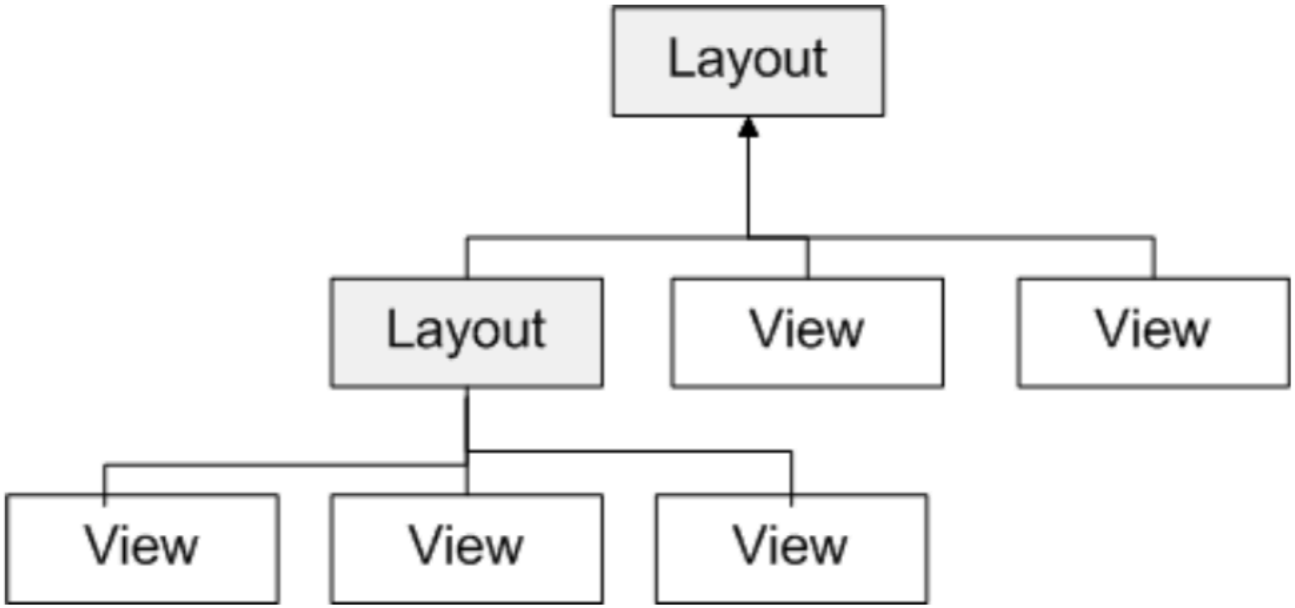
Graphical User Interfaces

The "View" Class

- The **View class** is the Android's most basic component from which users interfaces can be created. It acts as a container of displayable elements.
- A **View** occupies a rectangular area on the screen and is responsible for *drawing* and *event handling*.

The VIEW Class

- **Widgets** are subclasses of View. They are used to create interactive UI components such as buttons, checkboxes, labels, text fields, etc.
- **Layouts** are invisible structured containers used for holding other Views and nested layouts.



XML Layouts

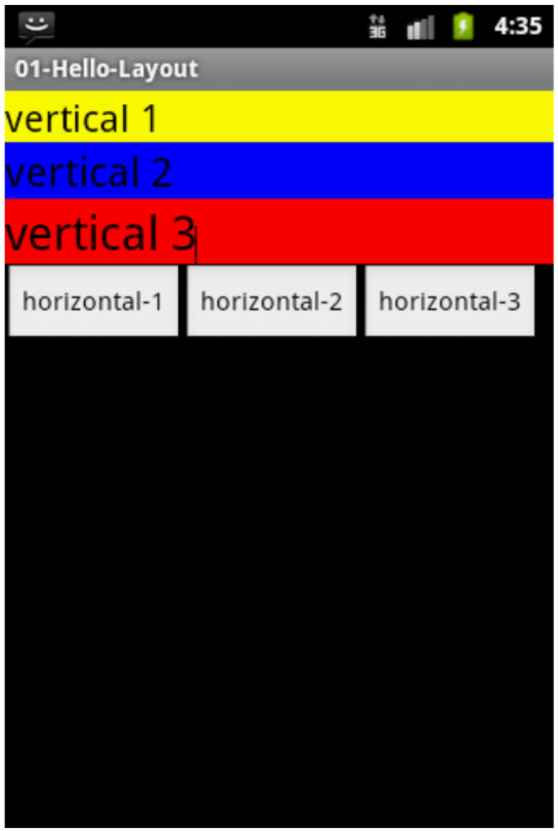
Using XML to represent UIs

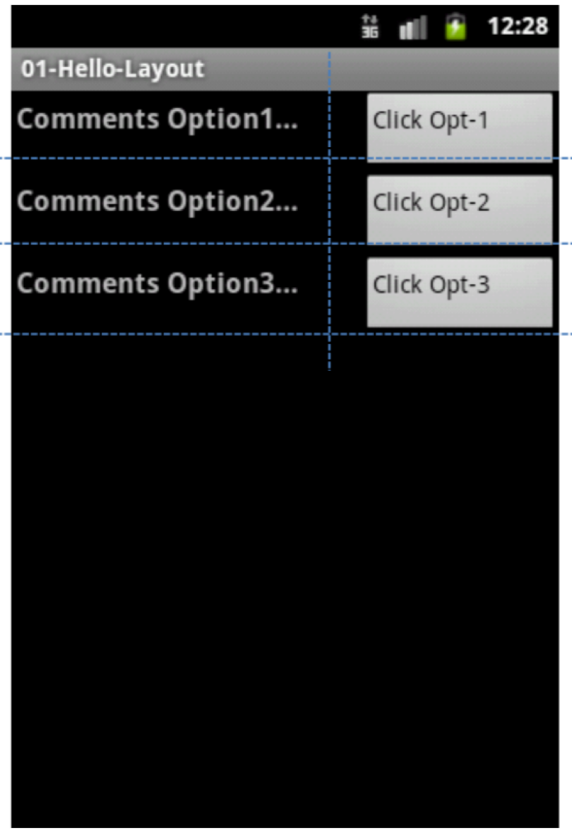
Actual UI displayed by the app

Text version: *activity_main.xml* file →

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="csu.matos.gui_demo.MainActivity" >

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="36dp"
        android:text="@string/edit_user_name"
        android:ems="12" >
        <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="48dp"
        android:text="@string/btn_go"  />

</RelativeLayout>
```

A Sample of Common Android LAYOUTS



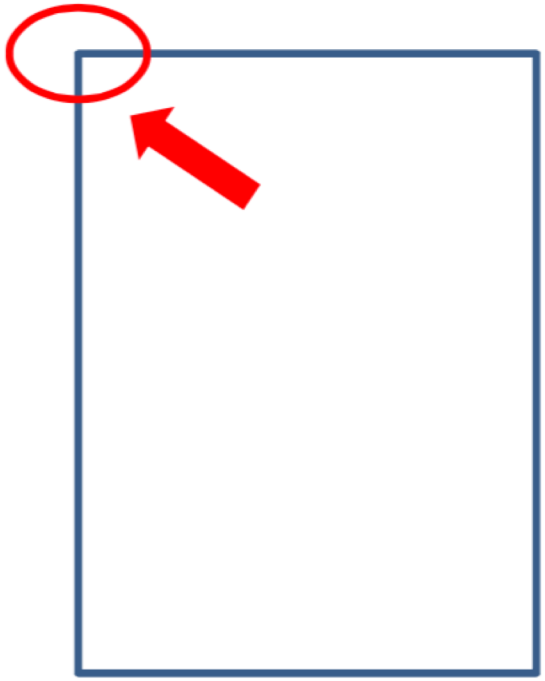


Linear Layout
A LinearLayout places its inner views either in horizontal or vertical disposition.

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.

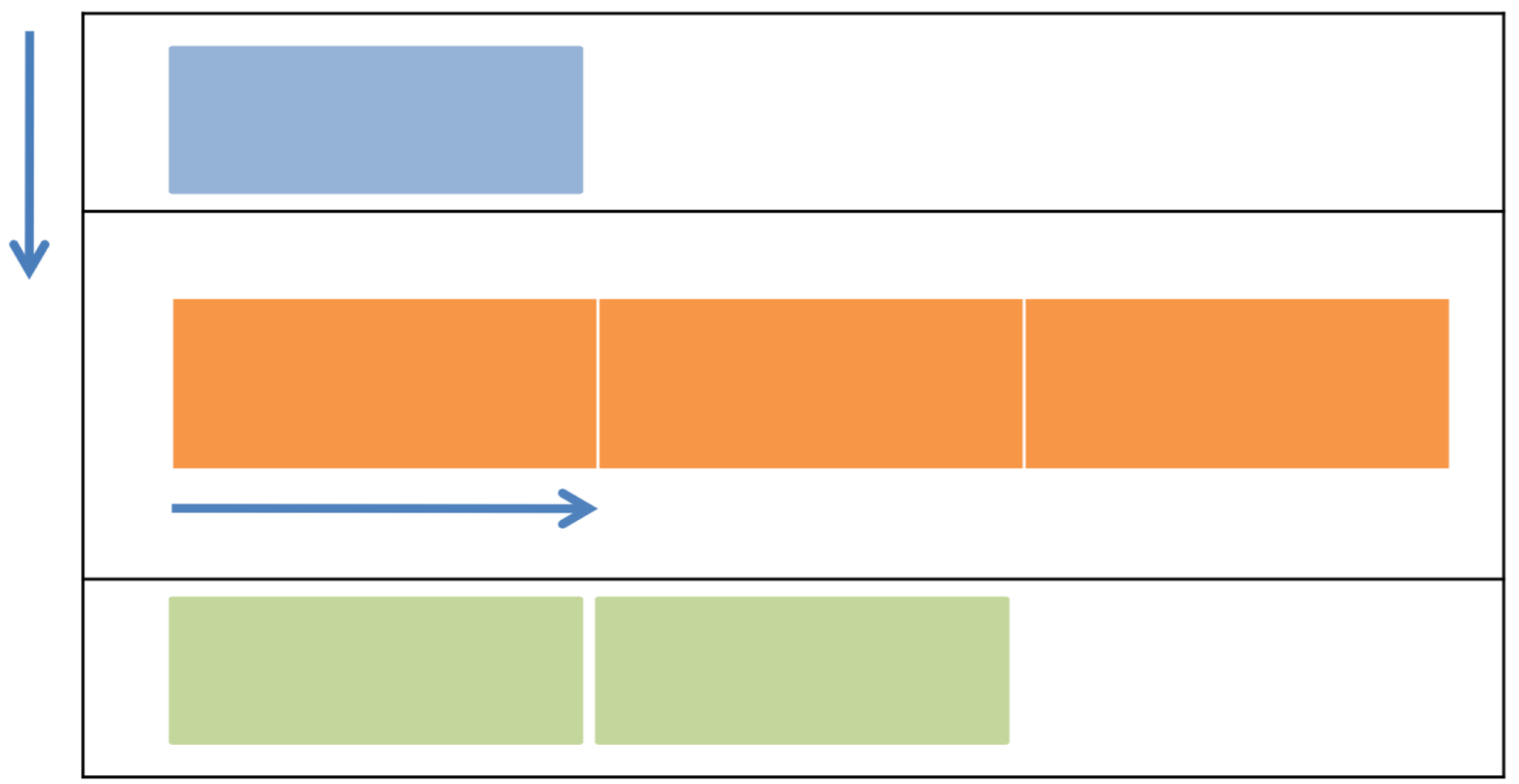
FrameLayout



- The **FrameLayout** is the simplest type of GUI container.
- It is useful as an *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.*;

LinearLayout

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



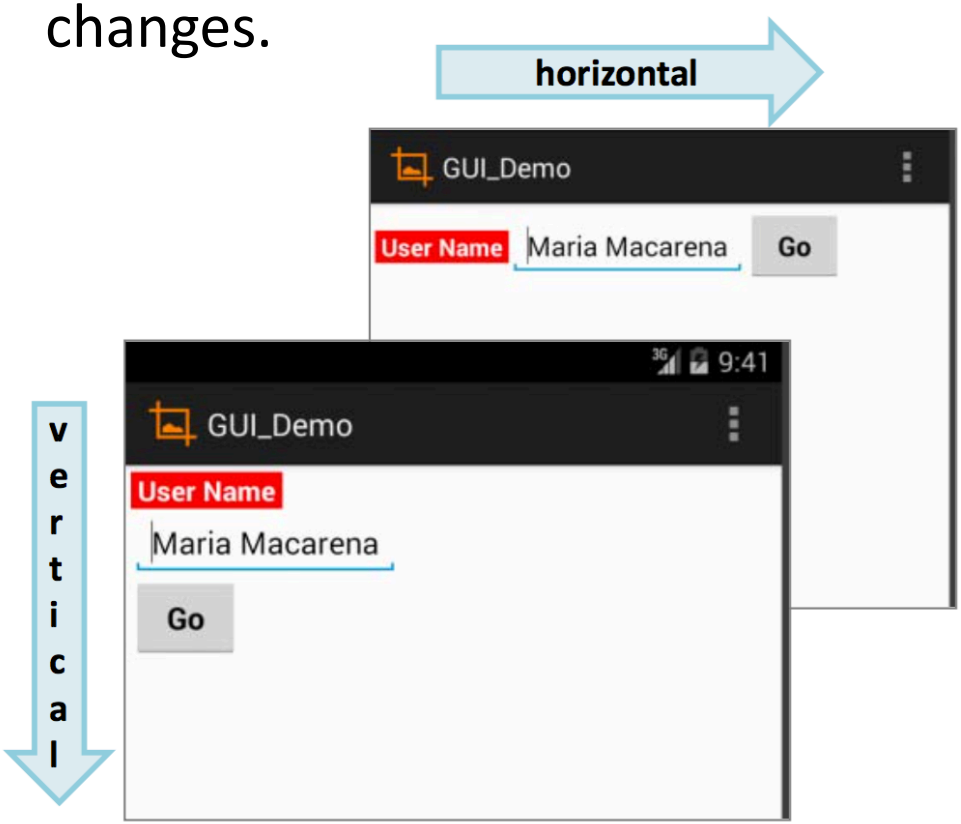
LinearLayout

- **Setting Attributes**

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

LinearLayout: Orientation

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

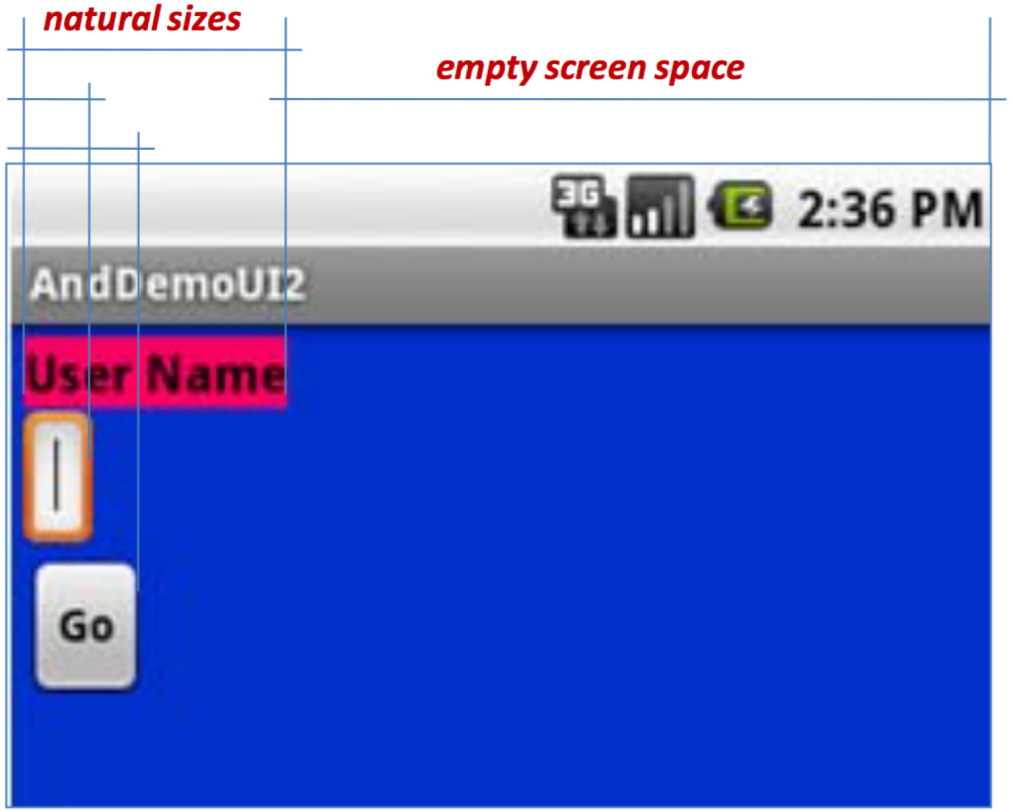


Shown on a Kitkat device

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myLinearLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:padding="4dp" >

    <TextView
        android:id="@+id/labelUserName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#ffff0000"
        android:text=" User Name "
        android:textColor="#ffffffff"
        android:textSize="16sp"
        android:textStyle="bold" />

    <EditText
        android:id="@+id/ediName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Maria Macarena"
        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>
```

LinearLayout : Fill Model

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



Shown on a Gingerbread device

LinearLayout : Fill Model

- All widgets inside a LinearLayout **must** include 'width' and 'height' attributes.
  android:layout_width
  android:layout_height
- Values used in defining height and width can be:

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

LinearLayout : Fill Model

Medium resolution is: 320 x 480 dpi.
Shown on a Gingerbread device

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myLinearLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#ff0033cc"
    android:orientation="vertical"
    android:padding="6dp" >

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

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

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

Row-wise

Use all the row

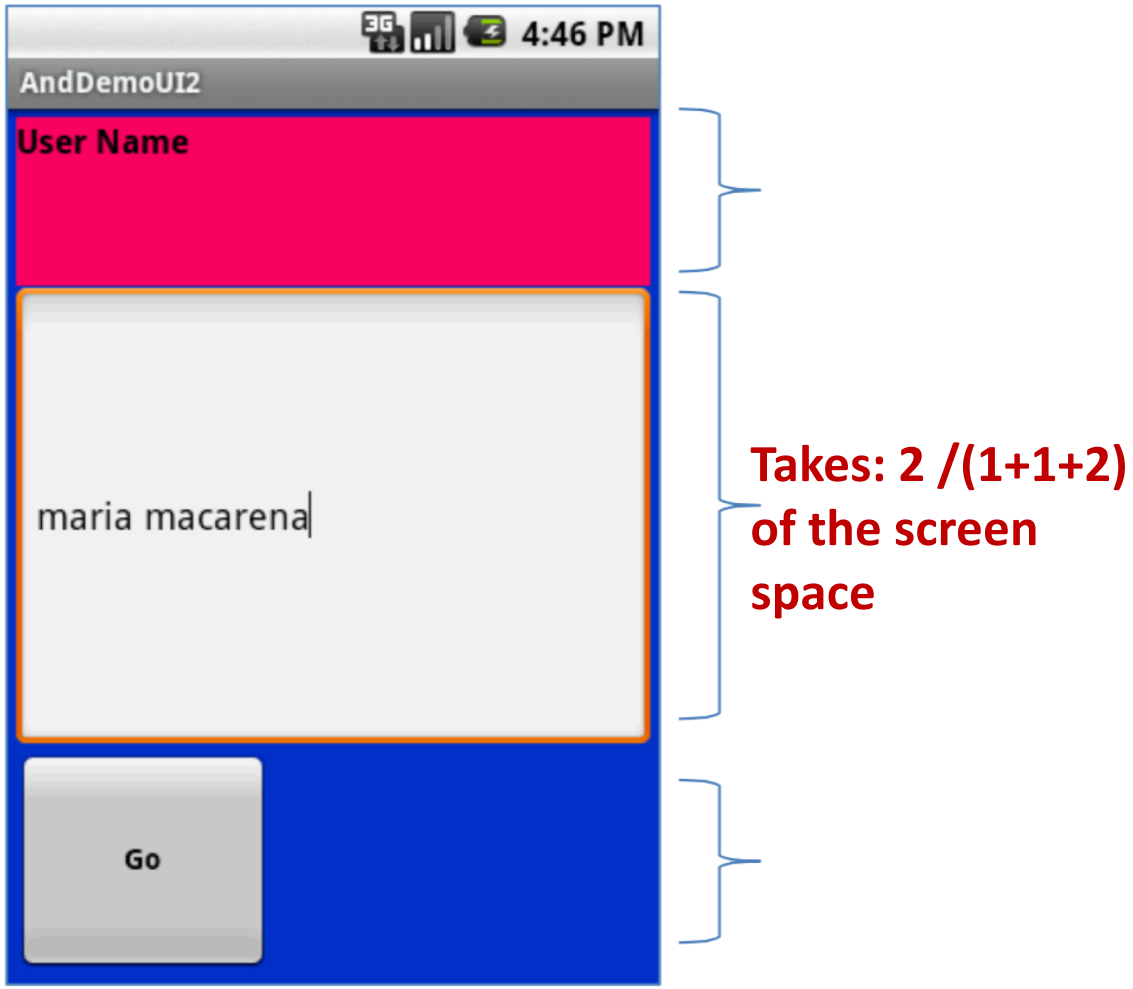
Specific size: 125dp

LinearLayout : Weight

- The bigger the **weight** the larger the extra space given to that widget. Use **0** if the view should not be stretched.

Example:

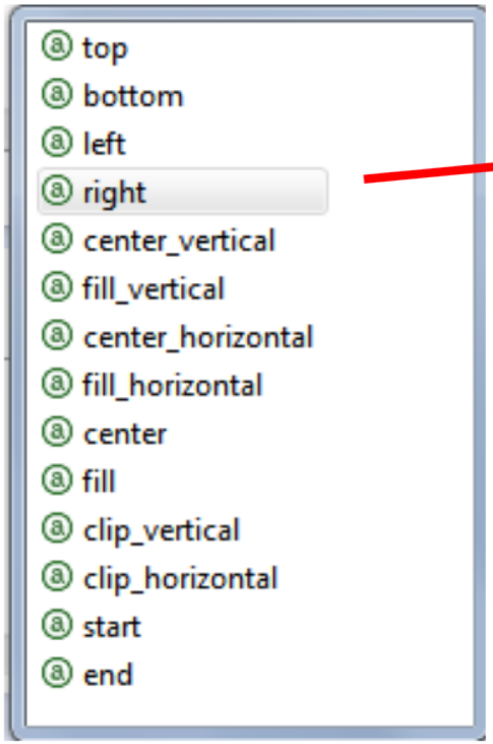
- The **TextView** and **Button** controls have the additional property **android:layout_weight="*1*"**
- The **EditText** control has **android:layout_weight="*2*"**
- *Remember, default value is 0*

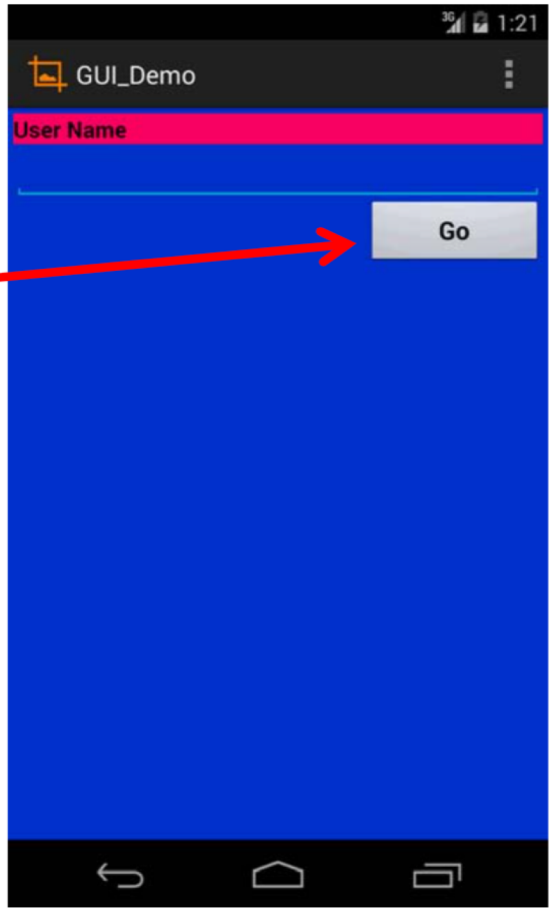


Gingerbread

LinearLayout : Gravity

- **Gravity** 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.





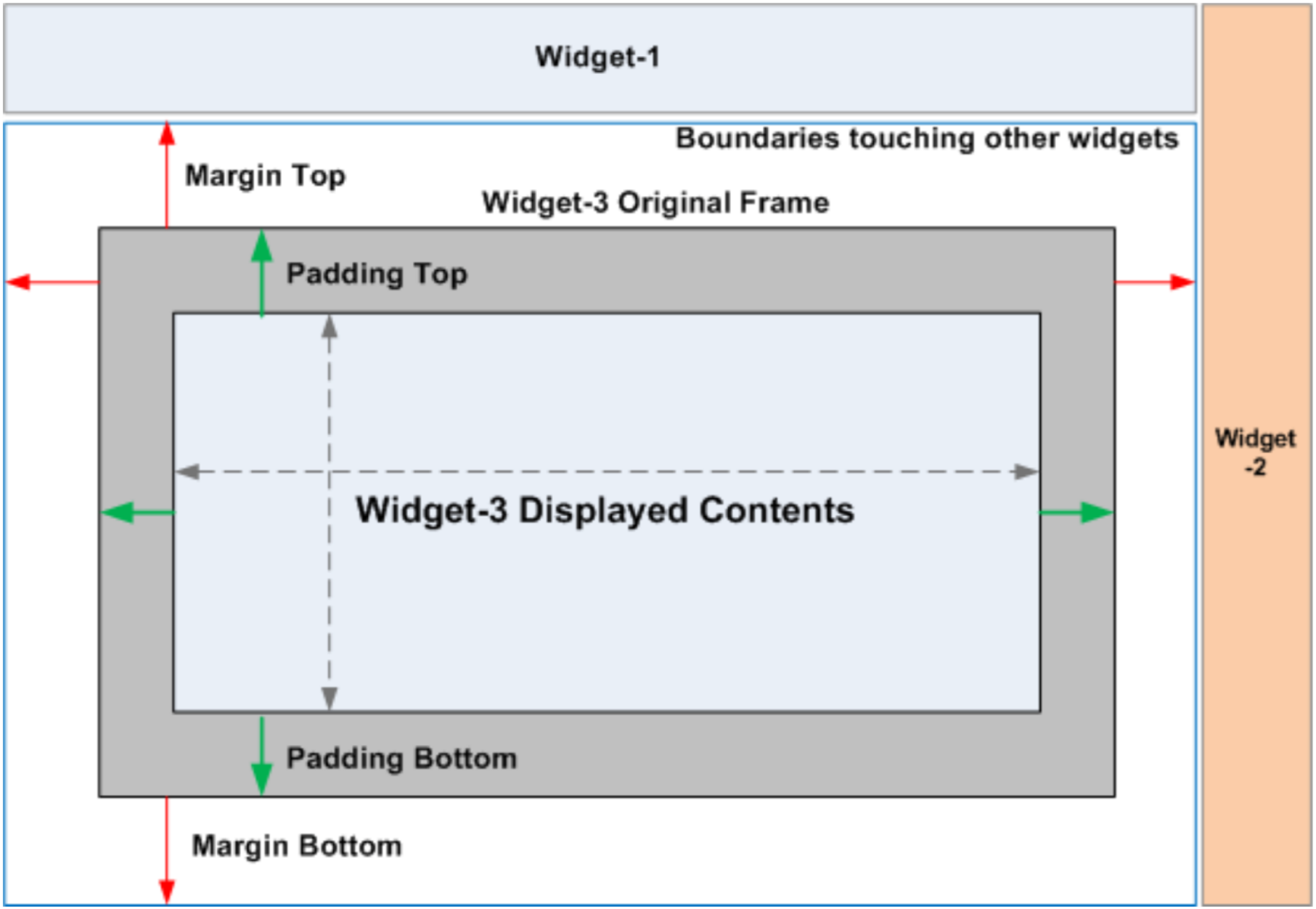
Button has **right** layout_gravity

LinearLayout : Padding

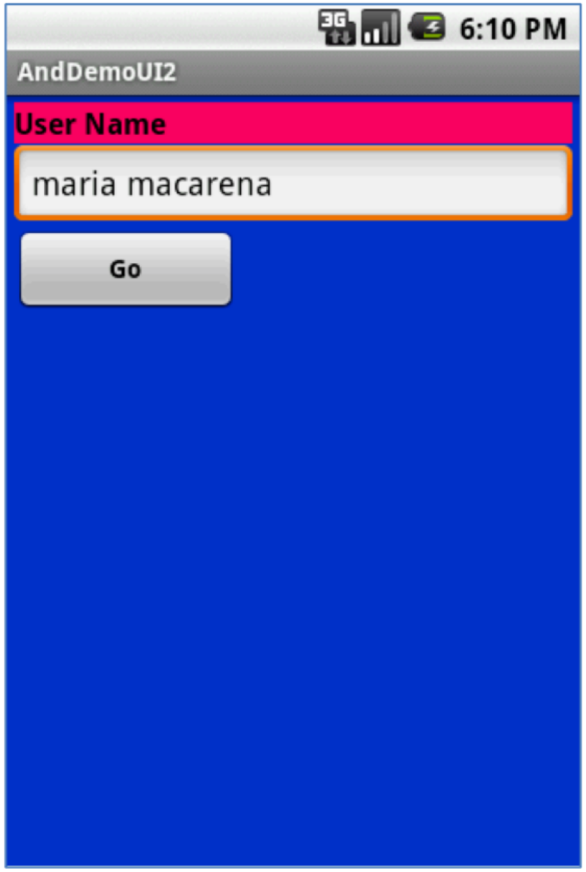
- The **padding** attribute specifies the widget's internal margin (in **dip** units).
- The internal margin is the extra space between the borders of the widget's "cell" and the actual widget contents.
- Either use
  - **android:padding** property or
  - call method **setPadding()** at runtime.

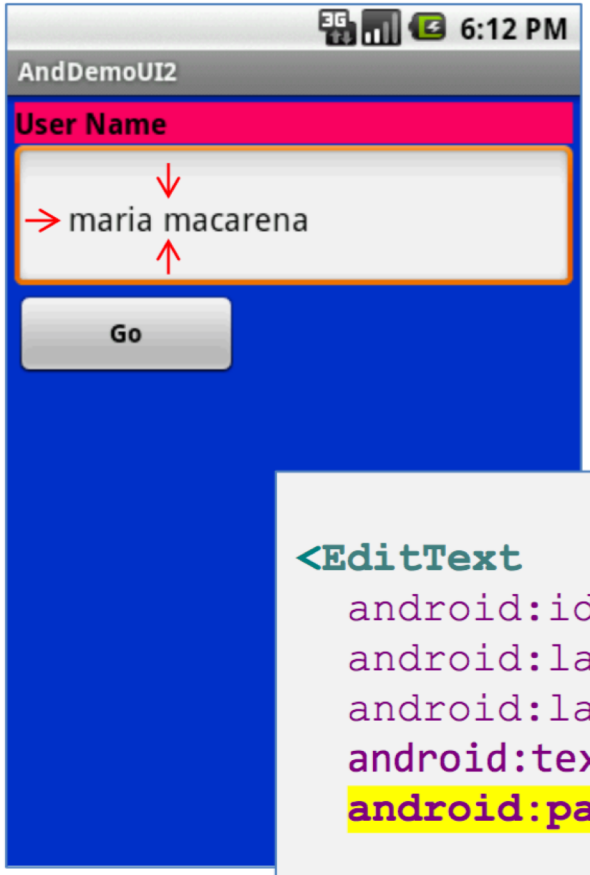
The 'blue' surrounding space around the text represents the inner view's padding

LinearLayout : Padding and Margin



LinearLayout : Set Internal Margins Using Padding

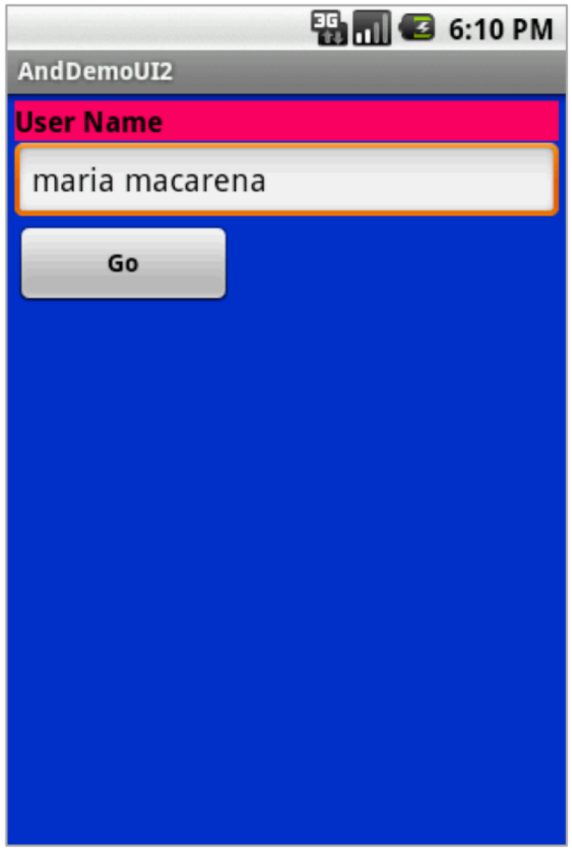




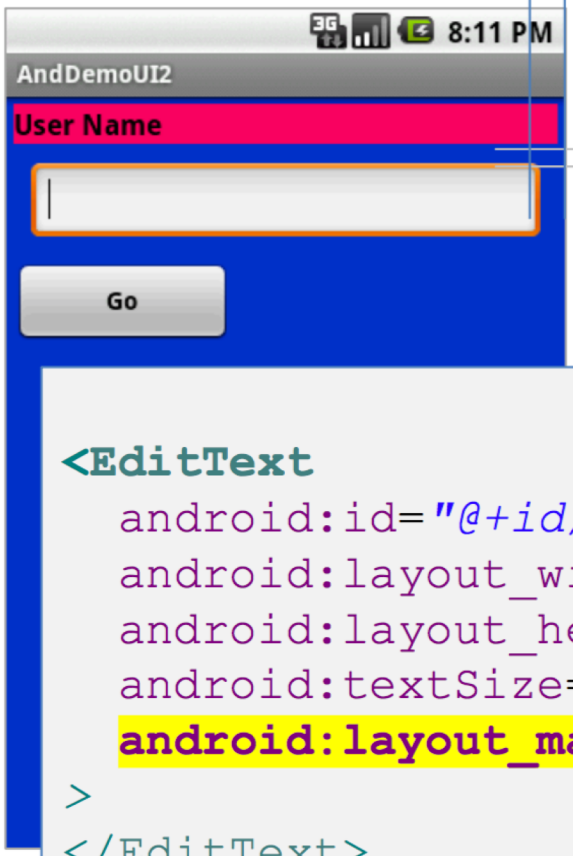

```
<EditText
   android:id="@+id/ediName"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:textSize="18sp"
   android:padding="30dp"  />

...
```

LinearLayout : Set External Margins

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



Using default spacing between widgets

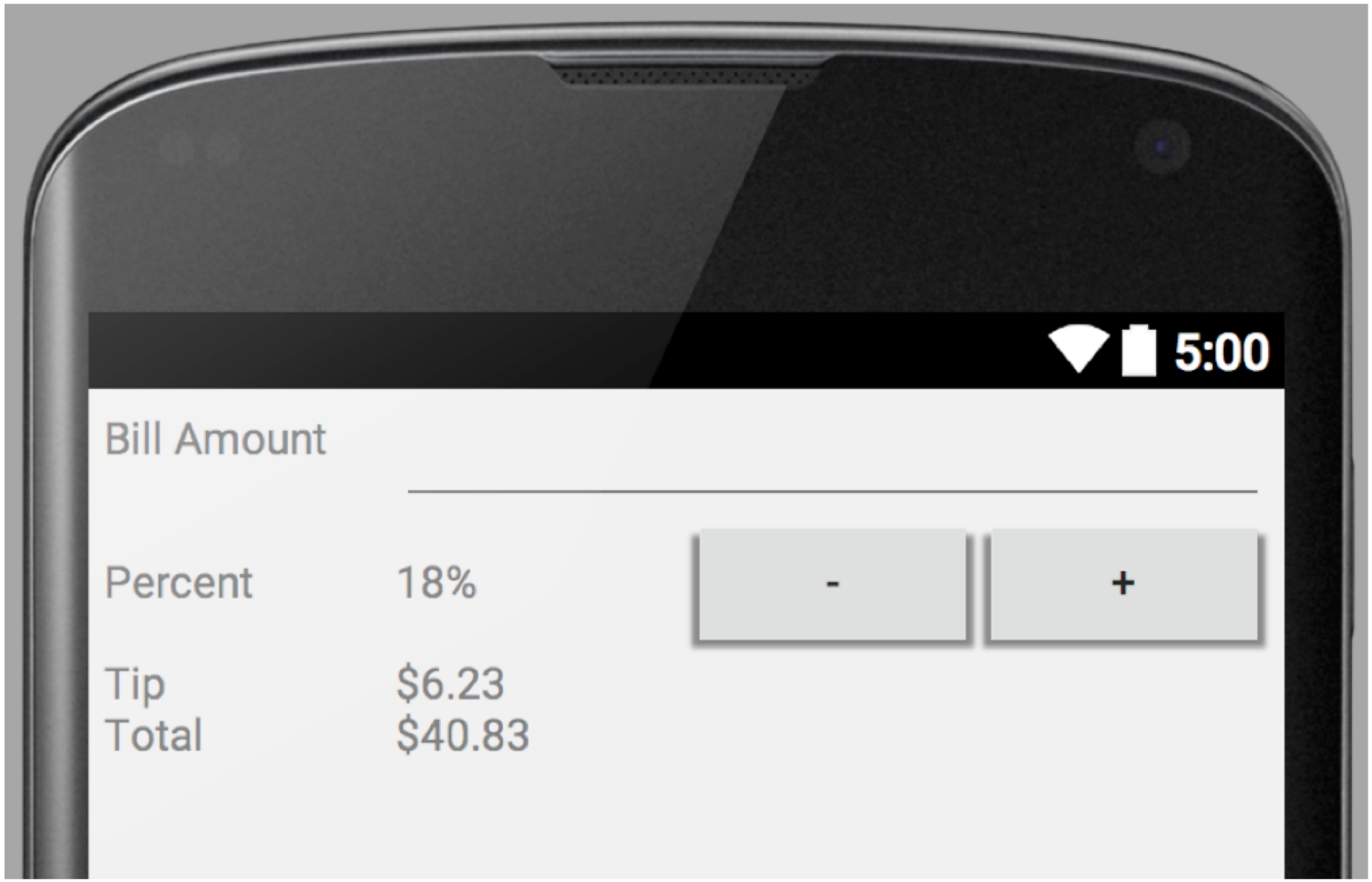


Increased inter-widget space

```
<EditText
   android:id="@+id/ediName"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:textSize="18sp"
   android:layout_margin="6dp"
>
</EditText>
...
```

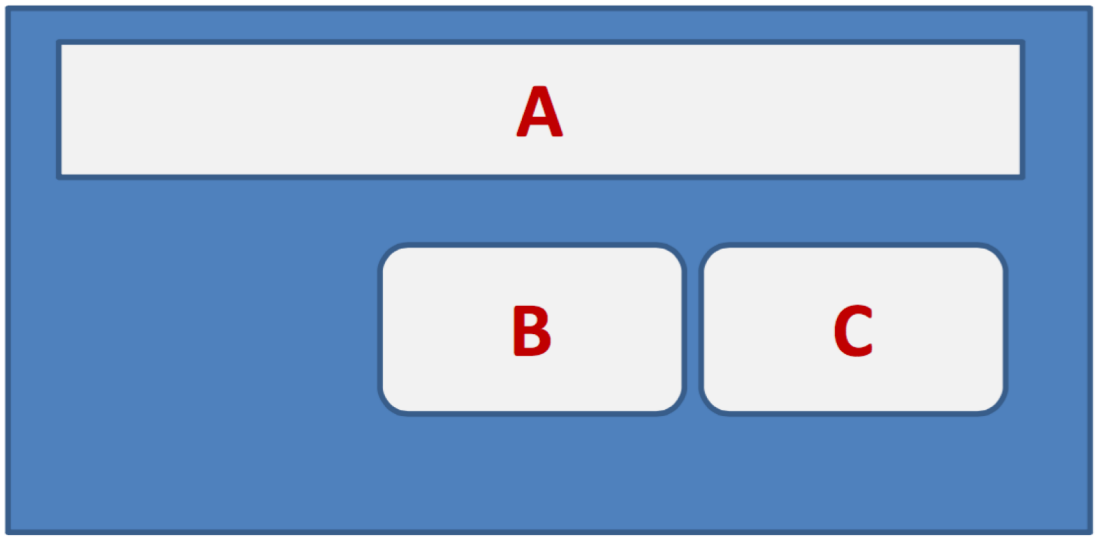
LinearLayout: Exercise 1

- Design GUI as the figure below using LinearLayout and their properties



Relative Layout

- The placement of a widget in a **RelativeLayout** is based on its *positional relationship* to other widgets in the container as well as the parent container.



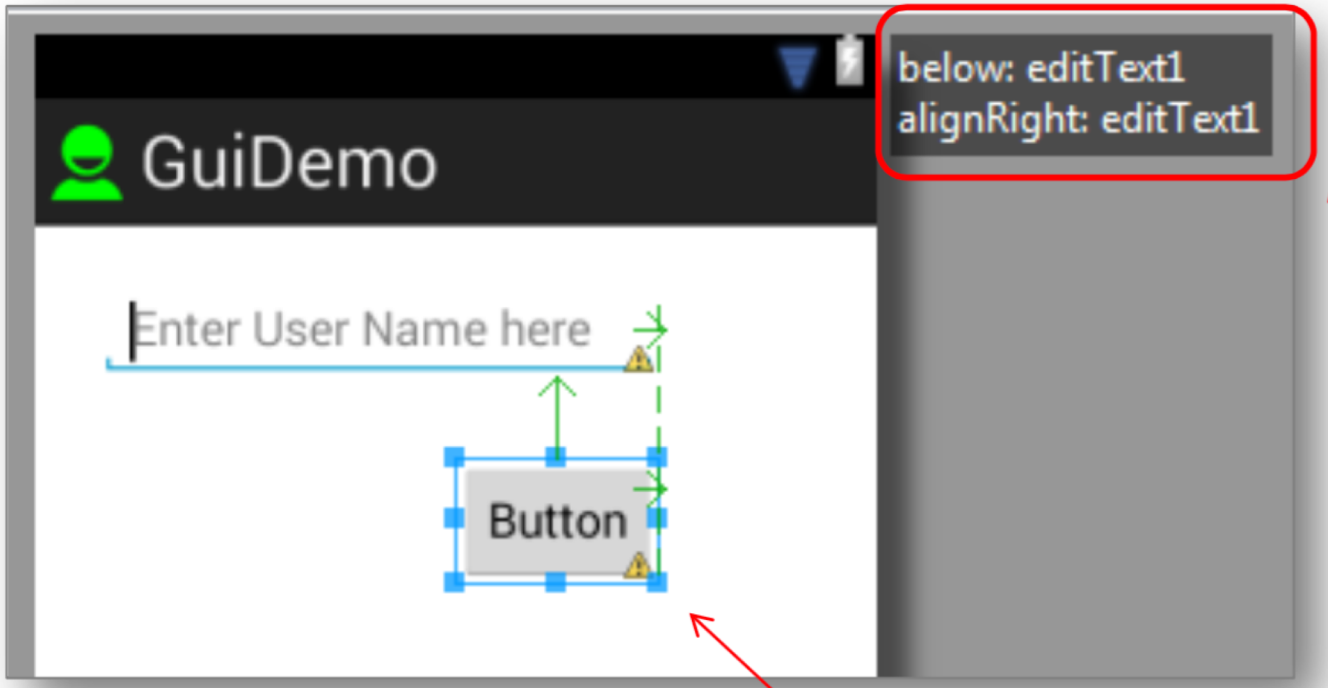
Example:

A is by the parent's top

C is below A, to its right

B is below A, to the left of C

Relative Layout

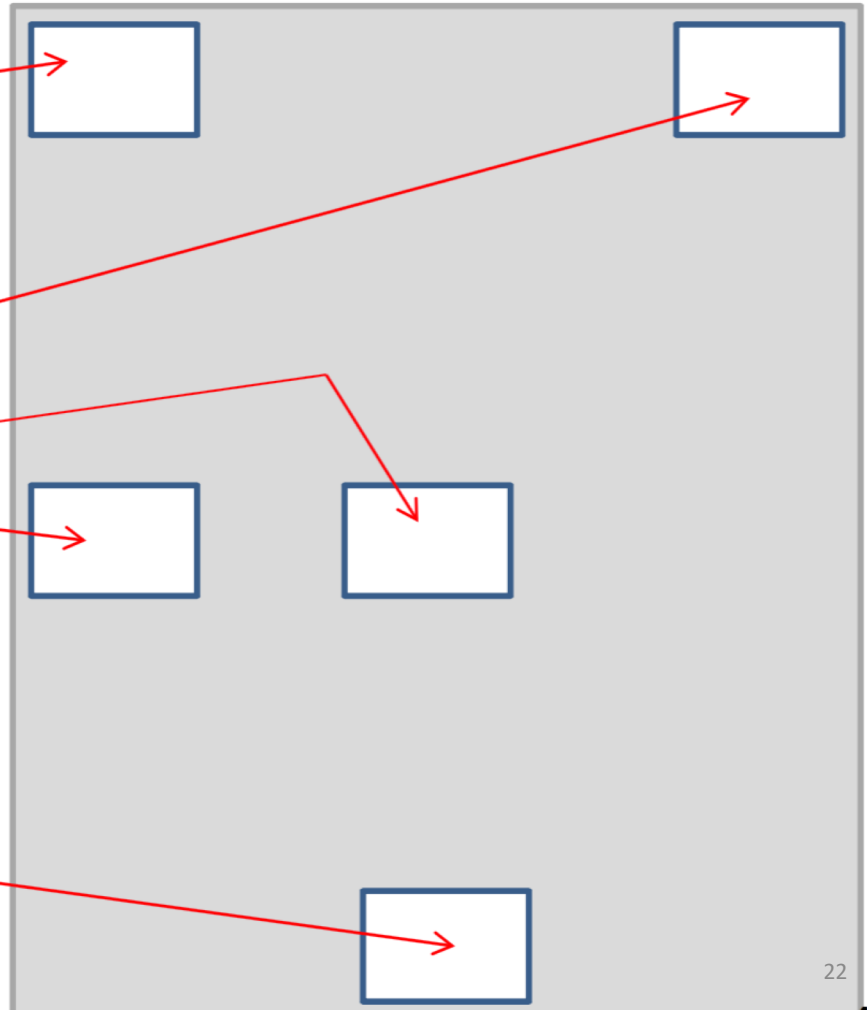


Location of the button is expressed in reference to its *relative* position with respect to the EditText box.

Relative Layout - Referring to the container

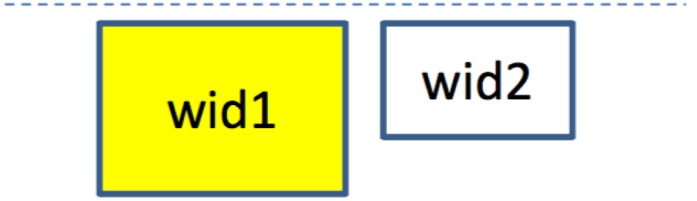
- Collocating a widget based on the location of its **parent** container *boolean properties* (**true/false**)

```
android:layout_alignParentTop
android:layout_alignParentBottom

android:layout_alignParentLeft
android:layout_alignParentRight

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

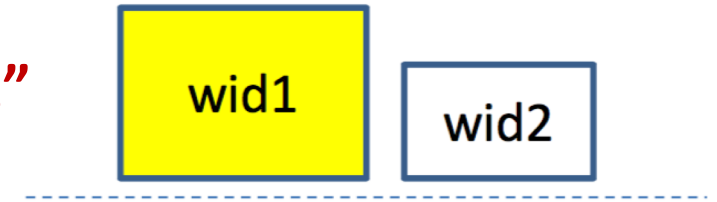


Relative Layout - Referring to Other Widgets

android:layout_alignTop="@+id/wid1"



android:layout_alignBottom ="@+id/wid1"



android:layout_alignLeft="@+id/wid1"

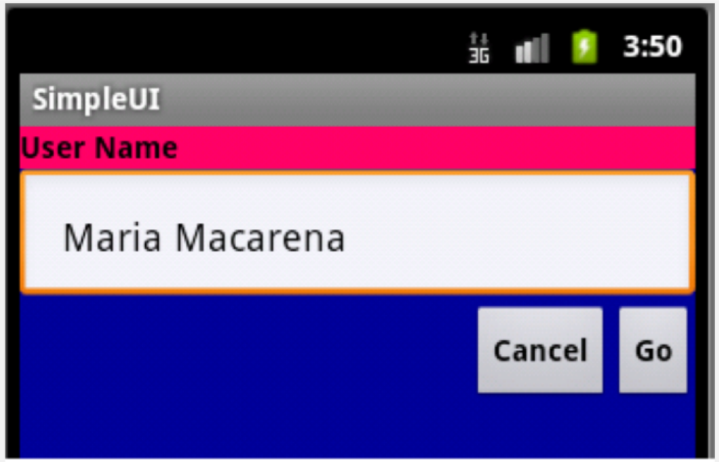
android:layout_alignRight="@+id/wid1"

Relative Layout - Referring to Other Widgets

- When using relative positioning you need to:

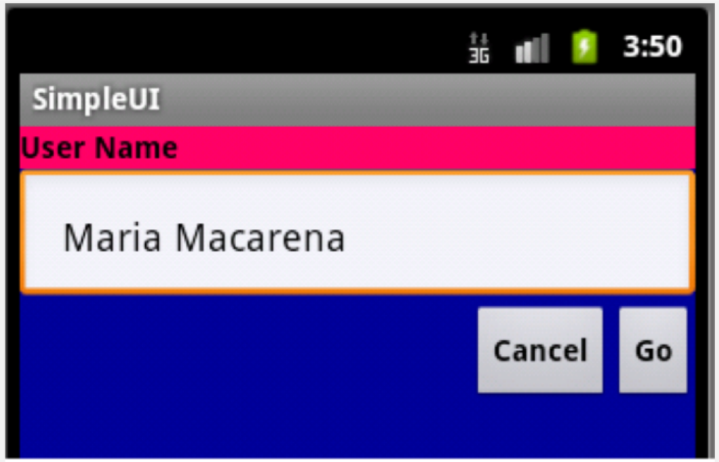
1. Use identifiers (**android:id** attributes) on *all elements* that you will be referring to.
2. XML elements are named using the prefix: **@+id/...** For instance an EditText box could be called: **android:id="*@+id/txtUserName***"
3. You must refer only to widgets that have been already defined. For instance a new control to be positioned below the *txtUserName* EditText box could refer to it using: ***android:layout_below="@+id/txtUserName"***

Relative Layout – Example



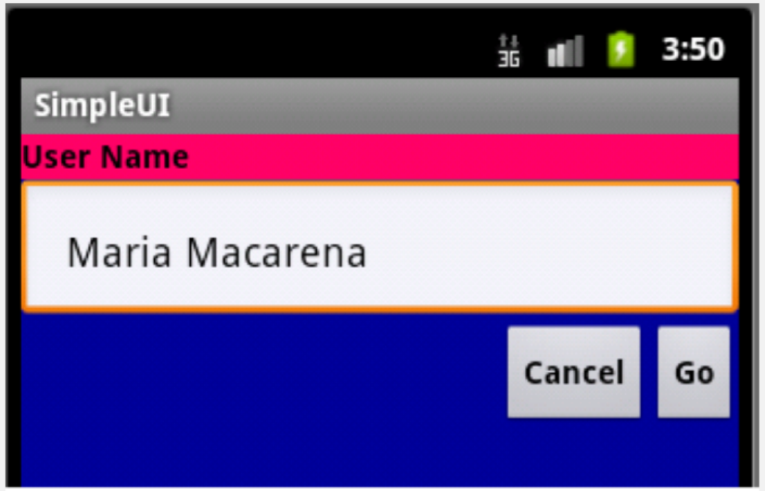

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myRelativeLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#ff000099" >

    <TextView
        android:id="@+id/lblUserName"
        android:layout_width="match_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>
```

Relative Layout - Example




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

RelativeLayout - Example




```
    <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>
```

RelativeLayout – Exercise 2

- Design GUI as the figure below using **RelativeLayout** and their properties

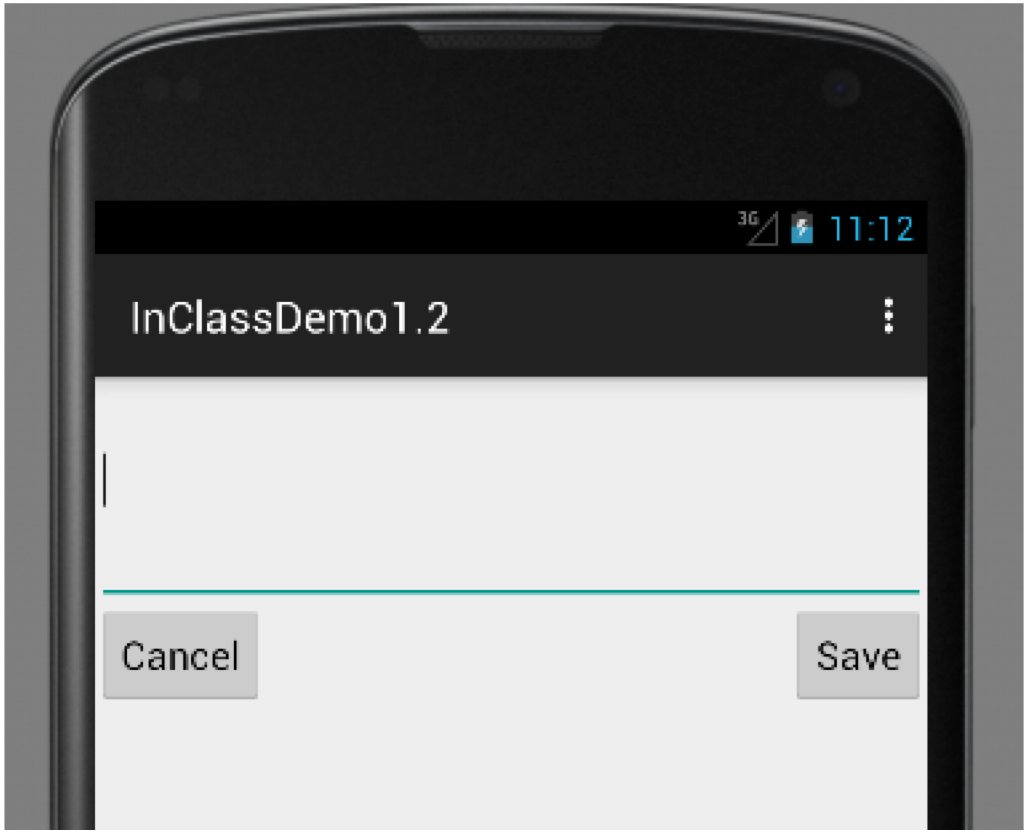


Table Layout

- Columns are *flexible*, they could *shrink* or *stretch* to accommodate their contents.
- The element **TableRow** is used to define a new row in which widgets can be allocated.
- The number of columns in a TableRow is determined by the total of side-by-side widgets placed on the row.

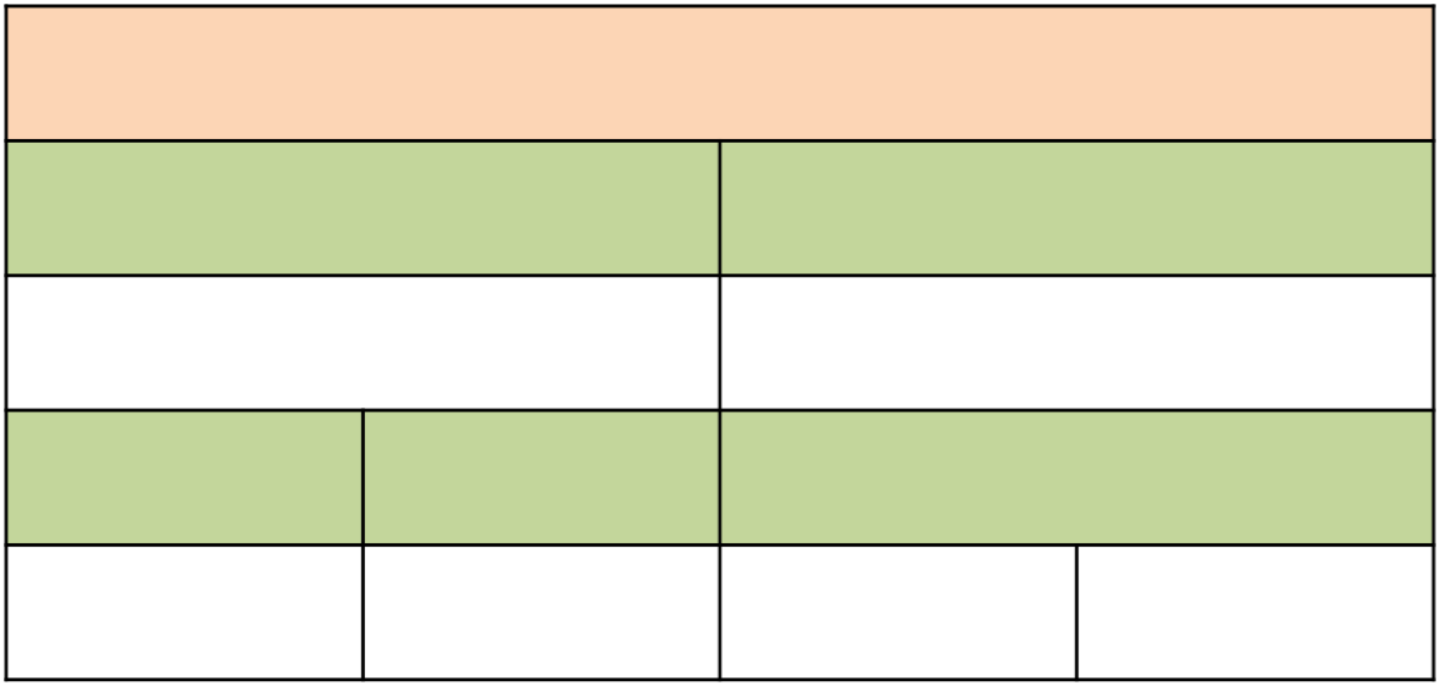
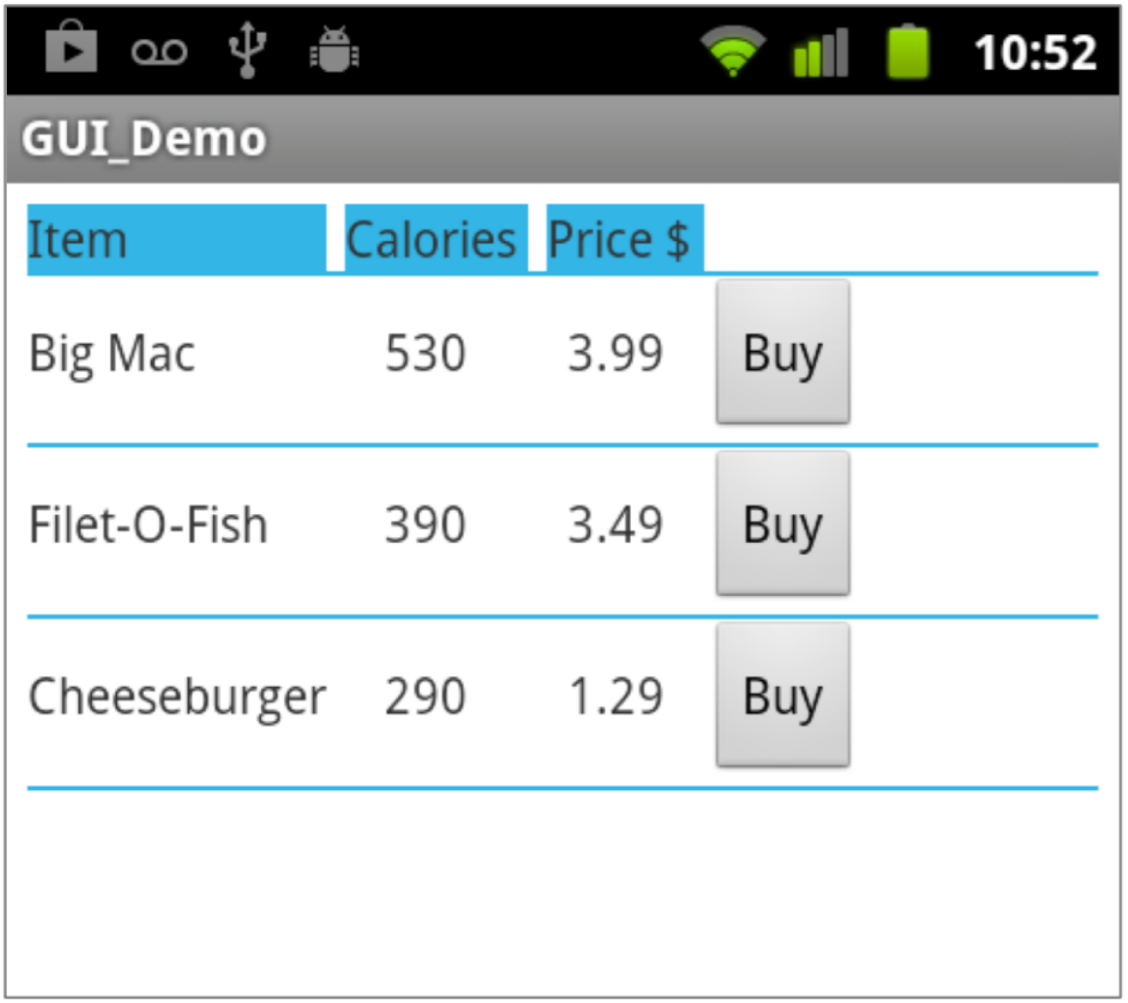


Table Layout – Setting Number of Columns

- ***The final number of columns in a table is determined by Android.***
- **Example**: If your *TableLayout* have three rows
  1. one row with two widgets,
  2. one with three widgets, and
  3. one final row with four widgets,
     there will be at least **four** columns in the table, with column indices: **0, 1, 2, 3.**

0		1	
0		1	2
0	1	2	3

Table Layout – Example



The screen shows various items from a McDonald's restaurant menu [*].
The **TableLayout** has four **TableRows**, with three columns in the first row (labels) and four cells in each of the other three rows (item, Calories, Price, and Buy button).

[*] Reference: Pages visited on Sept 8, 2014
http://nutrition.mcdonalds.com/getnutrition/nutritionfacts.pdf
http://hackthemenu.com/mcdonalds/menu-prices/

Table Layout – Example (*continuation*)

```
<TableLayout

xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myTableLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="6dp" >

    <TableRow>
        <TextView
            android:background="#FF33B5E5"
            android:text="Item " />
        <TextView
            android:layout_marginLeft="5dp"
            android:background="#FF33B5E5"
            android:text="Calories " />

        <TextView
            android:layout_marginLeft="5dp"
            android:background="#FF33B5E5"
            android:text="Price $ " />
    </TableRow>

    <View
        android:layout_height="1dp"
        android:background="#FF33B5E5" />

    <TableRow>

        <TextView
            android:text="Big Mac" />
        <TextView
            android:gravity="center"
            android:text="530" />
        <TextView
            android:gravity="center"
            android:text="3.99" />
        <Button
            android:id="@+id/btnBuyBigMac"
            android:gravity="center"
            android:text="Buy" />
    </TableRow>

    <View
        android:layout_height="1dp"
        android:background="#FF33B5E5" />

    <!-- other TableRows ommitted --!>


</TableLayout>
```

Table Layout – Stretching a Column

- A single widget in a **TableLayout** can occupy more than one column.
- The **android:layout_span** property indicates the number of columns the widget is allowed to expand.

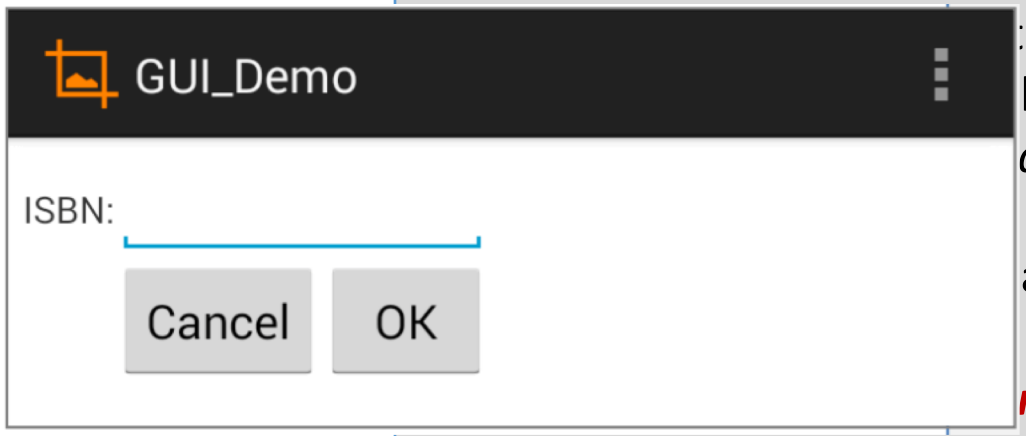
```
<TableRow>
    <TextView
              android:text="URL:" />
    <EditText
              android:id="@+id/txtData"
              android:layout_span="3" />
</TableRow>
```

Table Layout – Stretching a Column

- **Example:**
  1. The table shown below has four columns (*indices*: 0,1,2,3).
  2. The label ("**ISBN**") goes in the first column (*index 0*).
  3. The EditText to the right of the label uses the layout_span attribute to be placed into a spanned set of three columns (columns 1 through 3).

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

Table Layout – Example




```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/myTableLayout"
                                    :h="match_parent"
                                    ht="match_parent"
                                    dp" android:orientation="vertical" >

                                    android:text="ISBN:" />
                                    ndroid:id="@+id/ediISBN"
                                    ndroid:layout_span= “3” />
        </TableRow>
        <TableRow>
                <Button android:id="@+id/cancel”
                        android:layout_column=“2”
                        android:text="Cancel" />
                <Button android:id="@+id/ok”
                        android:text="OK" />
        </TableRow>
</TableLayout>
```

Table Layout – Stretching the Entire Table

- By default, a column is as wide as the “natural’ size of the widest widget collocated in this column
- A table does not necessarily take all the horizontal space available.
- If you want the table to (horizontally) match its container use the property:

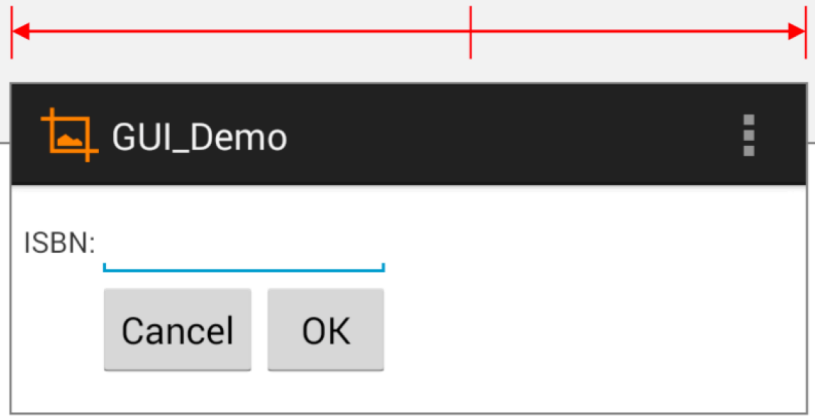
  android:stretchColumns=“column(s)”

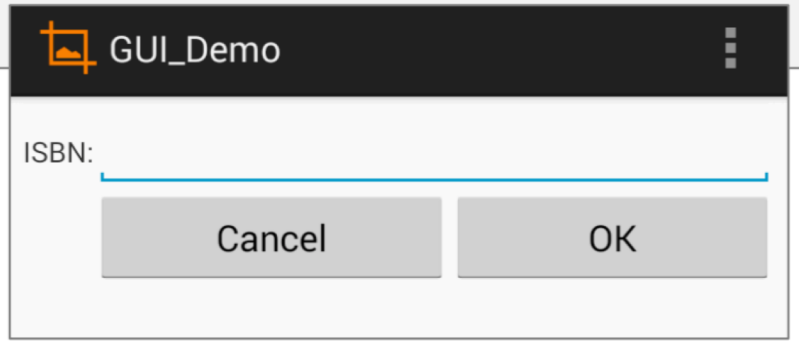
- Where ‘column(s)’ is the column-index to be stretched to take up any space still available on the row.
- For example, to stretch columns 0, and 2 of a table you set

  android:stretchColumns=“0,2”

Table Layout – Stretching the Entire Table

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





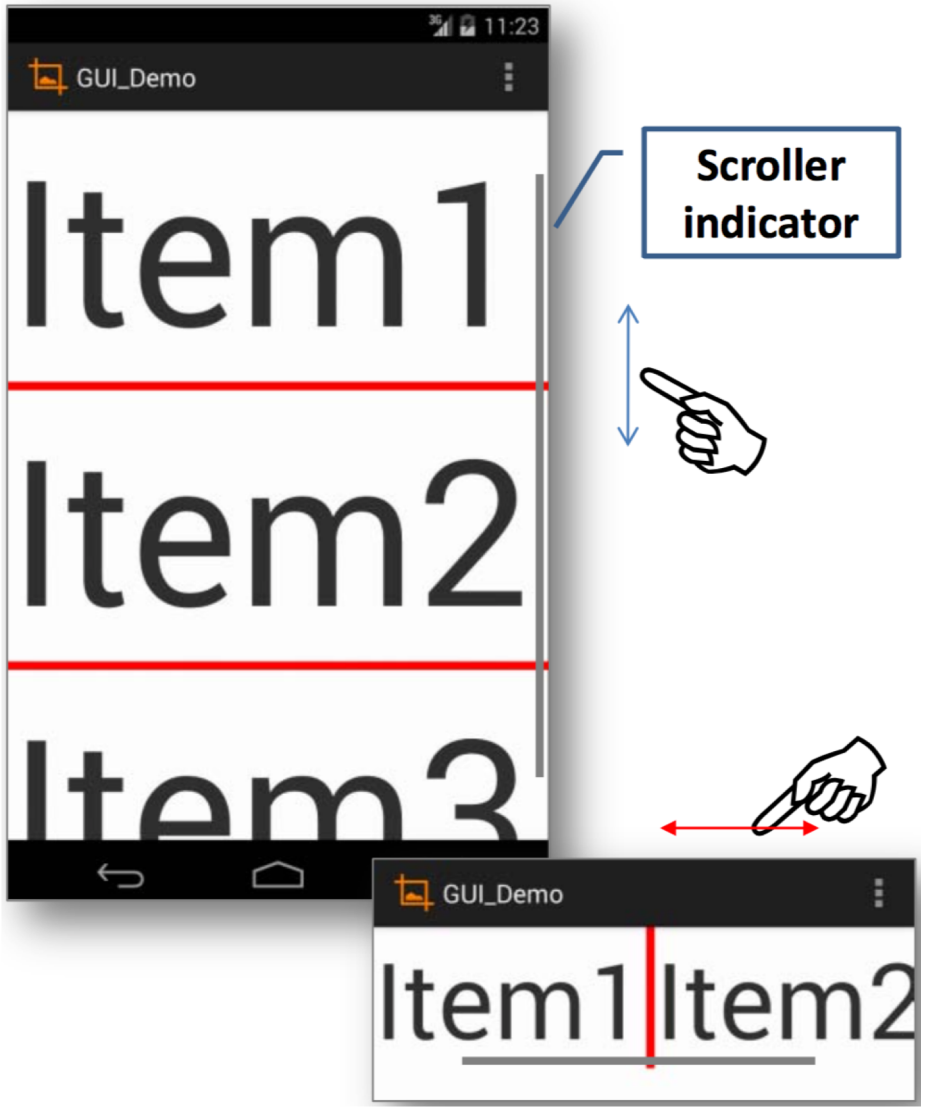
Screens shown before and after using the `android:stretchColumns` clause.

Table Layout – Exercise 3

Using **TableLayout** to design the GUI xml layout as the figure

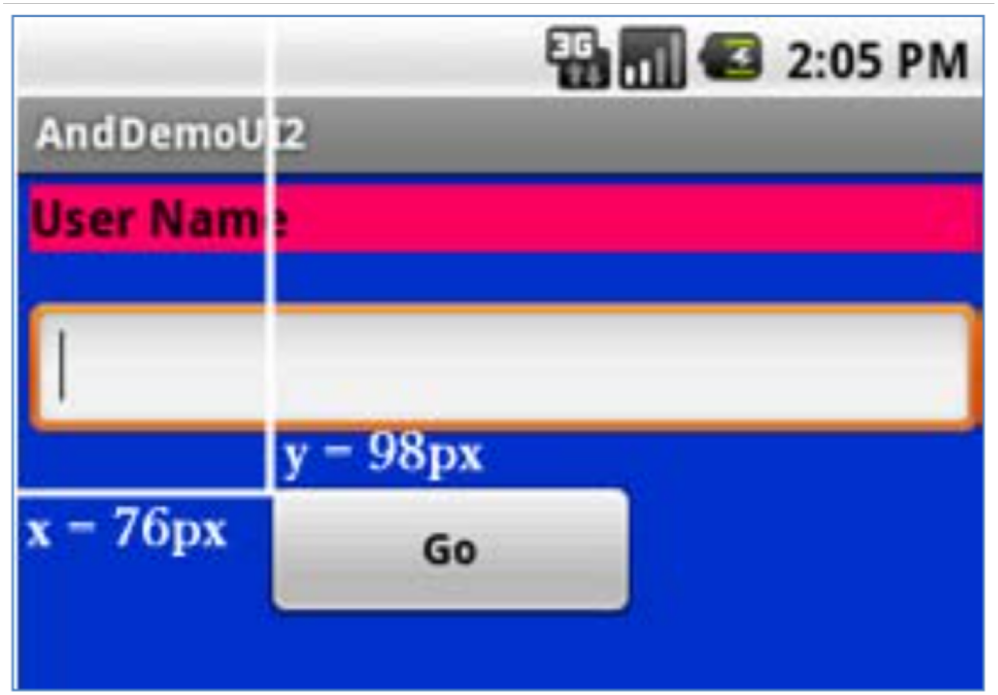
ScrollView Layout (Vertical & Horizontal)

- The **ScrollView** control is useful in situations in which we have *more data to show* than what a single screen could display.
- **ScrollViews** provide a vertical sliding (up/down) access to the data.
- The **HorizontalScrollView** provides a similar left/right sliding mechanism)
- Only a portion of the user's data can be seen at one time, however the rest is available for viewing.



Miscellaneous: Absolute Layout (Deprecated)

- This layout lets you specify exact locations (x/y coordinates) of its children.
- Absolute layouts are *less flexible* and harder to maintain than other types of layouts without absolute positioning.
- They DO NOT migrate well from one device to the other; not even from *portrait* to *landscape* modes in the same device!



Connecting Layouts to Java Code

findViewById(R.id.<xml_widget_id>)

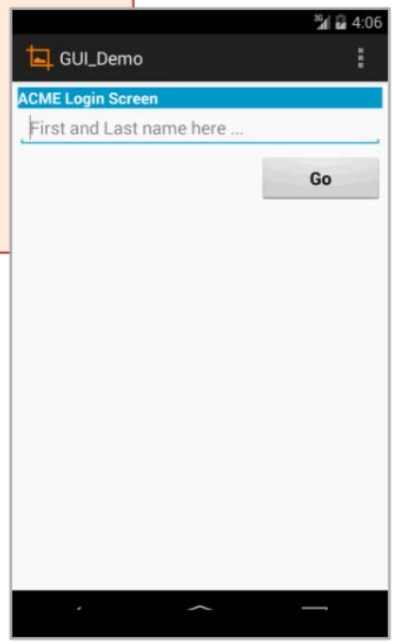
XLM Layout

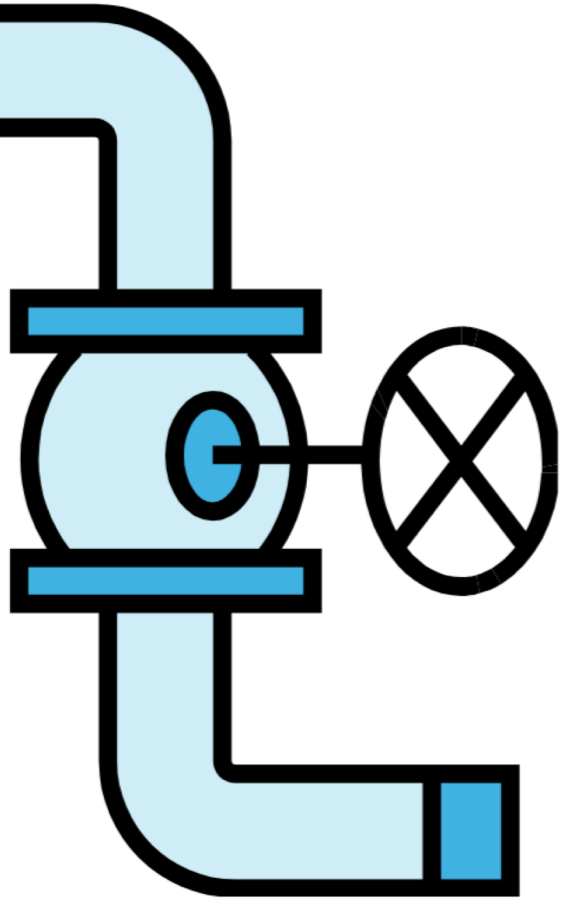
<xml....

. . .

. . .

</xml>

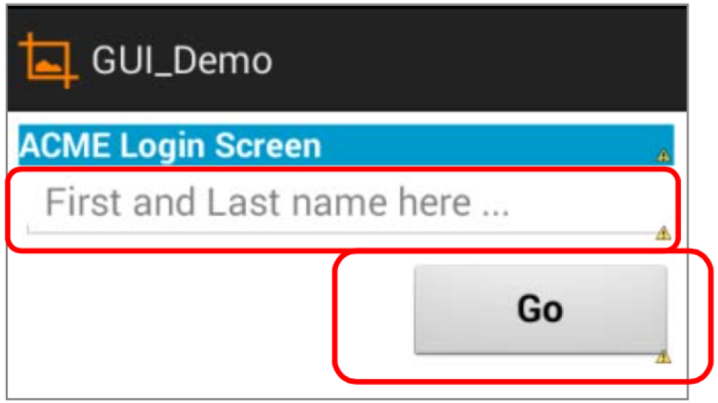




JAVA code

```
public class ...
{
  ...
  ...
}
```

Connecting Layouts to Java Code




```
<!- XML LAYOUT -->
<LinearLayout
     android:id="@+id/myLinearLayout"
     ... >

    <TextView
        android:text="ACME Login Screen"
        ...  />

    <EditText
        android:id="@+id/edtUserName"
        ... />

    <Button
        android:id="@+id/btnGo"
        ...  />

</LinearLayout>
```

Java code

```
package csu.matos.gui_demo;
import android…;

public class MainActivity extends Activity {

    EditText edtUserName;

    Button   btnGo;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);

     setContentView(R.layout.activity_main);

     edtUserName = (EditText) findViewById(R.id.edtUserName);

     btnGo = (Button) findViewById(R.id.btnGo);
     ...

   }

 ...
}
```

What is the meaning of an Android Context?

- On Android, a **Context** defines a logical **workspace** on which an app can load and access resources.
  1. For a simple '*one activity app*' -say MainActivity- the method **getApplicationContext()** and the reference **MainActivity.this** return the same result.
  2. An application could have **several activities**. Therefore, for a *multi- activity* app we have one app context, and a context for each of its activities, each good for accessing what is available in *that context.*

Connecting Layouts to Java Code

- Load and display a layout into an Activity

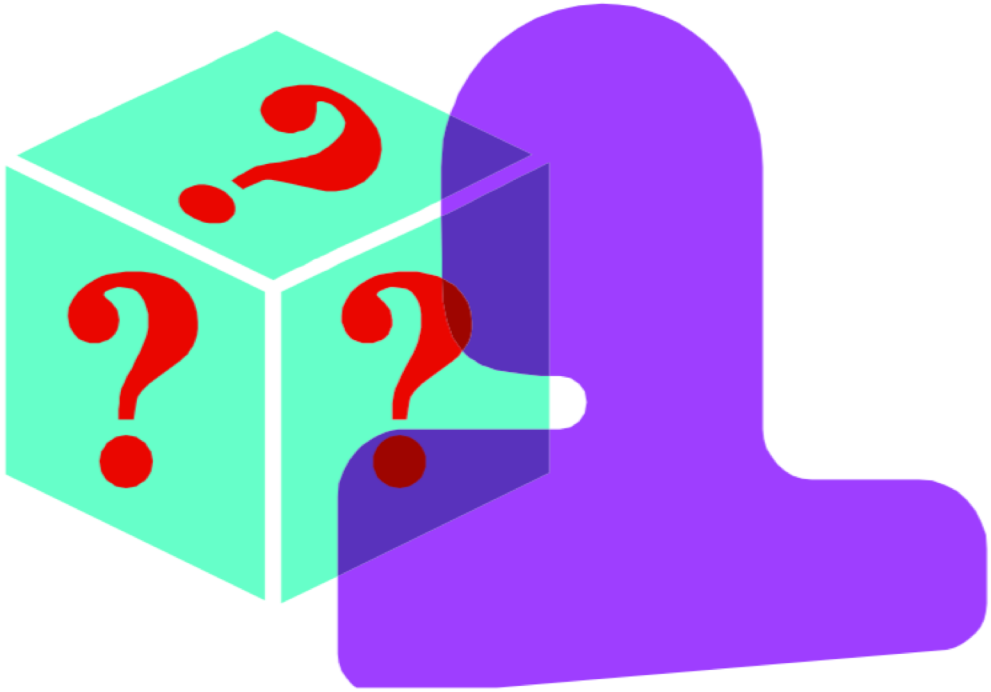
```
setContentView(R.layout.activity_main);
```

- Reference to a widget in a layout

```
Button btnGo= (Button) findViewById(R.id.btnGo);
```

- 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 (Use Eclipse's Package Explorer, look at your **/gen/package/R.java** contents).

Thanks for being here



Questions?

Homework 1

Using **TableLayout, LinearLayout** to design the GUI layout as the figure **(Using both XML & Java code)**

Homework 2

Using
TableLayout,
LinearLayout
to design the
GUI layout
as the figure

Html.fromHtml("°") -
http://www.ascii.cl/htmlcodes.htm

References

- Android User Interfaces, **Victor Matos,** Cleveland State University
- http://developer.android.com/training/basics/activity-lifecycle/index.html