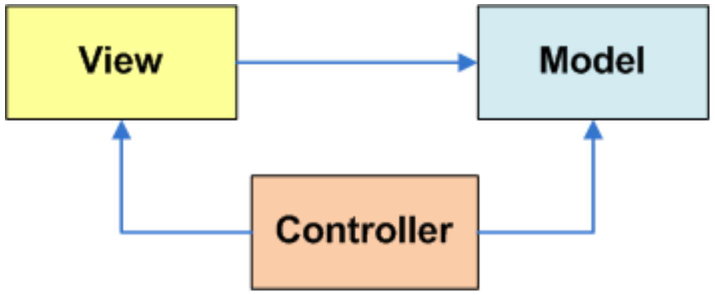
CMSC 491/628: Introduction to Mobile Computing
UI interface design

Nilanjan Banerjee

University of Maryland
Baltimore County
nilanb@umbc.edu
http://www.csee.umbc.edu/~nilanb/

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

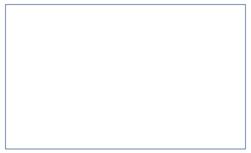
- **In a manner similar to a web page interaction**, when the Android user touches the screen, the controller interprets the input and determines what specific portion of the screen and gestures were involved. Basedon this information it tells the model about the interaction in such a way that the appropriate "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

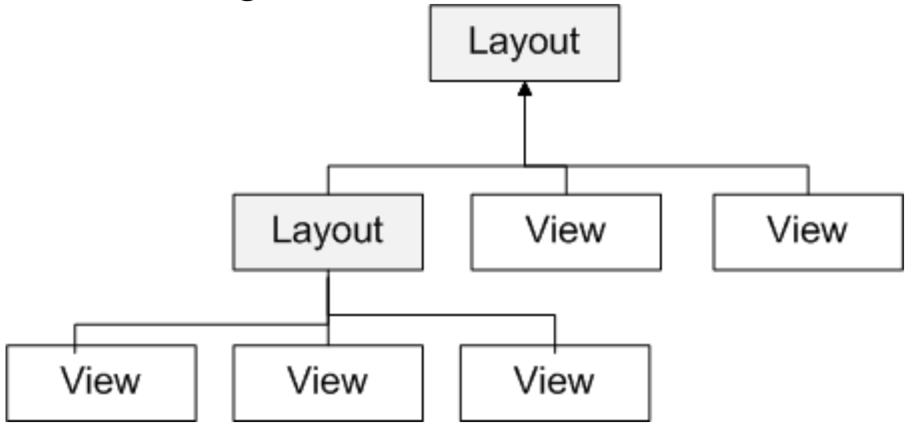
- The **View class** is the Android's most basic component from which users interfaces can be created. This element is similar to the Swing **JComponent** class for Java apps.

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

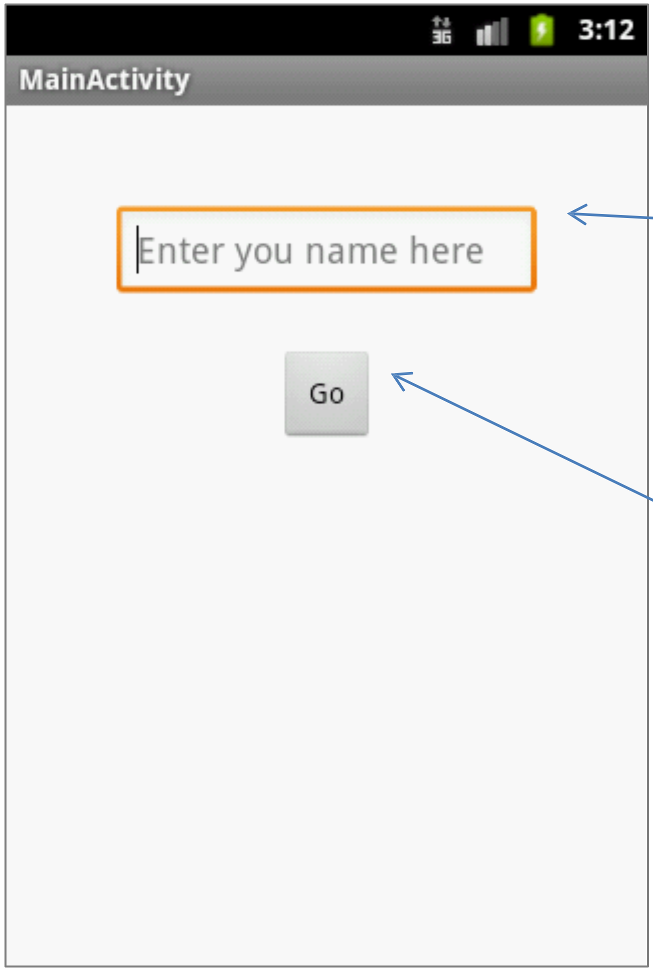


- **Widgets** are subclasses of View. They are used to create interactive UI components such as buttons, checkboxes, labels, text fields, etc.

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



Graphical UI ↔ XML Layout



```
<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>
```

Actual UI displayed by the app

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*.

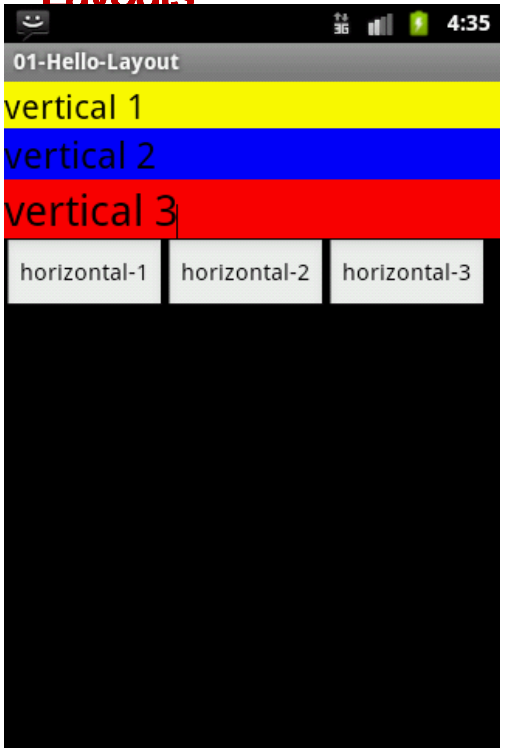
2. **Set up listeners:** For example, an image could be programmed to respond to various events such as: click, long-tap, mouse-over, etc.

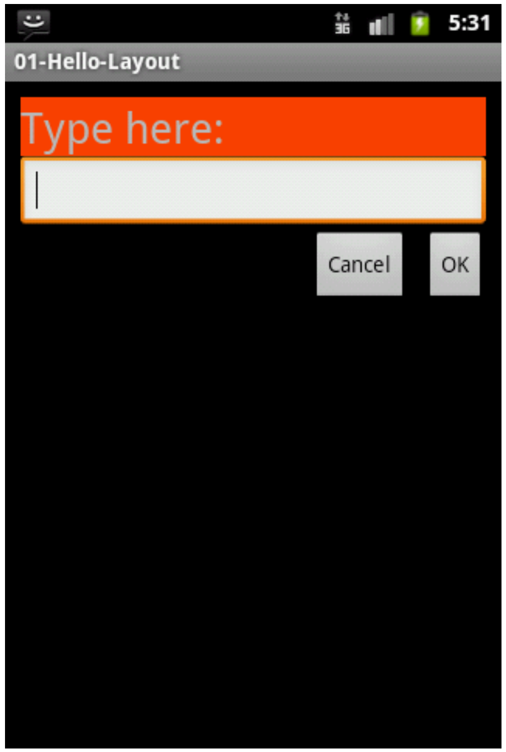
3. **Set focus:** To set focus on a specific view, you call the method requestFocus() or use XML tag <requestFocus />

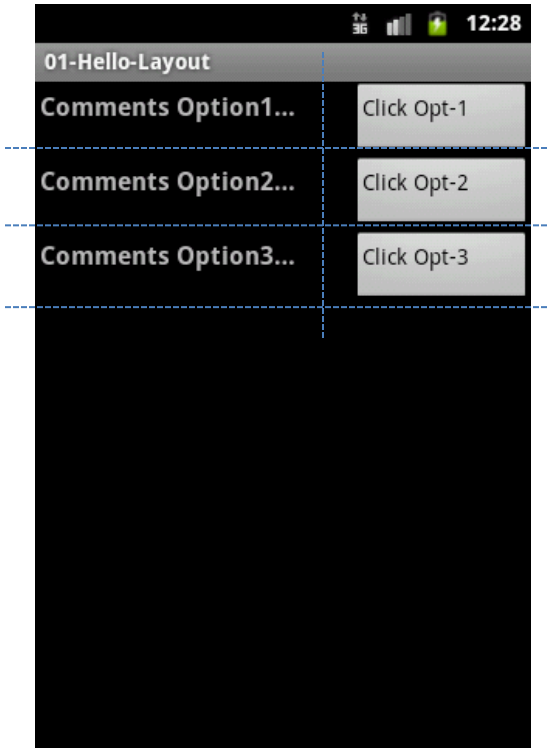
4. **Set visibility:** You can hide or show views using setVisibility(…).

A brief sample of UI components

Layouts







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

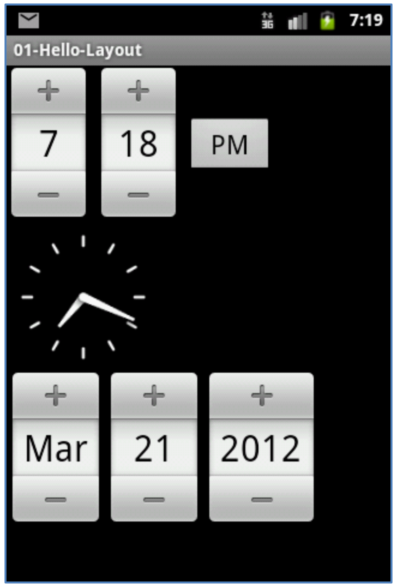
Relative Layout
A RelativeLayout is a ViewGroup that allows you to position elements relative to eachother.

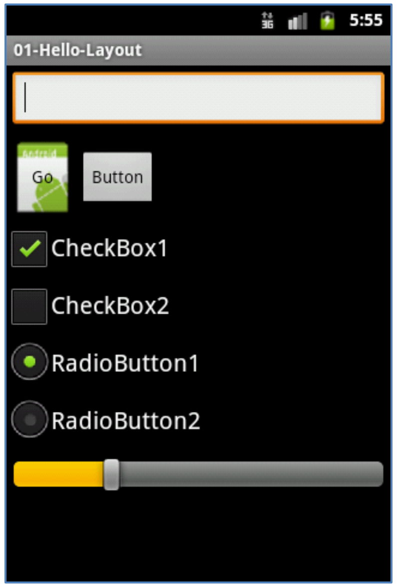
Table Layout
A TableLayout is a ViewGroup that places elements using a row & column disposition.

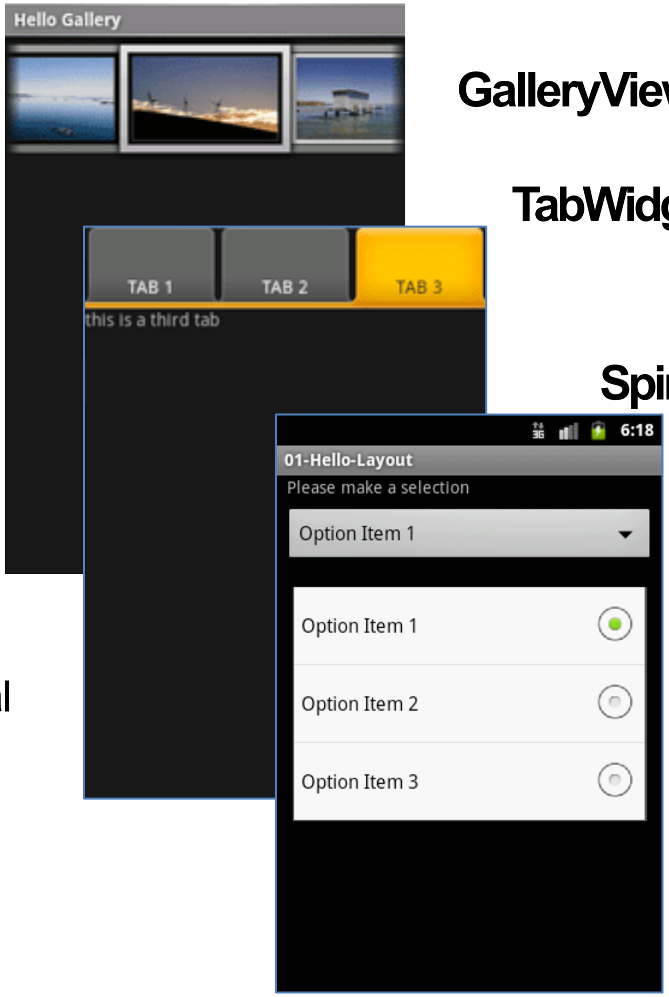
Reference: http://developer.android.com/guide/topics/ui/layout-objects.html

A brief sample of UI components

Widgets







GalleryView

TabWidget

Spinner

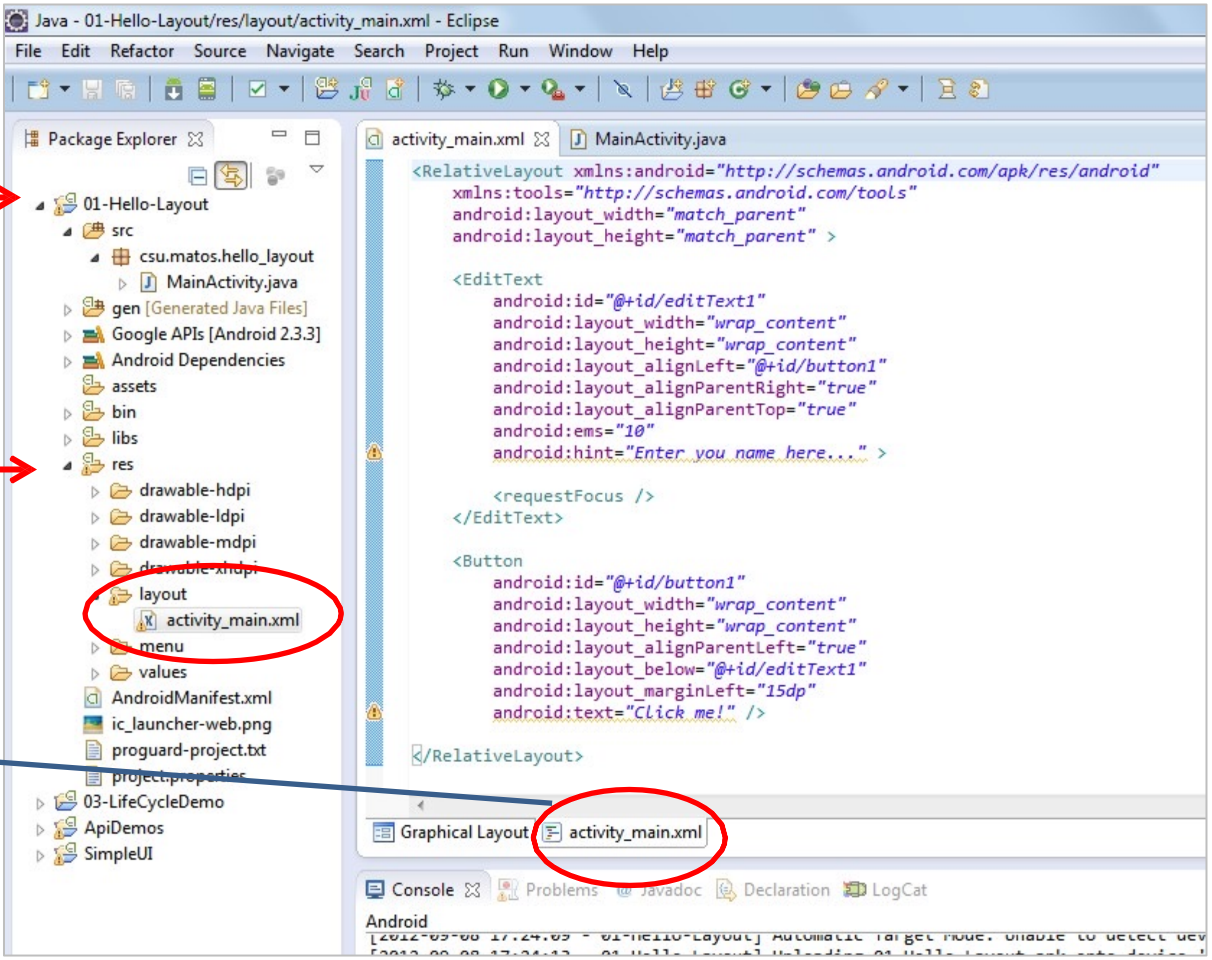
TimePicker
AnalogClock
DatePicker
A *DatePicke* is a widget that allows the user to select a month, day and year.

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

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.



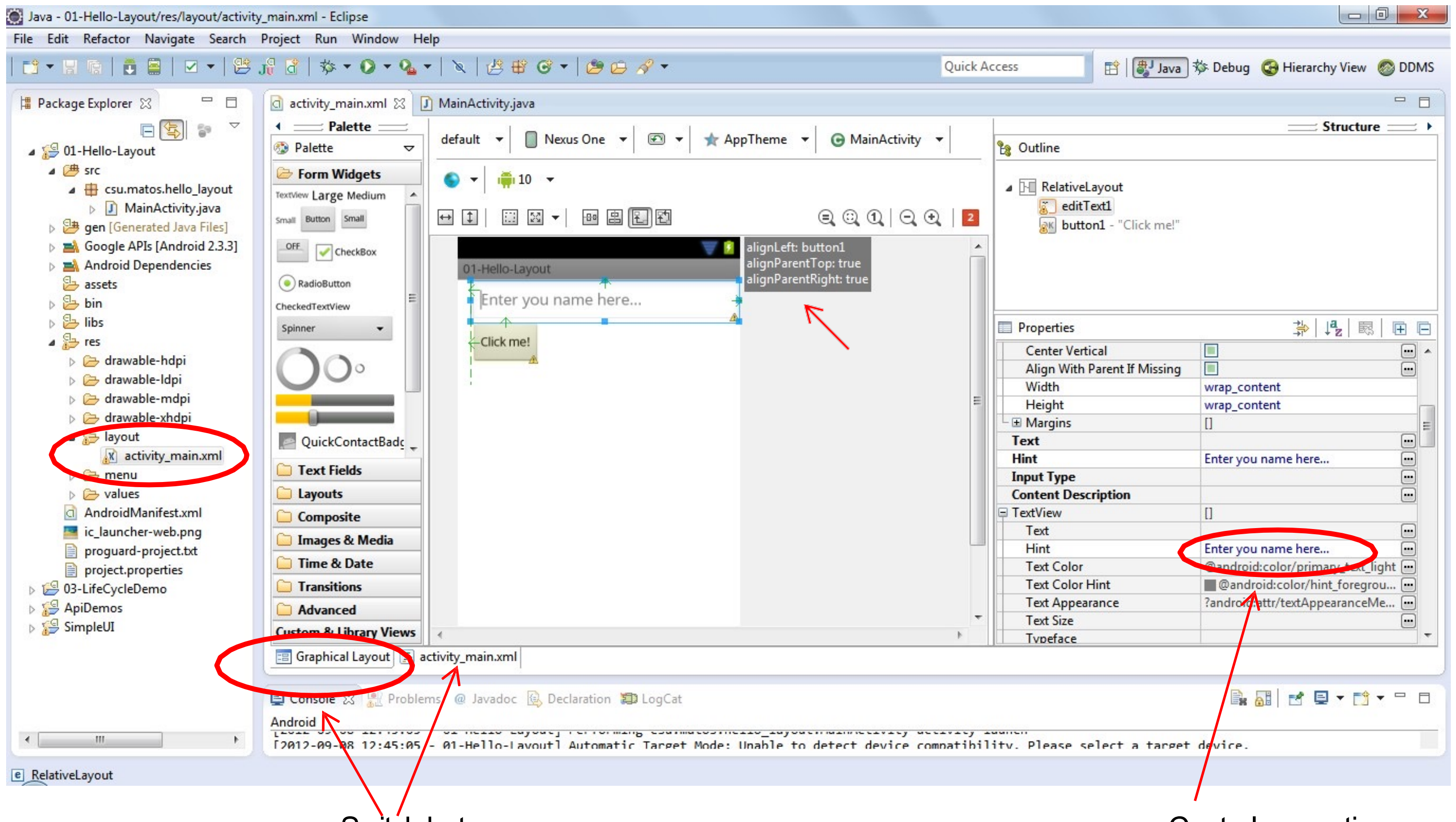
```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <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>
```

```
[2012-09-08 17:24:09 - 01-Hello-Layout] Automatic Target Mode: Unable to detect dev
[2012-09-08 17:24:13 - 01-Hello-Layout] Uploading 01-Hello-Layout.apk onto device '
```

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



Center Vertical	
Align With Parent If Missing	
Width	wrap_content
Height	wrap_content
Margins	[]
Text	
Hint	Enter you name here...
Input Type	
Content Description	
TextView	[]
Text	
Hint	Enter you name here...
Text Color	@android:color/primary_text_light
Text Color Hint	@android:color/hint_foregrou...
Text Appearance	?android:attr/textAppearanceMe...
Text Size	
Typeface	

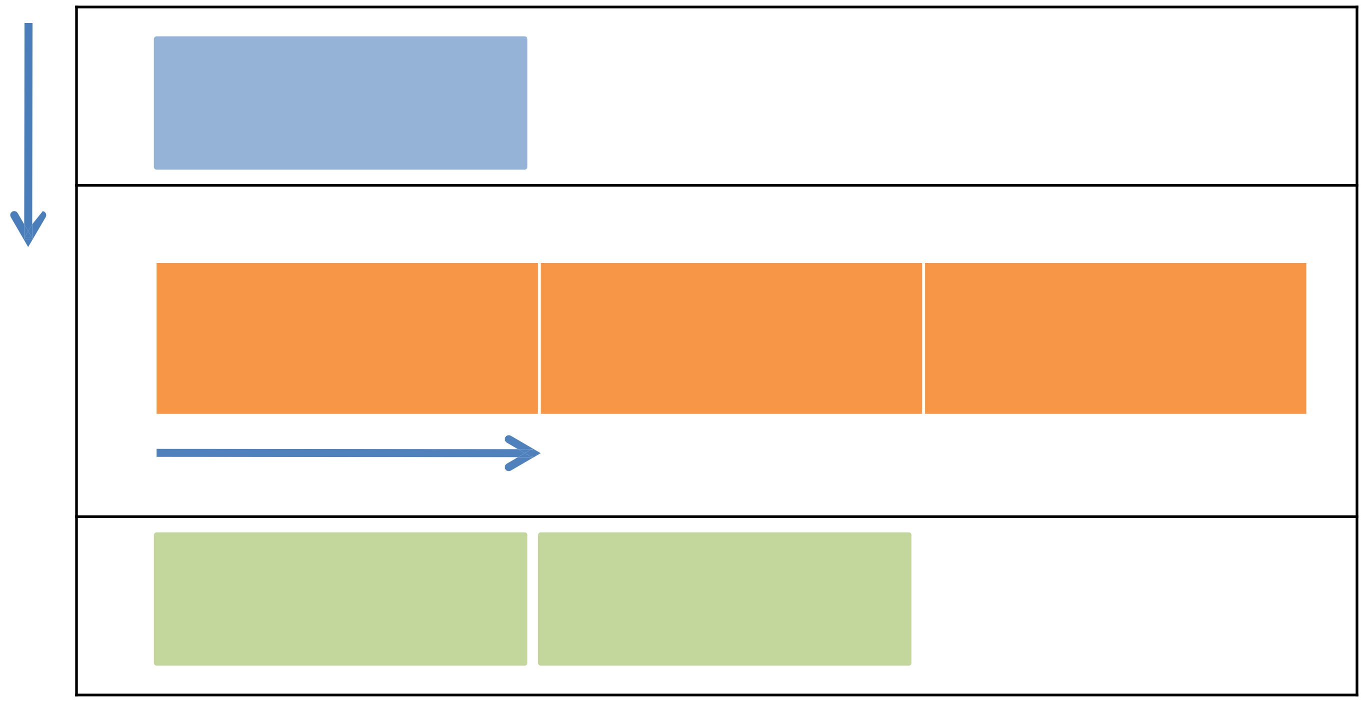
```
Android
[2012-09-08 12:45:05 - 01-Hello-Layout] Automatic Target Mode: Unable to detect device compatibility. Please select a target device.
```

Switch between
UI and XML view

Control properties

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.

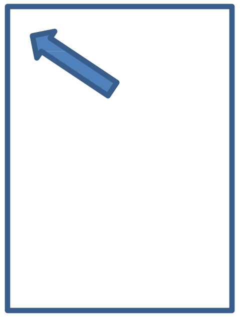


Common Layouts

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

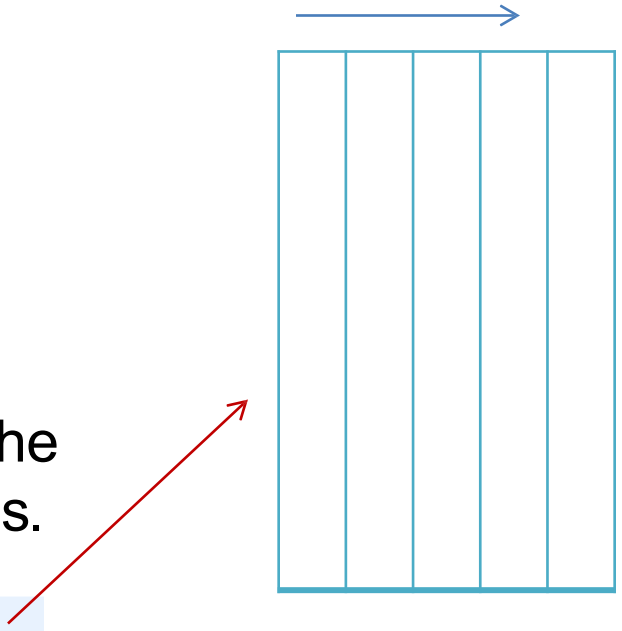
The LinearLayout

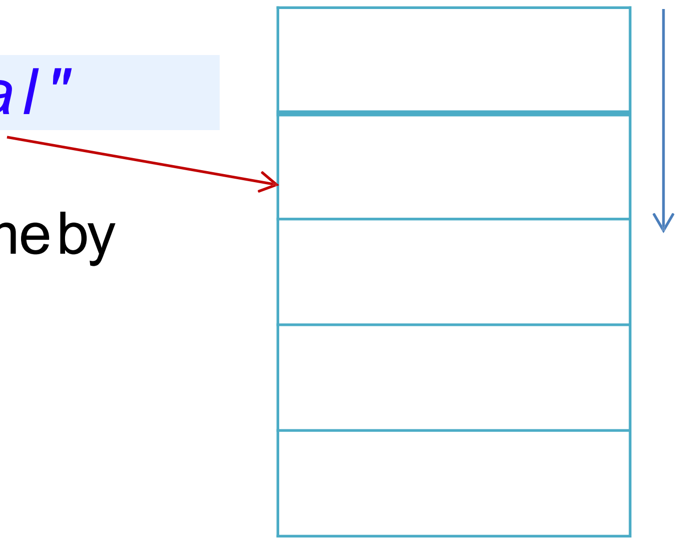
1. **Linear Layout**

Orientation

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

```
android:orientation="horizontal"

android:orientation="vertical"
```

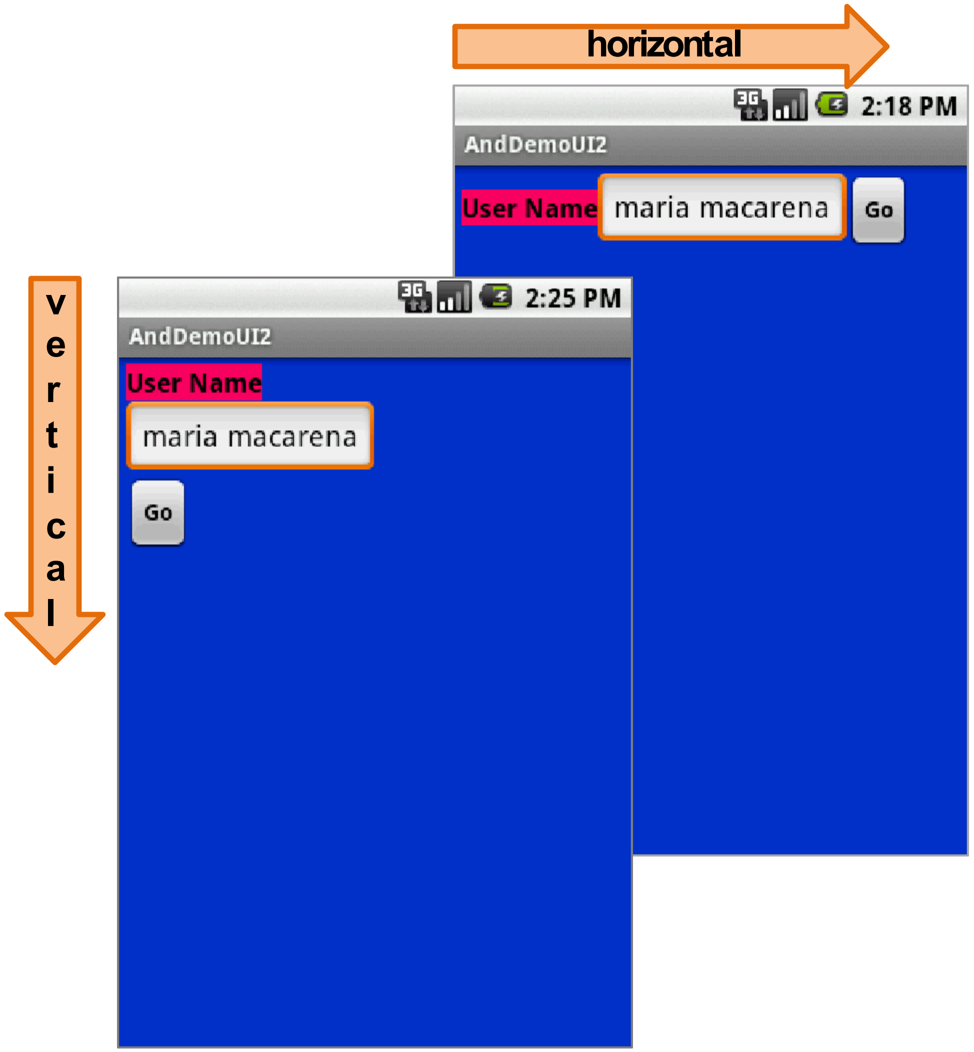




The orientation can be modified at runtime by invoking *setOrientation()*

The LinearLayout - Orientation

1.1 Linear Layout: Orientation

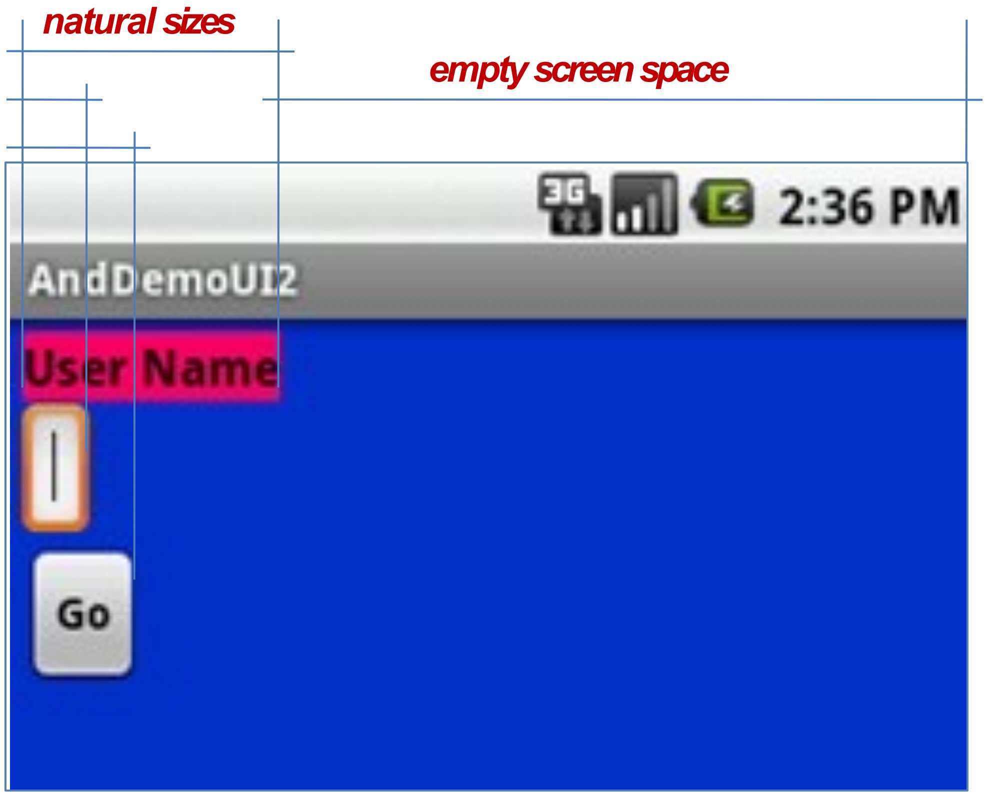


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
        android:id="@+id/myLinearLayout"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:background="#ff0033cc"
        android:padding="4dip"
        xmlns:android="http://schemas.android.com/apk/res/android"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

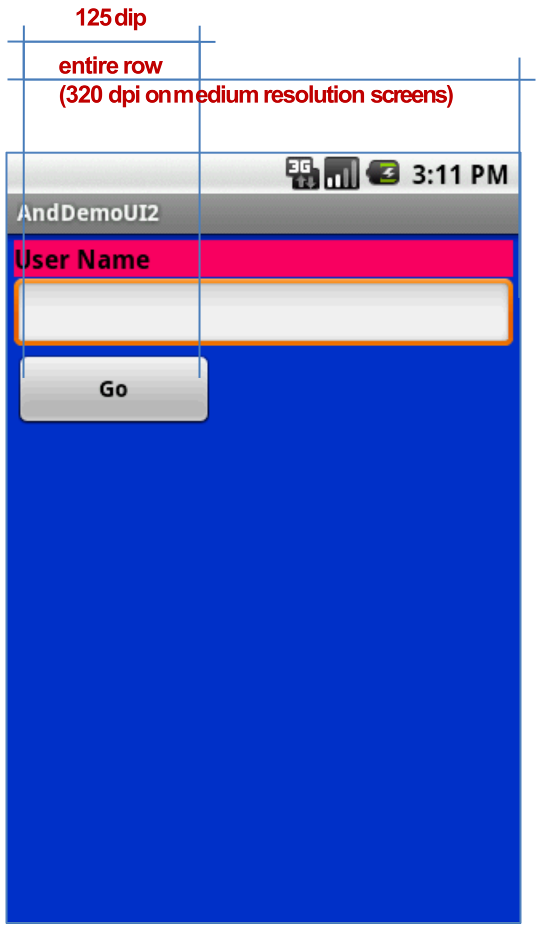
```
android:layout_width
android:layout_height
```

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

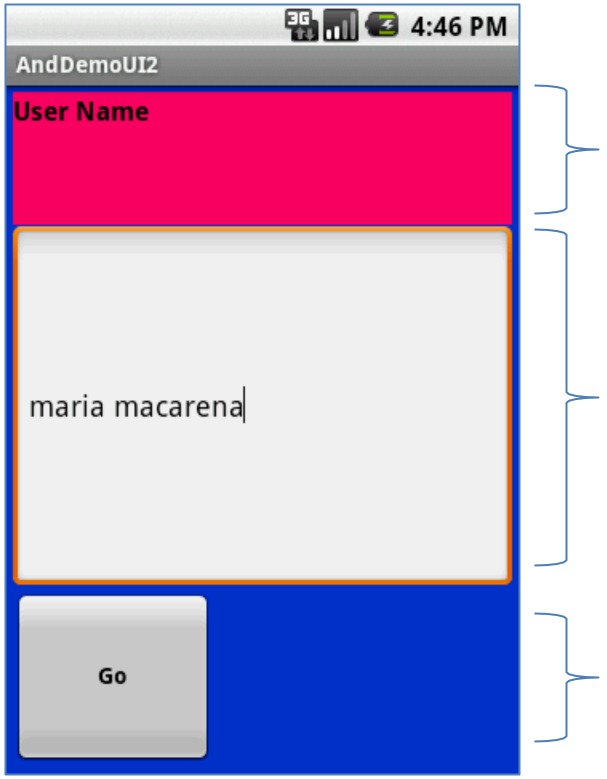
1.2 Linear Layout: Weight
Indicates how much of the extra space in the LinearLayout will be allocated to the view. Use **0** if the view should not be stretched. The bigger the weight the larger the extra space given to that widget.

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

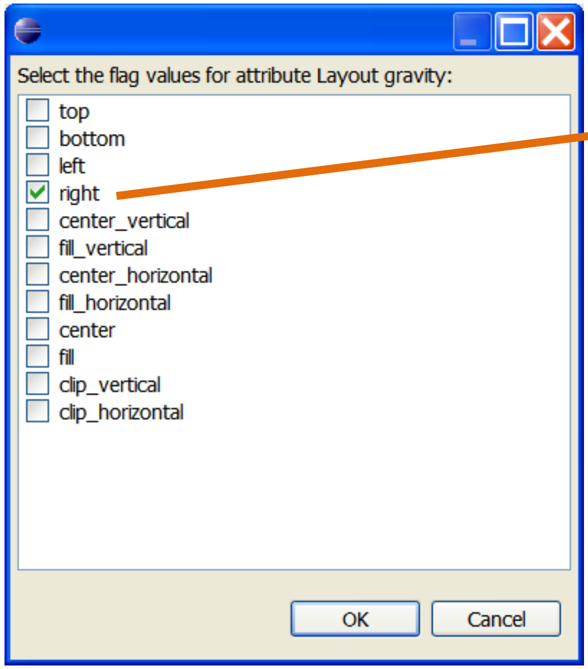


Takes: 2 /(1+1+2)
of the screen space

The LinearLayout – Gravity

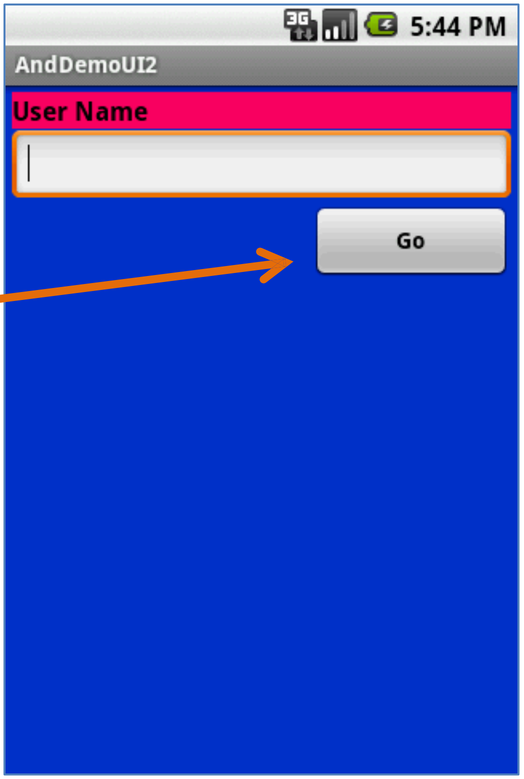
1.3 Layout_Gravity

- 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

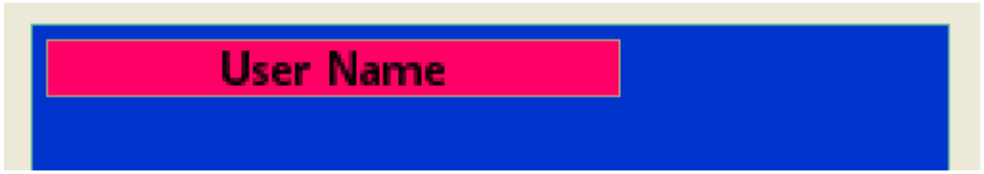
The LinearLayout – Gravity

1.3 CAUTION: gravity vs. layout_gravity

The difference between:

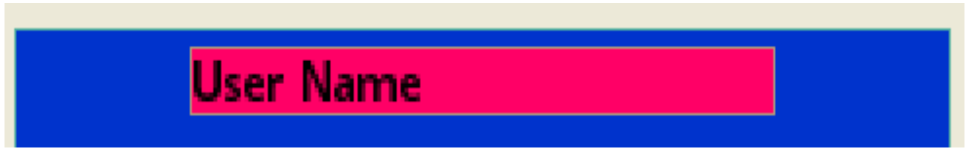
android:gravity
indicates how to place an object within a container. In the example the text is centered

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



android:layout_gravity
positions the view with respect to its

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



The LinearLayout – Padding

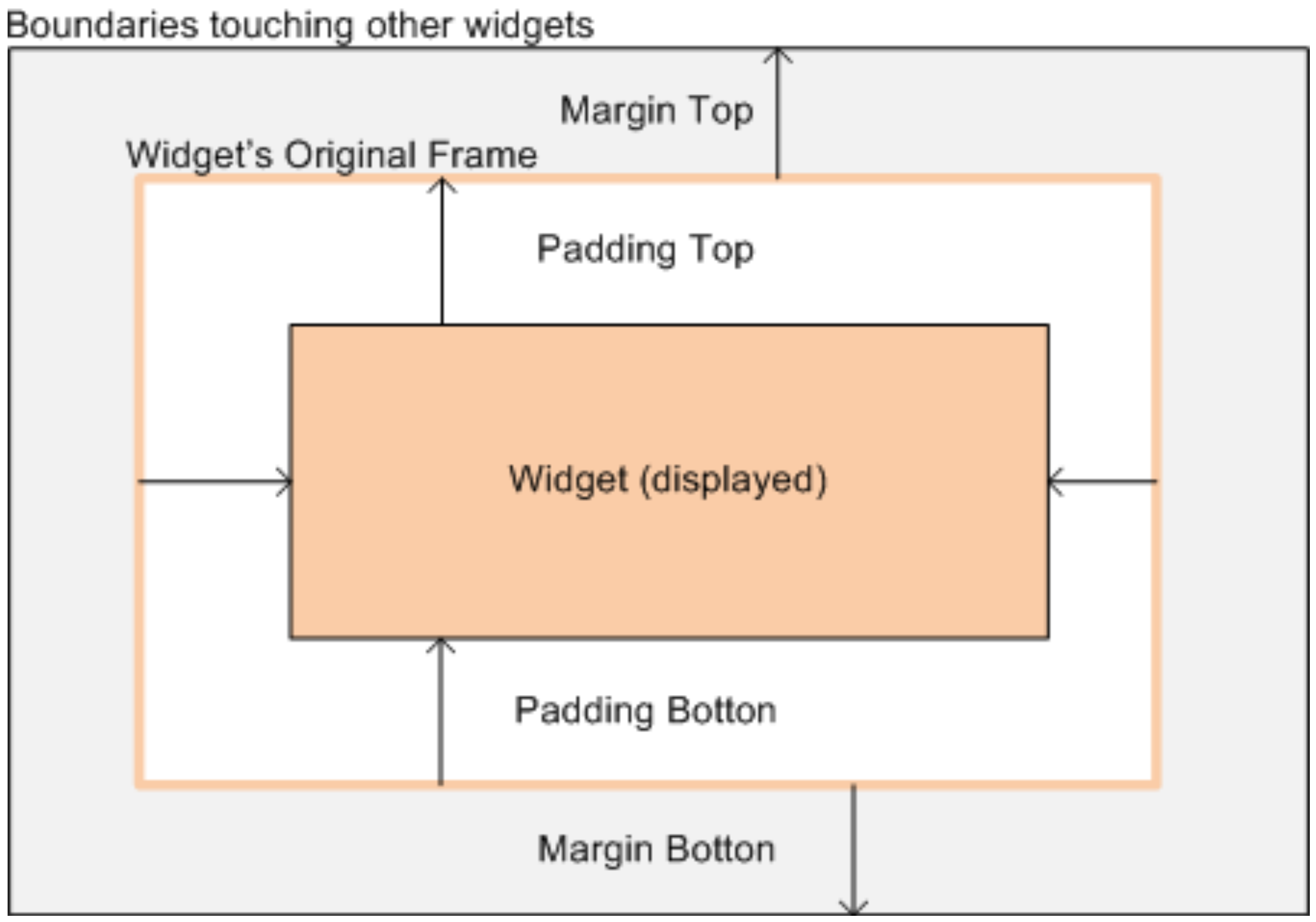
1.4 Linear Layout: Padding

- 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

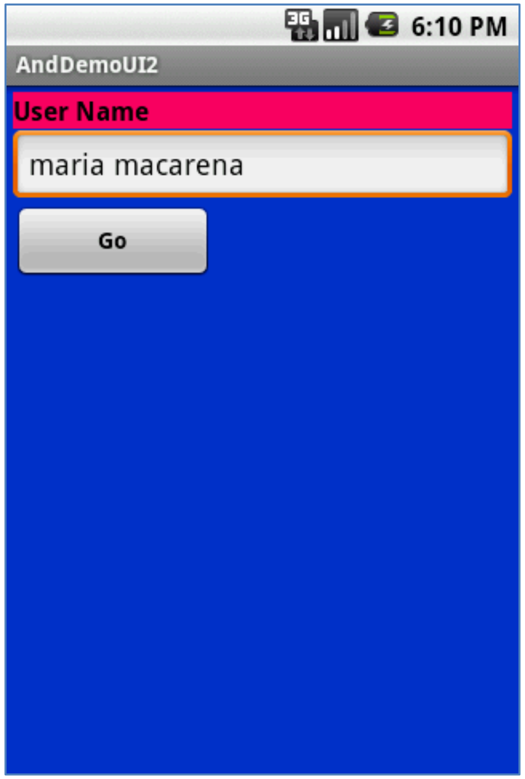


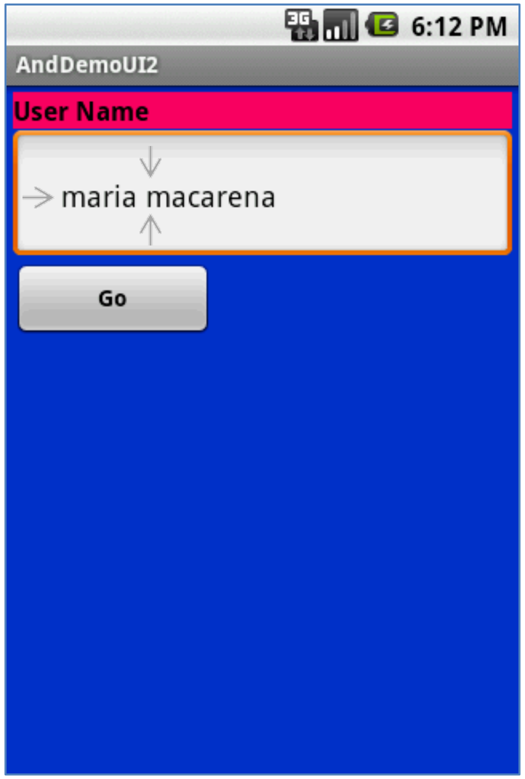
The LinearLayout – Padding

1.3 Linear Layout: Internal Margins Using Padding

Example:

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



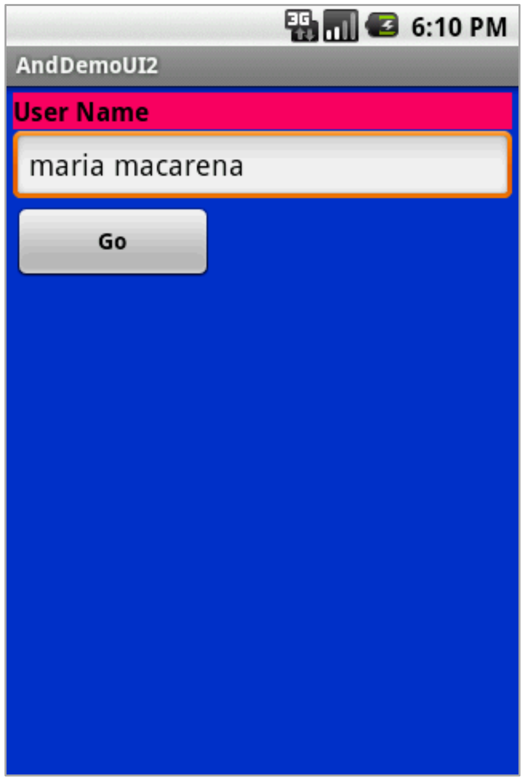


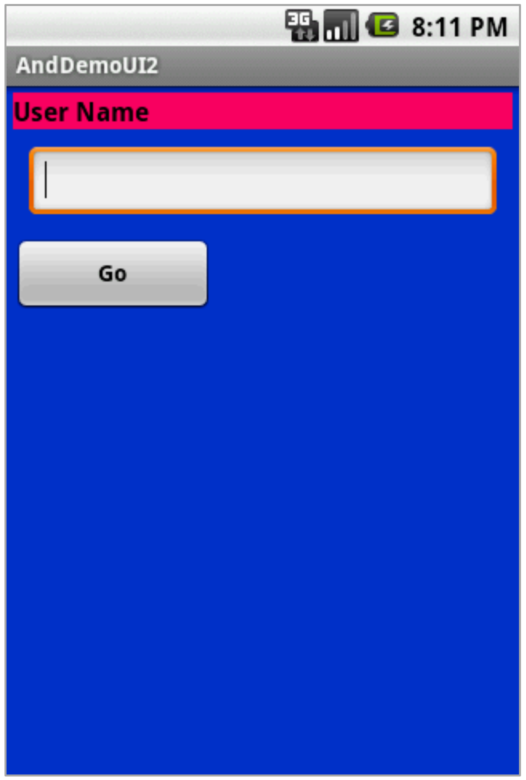
```
<EditText
android:id="@+id/ediName"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textSize="18sp"
android:padding="30dip"

>
</EditText>
...
```

The LinearLayout – Margin

1.4 Linear Layout: (External) Margin

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





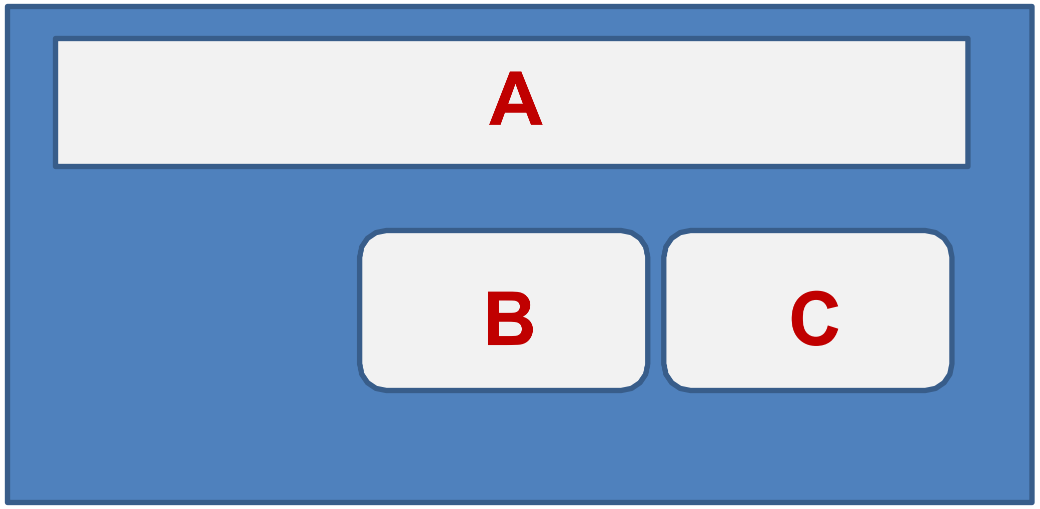
Increased inter-widget space

```
<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.

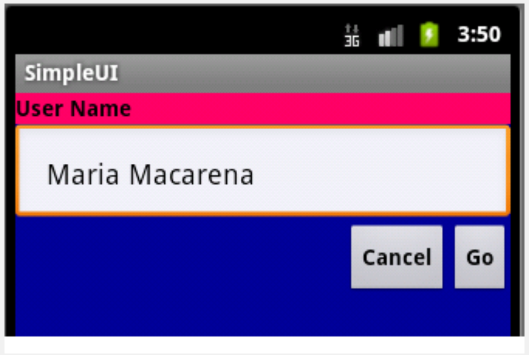
2. XML elements are named using: `@+id/...` For instance an EditText box could be called: `android:id="@+id/txtUserName"`

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

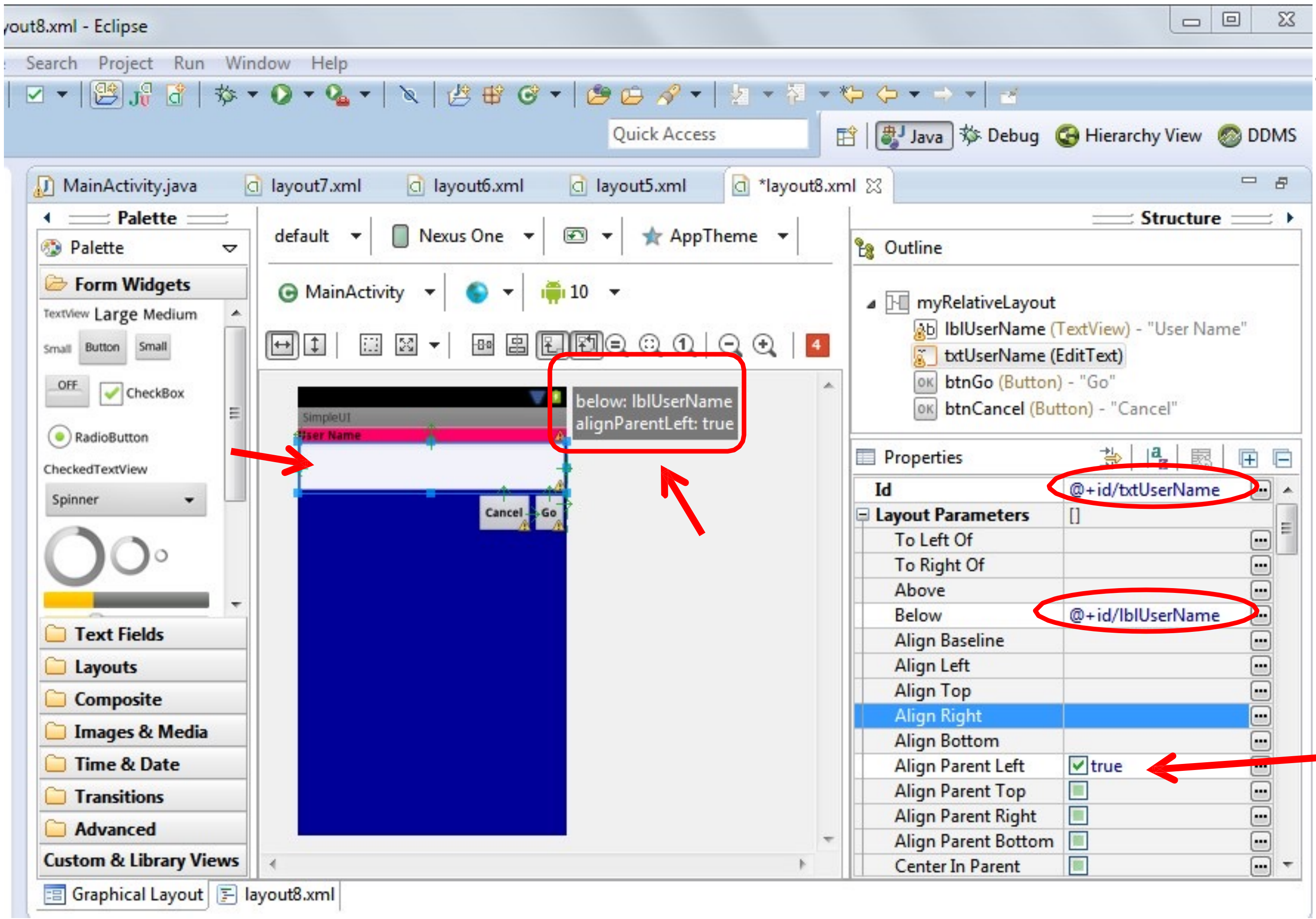
2. Relative Layout – Example

```
<?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)



yout8.xml - Eclipse
Search Project Run Window Help
Quick Access
Java Debug Hierarchy View DDMS
MainActivity.java layout7.xml layout6.xml layout5.xml *layout8.xml
Palette
Form Widgets
Text Fields
Layouts
Composite
Images & Media
Time & Date
Transitions
Advanced
Custom & Library Views
default Nexus One AppTheme
MainActivity 10
Structure
Graphical Layout layout8.xml

below: lblUserName
alignParentLeft: true

Outline

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

Properties

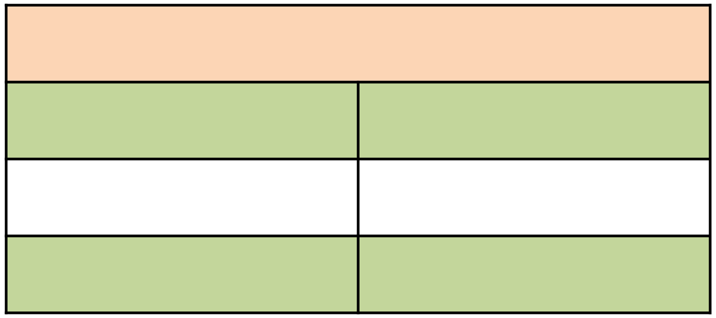
Id	@+id/txtUserName
Layout Parameters	[]
To Left Of	
To Right Of	
Above	
Below	@+id/lblUserName
Align Baseline	
Align Left	
Align Top	
Align Right	
Align Bottom	
Align Parent Left	✓ true
Align Parent Top	☐
Align Parent Right	☐
Align Parent Bottom	☐
Center In Parent	☐

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**)

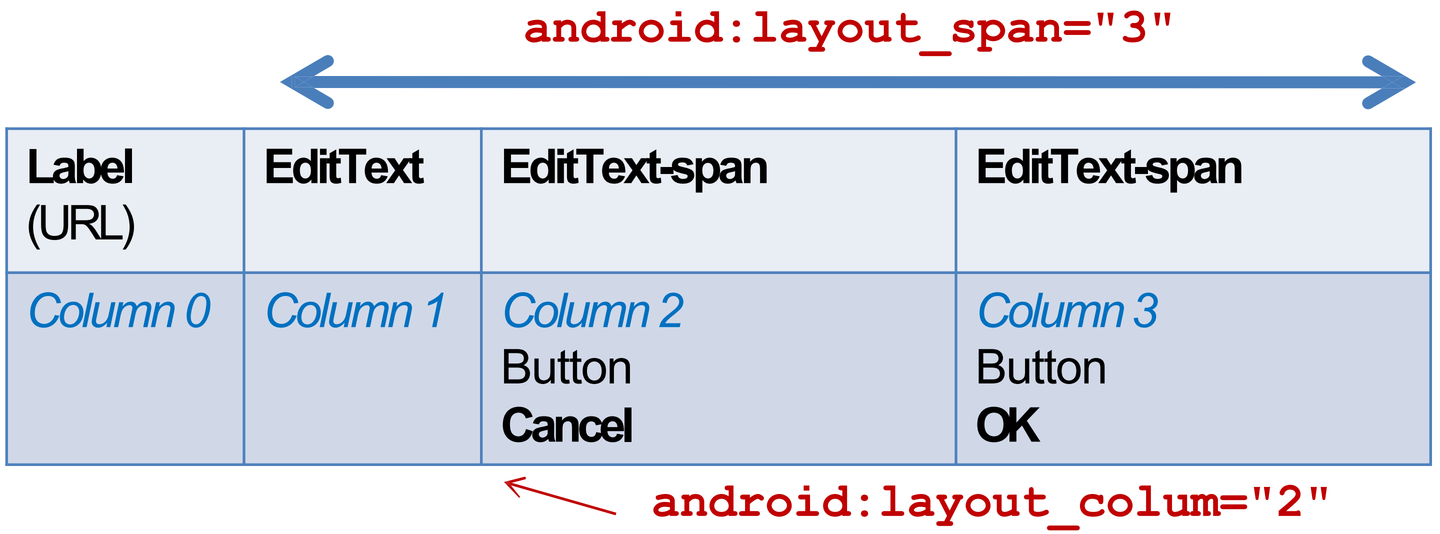
```
<TableRow>
   <TextView android:text="URL:" />
   <EditText
   android:id="@+id/entry"
   android:layout_span="3" />
</TableRow>
```

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"

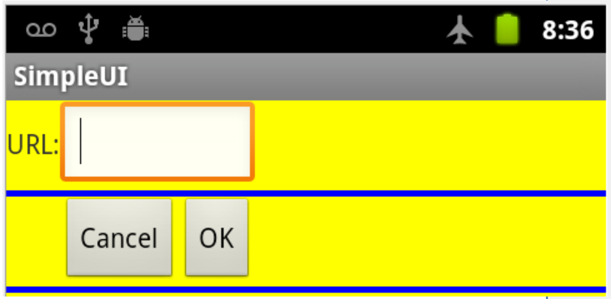
Basic XML Layouts - Containers

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



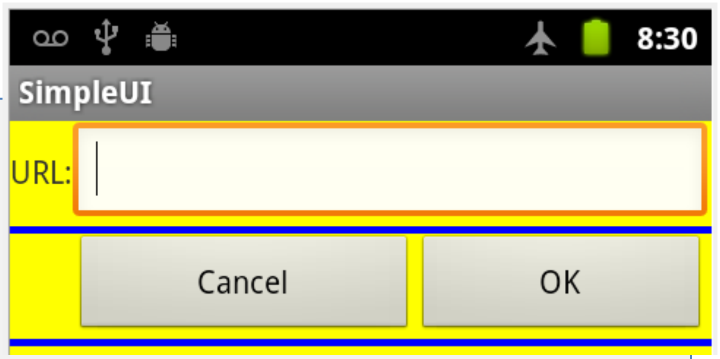
Note to the reader:
Experiment changing layout_span to 1, 2, 3

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 asingle 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.

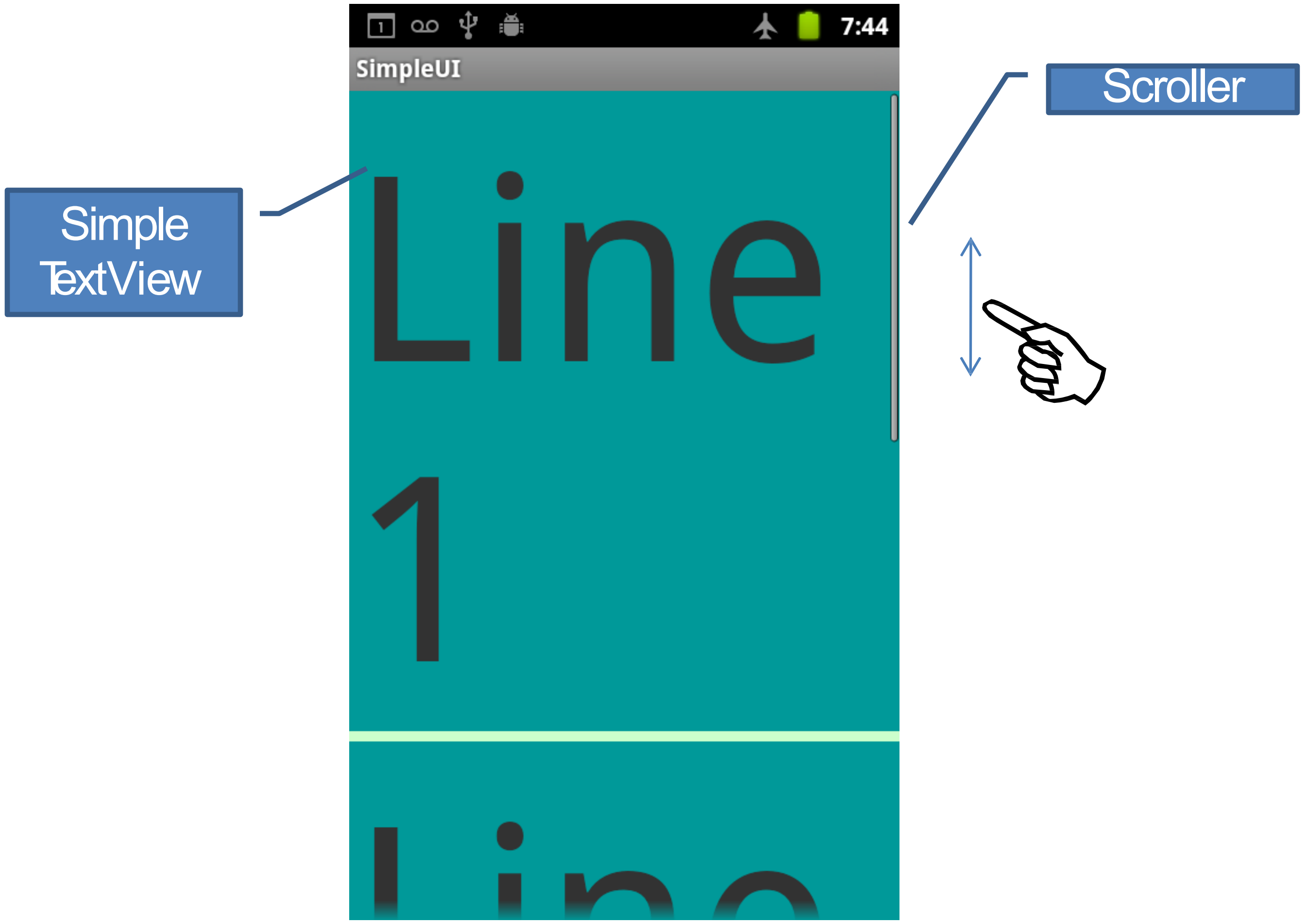
Basic XML Layouts - Containers

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"
            androi d : layout_height ="wrap_content"
            android:text="Line1"
            android:textSize="150dip" />
        <View
            androi d : layout_width ="f i ll_ parent "
            android:layout_height="6dip"
            android:background="#ffccffcc" />

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

        <View
            androi d : layout_width ="f i ll_ parent "
            android:layout_height="6dip"
            android:background="#ffccffcc" />

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

</ScrollView>
```

Basic XML Layouts - Containers

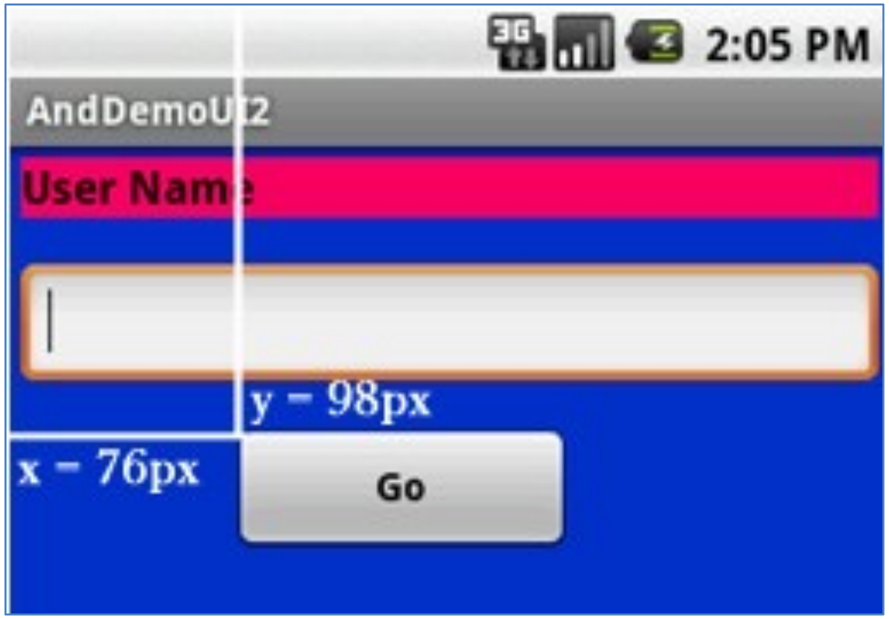
4. Example ScrollView Layout



Basic XML Layouts - Containers

5. Miscellaneous.
Absolute Layout

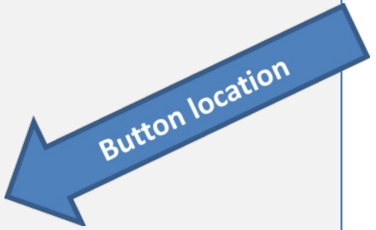
- A layout that lets you specify exact locations (x/y coordinates) of its children.
- Absolute layouts are *less flexible* and harder to maintain than other types of layouts without absolute positioning.



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>
```



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

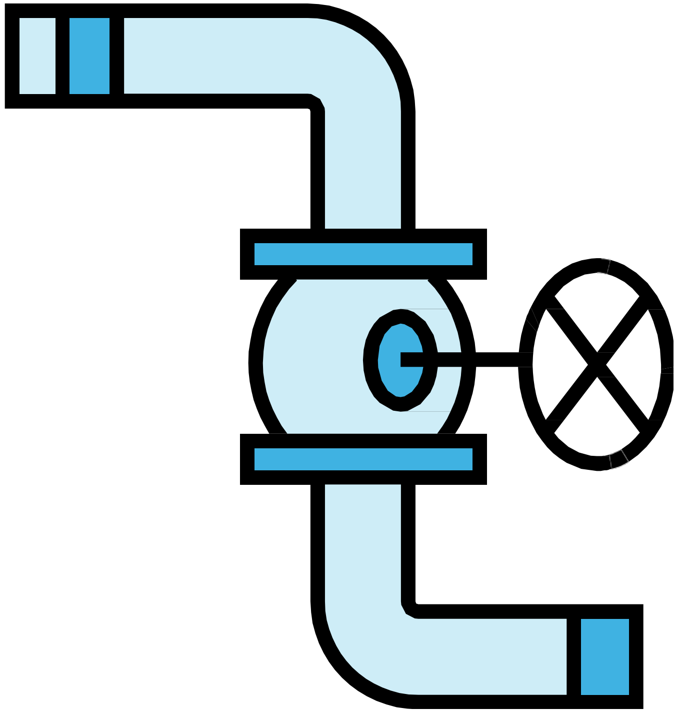
PopupWindow
ProgressBar
RadioButton
RadioGroup
RadioGroup.LayoutParams
RatingBar
RelativeLayout
RelativeLayout.LayoutParams
RemoteViews
ResourceCursorAdapter
ResourceCursorTreeAdapter
Scroller
ScrollView
SeekBar
SimpleAdapter
SimpleCursorAdapter
SimpleCursorTreeAdapter
SimpleExpandableListAdapter
SlidingDrawer
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:

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

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

CMSC 491/628: Introduction to Mobile Computing UI interface design

Nilanjan Banerjee

University of Maryland
Baltimore County
nilanb@umbc.edu
http://www.csee.umbc.edu/~nilanb/