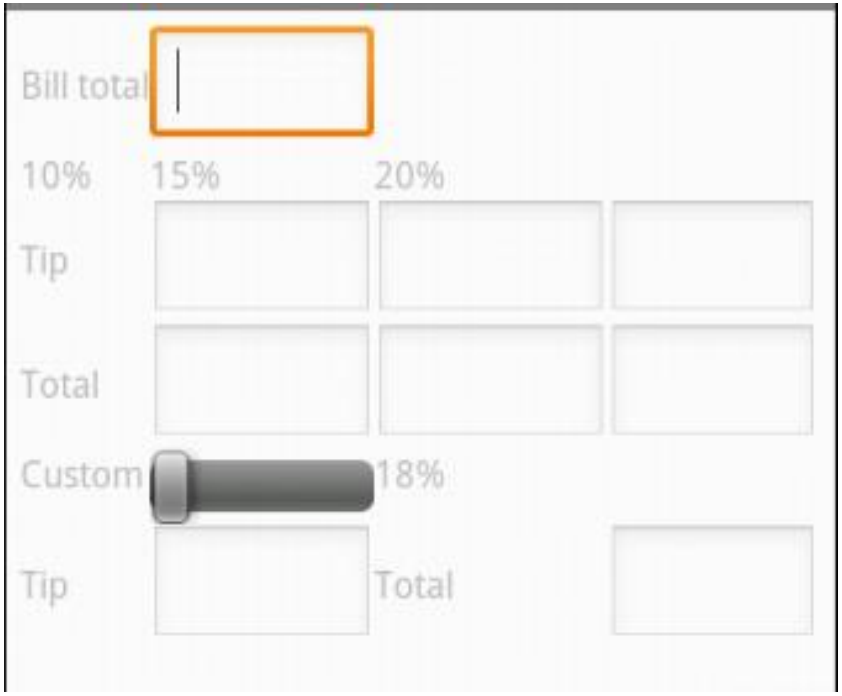
StretchColumns

```
<TableLayout xmlns:android="http://sch
    android:id="@+id/tableLayout"
    android:layout_width="match_parent
    android:layout_height="match_paren
    android:background="#FFF"
    android:padding="5dp"
    android:stretchColumns="1,2,3" >
```

- columns 0 indexed
- columns 1, 2, 3 stretch to fill layout width
- column 0 wide as widest element, plus any padding for that element

Initial UI

- Done via some Drag and Drop, Outline view, and editing XML
- Demo outline view
  - properties



Changes to UI

- change bill total and seekbar to span more columns

```
<EditText
    android:id="@+id/billEditText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_span="3"
    android:inputType="numberDecimal" >
```

- gravity and padding for text in column 0
- align text with seekBar
- set seekBar progress to 18
- set seekBar focusable to false - keep keyboard on screen

Changes to UI

- Prevent Editing in EditText
  - focusable, long clickable, and cursor visible properties to false
- Set text in EditText to 0.00