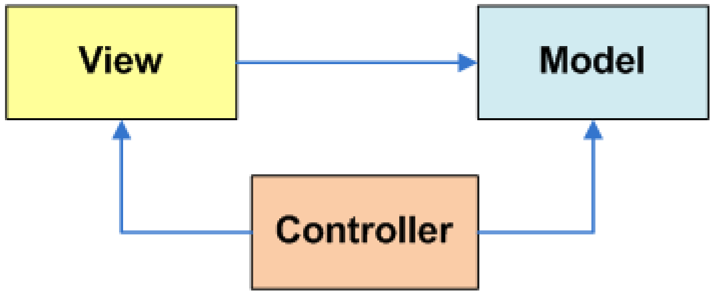
The Model-View-Control (**MVC**) Pattern

The *Model-View-Controller (MVC)* is an important software design pattern whose main goal is to separate the (1) user interface, (2) business, and (3) input logic.



How is this seen by the Android developer?

- **Model**. Consists of the Java code and objects used to manage the behavior and data of the application.
- **View**. Set of screens the user sees and interacts with.
- **Controller**. Implemented through the Android OS, responsible for interpretation of the user and system inputs. Input may come from a variety of sources such as the trackball, keyboard, touchscreen, GPS chip, background services, etc, and tells the Model and/or the View (usually through callbacks and registered listeners) to change as appropriate.

[Burbeck92] Burbeck, Steve. "Application Programming in Smalltalk-80: How to use Model-View-Controller (MVC)."*University of Illinois in Urbana-Champaign (UIUC) Smalltalk Archive.* Available at: http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html.

The Model-View-Control (**MVC**) Pattern

Getting ready to create MVC conforming solutions

The Android developer should be aware of …

- **Inputs** could be sent to the application from various physical/logical components. Reacting to those signals is typically handled by **callback methods**. Usually there are many of them, you want to learn how to choose the appropriate one.

- Moving to states in the **lifecycle** is tied to logic in the model. For instance, if forced to *Pause* you may want to save uncommitted data.

- A **notification** mechanism is used to inform the user of important events happening outside the application (such as arrival of a text message or email, phone calls, etc) and consequently choose how to proceed.

- **Views** are unlimited in terms of aesthetic and functionality. However physical constraints such as size, and hardware acceleration (or lack of) may affect how graphical components are managed.

Android & the **MVC** Pattern

The View - User Interfaces (Uis)
Android **graphical interfaces** are usually implemented as XML files (although they could also be dynamically created from code).

An Android UI is conceptually similar to a common HTML page

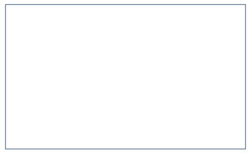
- **In a manner similar to a web page interaction**, when the Android user touches the screen, the controller interprets the input and determines what specific portion of the screen and gestures were involved. Based on this information it tells the model about the interaction in such a way that the appropriate "callback listener" or lifecycle state could be called into action.

- **Unlike a web application** (which refreshes its pages after explicit requests from the user) an asynchronous Android background service could quietly notify the controller about some change of state (such as reaching a given coordinate on a map) and in turn a change of the view's state could be triggered; all of these without user intervention.

The View Class

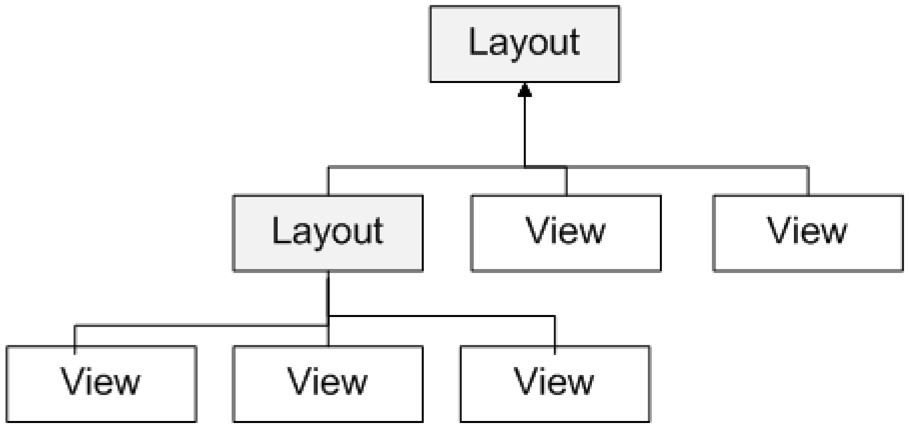
- The **View class** is the Android's most basic component from which users interfaces can be created. This element is similar to the Swing **JComponent** class for Java apps.

- A **View** occupies a rectangular area on the screen and is responsible for *drawing* and *event handling*.

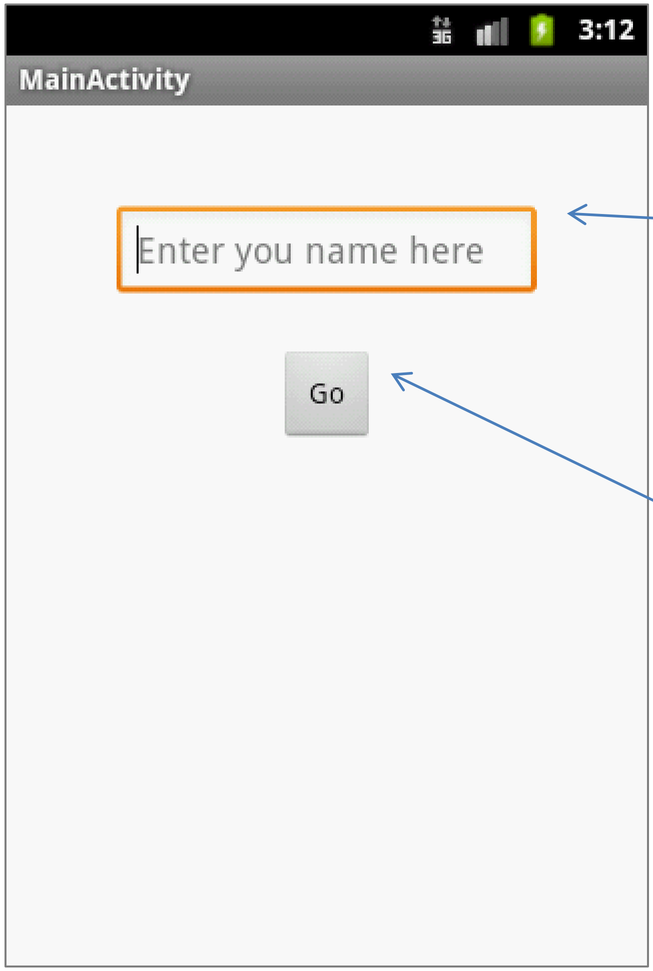


- **Widgets** are subclasses of View. They are used to create interactive UI components such as buttons, checkboxes, labels, text fields, etc.

- **Layouts** are invisible containers used for holding other Views and nested layouts.



Graphical UI ⟷ XML Layout



Actual UI displayed by the app

```
<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" >

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:hint="Enter you name here"
        android:layout_marginTop="50dp"
        android:ems="10" >

        <requestFocus />
    </EditText>

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/editText1"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="24dp"
        android:text=" Go " />

</RelativeLayout>
```

Text version: *activity_main.xml* file

Using Views

- An Android's **XML** view file consists of a **layout** holding a hierarchical arrangement of its contained elements.
- The inner elements could be simple widgets or nested layouts holding some complex viewgroups.
- An Activity uses the `setContentView(R.layout.xmlfilename)` method to render a view on the device's screen.

```
<?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" >

          Widgets and other nested layouts

</LinearLayout>
```

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*.

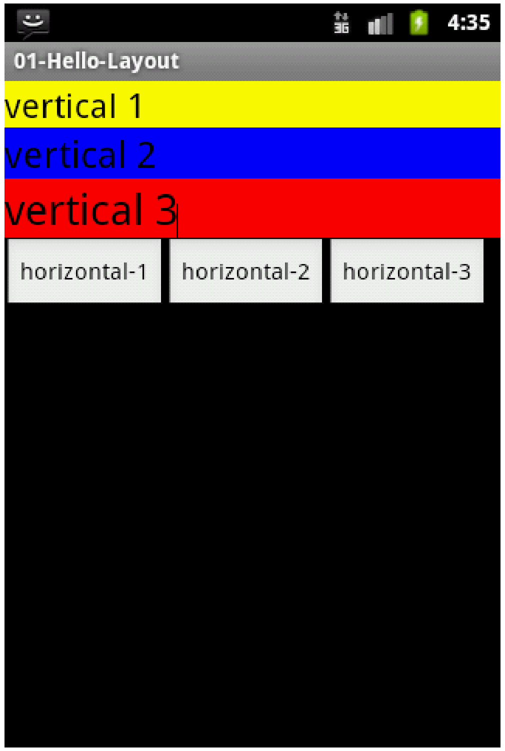
2. **Set up listeners:** For example, an image could be programmed to respond to various events such as: click, long-tap, mouse-over, etc.

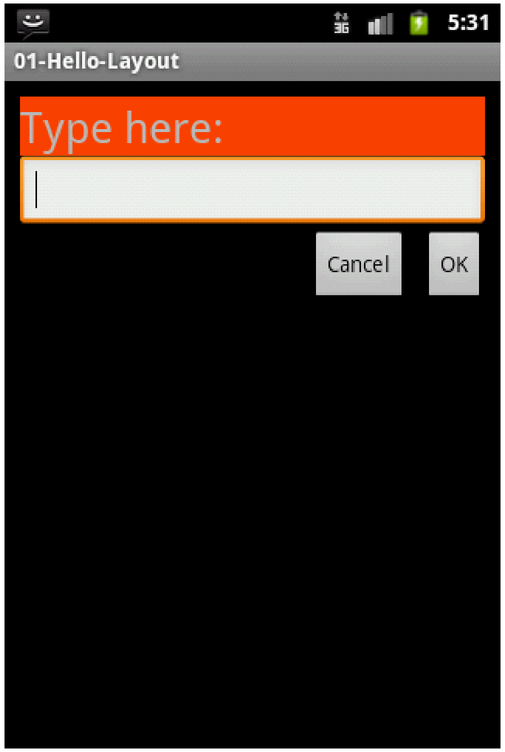
3. **Set focus:** To set focus on a specific view, you call the method requestFocus() or use XML tag <requestFocus />

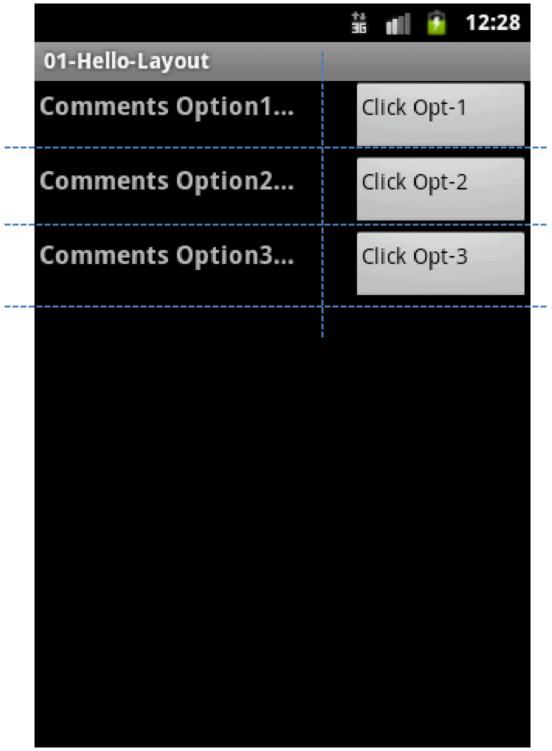
4. **Set visibility:** You can hide or show views using setVisibility(…).

A brief sample of UI components

Layouts







Linear Layout
A LinearLayout places its inner views either in horizontal or vertical disposition.

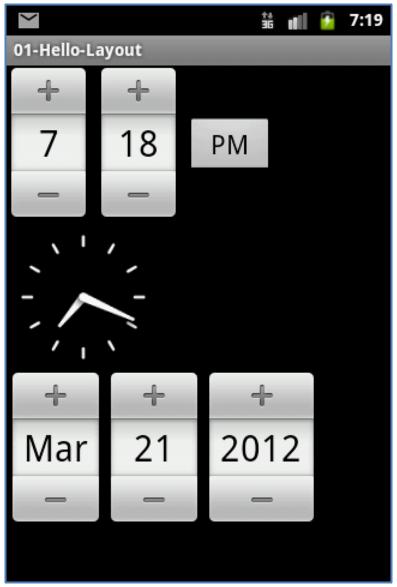
Relative Layout
A RelativeLayout is a ViewGroup that allows you to position elements relative to each other.

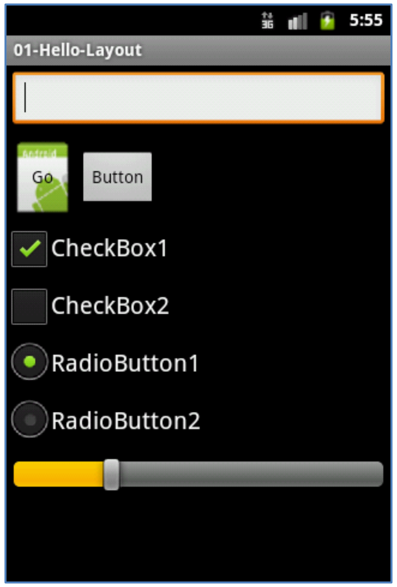
Table Layout
A TableLayout is a ViewGroup that places elements using a row & column disposition.

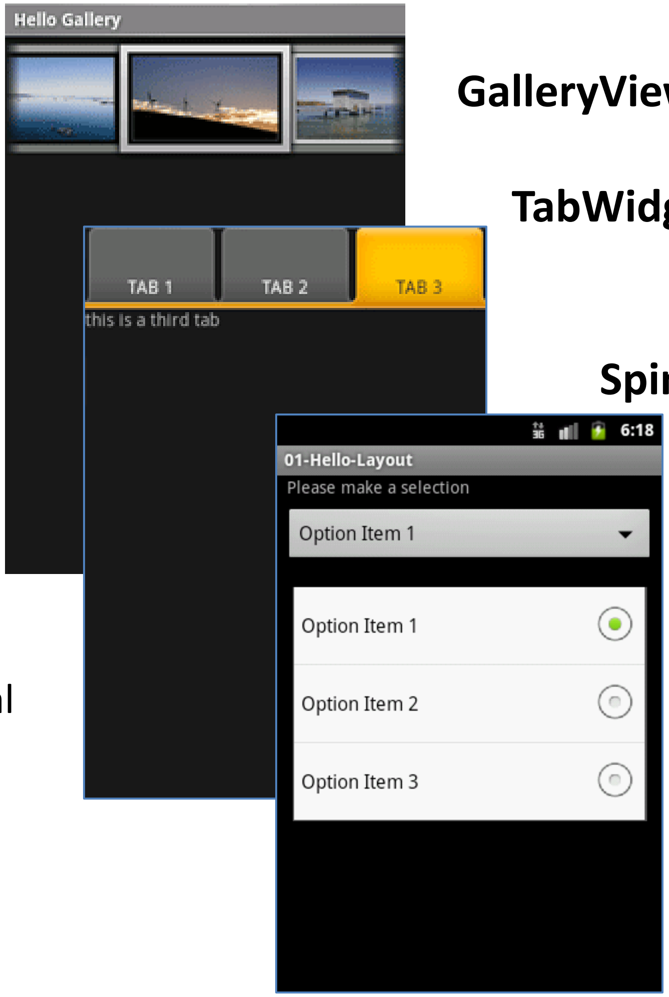
Reference: http://developer.android.com/guide/topics/ui/layout-objects.html

A brief sample of UI components

Widgets







GalleryView

TabWidget

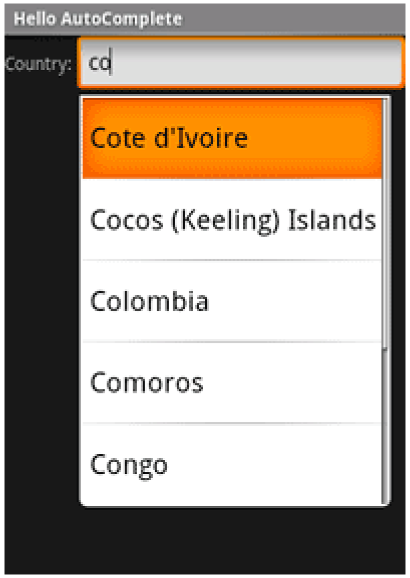
Spinner

TimePicker
AnalogClock
DatePicker
A *DatePicke* is a widget that allows the user to select a month, day and year.

Form Controls
Includes a variety of typical form widgets, like:
image buttons,
text fields,
checkboxes and
radio buttons.

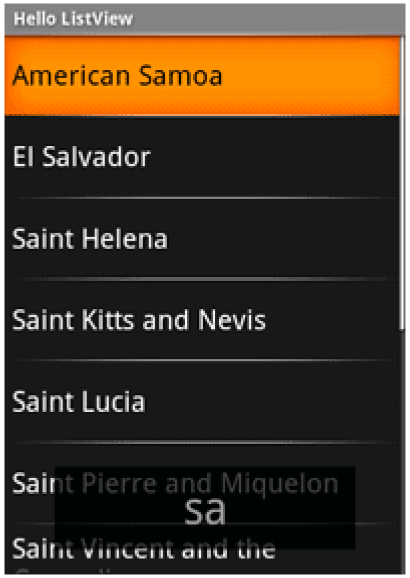
Reference: http://developer.android.com/guide/topics/ui/layout-objects.html

A brief sample of UI components


Hello AutoComplete

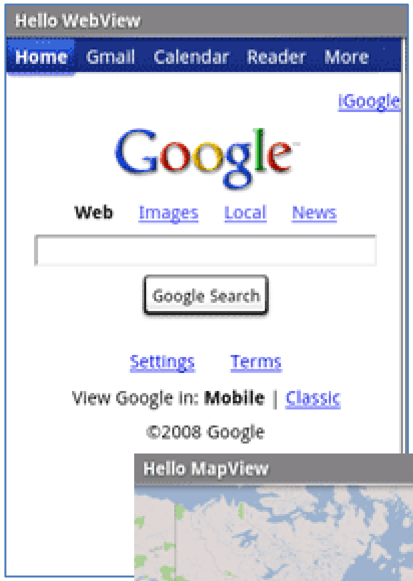
Country: co

Cote d'Ivoire
Cocos (Keeling) Islands
Colombia
Comoros
Congo


Hello ListView

American Samoa
El Salvador
Saint Helena
Saint Kitts and Nevis
Saint Lucia
Saint Pierre and Miquelon
Saint Vincent and the

WebView


Hello WebView

Home Gmail Calendar Reader More

iGoogle

Web Images Local News

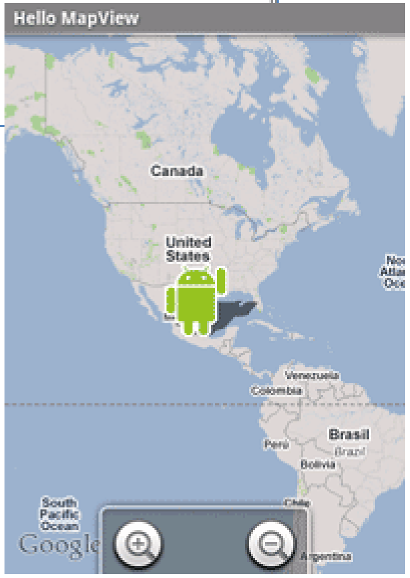
Google Search

Settings Terms

View Google in: Mobile | Classic

©2008 Google

MapView



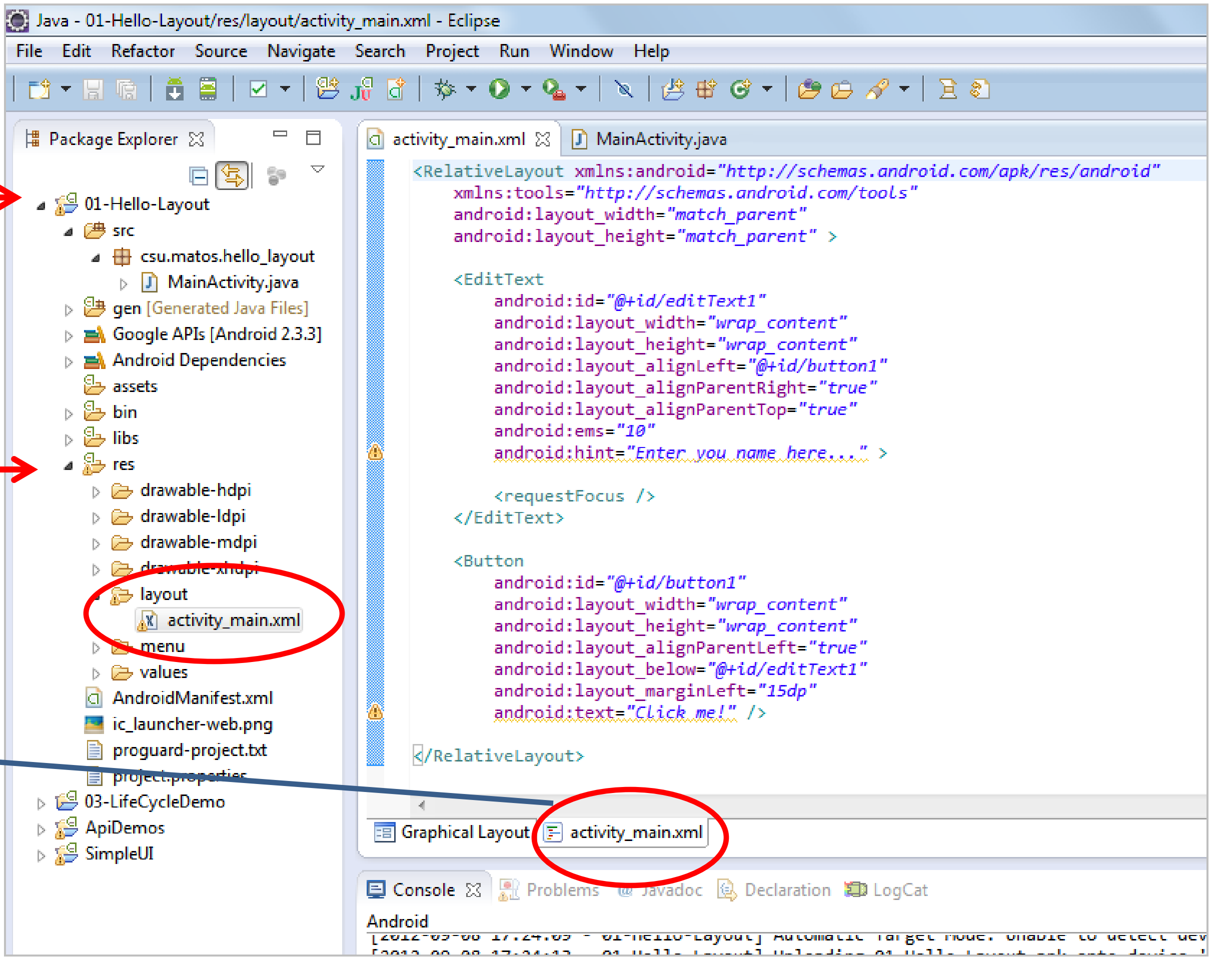
AutoCompleteTextView
It is a version of the *EditText* widget that will provide auto-complete suggestions as the user types. The suggestions are extracted from a collection of strings.

ListView
A *ListView* is a View that shows items in a vertically scrolling list. The items are acquired from a *ListAdapter*.

Reference: http://developer.android.com/guide/topics/ui/layout-objects.html

XML Layouts in Eclipse

Android considers XML-based layouts to be ***resources***, consequently layout files are stored in the **res/layout** directory inside your Android project.

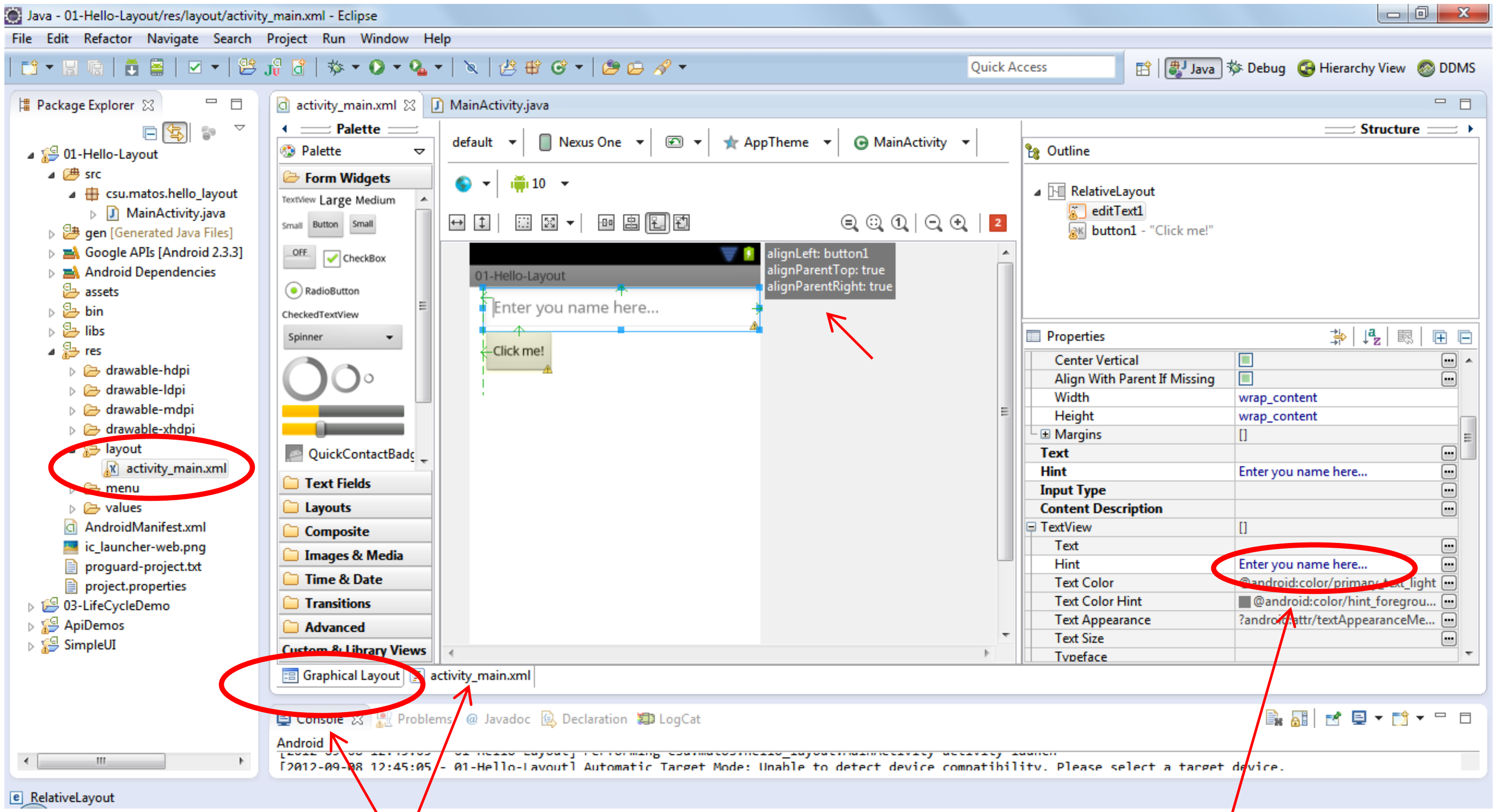


```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/button1"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:ems="10"
        android:hint="Enter you name here..." >

        <requestFocus />
    </EditText>

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/editText1"
        android:layout_marginLeft="15dp"
        android:text="Click me!" />

</RelativeLayout>
```

XML version of a window

XML Layouts in Eclipse

A reasonable UI representation of an XML file can be seen in Eclipse by clicking the [Graphical Layout] tab of the **res/layout/main.xml** resource



```
[2012-09-08 12:45:05 - 01-Hello-Layout] Automatic Target Mode: Unable to detect device compatibility. Please select a target device.
```

Switch between UI and XML view

Control properties

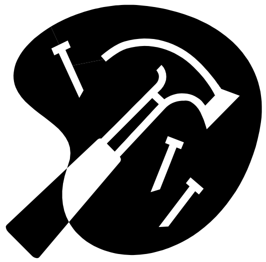
Tools for Android UI/XML Design

ASIDE - CREATING ANDROID UI & APPS

You could create Layout XML files using UI tools such as:

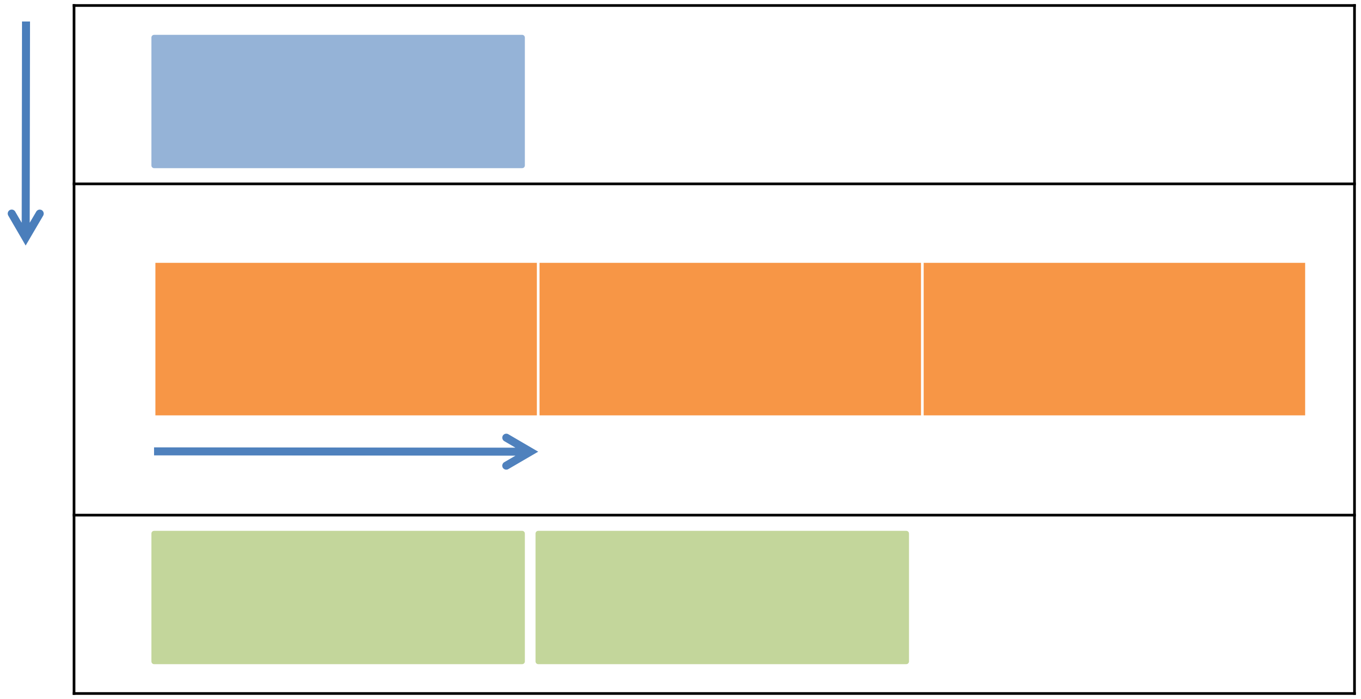
- **Eclipse ADT UI Designer.** It is getting better, integrates code & UI design in the same platform. Not yet at the same high level o Apple's iOS and Microsoft Visual Studio UI Tools.
- **DroidDraw** Very simple, incomplete, not integrated to the Eclipse IDE, aging! http://www.droiddraw.org/
- **App Inventor** (very promising & ambitious, 'hides' coding …) *http://appinventor.googlelabs.com/about/index.html*

More on this issue later…



How to create complex UIs?

- The **LinearLayout** is arguably the most common type of container.
- It offers a "box" model where inner elements could be placed side-by-side or up-and-down.
- In general, complex UI designs could be made by combining simpler *nested* boxes and stacking them in either a *horizontal* or *vertical* orientation.

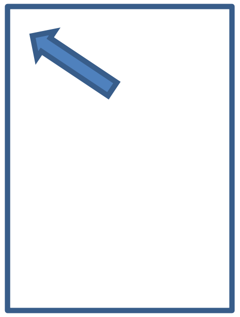


Common Layouts

We will discuss the following common and useful layouts:
Frame, Linear, Relative, Table, and **Absolute.**

1. FrameLayout

- FrameLayout is the simplest type of layout.
- Useful as outermost container holding a window.
- Allows you to define how much of the screen (high, width) is to be used.
- All its children elements are *aligned to the top left corner of the screen.*;



The LinearLayout

1. Linear Layout

The widgets or inner containers held in a LinearLayout are collocated one next to the other in either a *column* or a *row.*

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

- orientation,
- fill model,
- weight,
- gravity,
- padding ,
- margin

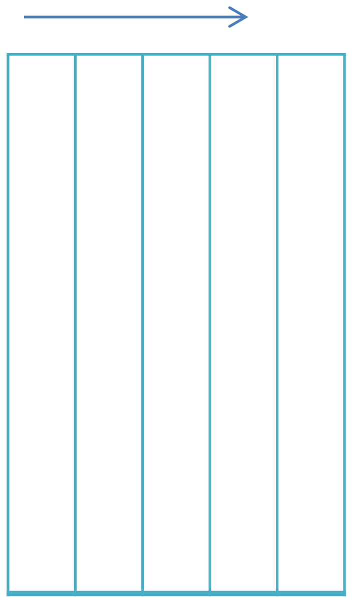
The LinearLayout

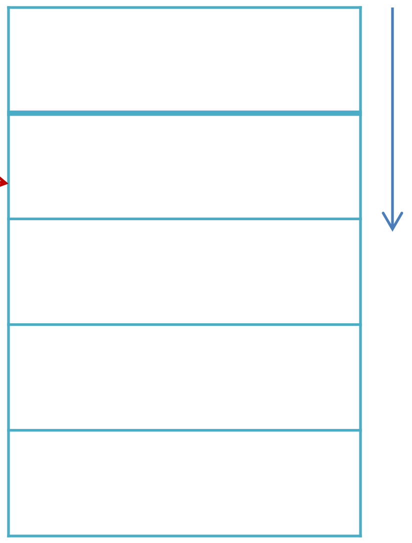
1. Linear Layout

Orientation

The **android:orientation** property can be set to the values: **horizontal** for rows or **vertical** for columns.

```
android:orientation="horizontal"

android:orientation="vertical"
```

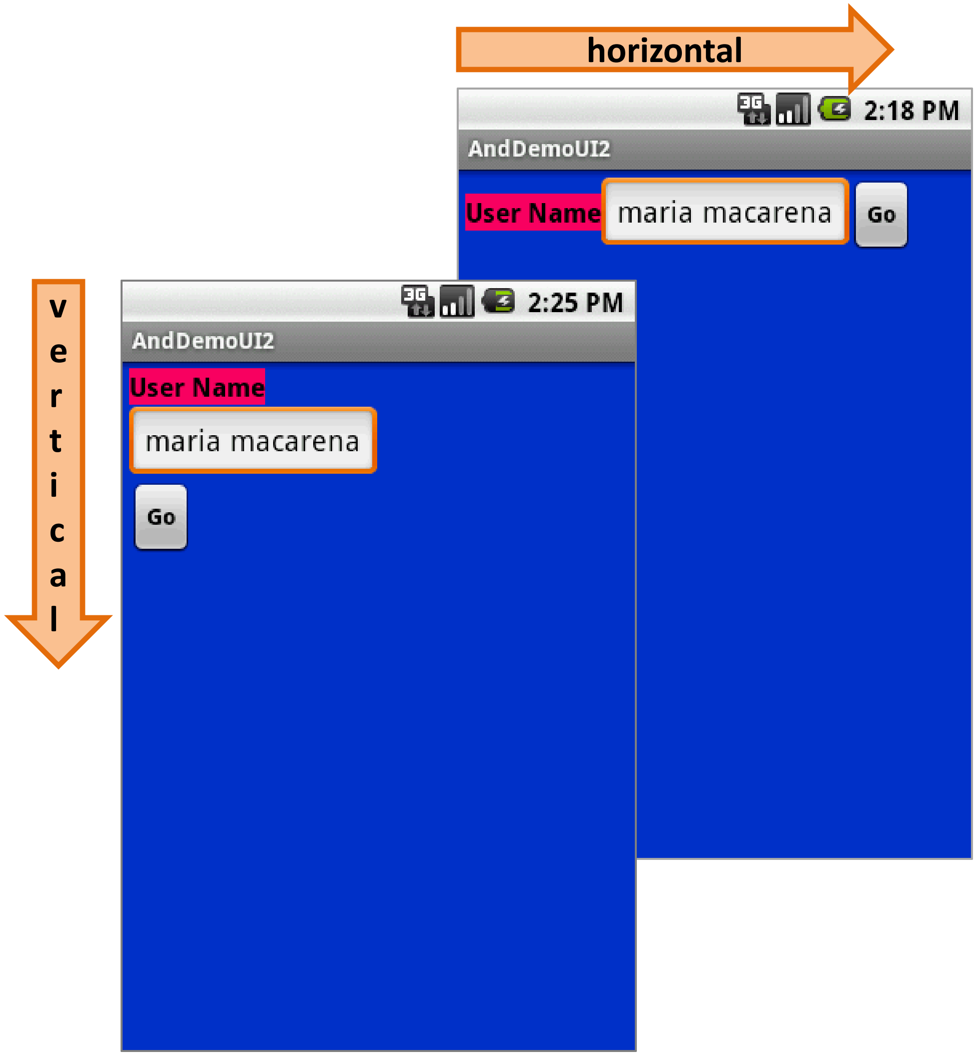




The orientation can be modified at runtime by invoking *setOrientation()*

The LinearLayout - Orientation

1.1 Linear Layout: Orientation

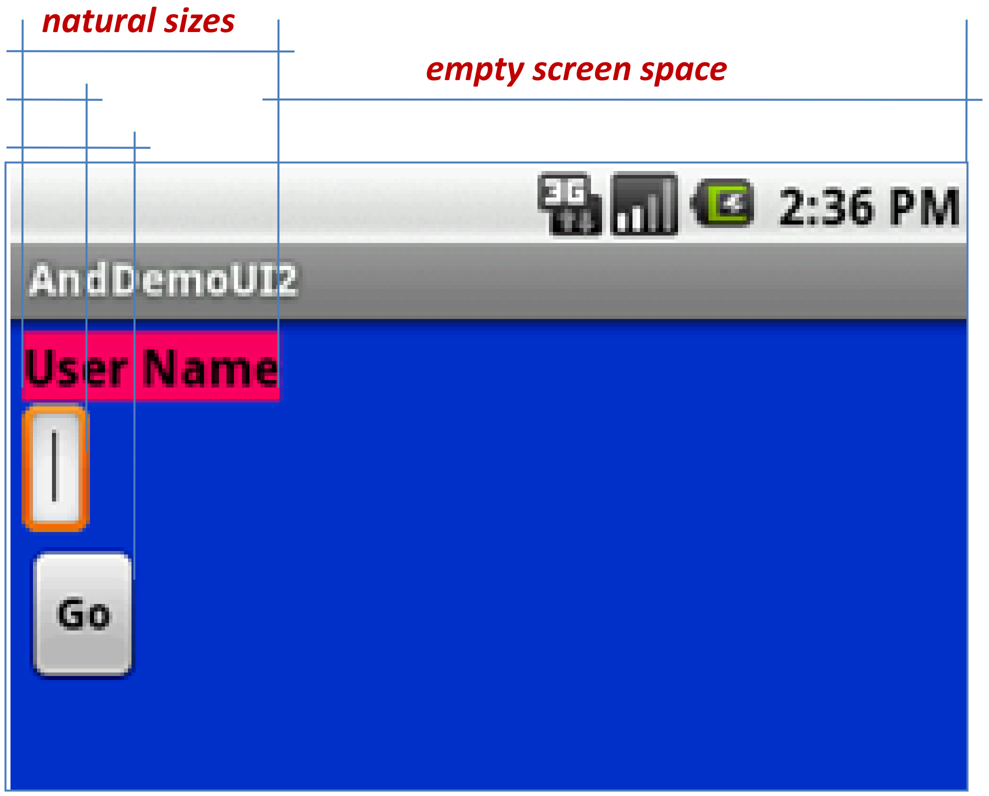


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
        android:id="@+id/myLinearLayout"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:background="#ff0033cc"
        android:padding="4dip"
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="horizontal"  >
<TextView
        android:id="@+id/labelUserName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#ffff0066"
        android:text="User Name"
        android:textSize="16sp"
        android:textStyle="bold"
        android:textColor="#ff000000"   />

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

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

</LinearLayout>
```

The LinearLayout – Fill Model

1.2 Linear Layout: Fill Model

- Widgets have a "natural" size based on their included text.
- You may want to specify how tall & wide a widget should be even if no text is involved (as is the case of the empty text box shown below).



The LinearLayout – Fill Model

1.2 Linear Layout: Fill Model

All widgets inside a LinearLayout **must** include ‘width’ and ‘height’ attributes to establish the issue of empty space around them.

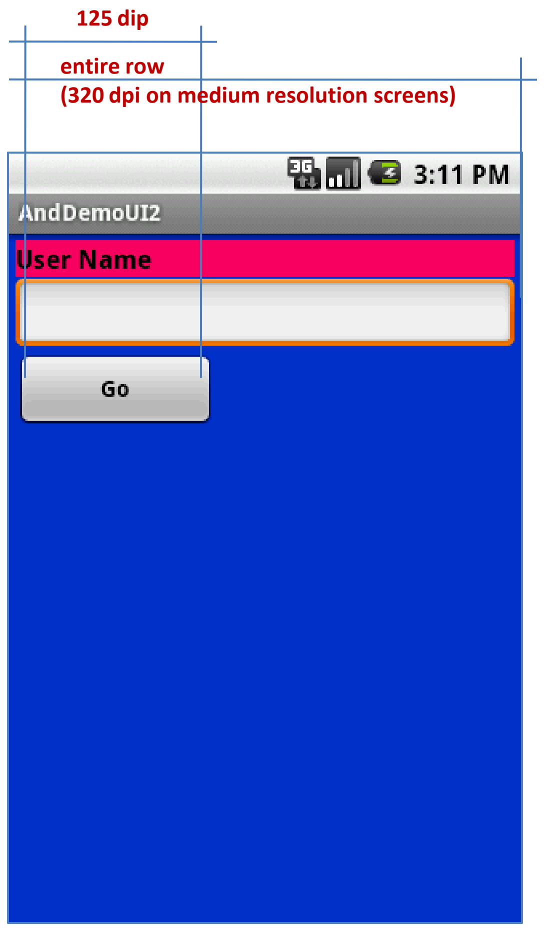
```
android:layout_width
android:layout_height
```

Values used in defining height and width can be:

1. A specific dimension such as **125dip** (device independent pixels, a.k.a. **dp**)
2. **wrap_content** indicates the widget should just fill up its natural space (if it is too big other options such as **word-wrap** could be used to make it fit).
3. **match_parent** (previously called ‘**fill_parent**’) indicates the widget wants to be as big as the enclosing parent.

The LinearLayout – Fill Model

1.2 Linear Layout: Fill Model



Medium resolution is: 320 x 480 dpi.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
        android:id="@+id/myLinearLayout"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical"
        android:background="#ff0033cc"
        android:padding="4dip"
        xmlns:android="http://schemas.android.com/apk/res/android"
>
<TextView
        android:id="@+id/labelUserName"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#ffff0066"
        android:text="User Name"
        android:textSize="16sp"
        android:textStyle="bold"
        android:textColor="#ff000000"  />

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

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

</LinearLayout>
```

Row-wise

Use all the row

Specific size: 125dip

The LinearLayout – Weight

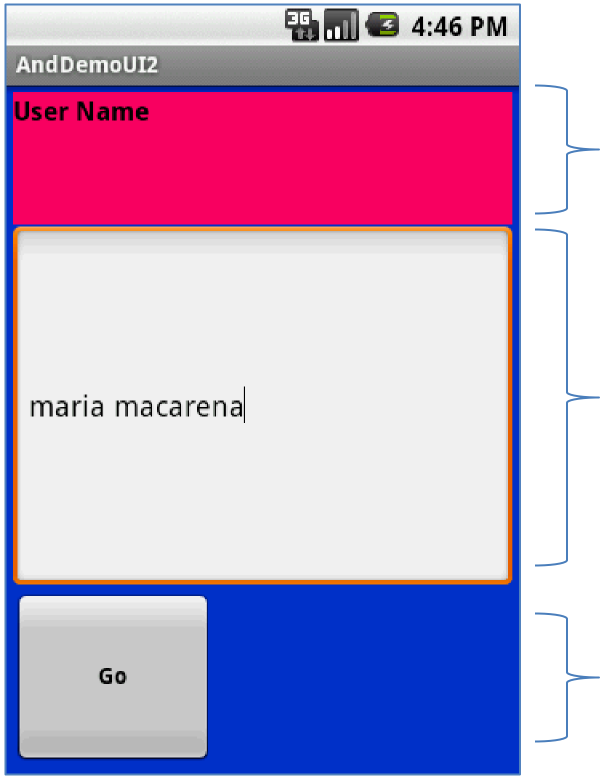
1.2 Linear Layout: Weight
Indicates how much of the extra space in the LinearLayout will be allocated to the view. Use **0** if the view should not be stretched. The bigger the weight the larger the extra space given to that widget.

Example
The XML specification for the window is very similar to the previous example.

The TextView and Button controls have the additional property
`android:layout_weight="1"`

whereas the EditText control has
`android:layout_weight="2"`

Default value is 0

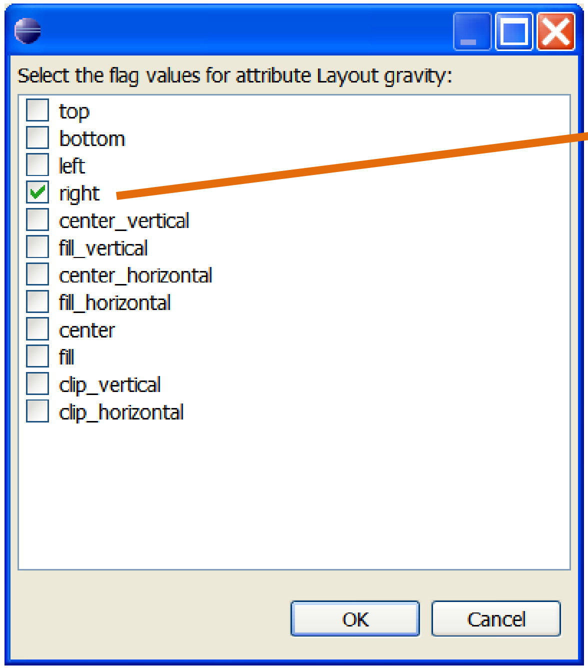


Takes: 2 /(1+1+2) of the screen space

The LinearLayout – Gravity

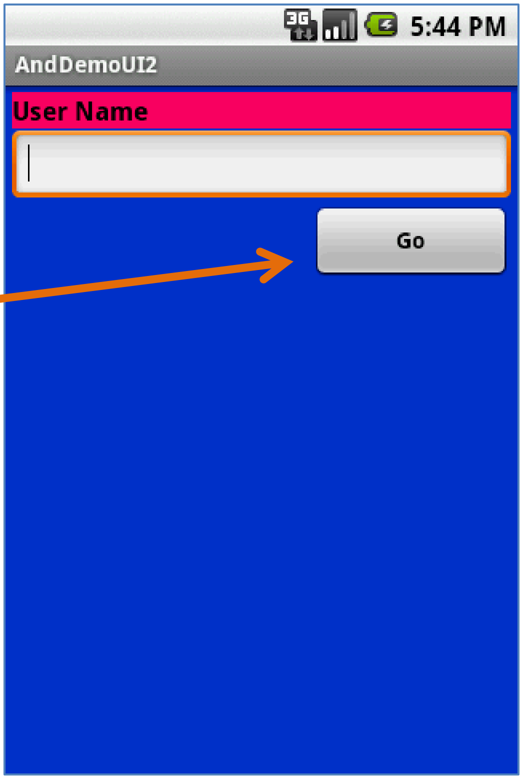
1.3 Layout_Gravity

- It is used to indicate how a control will align on the screen.
- By default, widgets are *left*- and *top*-aligned.
- You may use the XML property **android:layout_gravity="…"** to set other possible arrangements: *left, center, right, top, bottom,* etc.



Select the flag values for attribute Layout gravity:

- [ ] top
- [ ] bottom
- [ ] left
- [✓] right
- [ ] center_vertical
- [ ] fill_vertical
- [ ] center_horizontal
- [ ] fill_horizontal
- [ ] center
- [ ] fill
- [ ] clip_vertical
- [ ] clip_horizontal



Button has **right** layout_gravity

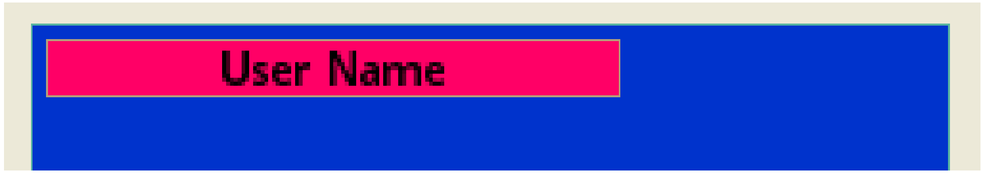
The LinearLayout – Gravity

1.3 CAUTION: gravity vs. layout_gravity

The difference between:

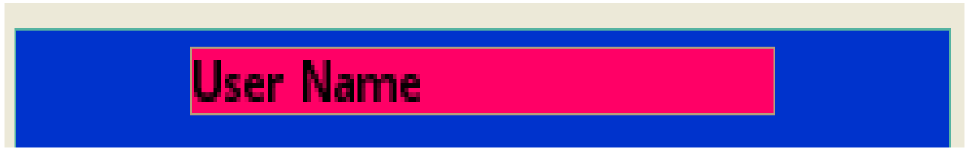
android:gravity
indicates how to place an object within a container. In the example the text is centered

```
android:gravity="center"
```



android:layout_gravity
positions the view with respect to its

```
android:layout_gravity="center"
```



The LinearLayout – Padding

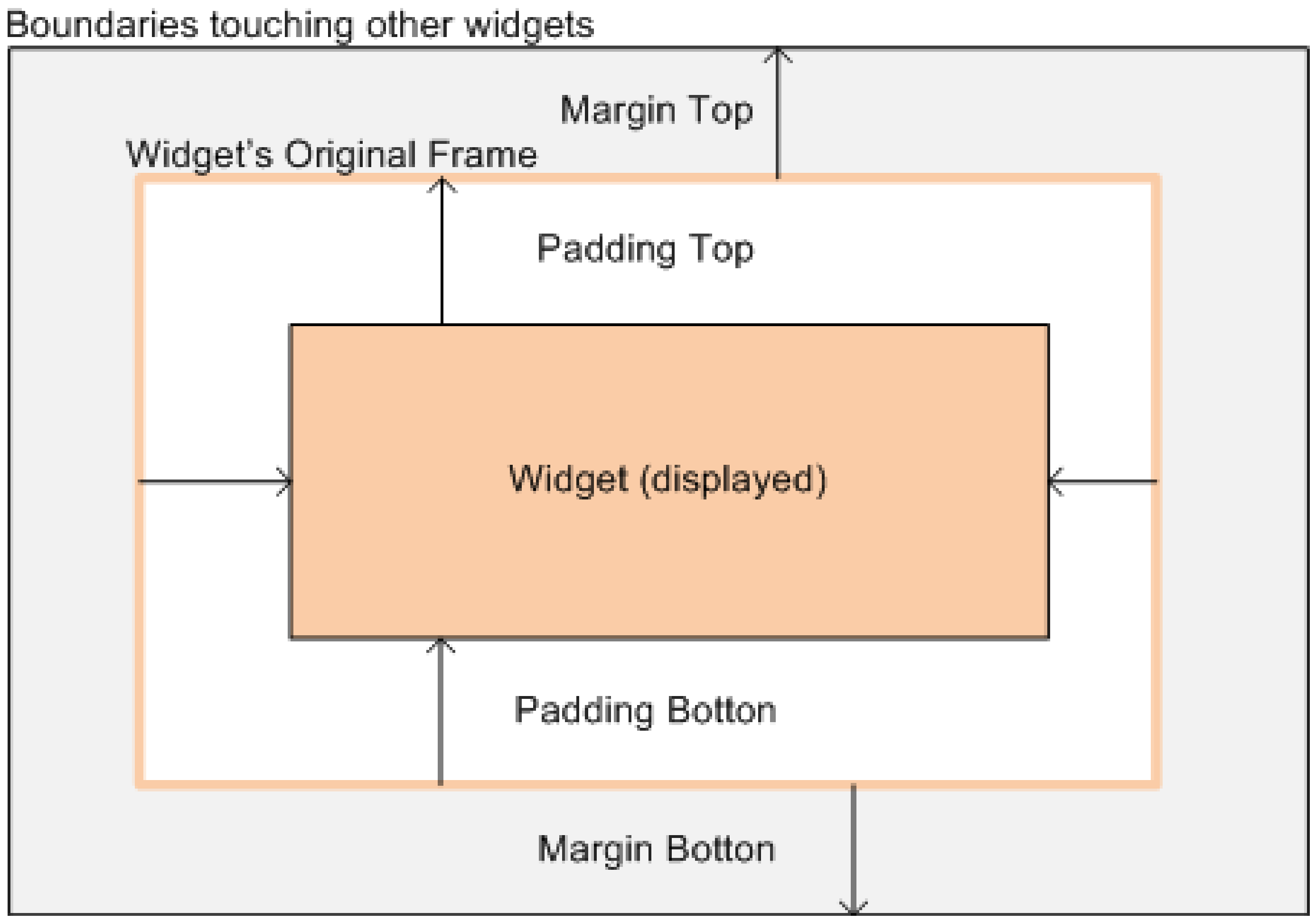
1.4 Linear Layout: Padding

- The padding specifies how much extra space there is between the boundaries of the widget's "cell" and the actual widget contents.

- Either use
  - **android:padding** property
  - or call method ***setPadding()*** at runtime.

The LinearLayout – Padding

1.3 Linear Layout: Padding and Marging

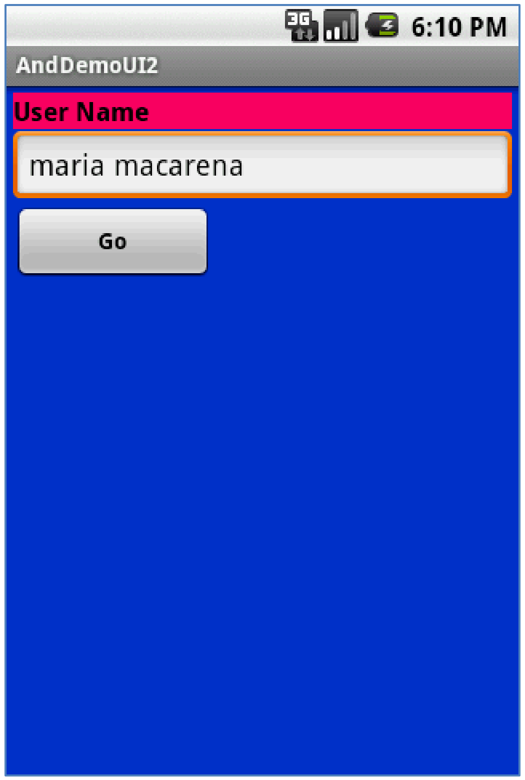


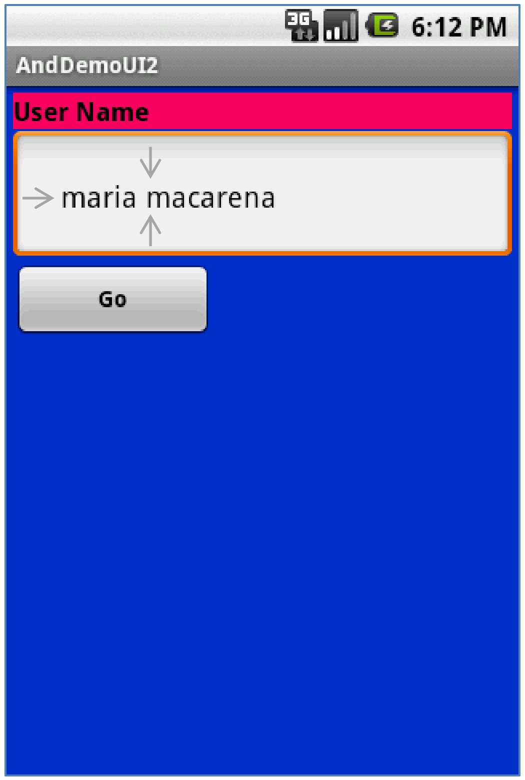
The LinearLayout – Padding

1.3 Linear Layout: Internal Margins Using Padding

Example:

The EditText box has been changed to display 30dip of padding all around



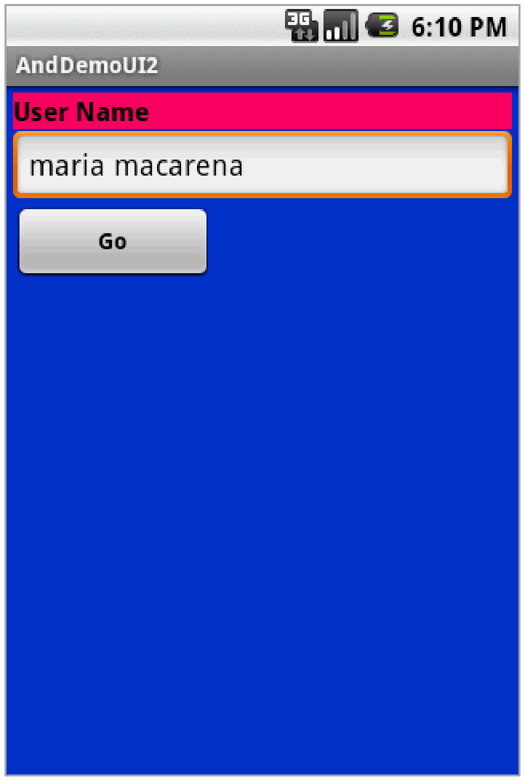


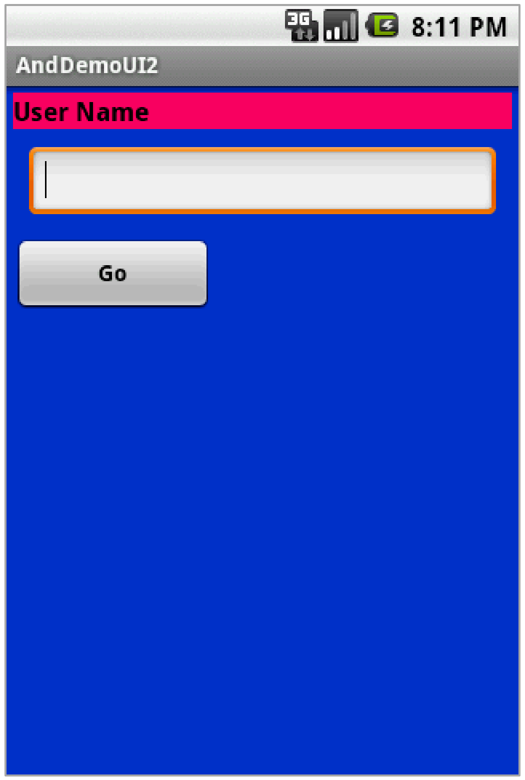
```
<EditText
android:id="@+id/ediName"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textSize="18sp"
android:padding="30dip"

>
</EditText>
...
```

The LinearLayout – Margin

1.4 Linear Layout: (External) Margin

- Widgets –by default– are tightly packed next to each other.
- To increase space between them use the **android:layout_margin** attribute





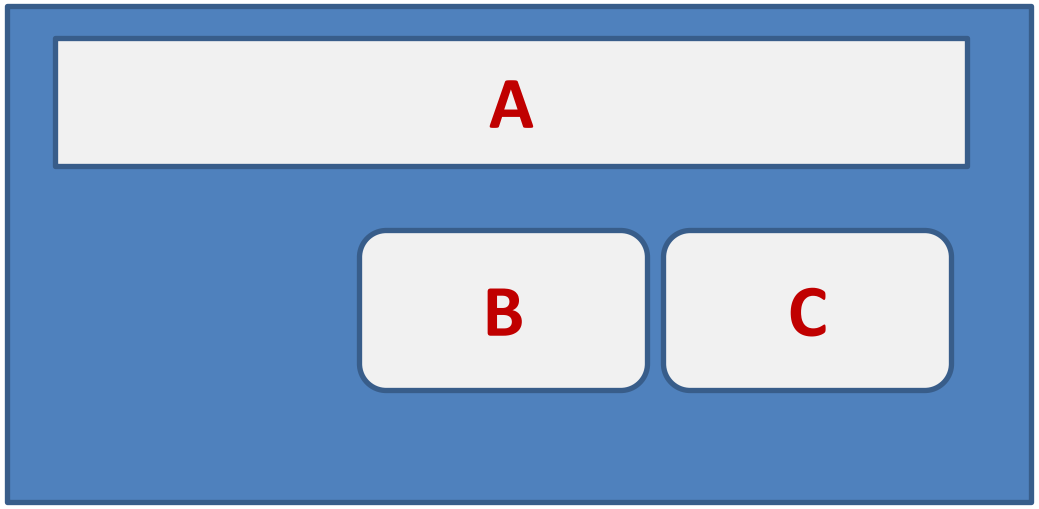
Increased inter-widget space

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

android:layout_margin="6dip"

>
</EditText>
...
```

The Relative Layout

2. Relative Layout

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

The Relative Layout

2. Relative Layout - Referring to the container

Below there is a list of some positioning XML boolean properties (true/false) mapping a widget according to its location **respect to the parent's place**.

- **android:layout_alignParentTop** the widget's top should align with the top of the container.
- **android:layout_alignParentBottom** the widget's bottom should align with the bottom of the container
- **android:layout_alignParentLeft** the widget's left side should align with the left side of the container
- **android:layout_alignParentRight** the widget's right side should align with the right side of the container

- **android:layout_centerInParent** the widget should be positioned both horizontally and vertically at the center of the container
- **android:layout_centerHorizontal** the widget should be positioned horizontally at the center of the container
- **android:layout_centerVertical** the widget should be positioned vertically at the center of the container

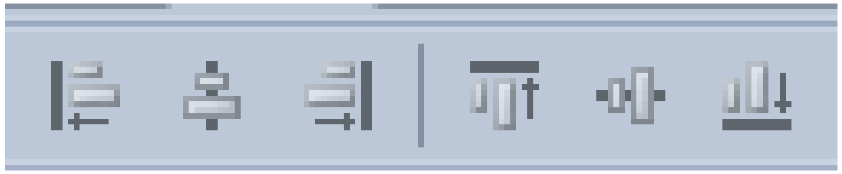
The Relative Layout

2. Relative Layout – Referring to other widgets

The following properties manage positioning of a widget **respect to other widgets:**

- **android:layout_above** indicates that the widget should be placed above the widget referenced in the property
- **android:layout_below** indicates that the widget should be placed below the widget referenced in the property
- **android:layout_toLeftOf** indicates that the widget should be placed to the left of the widget referenced in the property
- **android:layout_toRightOf** indicates that the widget should be placed to the right of the widget referenced in the property

The Relative Layout



2. Relative Layout – Referring to other widgets – cont.

- **android:layout_alignTop** indicates that the widget's top should be aligned with the top of the widget referenced in the property
- **android:layout_alignBottom** indicates that the widget's bottom should be aligned with the bottom of the widget referenced in the property
- **android:layout_alignLeft** indicates that the widget's left should be aligned with the left of the widget referenced in the property
- **android:layout_alignRight** indicates that the widget's right should be aligned with the right of the widget referenced in the property
- **android:layout_alignBaseline** indicates that the baselines of the two widgets should be aligned

The Relative Layout

2. Relative Layout – Referring to other widgets

When using relative positioning you need to:

1. Put identifiers (`android:id` attributes) on *all elements* that you will be referring to.

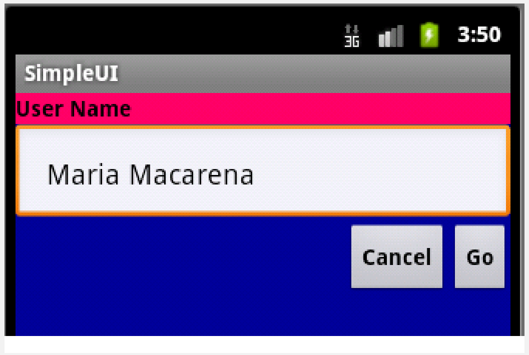
2. XML elements are named using: `@+id/...` For instance an EditText box could be called: `android:id="@+id/txtUserName"`

3. You must refer only to widgets that have been defined. For instance a new control to be positioned below the previous EditText box could refer to it using: `android:layout_below="@+id/txtUserName"`

The Relative Layout

2. Relative Layout – Example

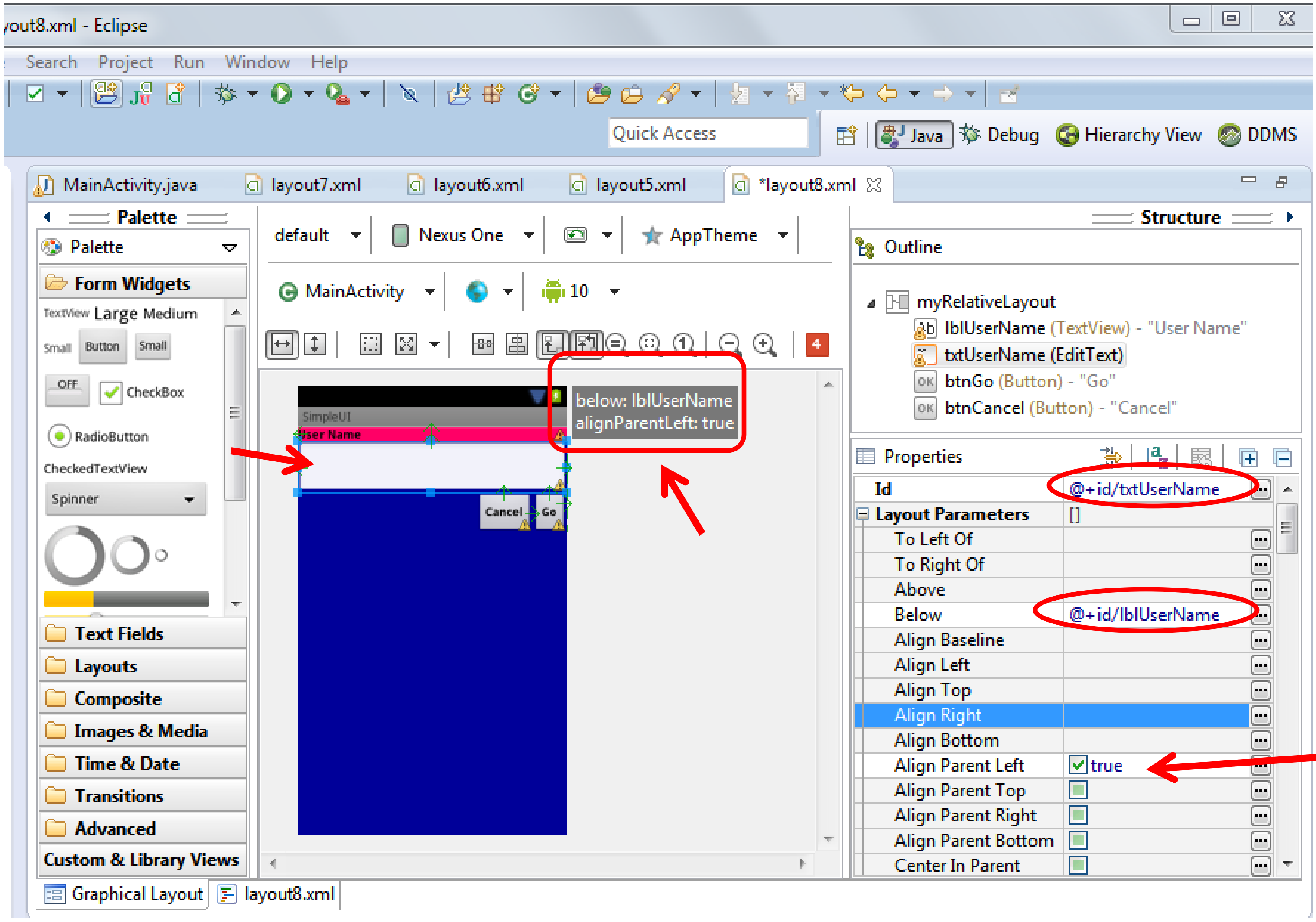
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myRelativeLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#ff000099" >

    <TextView
        android:id="@+id/lblUserName"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:background="#ffff0066"
        android:text="User Name"
        android:textColor="#ff000000"
        android:textStyle="bold" >
    </TextView>
```



```
    <EditText
        android:id="@+id/txtUserName"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/lblUserName"
        android:padding="20dip" >
    </EditText>

    <Button
        android:id="@+id/btnGo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignRight="@+id/txtUserName"
        android:layout_below="@+id/txtUserName"
        android:text="Go"
        android:textStyle="bold" >
    </Button>

    <Button
        android:id="@+id/btnCancel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/txtUserName"
        android:layout_toLeftOf="@+id/btnGo"
        android:text="Cancel"
        android:textStyle="bold" >
    </Button>

</RelativeLayout>
```

The Relative Layout

2. Relative Layout (as of Sept 2012)



ayout8.xml - Eclipse

Search Project Run Window Help

MainActivity.java | layout7.xml | layout6.xml | layout5.xml | *layout8.xml

below: lblUserName
alignParentLeft: true

Outline

- myRelativeLayout
  - lblUserName (TextView) - "User Name"
  - txtUserName (EditText)
  - btnGo (Button) - "Go"
  - btnCancel (Button) - "Cancel"

Properties

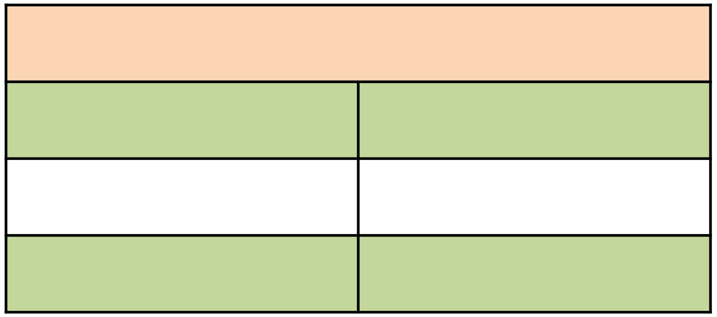
Id	@+id/txtUserName
Layout Parameters	[]
To Left Of	
To Right Of	
Above	
Below	@+id/lblUserName
Align Baseline	
Align Left	
Align Top	
Align Right	
Align Bottom	
Align Parent Left	✓ true
Align Parent Top	☐
Align Parent Right	☐
Align Parent Bottom	☐
Center In Parent	☐

Using **Eclipse ADT Layout Editor** for designing a *RelativeLayout*.

The Table Layout

3. Table Layout

1. Android's **TableLayout** uses a grid to position your widgets.
2. Cells in the grid are identifiable by *rows* and *columns*.
3. Columns might *shrink* or *stretch* to accommodate their contents.
4. The element **TableRow** is used to define a new row in which widgets can be allocated.
5. The number of columns in a TableRow is determined by the total of side-by-side widgets placed on the row.



Basic XML Layouts - Containers

3. Table Layout

The number of columns in a row is determined by Android.

So if you have three rows, one with two widgets, one with three widgets, and one with four widgets, there will be at least four columns.

0		1	
0		1	2
0	1	2	3

Basic XML Layouts - Containers

3. Table Layout

However, a single widget can take up more than one column by including the **android:layout_span** property, indicating the number of columns the widget spans (this is similar to the **colspan** attribute one finds in table cells in **HTML**)

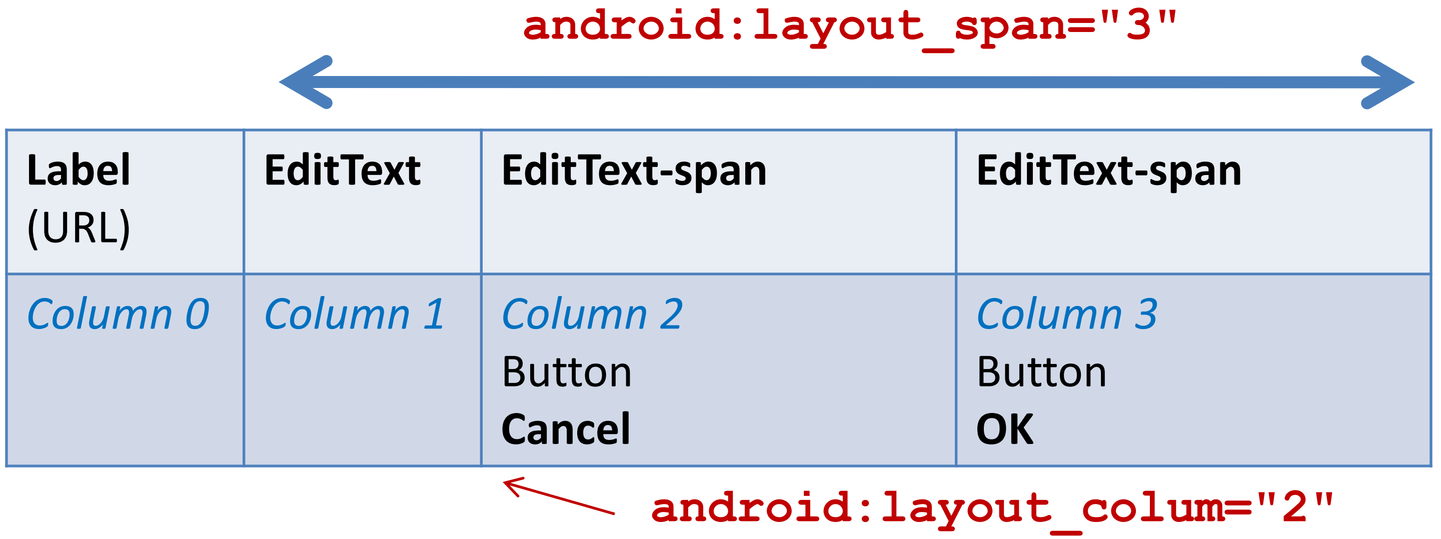
```
<TableRow>
   <TextView android:text="URL:" />
   <EditText
   android:id="@+id/entry"
   android:layout_span="3" />
</TableRow>
```

Basic XML Layouts - Containers

3. Table Layout

Ordinarily, widgets are put into the first available column of each row.

In the example below, the label (“*URL*”) would go in the first column (*column 0, as columns are counted starting from 0*), and the TextField would go into a spanned set of three columns (columns 1 through 3).



`android:layout_span="3"`

Label (URL)	**EditText**	**EditText-span**	**EditText-span**
Column 0	*Column 1*	*Column 2* Button **Cancel**	*Column 3* Button **OK**

`android:layout_colum="2"`

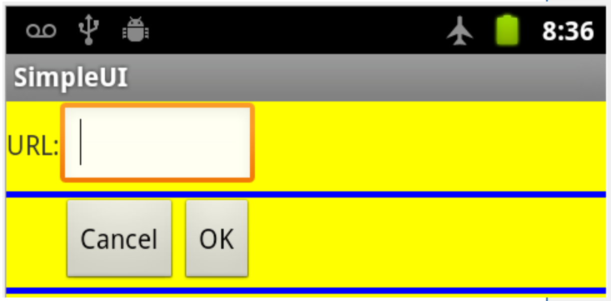
Basic XML Layouts - Containers

3. Table Layout Example

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myTableLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#ffffff00"
    android:orientation="vertical" >
    <TableRow>
        <TextView android:text="URL:" />
        <EditText
            android:id="@+id/ediUrl"
            android:layout_span="3" />
    </TableRow>
    <View
        android:layout_height="3dip"
        android:background="#0000FF" />

    <TableRow>
        <Button
            android:id="@+id/cancel"
            android:layout_column="2"
            android:text="Cancel" />
        <Button
            android:id="@+id/ok"
            android:text="OK" />
    </TableRow>
    <View
        android:layout_height="3dip"
        android:background="#0000FF" />
</TableLayout>
```

Strech up to column 3

Skip columns: 0, 1

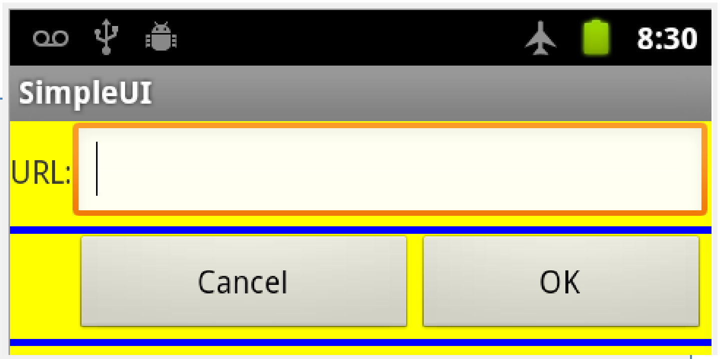


Note to the reader:
Experiment changing layout_span to 1, 2, 3

Basic XML Layouts - Containers

3. Table Layout

In our running example we stretch columns 2, 3, and 4 to fill the rest of the row.



```
...
<TableLayout
android:id="@+id/myTableLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#ff0033cc"
android:orientation="vertical"
android:stretchColumns ="2,3,4"
xmlns:android="http://schemas.android.com/apk/res/android"
>

. . .
```

***TODO**: try to stretch one column at the time 1, then 2, and so on.*

Basic XML Layouts - Containers

4. ScrollView Layout

When we have more data than what can be shown on a single screen you may use the **ScrollView** control.

It provides a sliding or scrolling access to the data. This way the user can only see part of your layout at one time, but the rest is available via scrolling.

This is similar to browsing a large web page that forces the user to scroll up the page to see the bottom part of the form.

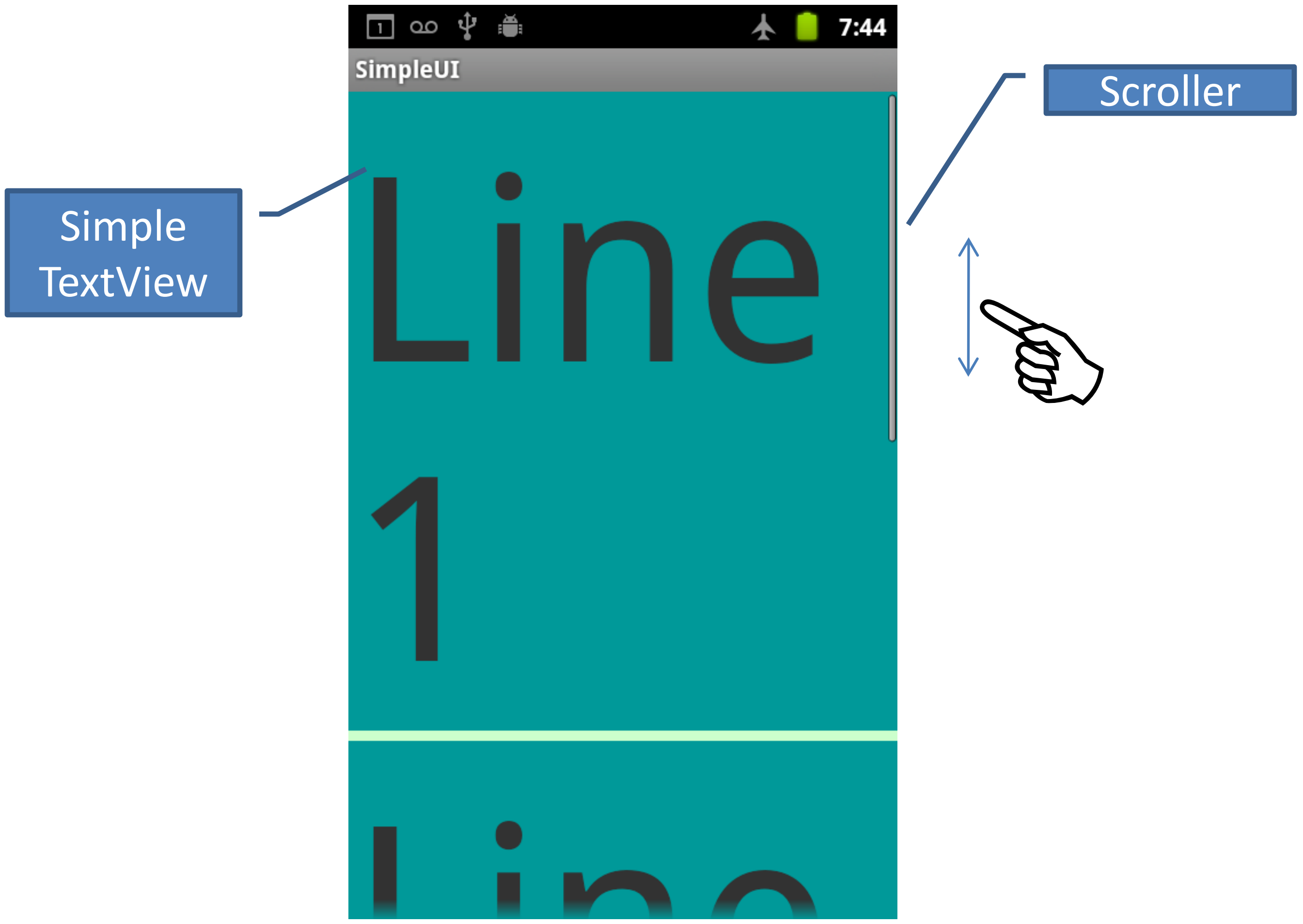
Basic XML Layouts - Containers

4. Example ScrollView Layout

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

<ScrollView
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myScrollView1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#ff009999" >

    <LinearLayout
        android:id="@+id/myLinearLayoutVertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical" >

        <TextView
            android:id="@+id/textView1"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Line1"
            android:textSize="150dip" />
        <View
            android:layout_width="fill_parent"
            android:layout_height="6dip"
            android:background="#ffccffcc" />

        <TextView
            android:id="@+id/textView2"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Line2"
            android:textSize="150dip" />

        <View
            android:layout_width="fill_parent"
            android:layout_height="6dip"
            android:background="#ffccffcc" />

        <TextView
            android:id="@+id/textView3"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Line3"
            android:textSize="150dip" />
    </LinearLayout>

</ScrollView>
```

Basic XML Layouts - Containers

4. Example ScrollView Layout

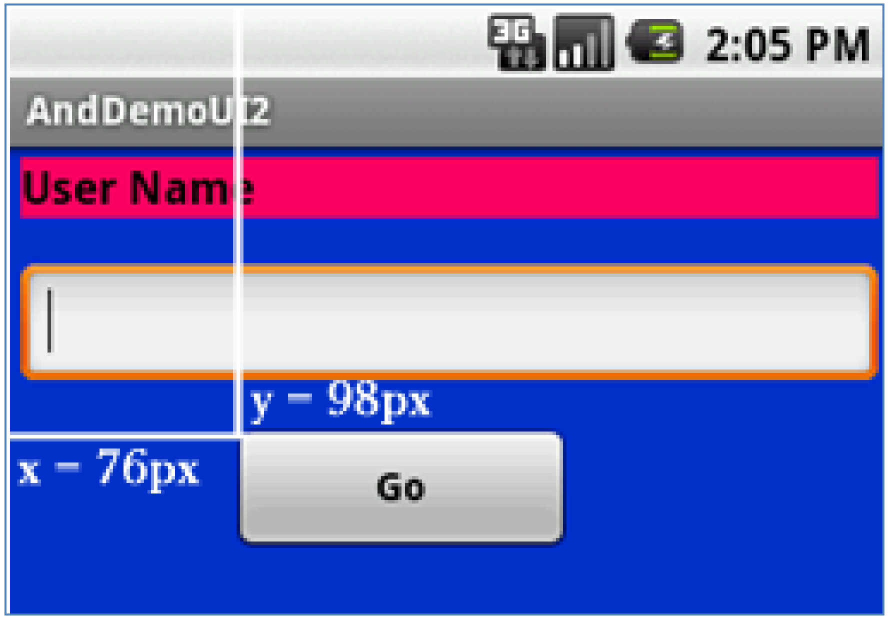


Basic XML Layouts - Containers

5. Miscellaneous.
Absolute Layout

- 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.



Basic XML Layouts - Containers

5. Miscellaneous Absolute Layout (cont.)

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

<TextView
android:id="@+id/tvUserName"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:background="#ffff0066"
android:text="User Name"
android:textSize="16sp"
android:textStyle="bold"
android:textColor="#ff000000"
android:layout_x="0dip"
android:layout_y="10dip"
>
</TextView>
<EditText
android:id="@+id/etName"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textSize="18sp"
android:layout_x="0dip"
android:layout_y="38dip"
>
</EditText>

<Button
android:layout_width="120dip"
android:text="Go"
android:layout_height="wrap_content"
android:textStyle="bold"
android:id="@+id/btnGo"
android:layout_x="100dip"
android:layout_y="170dip"   />
</AbsoluteLayout>
```

Button location

A Detailed List of Widgets

For a detailed list consult:

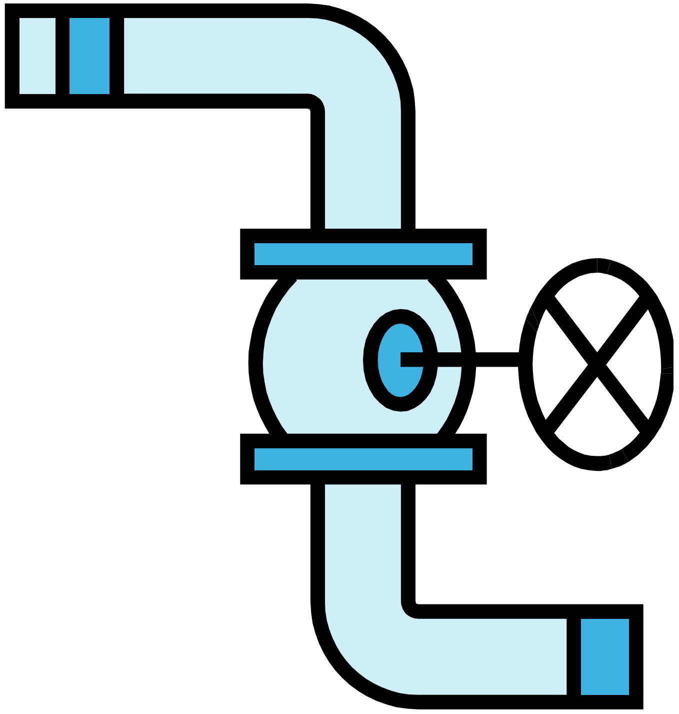
http://developer.android.com/reference/android/widget/package-summary.html

AbsListView
AbsListView.LayoutParams
AbsoluteLayout
AbsoluteLayout.LayoutParams
AbsSeekBar
AbsSpinner
AdapterView<T extends Adapter>
AdapterContextMenuInfo
AlphabetIndexer
AnalogClock
ArrayAdapter<T>
AutoCompleteTextView
BaseAdapter
BaseExpandableListAdapter
Button
CheckBox
CheckedTextView
Chronometer
CompoundButton
CursorAdapter
CursorTreeAdapter
DatePicker
DialerFilter
DigitalClock
EditText
ExpandableListView
ExpandableListContextMenuInfo
Filter
Filter.FilterResults
FrameLayout
FrameLayout.LayoutParams
Gallery
Gallery.LayoutParams
GridView
HeaderViewListAdapter
HorizontalScrollView
ImageButton
ImageSwitcher
ImageView
LinearLayout
LinearLayout.LayoutParams
ListView
ListView.FixedViewInfo
MediaController
MultiAutoCompleteTextView
CommaTokenizer
PopupWindow
ProgressBar
RadioButton
RadioGroup
RadioGroup.LayoutParams
RatingBar
RelativeLayout
RelativeLayout.LayoutParams
RemoteViews
ResourceCursorAdapter
ResourceCursorTreeAdapter
Scroller
ScrollView
SeekBar
SimpleAdapter
SimpleCursorAdapter
SimpleCursorTreeAdapter
SimpleExpandableListAdapter
SlidingDrawer
Spinner
TabHost
TabHost.TabSpec
TableLayout
TableLayout.LayoutParams
TableRow
TableRow.LayoutParams
TabWidget
TextSwitcher
TextView
TextView.SavedState
TimePicker
Toast
ToggleButton
TwoLineListItem
VideoView
ViewAnimator
ViewFlipper
ViewSwitcher
ZoomButton
ZoomControls

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.

XLM Layout
<xml….
. . .
. . .
</xml>



JAVA code
public class ….
{
. . .
. . .
}

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.

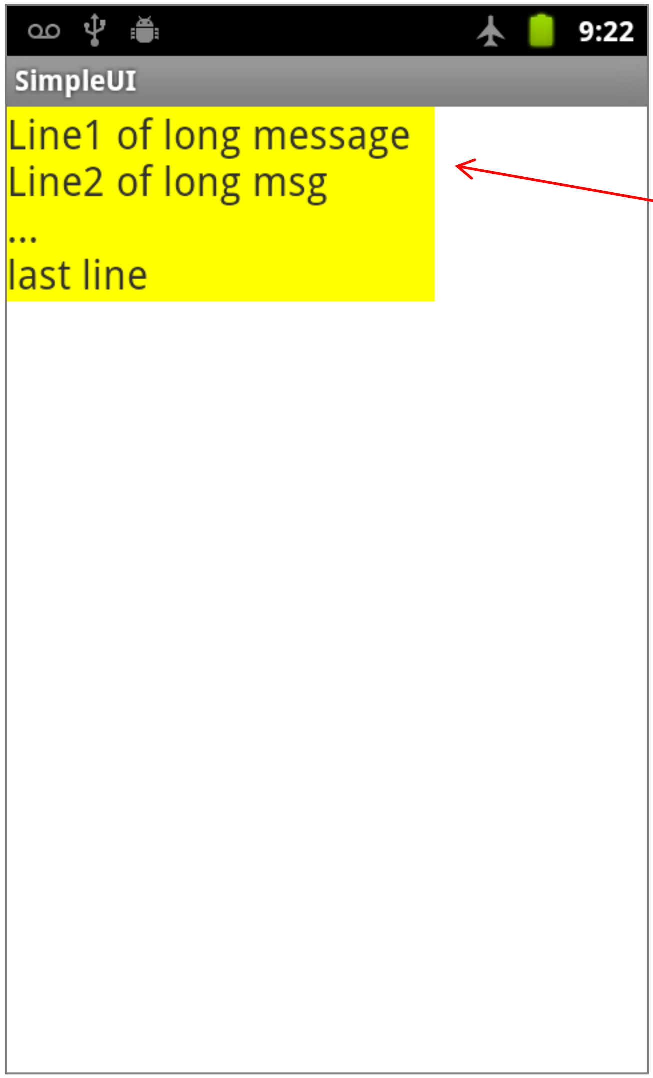
Attaching Layouts to Java Code

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:

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

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

Basic Widgets: Labels

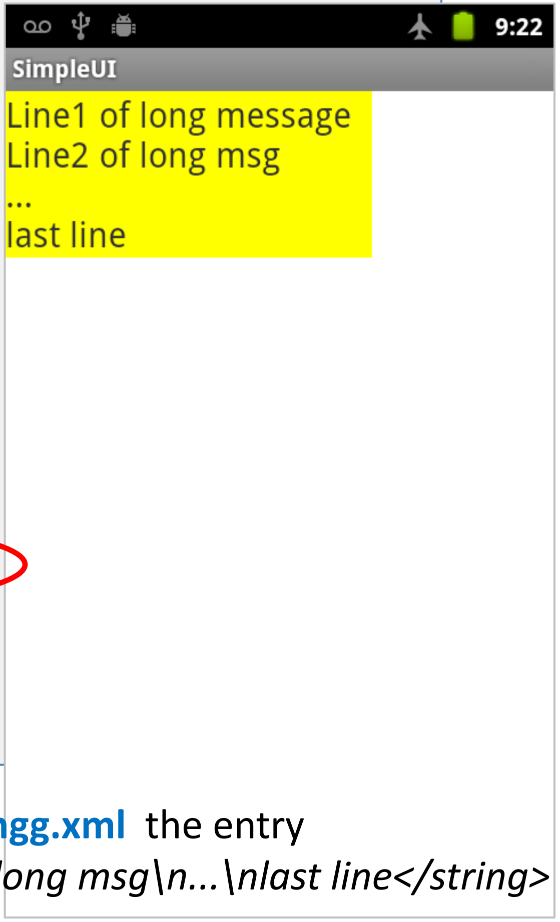


```
Line1 of long message
Line2 of long msg
...
last line
```

- A **label** is called in android a **TextView**.
- TextViews are typically used for output to display a caption.
- TextViews are *not* editable, therefore they take no input.

Basic Widgets: Labels

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/widget32"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/txt1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#ffffff00"
        android:inputType="none"
        android:text="@string/long_msg_1"
        android:textSize="20sp" />

</LinearLayout>
```



```
Line1 of long message
Line2 of long msg
...
last line
```

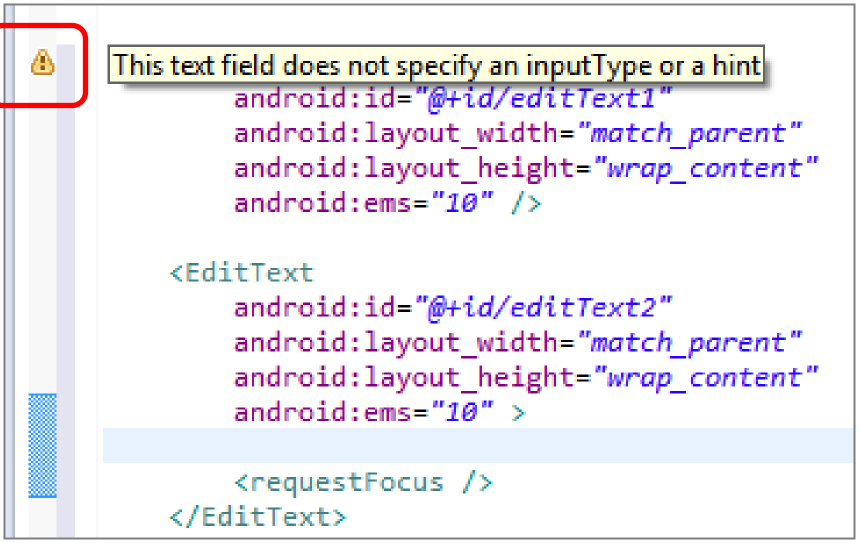
Hint on Better Programming Style: Add to the **res/values/stringg.xml** the entry
<string name=“*long_msg_1">Line1 of long message\nLine2 of long msg\n...\nlast line</string>*

EditText Caution

WARNING

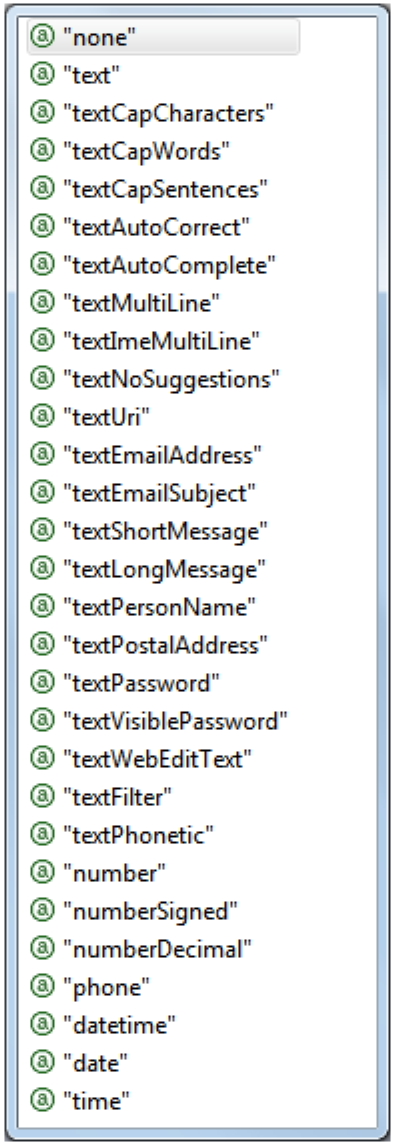
This text field does not specify an InputType or a hint

is just a warning requesting your help to improve the working of a TextView. Add the clause **android:hint="…some hint here…"** and/or **android:InputType="…choice…"** where choices are



This text field does not specify an inputType or a hint

```
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10" />

    <EditText
        android:id="@+id/editText2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10" >

        <requestFocus />
    </EditText>
```



"none"
"text"
"textCapCharacters"
"textCapWords"
"textCapSentences"
"textAutoCorrect"
"textAutoComplete"
"textMultiLine"
"textImeMultiLine"
"textNoSuggestions"
"textUri"
"textEmailAddress"
"textEmailSubject"
"textShortMessage"
"textLongMessage"
"textPersonName"
"textPostalAddress"
"textPassword"
"textVisiblePassword"
"textWebEditText"
"textFilter"
"textPhonetic"
"number"
"numberSigned"
"numberDecimal"
"phone"
"datetime"
"date"
"time"

Basic Widgets / Attributes & Methods: TextView

http://developer.android.com/reference/android/widget/TextView.html

XML Attribute / Equivalent Method	**Description**
android:autoLink setAutoLinkMask(int)	Controls whether links such as urls and email addresses are automatically found and converted to clickable links.
android:autoText setKeyListener(KeyListener)	If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
android:bufferType setText(CharSequence,TextView.BufferType)	Determines the minimum type that getText() will return.
android:capitalize setKeyListener(KeyListener)	If set, specifies that this TextView has a textual input method and should automatically capitalize what the user types.
android:cursorVisible setCursorVisible(boolean)	Makes the cursor visible (the default) or invisible.
android:digits setKeyListener(KeyListener)	If set, specifies that this TextView has a numeric input method and that these specific characters are the ones that it will accept.
android:drawableBottom setCompoundDrawablesWithIntrinsicBounds( )	The drawable to be drawn below the text.

Basic Widgets / Attributes & Methods: TextView *cont.*

http://developer.android.com/reference/android/widget/TextView.html

XML Attribute / Equivalent Method	Description
android:drawableEnd	The drawable to be drawn to the end of the text.
android:drawableLeft setCompoundDrawablesWithIntrinsicBounds(int...)	The drawable to be drawn to the left of the text.
android:drawablePadding setCompoundDrawablePadding(int)	The padding between the drawables and the text.
android:drawableRight setCompoundDrawablesWithIntrinsicBounds(int...)	The drawable to be drawn to the right of the text.
android:drawableStart	The drawable to be drawn to the start of the text.
android:drawableTop setCompoundDrawablesWithIntrinsicBounds( )	The drawable to be drawn above the text.
android:editable	If set, specifies that this TextView has an input method.
android:editorExtras setInputExtras(int)	Reference to an **<input-extras>** XML resource containing additional data to supply to an input method, which is private to the implementation of the input method.

Basic Widgets / Attributes & Methods: TextView *cont.*

http://developer.android.com/reference/android/widget/TextView.html

XML Attribute / Equivalent Method	**Description**
android:ellipsize setEllipsize(TextUtils.TruncateAt)	If set, causes words that are longer than the view is wide to be ellipsized instead of broken in the middle.
android:ems setEms(int)	Makes the TextView be exactly this many ems wide.
android:fontFamily setTypeface(Typeface)	Font family (named by string) for the text.
android:freezesText setFreezesText(boolean)	If set, the text view will include its current complete text inside of its frozen icicle in addition to meta-data such as the current cursor position.
android:gravity setGravity(int)	Specifies how to align the text by the view's x- and/or y-axis when the text is smaller than the view.
android:height setHeight(int)	Makes the TextView be exactly this many pixels tall.
android:hint setHint(int)	Hint text to display when the text is empty.
android:imeActionId setImeActionLabel(CharSequence.)	Supply a value for **EditorInfo.actionId** used when an input method is connected to the text view.

Basic Widgets / Attributes & Methods: TextView *cont.*

http://developer.android.com/reference/android/widget/TextView.html

XML Attribute / Equivalent Method	**Description**
android:imeActionLabel setImeActionLabel(CharSequence.)	Supply a value for **EditorInfo.actionLabel** used when an input method is connected to the text view.
android:imeOptions setImeOptions(int)	Additional features you can enable in an IME associated with an editor to improve the integration with your application.
android:includeFontPadding setIncludeFontPadding(boolean)	Leave enough room for ascenders and descenders instead of using the font ascent and descent strictly.
android:inputMethod setKeyListener(KeyListener)	If set, specifies that this TextView should use the specified input method (specified by fully-qualified class name).
android:inputType setRawInputType(int)	The type of data being placed in a text field, used to help an input method decide how to let the user enter text.
android:lineSpacingExtra setLineSpacing(float.)	Extra spacing between lines of text.
android:lineSpacingMultiplier setLineSpacing(float.)	Extra spacing between lines of text, as a multiplier.

Basic Widgets / Attributes & Methods: TextView *cont.*

http://developer.android.com/reference/android/widget/TextView.html

XML Attribute / Equivalent Method	Description
android:minEms setMinEms(int)	Makes the TextView be at least this many ems wide.
android:minHeight setMinHeight(int)	Makes the TextView be at least this many pixels tall.
android:minLines setMinLines(int)	Makes the TextView be at least this many lines tall.
android:minWidth setMinWidth(int)	Makes the TextView be at least this many pixels wide.
android:numeric setKeyListener(KeyListener)	If set, specifies that this TextView has a numeric input method.
android:password setTransformationMethod(TransformationMethod)	Whether the characters of the field are displayed as password dots instead of themselves.
android:phoneNumber setKeyListener(KeyListener)	If set, specifies that this TextView has a phone number input method.

Basic Widgets / Attributes & Methods: TextView *cont.*

http://developer.android.com/reference/android/widget/TextView.html

XML Attribute / Equivalent Method	Description
android:privateImeOptions setPrivateImeOptions(String)	An addition content type description to supply to the input method attached to the text view, which is private to the implementation of the input method.
android:scrollHorizontally setHorizontallyScrolling(boolean)	Whether the text is allowed to be wider than the view (and therefore can be scrolled horizontally).
android:selectAllOnFocus setSelectAllOnFocus(boolean)	If the text is selectable, select it all when the view takes focus.
android:shadowColor setShadowLayer(float...)	Place a shadow of the specified color behind the text.
android:shadowDx setShadowLayer(float...)	Horizontal offset of the shadow.
android:shadowDy setShadowLayer(float...)	Vertical offset of the shadow.
android:shadowRadius setShadowLayer(float...)	Radius of the shadow.
android:singleLine setTransformationMethod(TransformationMethod)	Constrains the text to a single horizontally scrolling line instead of letting it wrap onto multiple lines, and advances focus instead of inserting a newline when you press the enter key.

Basic Widgets / Attributes & Methods: TextView *cont.*

http://developer.android.com/reference/android/widget/TextView.html

XML Attribute / Equivalent Method	Description
android:privateImeOptions setPrivateImeOptions(String)	An addition content type description to supply to the input method attached to the text view, which is private to the implementation of the input method.
android:scrollHorizontally setHorizontallyScrolling(boolean)	Whether the text is allowed to be wider than the view (and therefore can be scrolled horizontally).
android:selectAllOnFocus setSelectAllOnFocus(boolean)	If the text is selectable, select it all when the view takes focus.
android:shadowColor setShadowLayer(float...)	Place a shadow of the specified color behind the text.
android:shadowDx setShadowLayer(float...)	Horizontal offset of the shadow.
android:shadowDy setShadowLayer(float...)	Vertical offset of the shadow.
android:shadowRadius setShadowLayer(float...)	Radius of the shadow.
android:singleLine setTransformationMethod(TransformationMethod)	Constrains the text to a single horizontally scrolling line instead of letting it wrap onto multiple lines, and advances focus instead of inserting a newline when you press the enter key.

Basic Widgets / Attributes & Methods: TextView *cont.*

http://developer.android.com/reference/android/widget/TextView.html

XML Attribute / Equivalent Method	**Description**
android:text setText(CharSequence,TextView.BufferType)	Text to display.
android:textAllCaps setAllCaps(boolean)	Present the text in ALL CAPS.
android:textAppearance	Base text color, typeface, size, and style.
android:textColor setTextColor(int)	Text color.
android:textColorHighlight setHighlightColor(int)	Color of the text selection highlight.
android:textColorHint setHintTextColor(int)	Color of the hint text.
android:textColorLink setLinkTextColor(int)	Text color for links.
android:textIsSelectable isTextSelectable()	Indicates that the content of a non-editable text can be selected.

Basic Widgets / Attributes & Methods: TextView *cont.*

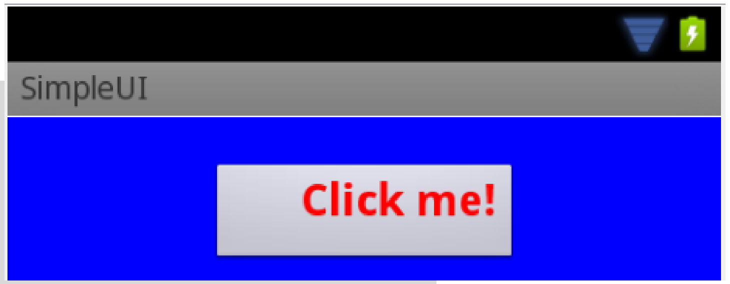
http://developer.android.com/reference/android/widget/TextView.html

XML Attribute / Equivalent Method	**Description**
android:textScaleX setTextScaleX(float)	Sets the horizontal scaling factor for the text.
android:textSize setTextSize(int.)	Size of the text.
android:textStyle setTypeface(Typeface)	Style (bold, italic, bolditalic) for the text.
android:typeface setTypeface(Typeface)	Typeface (normal, sans, serif, monospace) for the text.
android:width setWidth(int)	Makes the TextView be exactly this many pixels wide.

Basic Widgets: Buttons

- A **Button** widget allows the simulation of a clicking action on a GUI.

- **Button** is a subclass of **TextView**. Therefore formatting a button's face is similar to the setting of a **TextView**.



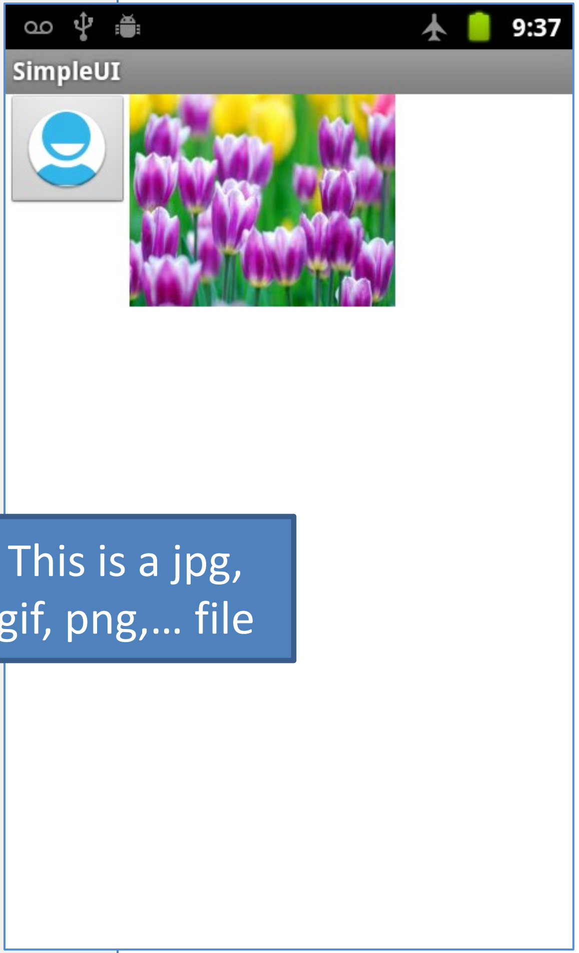
```
<Button
    android:id="@+id/button1"
    android:layout_width="140dp"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:gravity="right“
    android:layout_marginTop="20dp"
    android:padding="5dp"
    android:text="@string/button1_caption"
    android:textColor="#ffff0000"
    android:textSize="20sp"
    android:textStyle="bold" />
```

Basic Widgets: Images

- **ImageView** and **ImageButton** are two Android widgets that allow embedding of images in your applications.
- Both are *image-based widgets* analogue to *TextView* and *Button*, respectively.
- Each widget takes an **android:src** or **android:background** attribute (in an XML layout) to specify what picture to use.
- Pictures are usually a reference to a *drawable* resource.
- **ImageButton**, is a subclass of ImageView. It adds the standard *Button* behavior for responding to *click* events.

Basic Widgets: Images

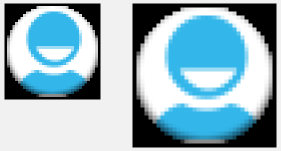
```
<LinearLayout
    . . .

    <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: Images

Icons are small images used to graphically represent your application and/or parts of it. They may appear in different places of the device including:



- Home screen
- Launcher window.
- Options menu
- Action Bar
- Status bar
- Multi-tab interface.
- Pop-up dialog boxes
- List view

Detailed information at:
http://developer.android.com/guide/practices/ui_guidelines/icon_design.html

HINT
Several websites allow you to convert your pictures into arbitrary image files under a variety of formats & sizes (.png, .jpg, .gif, etc). For instance try;
http://www.prodraw.net/favicon/index.php
http://converticon.com/

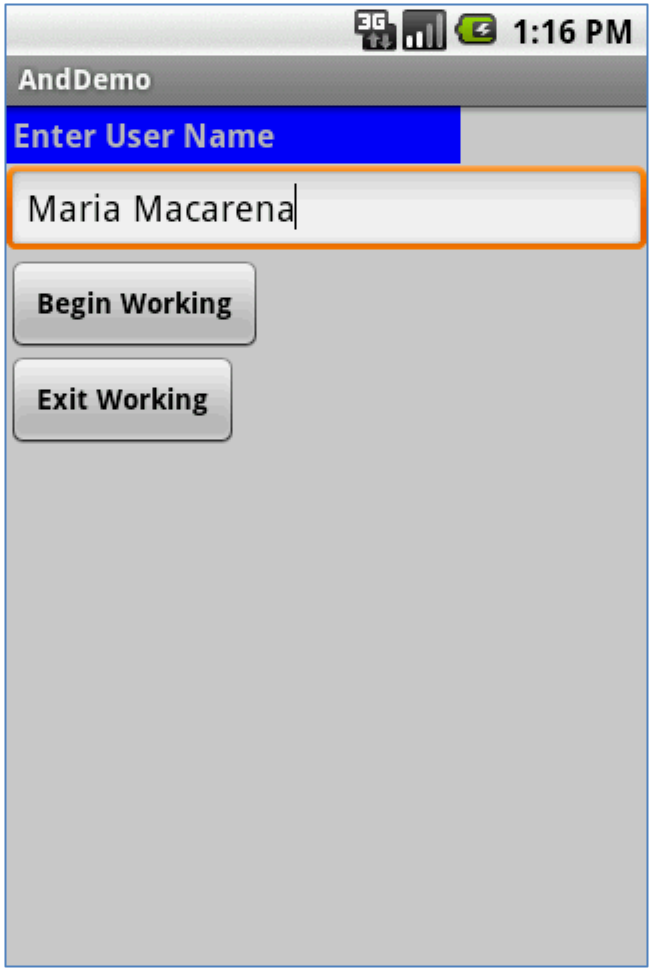
Basic Widgets: EditText

- The **EditText** (or *textBox*) widget is an extension of *TextView* that allows updates.

- The control configures itself to be *editable.*

- Important Java methods are:

  txtBox.setText("someValue")
  and
  txtBox.getText().toString()



Basic Widgets: EditText

In addition to the standard TextView's properties, EditText has many other (now) *deprecated* features such as:

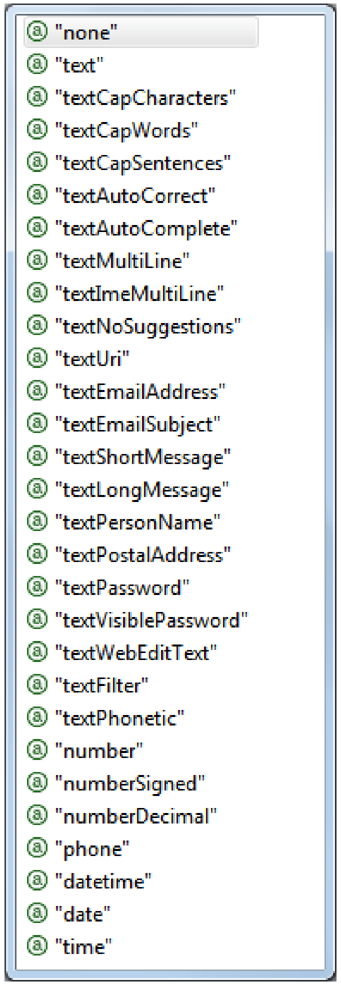
DEPRECATED

- **android:autoText**, (true/false) provides automatic spelling assistance
- **android:capitalize**, (*words/sentences*) automatic capitalization
- **android:digits**, to configure the field to accept only certain digits
- **android:singleLine**, is the field for single-line / multiple-line input
- **android:password**, (*true/false*) controls field's visibility
- **android:numeric**, (*integer, decimal, signed*) controls numeric format
- **android:phonenumber**, (true/false) Formatting phone numbers

Instead use the newer clause

NEW

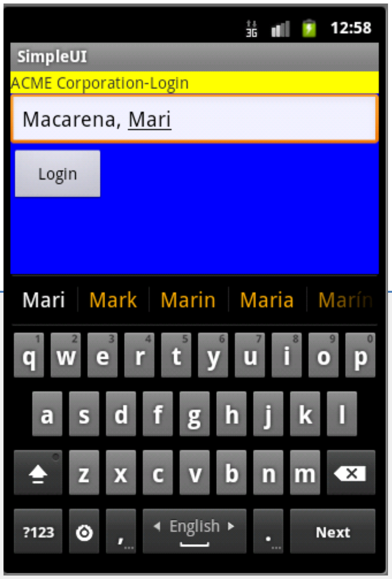
android:InputType="…choices…"
where choices include


"none"
"text"
"textCapCharacters"
"textCapWords"
"textCapSentences"
"textAutoCorrect"
"textAutoComplete"
"textMultiLine"
"textImeMultiLine"
"textNoSuggestions"
"textUri"
"textEmailAddress"
"textEmailSubject"
"textShortMessage"
"textLongMessage"
"textPersonName"
"textPostalAddress"
"textPassword"
"textVisiblePassword"
"textWebEditText"
"textFilter"
"textPhonetic"
"number"
"numberSigned"
"numberDecimal"
"phone"
"datetime"
"date"
"time"

Basic Widgets: EditViews

Example

```
...

<EditText
  android:id="@+id/txtUserName"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"

  android:inputType="textCapWords|textAutoCorrect"

  android:hint="Enter your First and Last Name"
  android:textSize="18sp" >
...
```



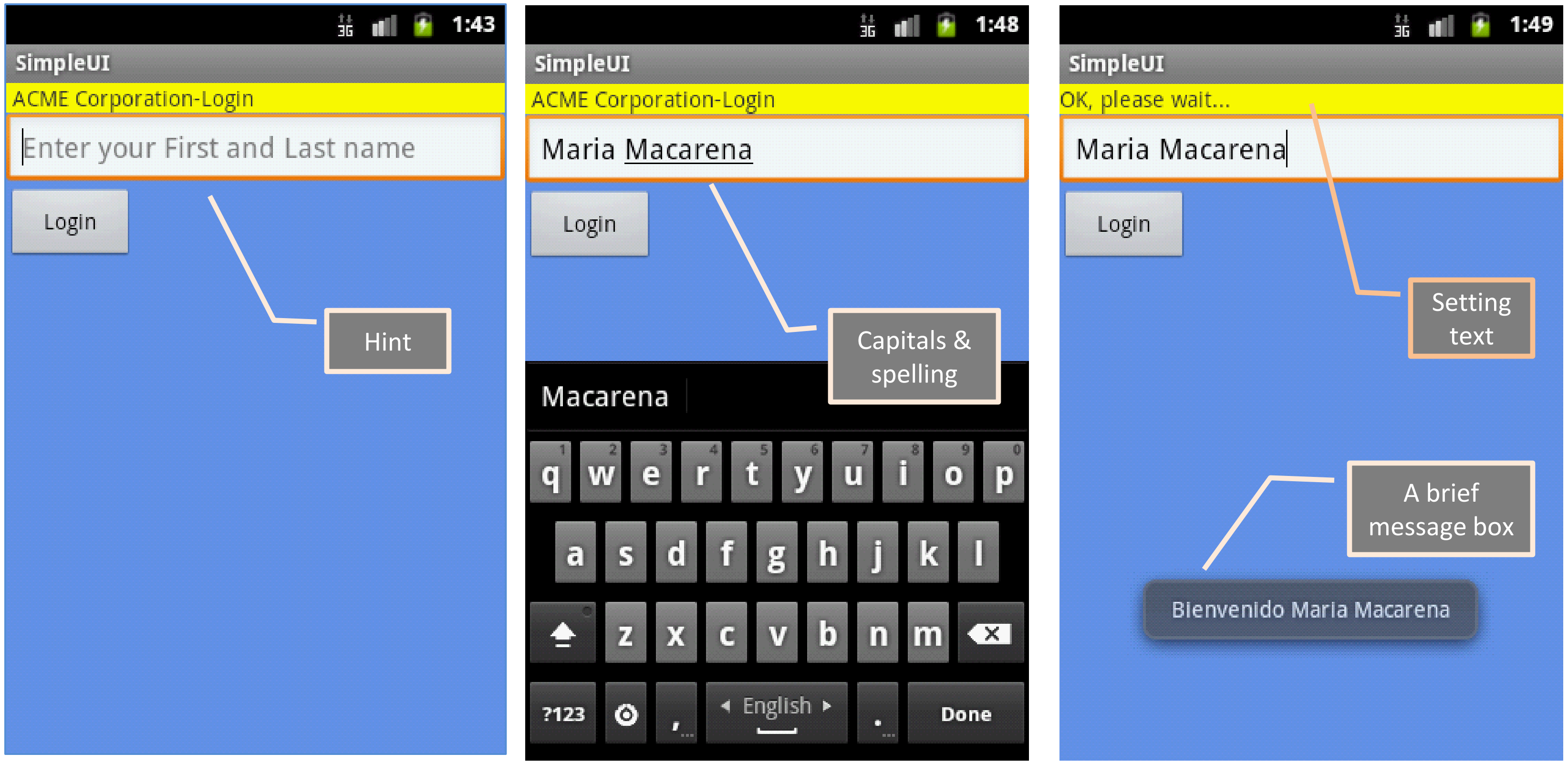
Enter "**teh**" It will be changed to: "**the**"

Each word is capitalized

Suggestion (grey out)

Basic Widgets: Example 1

In this little example we will create and use a simple login screen holding a label(**TexView**), a textBox (**EditText**), and a **Button**.



Bienvenido Maria Macarena

Basic Widgets: Example 1

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 android:background="#ff6495ed"
 android:orientation="vertical" >

 <TextView
     android:id="@+id/textView1"
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:background="#ffffff00"
     android:text=" ACME Corporation-Login "   />

 <EditText
     android:id="@+id/txtUserName"
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"

android:inputType="textCapWords|textAutoCorrect"

     android:hint="Enter your First and Last name"
     android:textSize="18sp" >

     <requestFocus />
 </EditText>

<Button
     android:id="@+id/button1"
     android:layout_width="82dp"
     android:layout_height="wrap_content"
     android:text="Login" />

</LinearLayout>
```

Basic Widgets: Example 1

Android’s Application (1 of 2)

```
package cis493.gui;
import ...

////////////////////////////////////////////////////////////////////////
// "LOGIN" - a gentle introduction to UI controls

public class AndDemo extends Activity {
    TextView labelUserName;
    EditText txtUserName;
    Button btnBegin;

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

        //binding the UI's controls defined in "main.xml" to Java code
        labelUserName = (TextView) findViewById(R.id.textView1);
        txtUserName = (EditText) findViewById(R.id.txtUserName);
        btnBegin = (Button) findViewById(R.id.button1);
```

Basic Widgets: Example 1

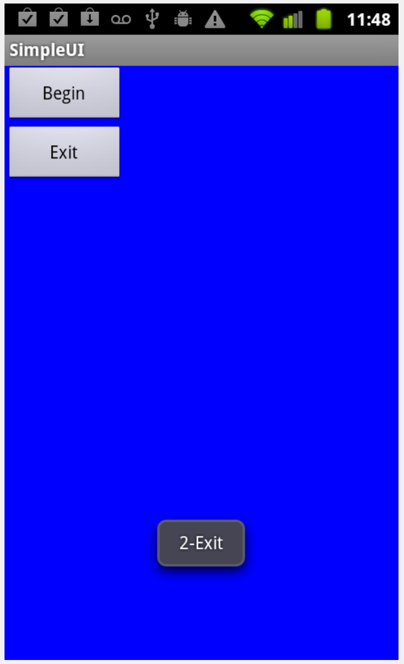
Android's Application (2 of 2)

```
        //LISTENER: wiring the button widget to events-&-code
        btnBegin.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            String userName = txtUserName.getText().toString();
            if (userName.compareTo("Maria Macarena")==0){
                labelUserName.setText("OK, please wait...");
                Toast.makeText(getApplicationContext(),
                        "Bienvenido " + userName,
                        Toast.LENGTH_SHORT).show();
            }
            Toast.makeText(getApplicationContext(),
                    "Bienvenido " + userName,
                    Toast.LENGTH_SHORT).show();
        }

         });// onClick

    }//onCreate

}//class
```

Basic Widgets: Example 2

Note: Another way of defining a Listener for multiple button widgets

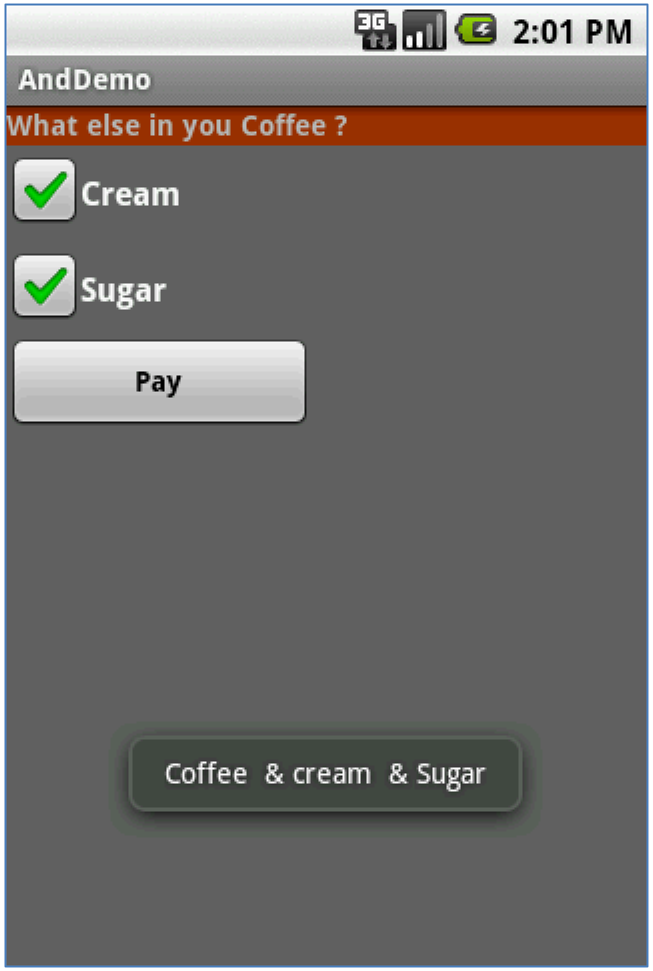
```
public class SimpleUI extends Activity implements OnClickListener {
    Button btnBegin;
    Button btnExit;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        btnBegin = (Button) findViewById(R.id.btnBegin);
        btnExit = (Button) findViewById(R.id.btnExit);

        btnBegin.setOnClickListener(this);
        btnExit.setOnClickListener(this);
    }//onCreate

   @Override
   public void onClick(View v) {
      if (v.getId()==btnBegin.getId() ){
      Toast.makeText(getApplicationContext(), "1-Begin", 1).show();
      }
      if (v.getId()==btnExit.getId() ){
      Toast.makeText(getApplicationContext(), "2-Exit", 1).show();
      }
   }//onClick
}//class
```



Basic Widgets: CheckBox

A checkbox is a specific type of two-states button that can be either *checked* or *unchecked*.

A example usage of a checkbox inside your activity would be the following:



Example 3: CheckBox

Complete code for the **checkBox** demo (1 of 3)

Layout: main.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
 android:id="@+id/linearLayout"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 android:background="#ff666666"
 android:orientation="vertical"
xmlns:android="http://schemas.android.com/a
pk/res/android"
>

<TextView
 android:id="@+id/labelCoffee"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:background="#ff993300"
 android:text="What else in you Coffee ?"
 android:textStyle="bold"
>
</TextView>
<CheckBox
 android:id="@+id/chkCream"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Cream"
 android:textStyle="bold"
>
</CheckBox>
<CheckBox
 android:id="@+id/chkSugar"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Sugar"
 android:textStyle="bold"
>
</CheckBox>
<Button
 android:id="@+id/btnPay"
 android:layout_width="153px"
 android:layout_height="wrap_content"
 android:text="Pay"
 android:textStyle="bold"
>
</Button>
</LinearLayout>
```

Example 2: CheckBox

Complete code for the checkBox demo (2 of 3)

```
public class MainActivity Activity {
    CheckBox chkCream;
    CheckBox chkSugar;
    Button btnPay;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //binding XMl controls with Java code
        chkCream = (CheckBox)findViewById(R.id.chkCream);
        chkSugar = (CheckBox)findViewById(R.id.chkSugar);
        btnPay = (Button) findViewById(R.id.btnPay);
```

Example 2: CheckBox

Complete code for the checkBox demo (3 of 3)

```
 //LISTENER: wiring button-events-&-code
        btnPay.setOnClickListener(new OnClickListener() {

        @Override
        public void onClick(View v) {
           String msg = "Coffee ";
           if (chkCream.isChecked()) {
              msg += " & cream ";
           }
           if (chkSugar.isChecked()){
              msg += " & Sugar";
           }
           Toast.makeText(getApplicationContext(),
                 msg, Toast.LENGTH_SHORT).show();
           //go now and compute cost...

        }//onClick

        });
    }//onCreate
}//class
```

Basic Widgets: RadioButtons

- A radio button is a two-states button that can be either *checked* or *unchecked*.
- When the radio button is unchecked, the user can press or click it to check it.
- Radio buttons are normally used together in a **RadioGroup**.

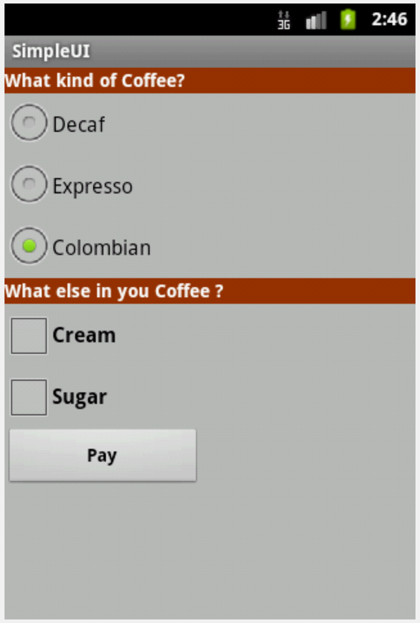
- When several radio buttons live inside a radio group, checking one radio button *unchecks* all the others.
- RadioButton inherits from … TextView. Hence, all the standard TextView properties for *font face, style, color,* etc. are available for controlling the look of radio buttons.
- Similarly, you can call ***isChecked()*** on a RadioButton to see if it is selected, **toggle()** to select it, and so on, like you can with a CheckBox.

Basic Widgets: RadioButtons

Example

We extend the previous example by adding a *RadioGroup* and three *RadioButtons*. Only new XML and Java code is shown:

```
<TextView
        android:id="@+id/textView1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#ff993300"
        android:text="What kind of Coffee?"
        android:textColor="#ffffff"
        android:textStyle="bold" />
```



```
<RadioGroup
        android:id="@+id/radioGroupCoffeeType"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <RadioButton
            android:id="@+id/radDecaf"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Decaf" />

        <RadioButton
            android:id="@+id/radExpresso"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Expresso" />


        <RadioButton
            android:id="@+id/radColombian"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:checked="true"
            android:text="Colombian" />

    </RadioGroup>
```

Basic Widgets: RadioButtons

```
public class MainActivity extends Activity {
   CheckBox chkCream;
   CheckBox chkSugar;
   Button btnPay;

   RadioGroup radCoffeeType;
   RadioButton radDecaf;
   RadioButton radExpresso;
   RadioButton radColombian;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      chkCream = (CheckBox) findViewById(R.id.chkCream);
      chkSugar = (CheckBox) findViewById(R.id.chkSugar);
      btnPay = (Button) findViewById(R.id.btnPay);

      radCoffeeType = (RadioGroup) findViewById(R.id.radioGroupCoffeeType);

      radDecaf = (RadioButton) findViewById(R.id.radDecaf);
      radExpresso = (RadioButton) findViewById(R.id.radExpresso);
      radColombian = (RadioButton) findViewById(R.id.radColombian);
```

Basic Widgets: RadioButtons

```
        // LISTENER: wiring button-events-&-code
        btnPay.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                String msg = "Coffee ";
                if (chkCream.isChecked())
                    msg += " & cream ";
                if (chkSugar.isChecked())
                    msg += " & Sugar";

                // get radio buttons ID number
                int radioId = radCoffeeType.getCheckedRadioButtonId();

                // compare selected's Id with individual RadioButtons ID
                if (radColombian.getId() == radioId)
                    msg = "Colombian " + msg;
                // similarly you may use .isChecked() on each RadioButton
                if (radExpresso.isChecked())
                    msg = "Expresso " + msg;
                // similarly you may use .isChecked() on each RadioButton
                if (radDecaf.isChecked())
                    msg = "Decaf " + msg;

                Toast.makeText(getApplicationContext(), msg, 1).show();
                // go now and compute cost...
            }// onClick
        });
    }// onCreate

}// class
```

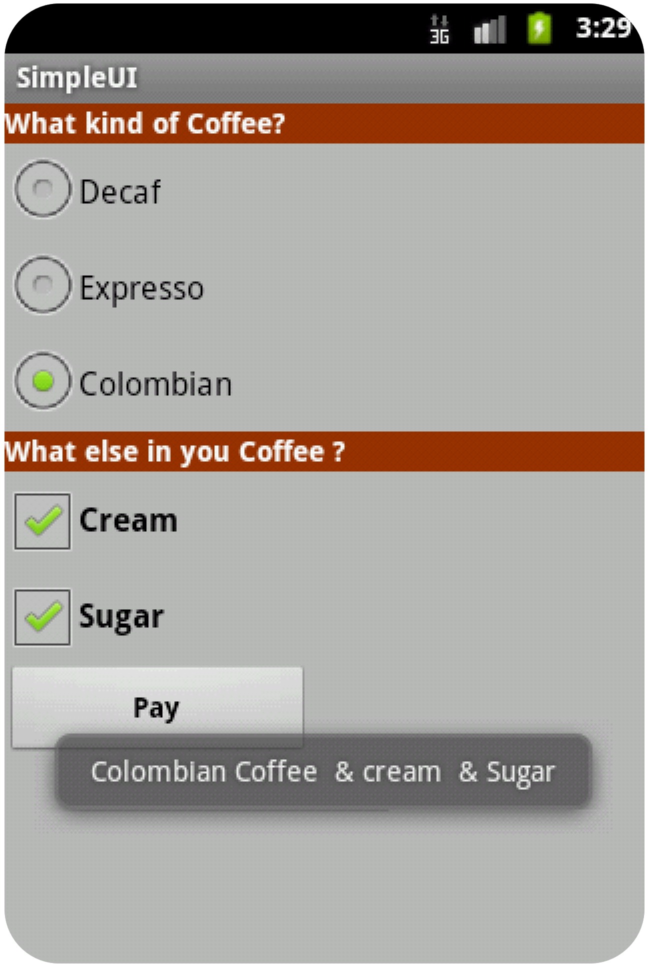
Basic Widgets: RadioButtons

Example

This UI uses *RadioButtons* and *CheckBoxes* to define choices

RadioGroup

Summary of choices



UI – Other Features

XML Controls the focus sequence:

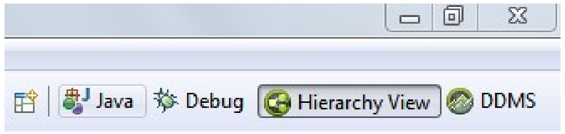
android:visibility
qndroid:background
<requestFocus />

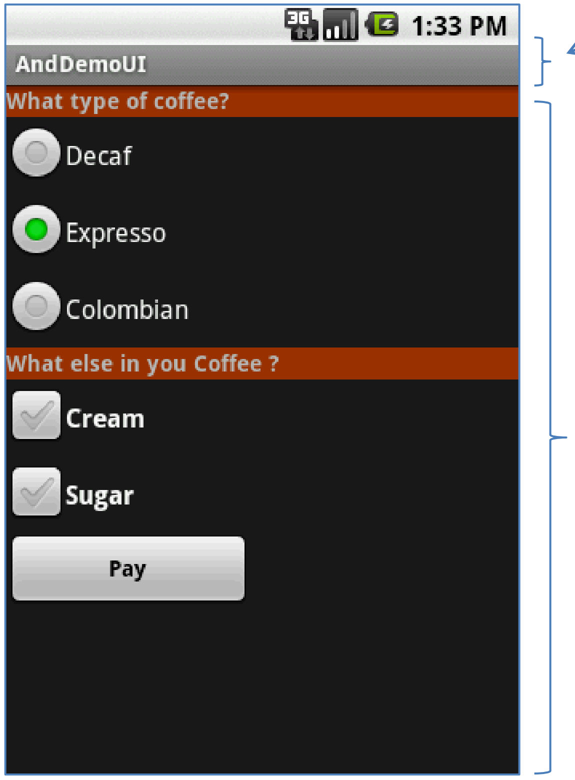
Java methods

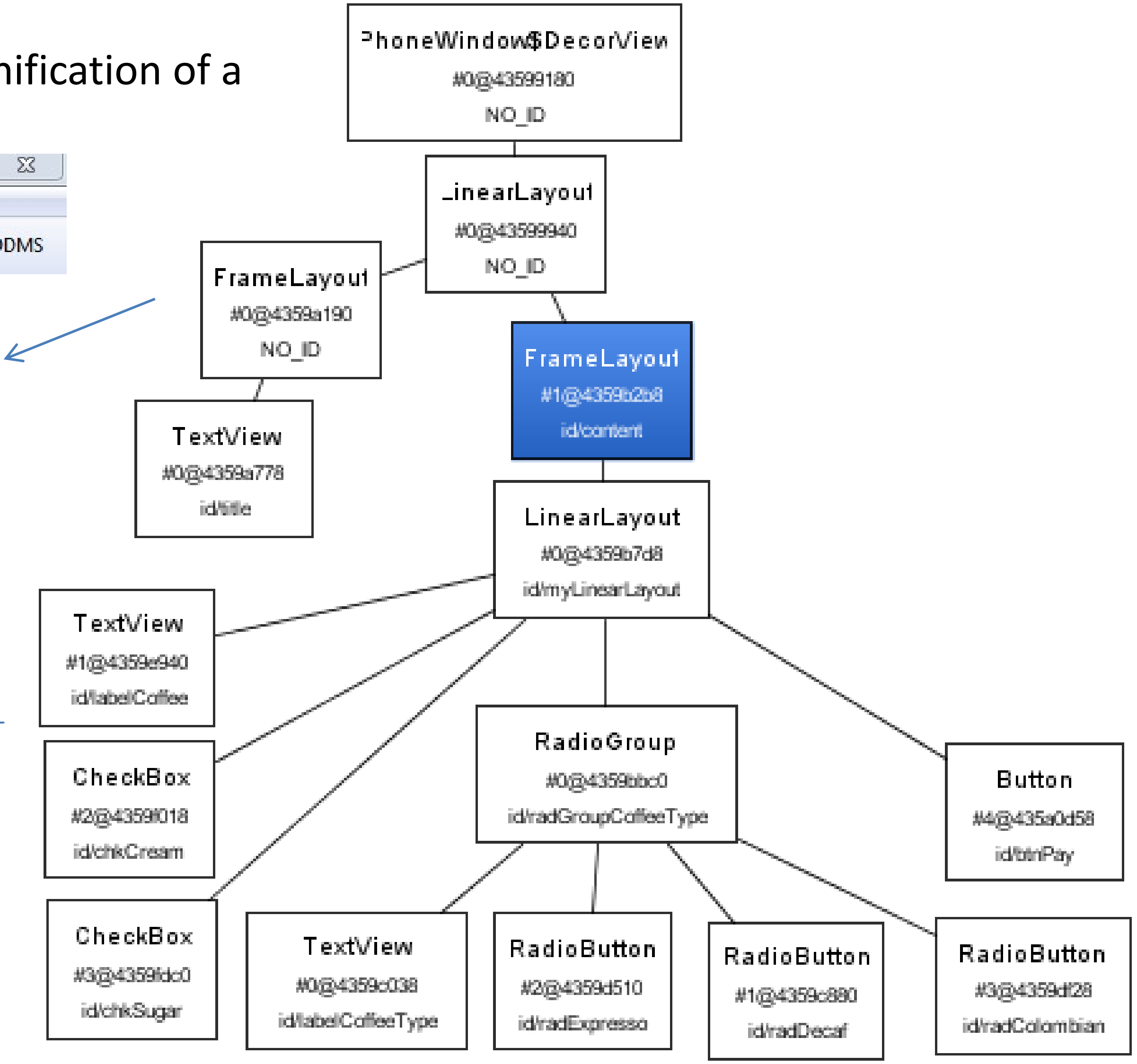
myButton.requestFocus()
myTextBox.isFocused()
myWidget.setEnabled()
myWidget.isEnabled()

Hierarchy Viewer Tools

Can be added to the Eclipse IDE as a Perspective option.
Allows exploration/magnification of a displayed UI






PhoneWindow$DecorView #0@43599180 NO_ID
LinearLayout #0@43599940 NO_ID
FrameLayout #0@4359a190 NO_ID
TextView #0@4359a778 id/title
FrameLayout #1@4359b2b8 id/content
LinearLayout #0@4359b7d8 id/myLinearLayout
TextView #1@4359e940 id/labelCoffee
CheckBox #2@4359f018 id/chkCream
CheckBox #3@4359fdc0 id/chkSugar
RadioGroup #0@4359bbc0 id/radGroupCoffeeType
Button #4@435a0d58 id/btnPay
TextView #0@4359c038 id/labelCoffeeType
RadioButton #2@4359d510 id/radExpresso
RadioButton #1@4359c880 id/radDecaf
RadioButton #3@4359df28 id/radColombian