Graphical User Interfaces

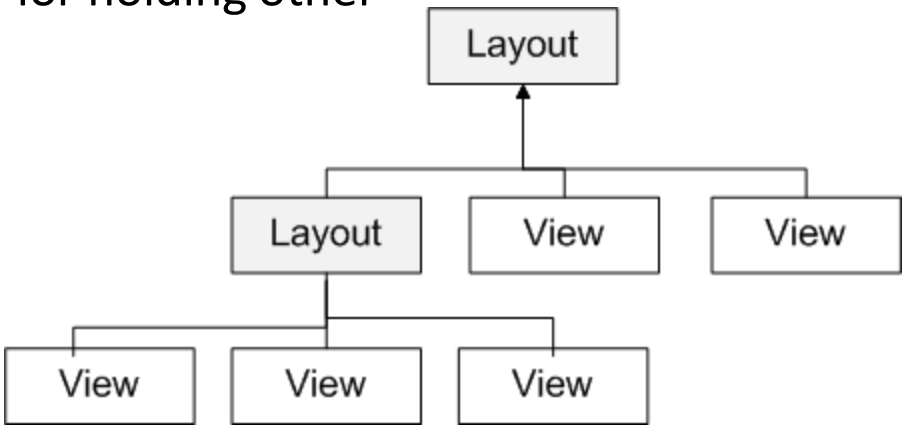
The View Class

- The **View class** is the Android's most basic UI component.

- 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

Text version: *activity_main.xml* file ⟶

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

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

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

</RelativeLayout>
```

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*.

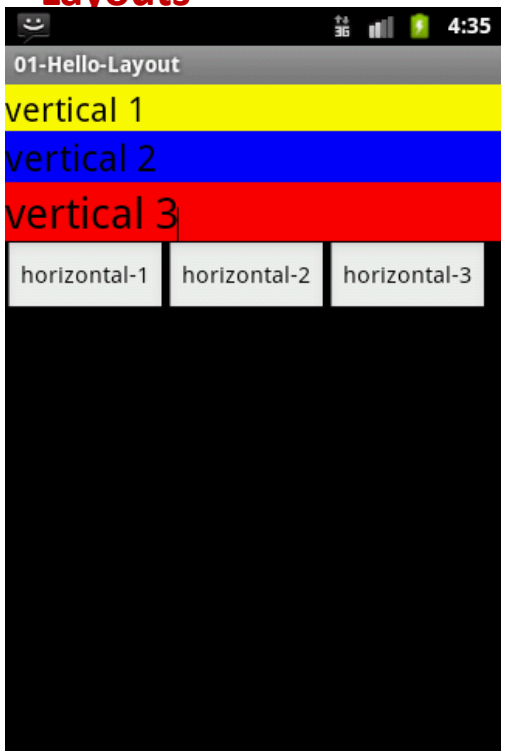
2. **Set up listeners:** For example, an image could be programmed to respond to various events such as: click, long-tap, mouse-over, etc.

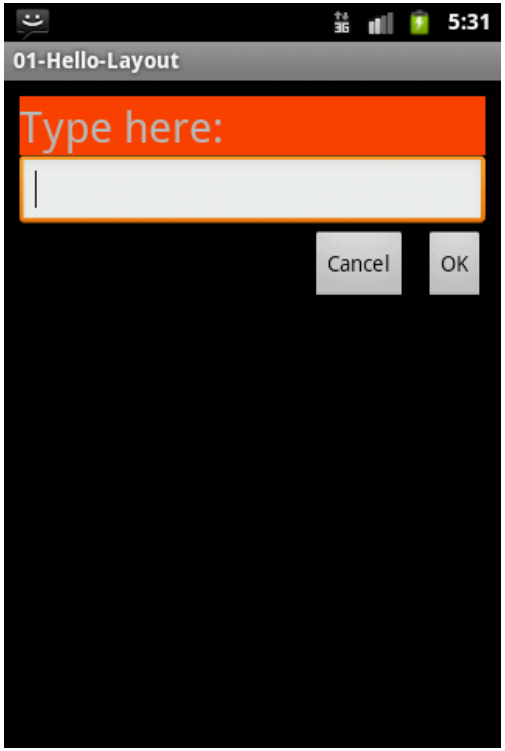
3. **Set focus:** To set focus on a specific view, you call the method requestFocus() or use XML tag <requestFocus />

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.

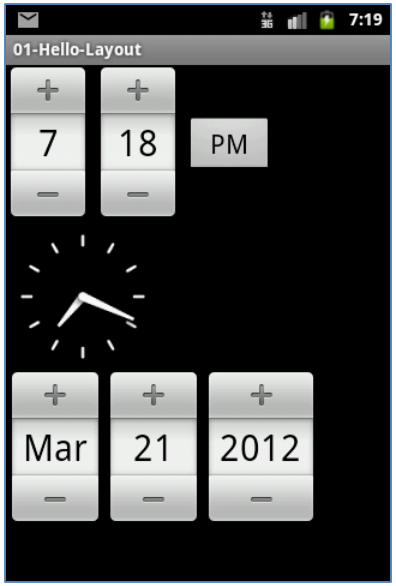
Relative Layout

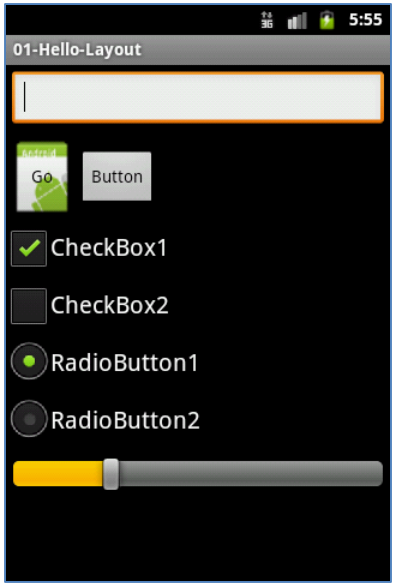
A RelativeLayout is a ViewGroup that allows you to position elements relative to each other.

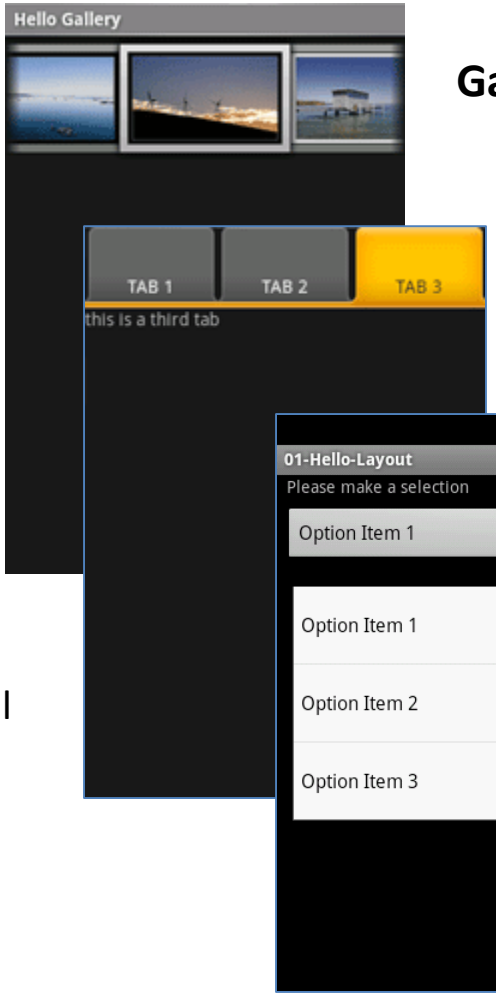
Reference: http://developer.android.com/guide/topics/ui/layout-objects.html

A brief sample of UI components

Widgets







GalleryView

TabWidget

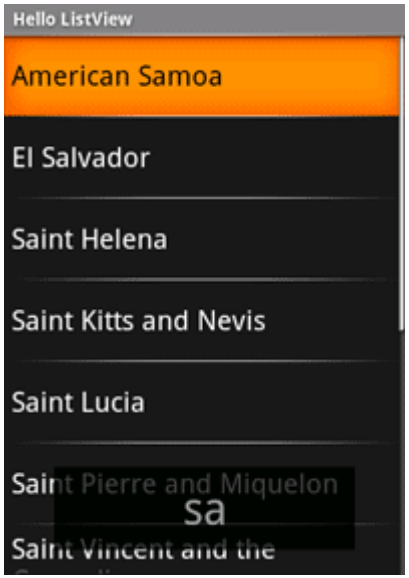
Spinner

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

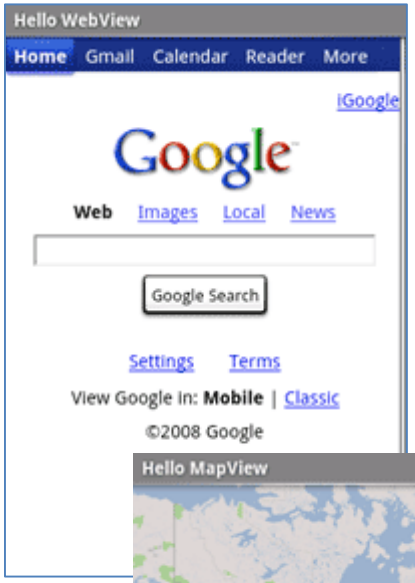
Form Controls
Includes a variety of typical form widgets, like:
image buttons,
text fields,
checkboxes and
radio buttons.

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

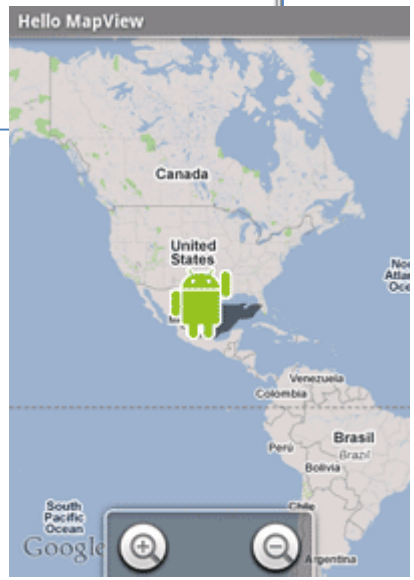
A brief sample of UI components



WebView



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

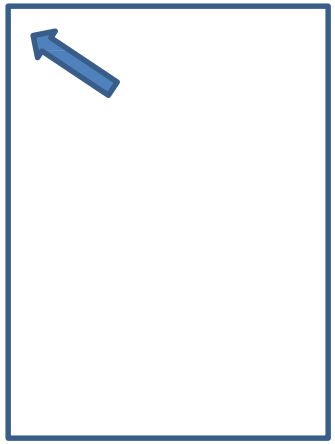
How to create Android GUIs?

- Android *Layouts* are GUI containers having a predefined structure and placement policy.

- **Layouts can be nested**, therefore a cell, row, or column of a given layout could be another layout.

Common Layouts

FrameLayout

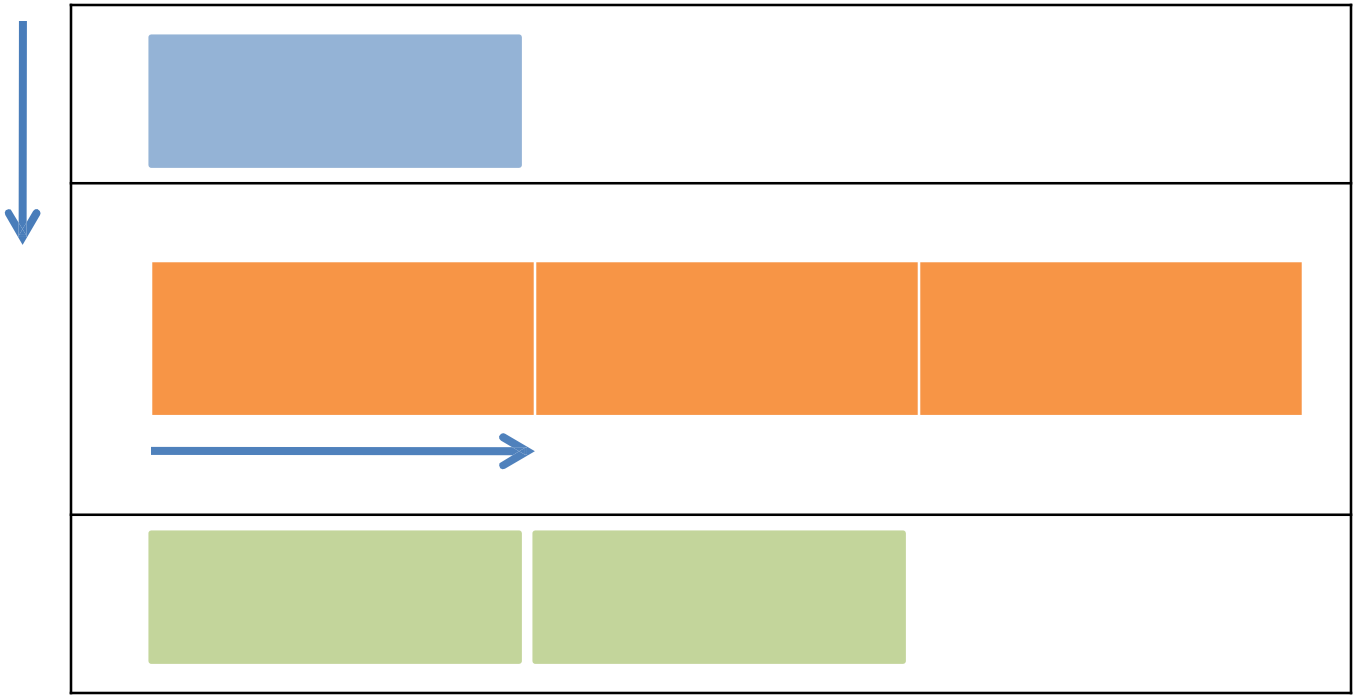


- FrameLayout is the simplest type of GUI container.
- Useful as outermost container holding a window.
- Allows you to define how much of the screen (high, width) is to be used.
- All its children elements are *aligned to the top left corner of the screen.*;

The Linear Layout

1. Linear Layout

- 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.



The Linear Layout

1. LinearLayout: Setting Attributes

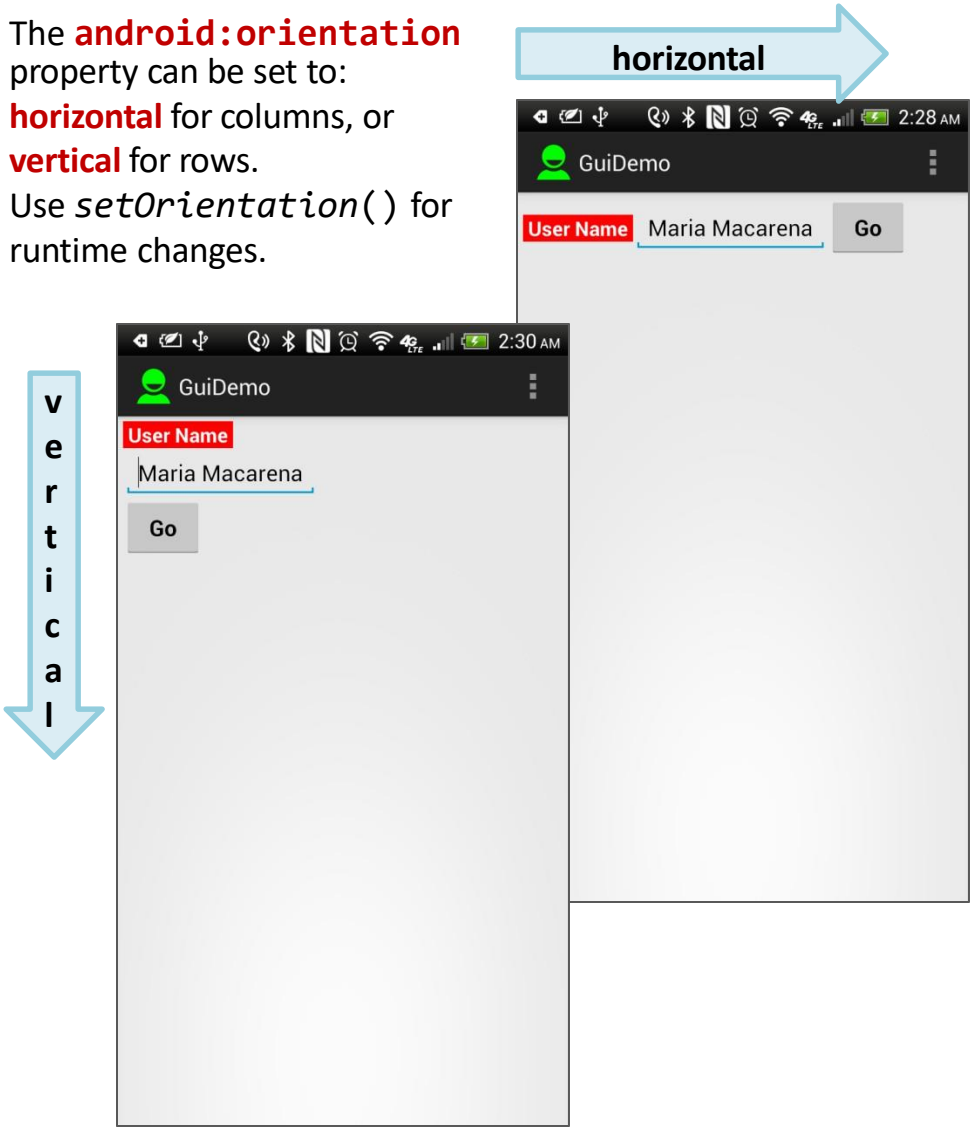
Configuring a **LinearLayout** usually requires you to set the following attributes:

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

The LinearLayout - Orientation

1.1 Attribute: Orientation

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

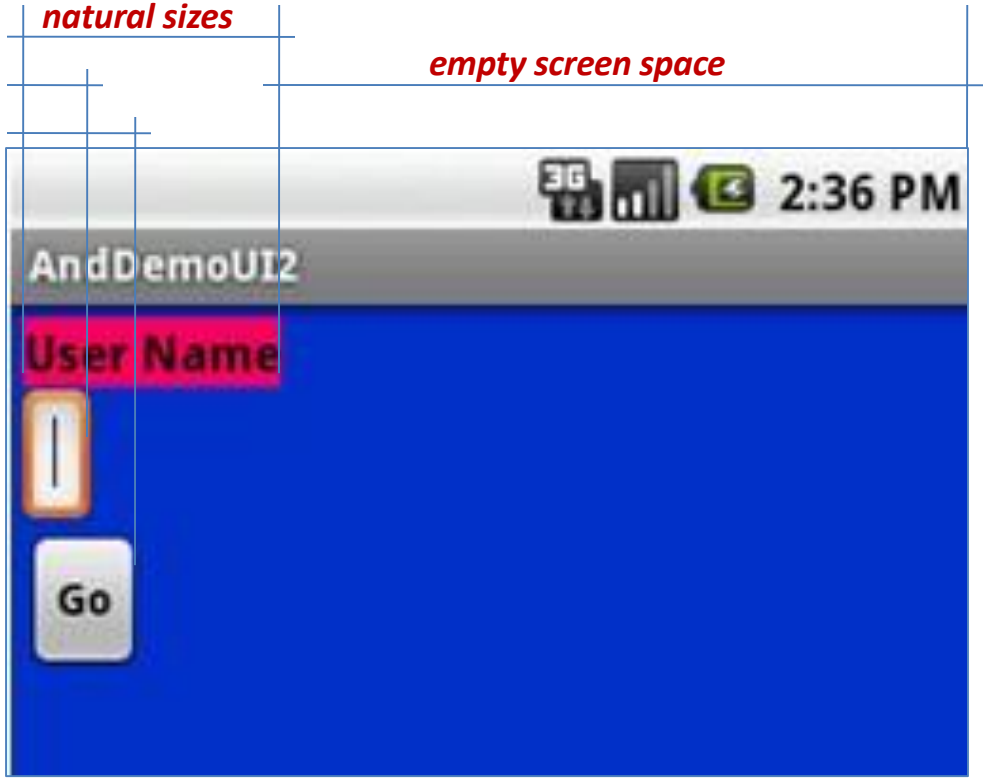



```xml
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myLinearLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:padding="4dp" >

    <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>
```

The LinearLayout – Fill Model

1.2 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).



The LinearLayout – Fill Model

1.2 Fill Model

All widgets inside a LinearLayout **must** include 'width' and 'height' attributes.

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

Values used in defining height and width can be:

1. A specific dimension such as **125dp** (device independent pixels, a.k.a. **dip**)

2. **wrap_content** indicates the widget should just fill up its natural space.

3. **match_parent** (previously called '**fill_parent**') indicates the widget wants to be as big as the enclosing parent.

The LinearLayout – Fill Model

1.2 Fill Model

Medium resolution is: 320 x 480 dpi.

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

    <TextView
        android:id="@+id/labelUserName"
        android:layout_width="match_pa
        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

The LinearLayout – Weight

1.2 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 this window is similar to the previous example.

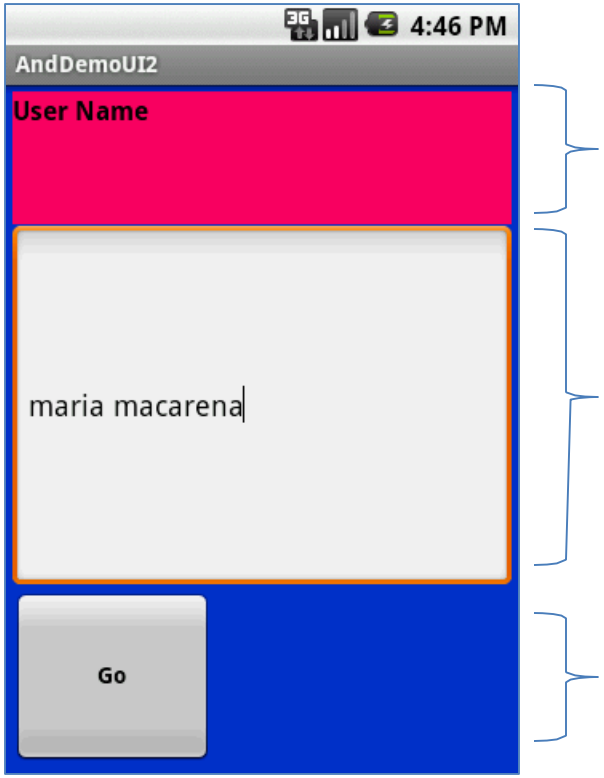
The TextView and Button controls have the additional property

```
android:layout_weight="1"
```

whereas the EditText control has

```
android:layout_weight="2"
```

Default value is 0

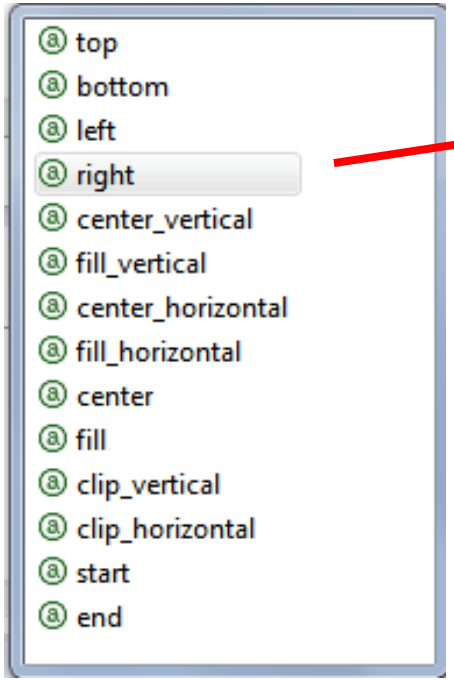


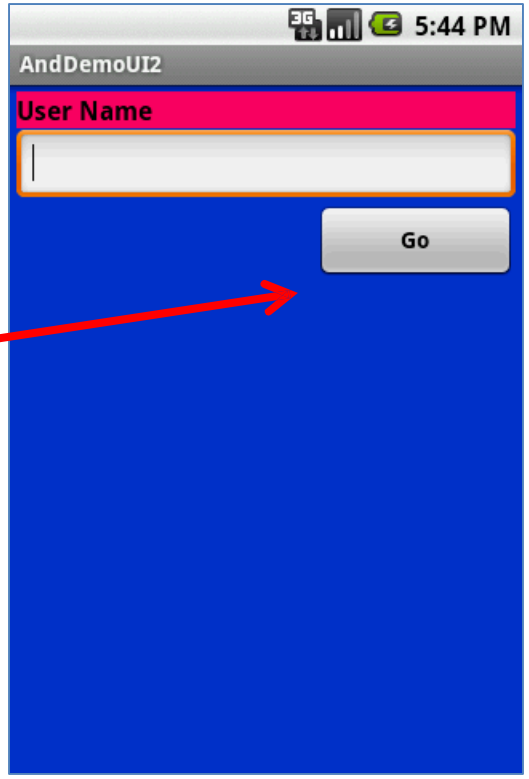
Takes: 2 /(1+1+2) of the screen space

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.





Button has **right** layout_gravity

The LinearLayout – Gravity

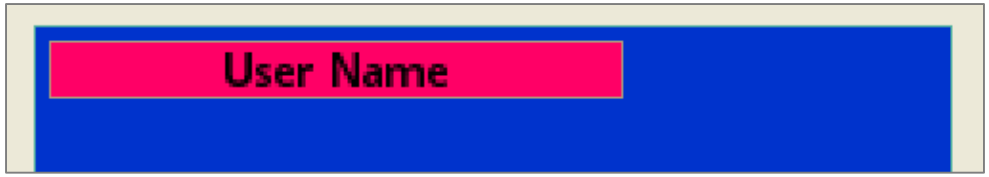
1.3 CAUTION: gravity vs. layout_gravity

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

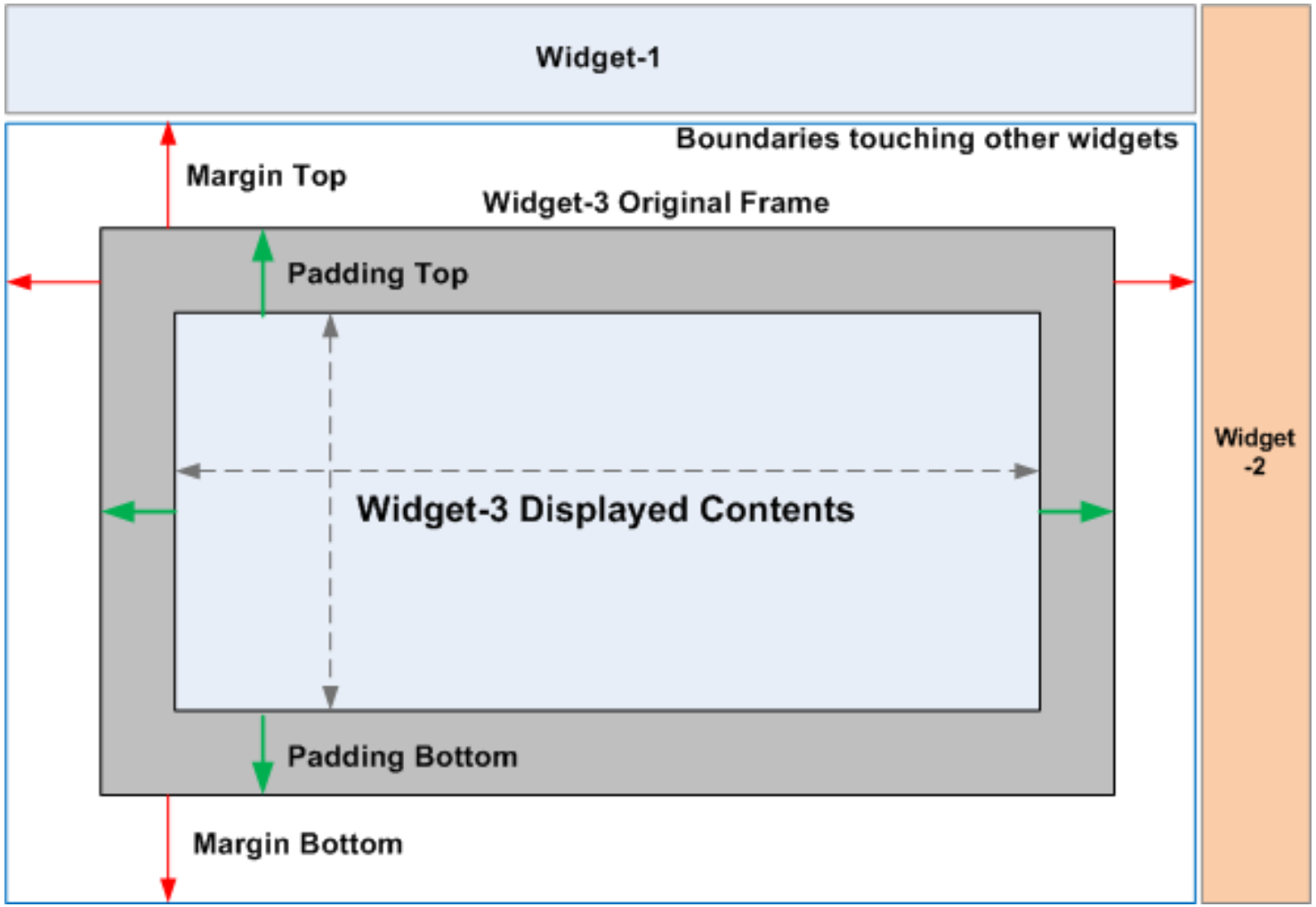
1.4 Padding

- The **padding** attribute specifies the widget's internal margin (in **dp** 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 LinearLayout – Padding

1.3 Padding and Margin

Padding and Margin represent the internal and external spacing between a widget and its included and surrounding context (respectively).

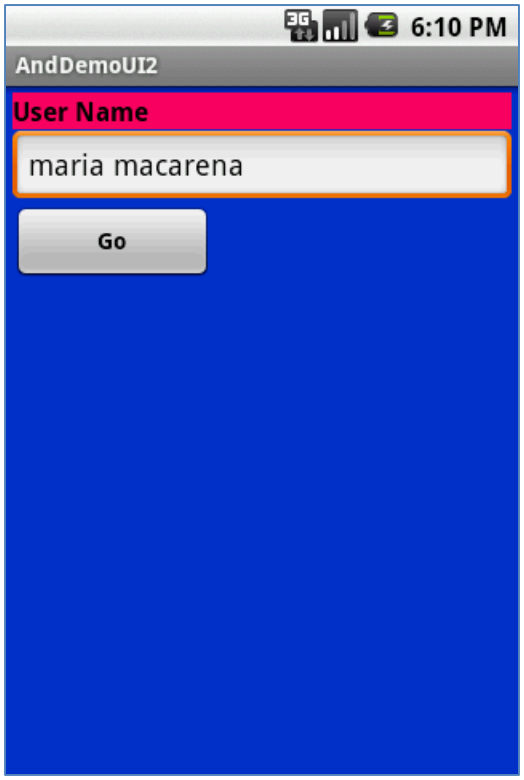


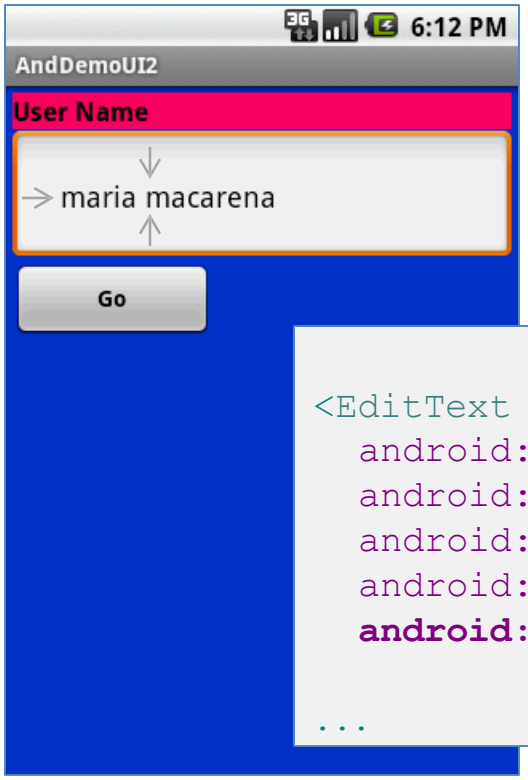
The LinearLayout – Padding

1.3 Internal Margins Using Padding

Example:

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



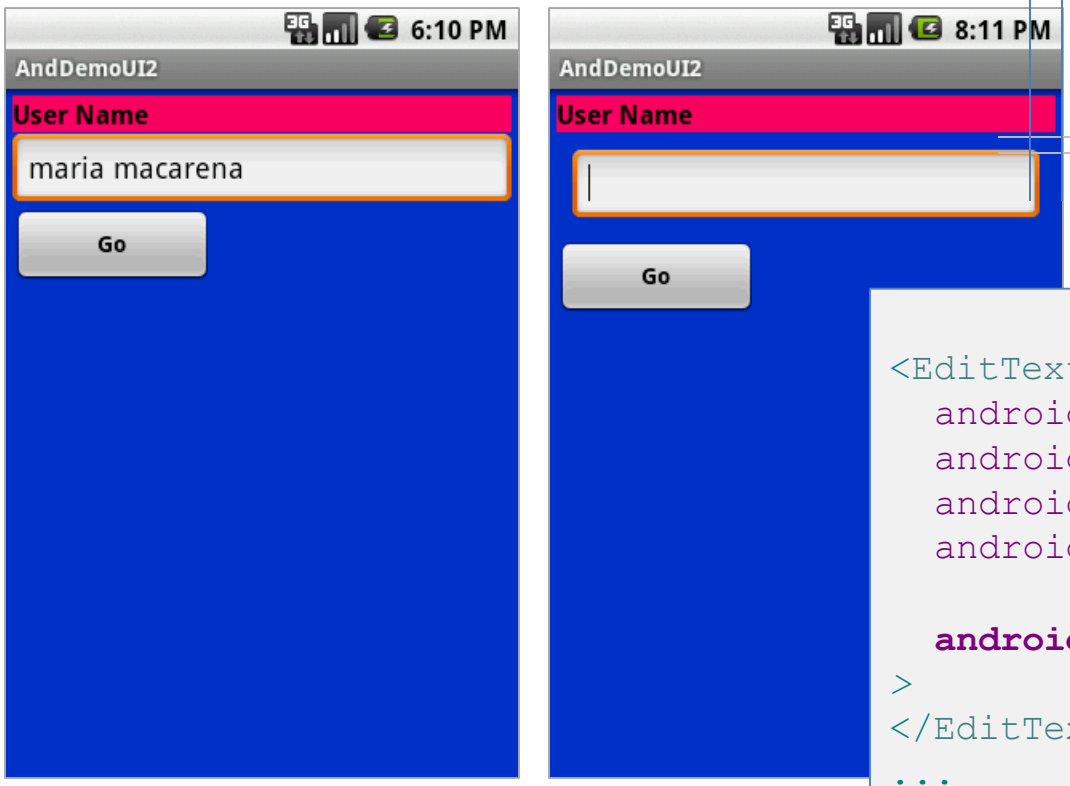



```
<EditText
  android:id="@+id/ediName"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:textSize="18sp"
  android:padding="30dp"   />

...
```

The LinearLayout – Margin

1.4 (External) Margin

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



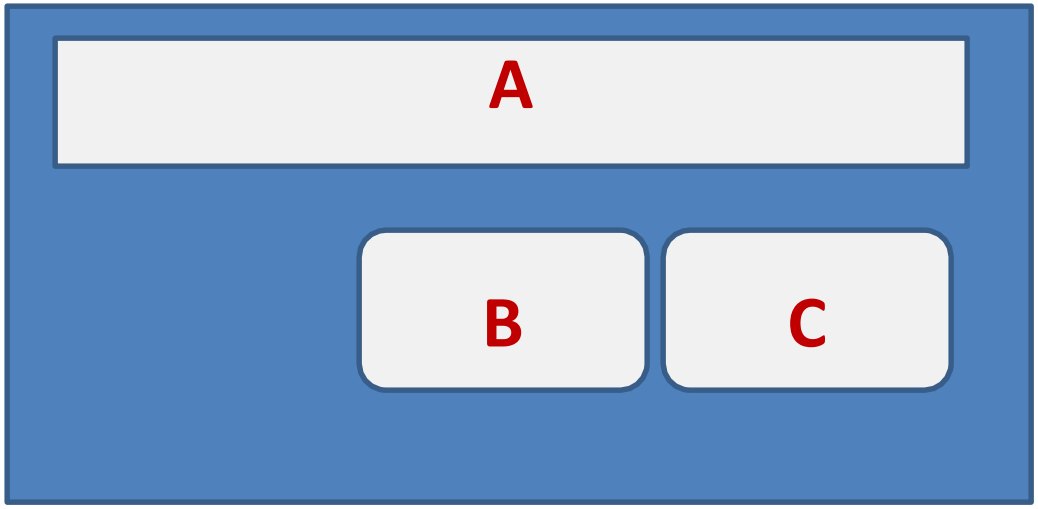
Increased inter-widget space

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

  android:layout_margin="6dp"
>
</EditText>
...
```

The Relative Layout

2. Relative Layout

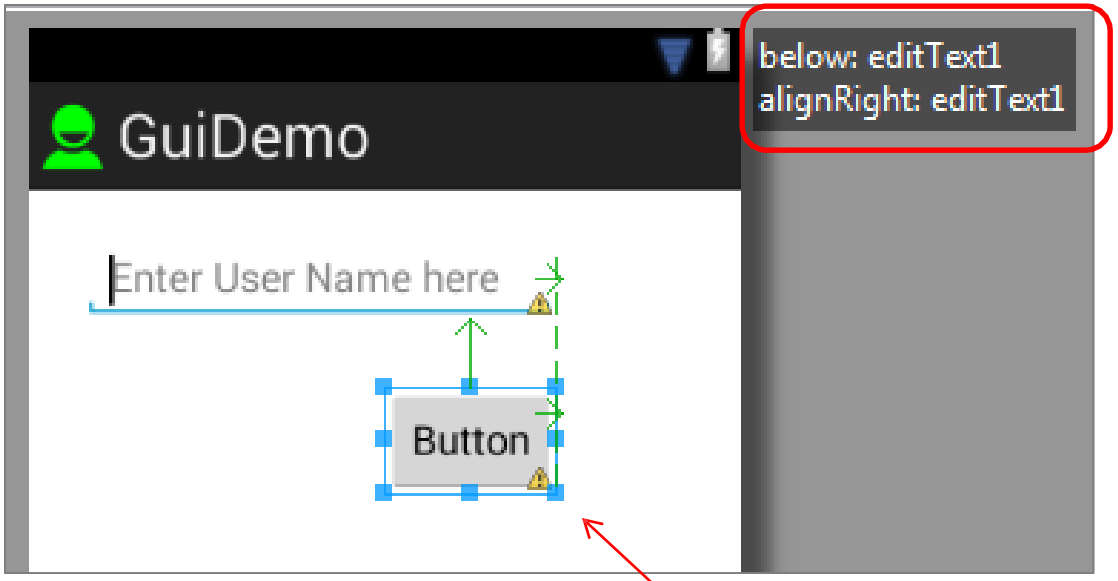
The placement of widgets in a **RelativeLayout** is based on their *positional relationship* to other widgets in the container and the parent container.



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. Example: Relative Layout



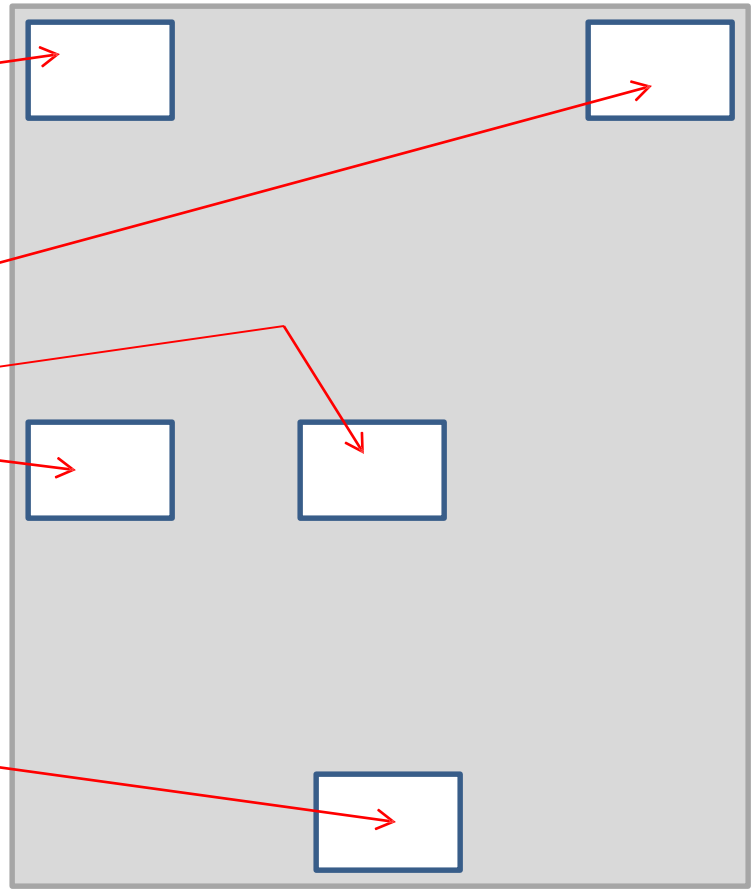
Location of the button is expressed in reference to its relative position with respect to the EditText box.

The Relative Layout

2. Referring to the container

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

```
android:layout_alignParentTop
android:layout_alignParentBottom

android:layout_alignParentLeft
android:layout_alignParentRight

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

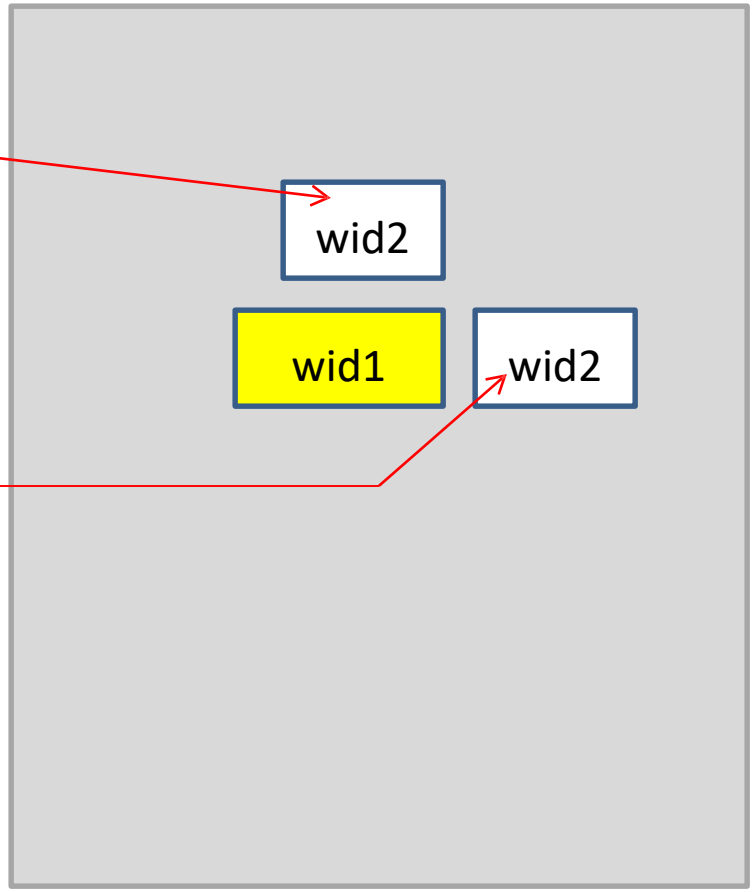


The Relative Layout

2. Referring to other widgets

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

```
android:layout_above="@+id/wid1"

android:layout_below

android:layout_toLeftOf

android:layout_toRightOf
```

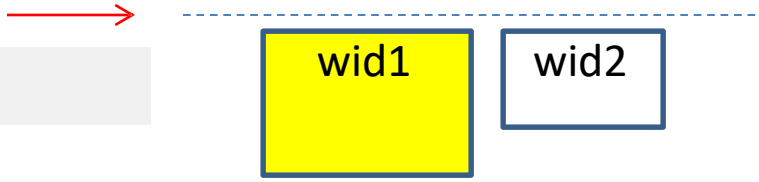


In this example widget “wid2” is map relative to wid1 (known as “@+id/wid1”)

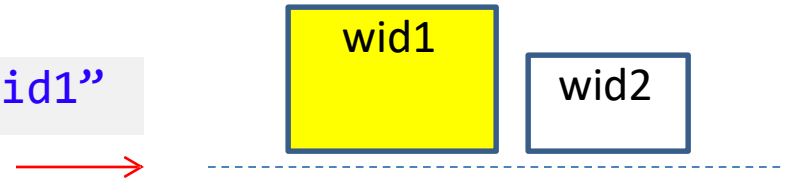
The Relative Layout

2. Referring to other widgets – cont.

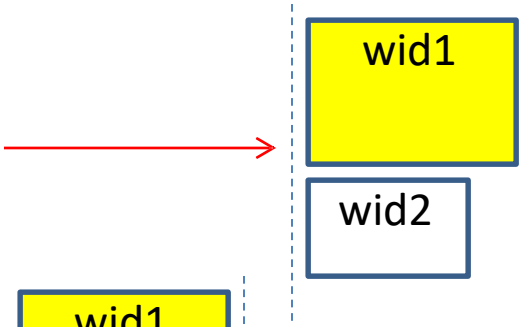
```
android:layout_alignTop="@+id/wid1"
```



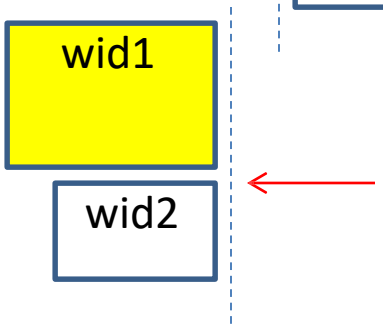

```
android:layout_alignBottom ="@+id/wid1"
```




```
android:layout_alignLeft="@+id/wid1"
```




```
android:layout_alignRight="@+id/wid1"
```



The Relative Layout

2. 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.

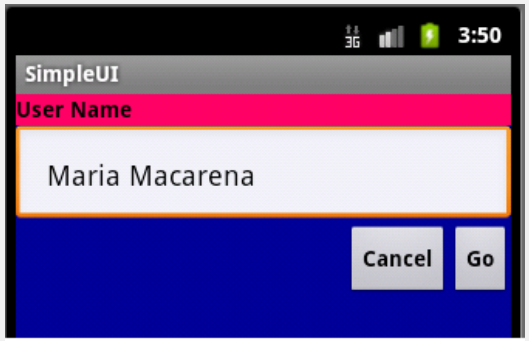
2. XML elements are named using the prefix: **`@+id/...`** For instance an EditText box could be called: `android:id="@+id/txtUserName"`

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"`*

The Relative Layout

2. 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>
```

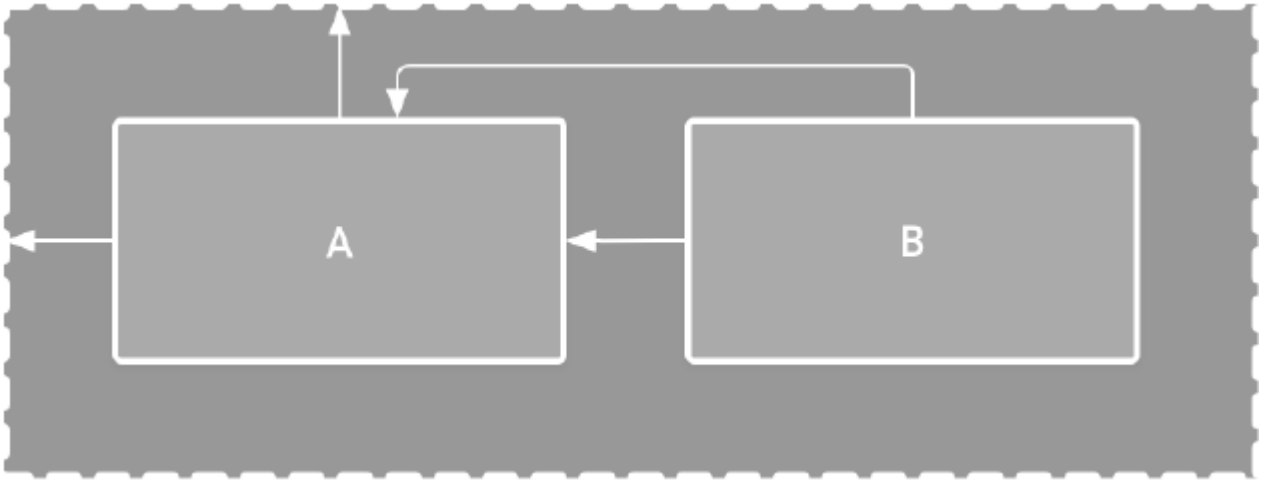



```
    <EditText
        android:id="@+id/txtUserName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/lblUserName"
        android:padding="20dp" >
    </EditText>

    <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>

    <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 Constraint Layout

2. Constraint Layout

- A layout that defines the position for each view based on constraints to sibling views and the parent layout
- Reduce layout nesting



The Constraint Layout

Constraint

- a connection
- between *View* and another *View*
- or to *View* and *parent* (ConstraintLayout)
- optional margin to create a gap
- … can be done with `RelativeLayout`

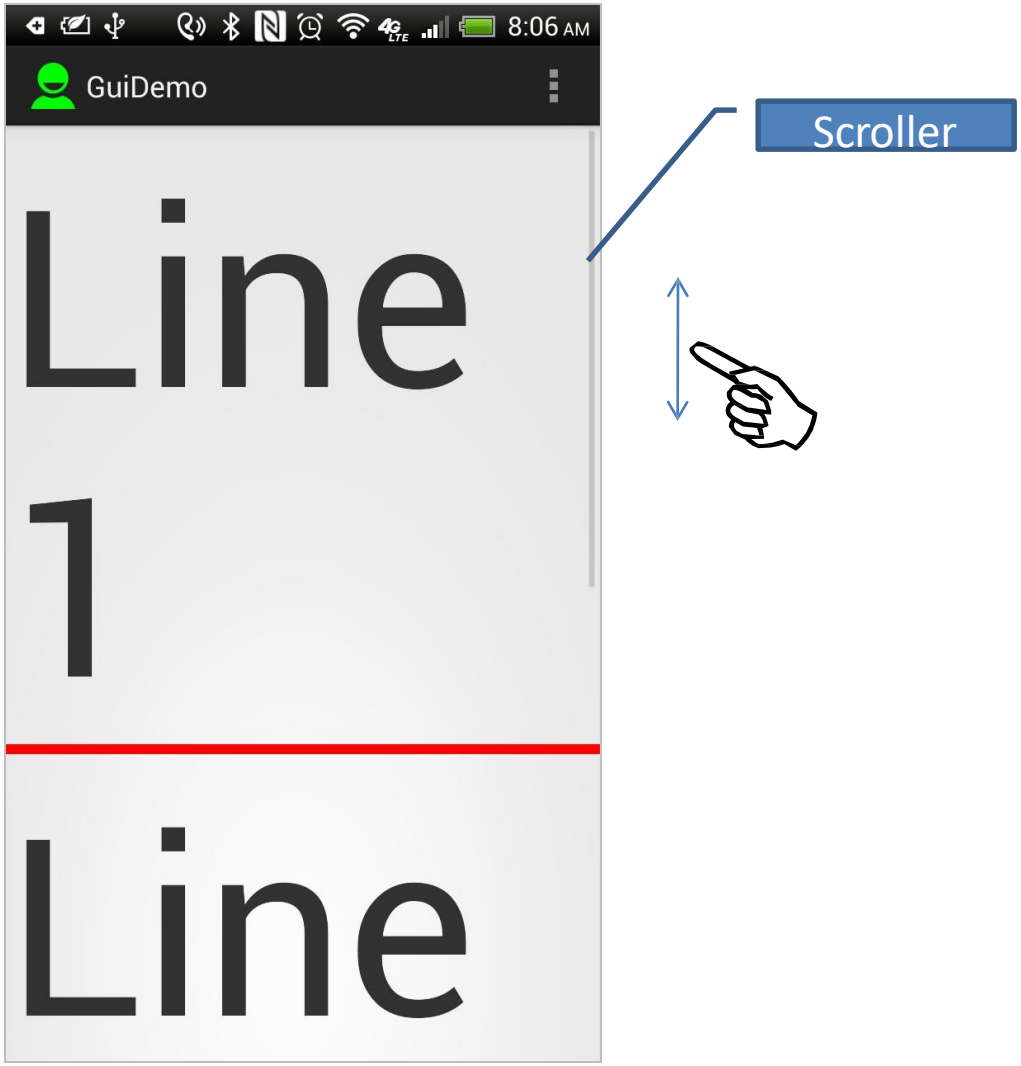
The Constraint Layout

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"  android:layout_height="match_parent"
  tools:context="pl.pelotasplus.constraintlayoutworkshop.MainActivity"
  tools:layout_editor_absoluteX="0dp"  tools:layout_editor_absoluteY="81dp">

  <TextView
     android:id="@+id/textView2"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:layout_marginLeft="50dp"
     android:layout_marginTop="50dp"
     android:text="First Name"
     android:textAppearance="@style/TextAppearance.AppCompat.Large"
     android:textSize="40sp"  app:layout_constraintLeft_toLeftOf="parent"
     app:layout_constraintTop_toTopOf="parent" />

  <TextView
     android:id="@+id/textView"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:layout_marginLeft="0dp"
     android:layout_marginTop="50dp"
     android:text="LastName"
     android:textAppearance="@style/TextAppearance.AppCompat.Large"
     android:textSize="40sp"
     app:layout_constraintLeft_toLeftOf="@+id/textView2"
     app:layout_constraintTop_toBottomOf="@+id/textView2" />

</android.support.constraint.ConstraintLayout>
```

Basic XML Layouts - Containers

3. ScrollView Layout

- 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 sliding access to the data.
- Only a portion of the user's data can be seen at one time, however the rest is available via scrolling.



Basic XML Layouts - Containers

3. 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="match_parent"
    android:layout_height="match_parent" >

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

        <TextView
            android:id="@+id/textView1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Line1"
            android:textSize="150dp" />
        <View
            android:layout_width="match_parent"
            android:layout_height="6dp"
            android:background="#ffff0000" />
        <TextView
            android:id="@+id/textView2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Line2"
            android:textSize="150dp" />

        <View
            android:layout_width="match_parent"
            android:layout_height="6dp"
            android:background="#ffff0000" />

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

</ScrollView>
```

Basic XML Layouts - Containers

4. 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. Absolute Layout (cont.)

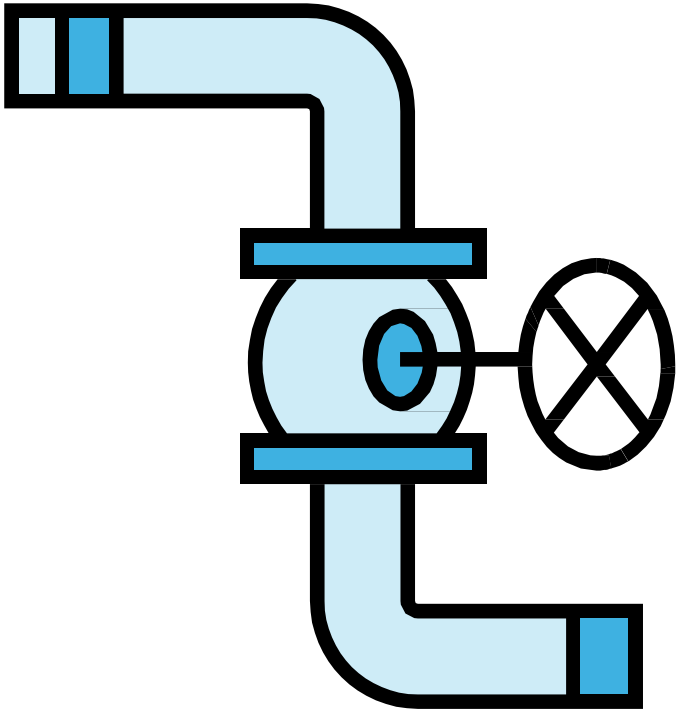
```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
android:id="@+id/myLinearLayout"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:background="#ff0033cc"
android:padding="4dp"
xmlns:android="http://schemas.android.com
/apk/res/android"
>

<TextView
android:id="@+id/tvUserName"
android:layout_width="match_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="0dp"
android:layout_y="10dp"
>
</TextView>
<EditText
android:id="@+id/etName"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:textSize="18sp"
android:layout_x="0dp"
android:layout_y="38dp"
>
</EditText>

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


Button location

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.

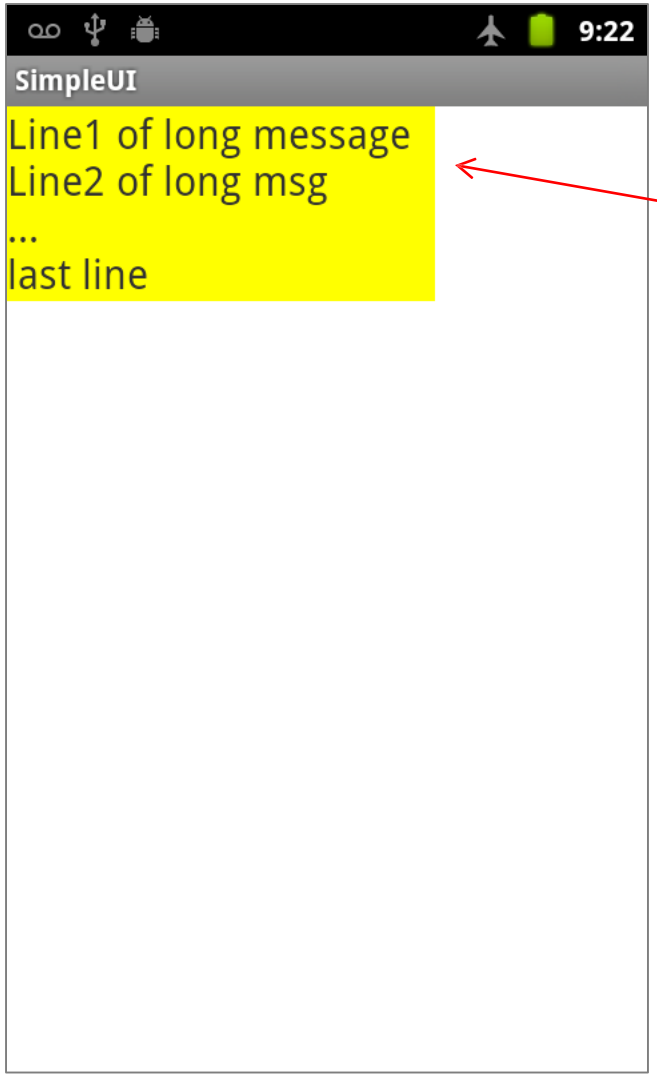
Attaching Layouts to Java Code

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());
}
```

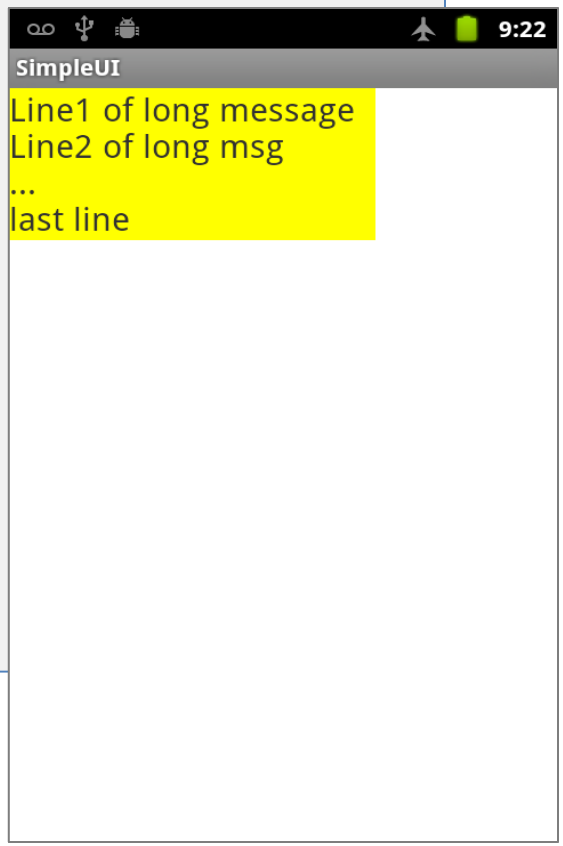
Basic Widgets: Labels



- 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="match_parent"
    android:layout_height="match_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>
```



Hint on Better Programming Style:

Add to the **res/values/strings.xml** the entry

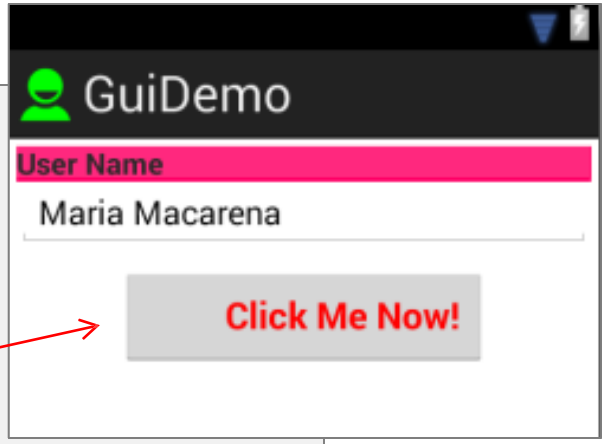
<string name="*long_msg_1">Line1 of long message\nLine2 of long msg\n...\nlast line*</string>

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="300dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="5dp"
        android:gravity="right"
        android:padding="5dp"
        android:text="@string/button1_caption"
        android:textColor="#ffff0000"
        android:textSize="20sp"
        android:textStyle="bold" />
```



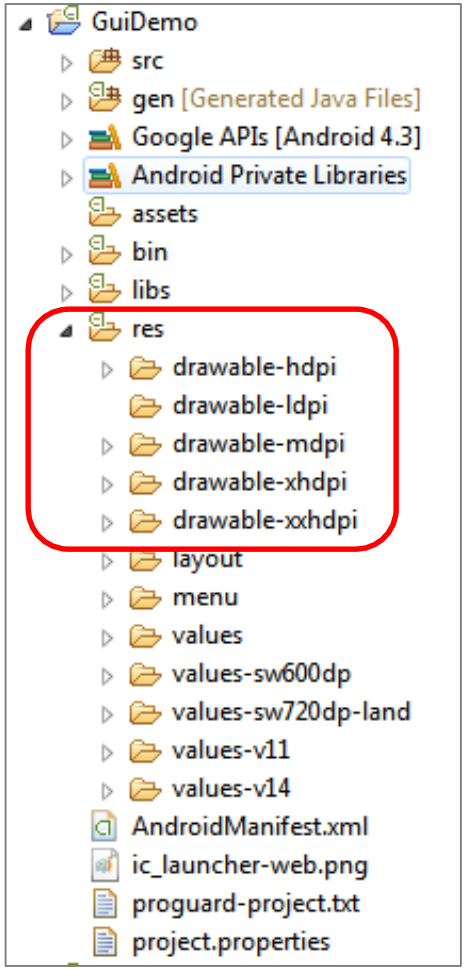
Basic Widgets: Images

- **ImageView** and **ImageButton** are two Android widgets that allow embedding of images in your applications.

- Analogue to *TextView* and *Button* controls (respectively).

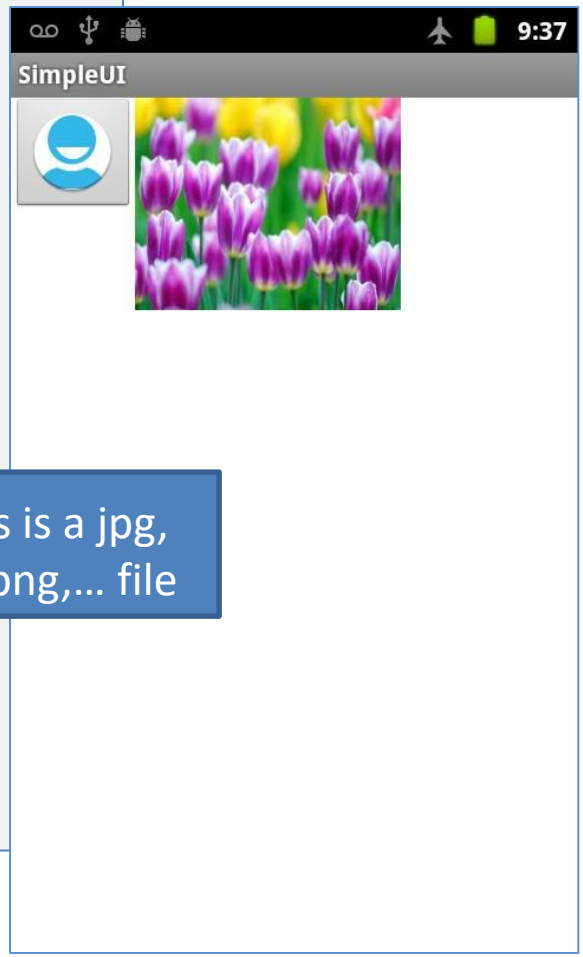
- Each widget takes an `android:src` or `android:background` attribute (in an XML layout) to specify what picture to use.

- Pictures are usually stored in the **res/drawable** folder (optionally a low, medium, and high definition version of the same image could be stored to later be used with different types of screens)



Basic Widgets: Images

```
<LinearLayout
    . . .

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

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

</LinearLayout>
```

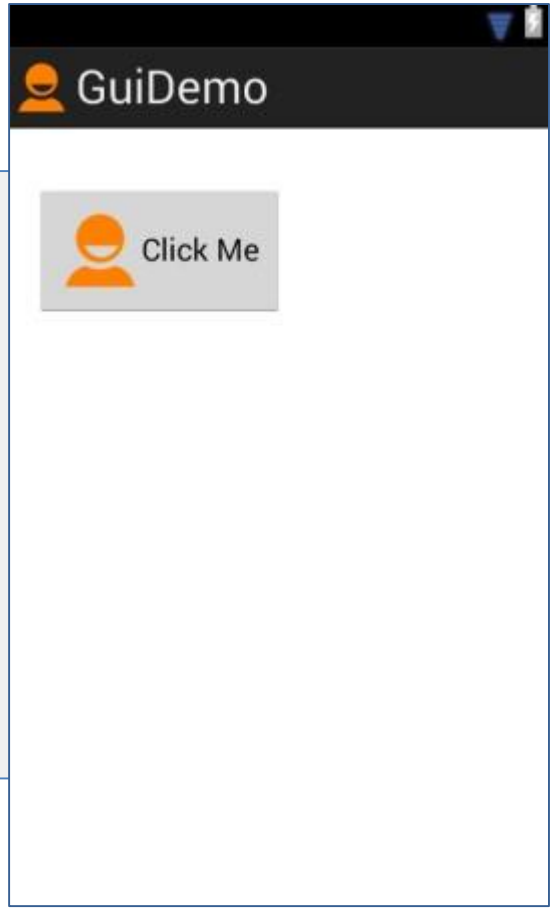


This is a jpg, gif, png,… file

Basic Widgets: Combining Images & Text

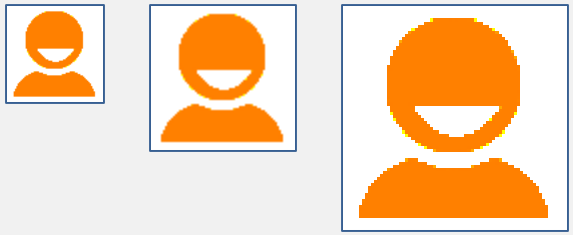
A common **Button** could display text and a simple image as shown below

```
<LinearLayout
    . . .

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


</LinearLayout>
```



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 to 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 user's input.

- The control can display *editable* text (uses HTML-styles: bold, ...).

- Important Java methods are:

```
txtBox.setText("someValue")
```

and

```
txtBox.getText().toString()
```

```
EditText txtBox = (EditText)
findViewById(R.id.myedittext1);
```

Basic Widgets: EditText

- The **EditText** (or *textBox*) widget is an extension of *TextView* that allows user's input.

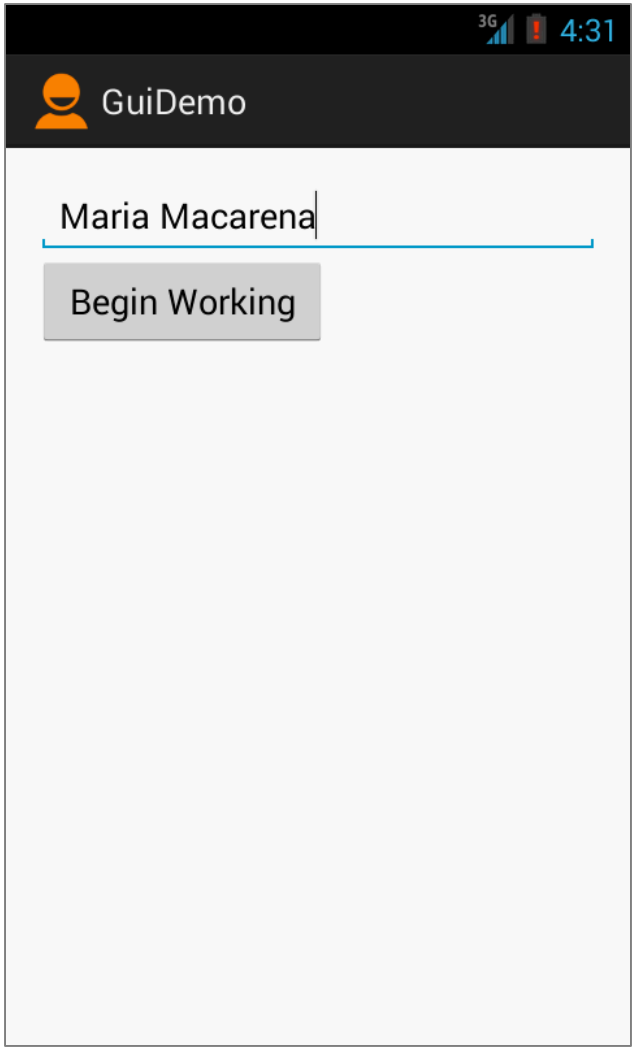
- Important Java I/O methods are:

  `txtBox.setText("someValue")`

  and

  `txtBox.getText().toString()`

- The control can display *editable* or *HTML-formatted* text by means of `Html.fromHtml(text)`



Basic Widgets: EditText

CAUTION: Deprecated Methods

DEPRECATED

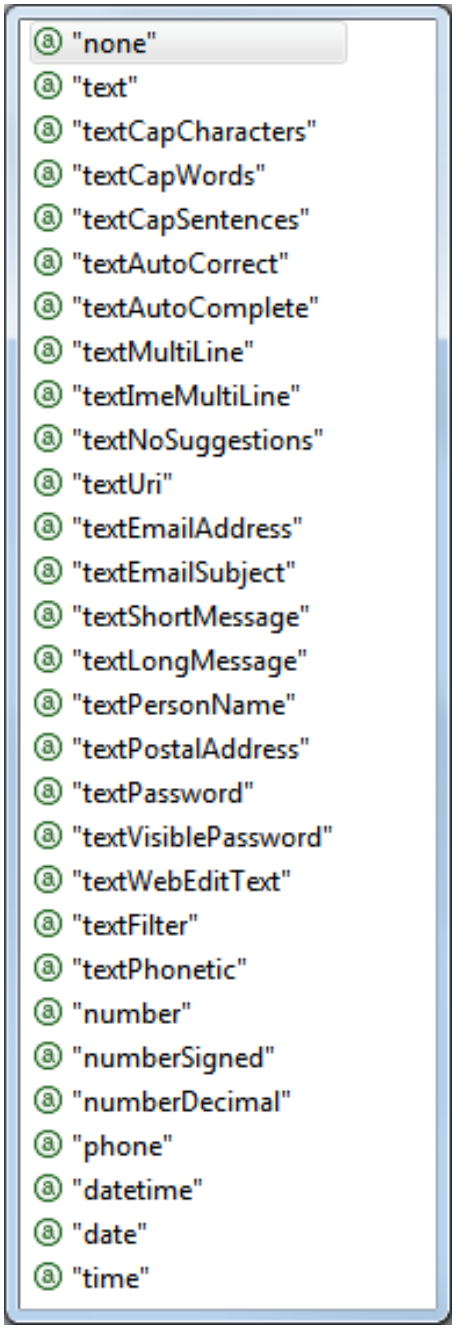
- `android:autoText`
- `android:capitalize`
- `android:digits`
- `android:singleLine`
- `android:password`
- `android:numeric`
- `android:phonenumber`

Instead use the newer atttribute:

NEW

`android:inputType="…choices…"`

where choices include



Basic Widgets: EditViews

Example

```
...

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

  android:inputType="textCapWords|textAutoCorrect"


  android:hint="@string/enter_your_first_and_last_name"

  android:textSize="18sp" />

...
```

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

Each word is capitalized

Suggestion (grey out)

Example 1: Login Screen

In this example we will create and use a simple login screen holding a label(**TexView**), a textBox (**EditText**), and a **Button**. A fragment of its functionality is shown below.

Example 1: Login Screen

Layout Design 1 of 2

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#886495ed"
    android:orientation="vertical"
    android:padding="2dp" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="1dp"
        android:background="#ffffff00"
        android:text="@string/ACME_Corp_Caption" />

    <EditText
        android:id="@+id/txtUserName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="1dp"
        android:hint="@string/Enter_your_First_and_Last_name"
        android:inputType="textCapWords|textAutoCorrect"
        android:textSize="18sp" >

        <requestFocus />
    </EditText>
```

Example 1: Login Screen

Layout Design 2of 2

```
    <Button
        android:id="@+id/button1"
        android:layout_width="82dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="1dp"
        android:text="@string/login" />

</LinearLayout>
```

Resource Captions: res/values/strings

```
<?xml version="1.0" encoding="utf-8"?>
<!-- this is the res/values/strings.xml file -->
<resources>

    <string name="app_name">GuiDemo</string>
    <string name="action_settings">Settings</string>
    <string name="login">login</string>
    <string name="ACME_Corp_Caption">login</string>
    <string name="Enter_your_First_and_Last_name">Enter your First and Last name</string>

</resources>
```

Example 1: Login Screen

MainActivity.java Class (1 of 2)

```
package csu.matos.guidemo;
import ...
// "LOGIN" - a gentle introduction to UI controls

public class MainActivity extends Activity {

    //class variables representing UI controls to be controlled from the program
    TextView labelUserName;
    EditText txtUserName;
    Button btnBegin;

    //variables used with the Toast message class
    private Context context;
    private int duration = Toast.LENGTH_SHORT;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //show the login screen
        setContentView(R.layout.activity_main);
        context = getApplicationContext();
```

Example 1: Login Screen

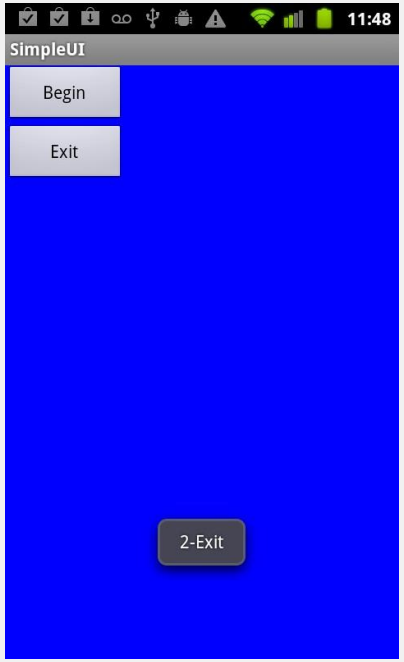
MainActivity.java Class (2 of 2)

```
        //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);

        //LISTENER: allowing the button widget to react to user interaction
        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(context,
                         "Bienvenido " + userName,
                         duration).show();
              }
              Toast.makeText(context,
                     userName + "  is not a valid USER" ,
                     duration).show();
          }
               });// onClick

    }//onCreate

}//class
```

Example 2: Wiring Multiple Button Widgets

Note: The example below shows an alternative way of defining a single Listener for multiple buttons.

```
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
```

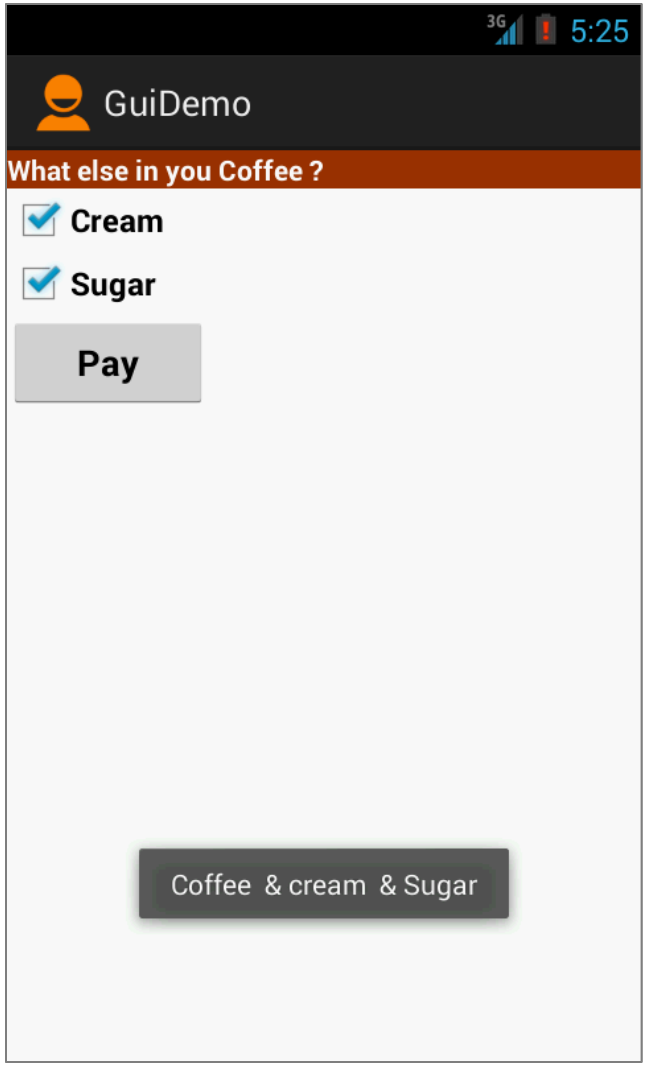


Basic Widgets: CheckBox

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

The screen displays two CheckBox controls for selecting 'Cream' and 'Sugar' options. In this image both boxes are 'checked'.

When the user pushes the 'Pay' button a Toast-message is issue telling what is the current combination of choices held by the checkboxes.



Example 3: CheckBox

The following Coffee-App shows us how to use CheckBoxes.

Layout 1 of 2

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="5dp"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/labelCoffee"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ff993300"
        android:text="@string/coffee_addons"
        android:textColor="@android:color/white"
        android:textStyle="bold" />

    <CheckBox
        android:id="@+id/chkCream"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cream"
        android:textStyle="bold" />
```

Example 3: CheckBox

Coffee-App

Layout 2 of 2

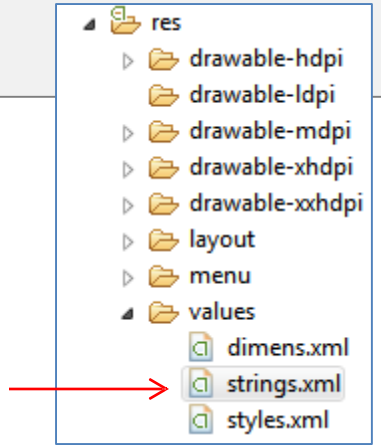
```
<CheckBox
        android:id="@+id/chkSugar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/sugar"
        android:textStyle="bold" />

    <Button
        android:id="@+id/btnPay"
        android:layout_width="153dp"
        android:layout_height="wrap_content"
        android:text="@string/pay"
        android:textStyle="bold" />

</LinearLayout>
```

Example 3: CheckBox

Coffee-App

Resources: res/values/strings

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

    <string name="app_name">GuiDemo</string>
    <string name="action_settings">Settings</string>

    <string name="click_me">Click Me</string>
    <string name="sugar">Sugar</string>
    <string name="cream">Cream</string>
    <string name="coffee_addons">What else do you like in your coffee?</string>
    <string name="pay">Pay</string>
</resources>
```



Example 3: CheckBox

Java Code – 1 of 2

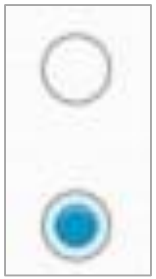
```
public class MainActivity extends Activity {
    CheckBox chkCream;
    CheckBox chkSugar;
    Button btnPay;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_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 3: 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.

Example 4: RadioButtons

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="match_parent"
      android:layout_height="wrap_content"
      android:background="#ff993300"
      android:text="@string/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="@string/decaf" />

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

      <RadioButton
          android:id="@+id/radColombian"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:checked="true"
          android:text="@string/colombian" />
  </RadioGroup>
```

Example 4: 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);
```

Example 4: 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
```

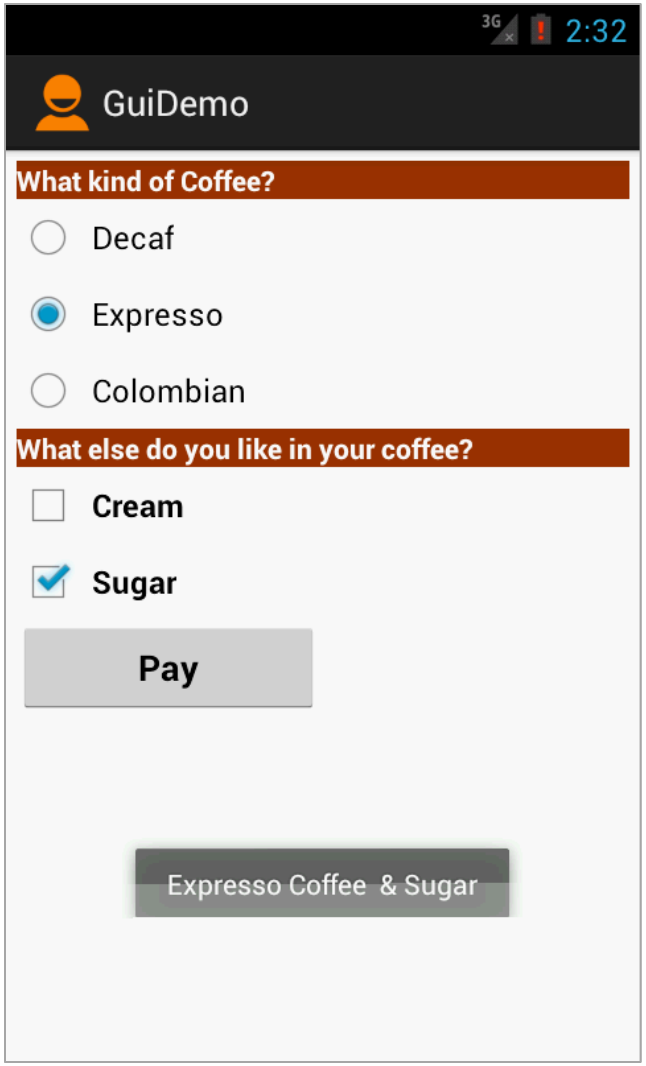
Example 4: RadioButtons

Example

This UI uses
RadioButtons
and
CheckBoxes
to define choices

RadioGroup

Summary of choices



Miscellaneous:
UI Attributes & Java Methods

XML Controls the focus sequence:

```
android:visibility
android:background
<requestFocus />
```

Java methods

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

User Interfaces

This image was made using the *Device Frame Generator*, which is part of the **Android Asset Studio tool**

Appendix B:

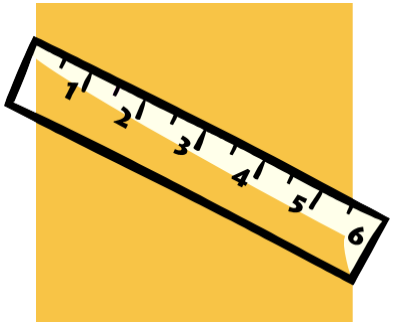
Android Asset Studio

LINK: http://android-ui-utils.googlecode.com/hg/asset-studio/dist/index.html

This tool offers a number of options to craft high-quality icons and other displayed elements typically found in Android apps.

Icon Generators	Other Generators	Community Tools
Launcher icons Action bar and tab icons Notification icons Navigation drawer indicator Generic icons	Device frame generator Simple nine-patch gen.	Android Action Bar Style Generator Android Holo Colors Generator

Appendix C: Measuring Graphic Elements



Q. What is **dpi** (also know as **ppi**) ?

Stands for *dots per inch*. It suggests a measure of screen quality.

You can compute it using the following formula:

$$dpi = \sqrt{widthPixels^2 + heightPixels^2} \;/\; diagonalInches$$

G1 (base device 320x480)	155.92 dpi	(3.7 in diagonally)
Nexus (480x800)	252.15 dpi	
HTC One (1080x1920)	468 dpi	(4.7 in)
Samsung S4 (1080x1920)	441 dpi	(5.5 in)

Q. What is the difference between **dp, dip** and **sp** units in Android?

dp (also known as **dip)** *Density-independent Pixels* – is an abstract unit based on the physical density of the screen. These units are relative to a 160 dpi screen, so one dp is one pixel (dp) on a 160 dpi screen. Use it for measuring anything but fonts – DO NOT USE dp, in. mm

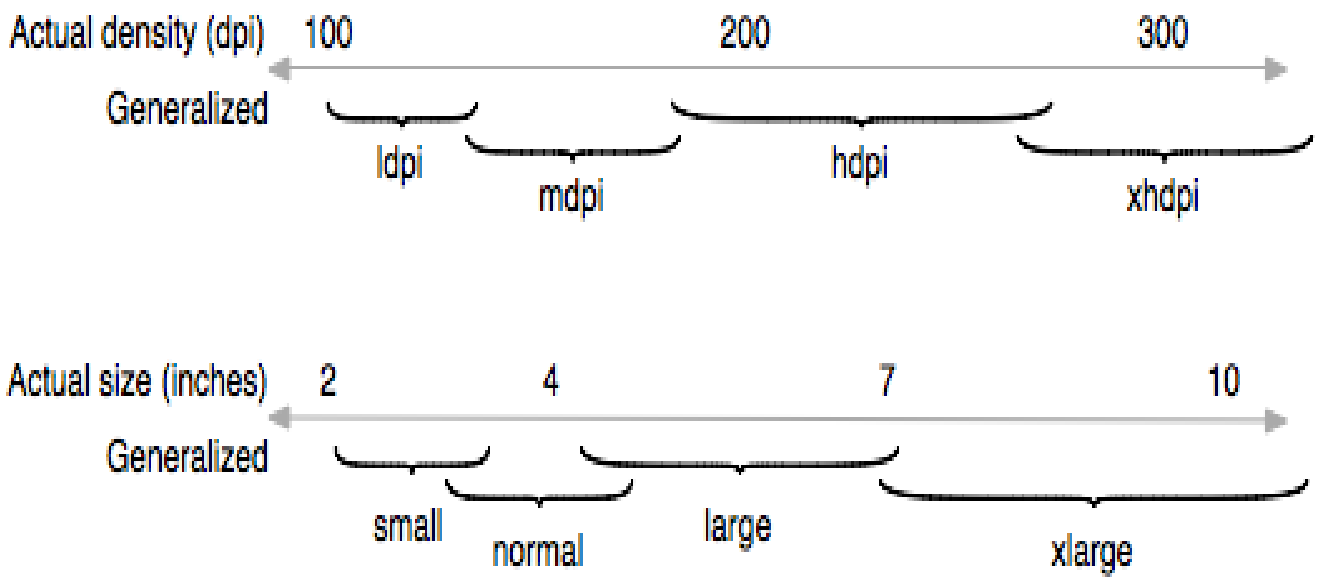
sp

Scale-independent Pixels – similar to the relative density dp unit, but used for font size preference.

Appendix C: Measuring Graphic Elements

Q. How Android deals with screen resolutions?

Illustration of how the Android platform maps actual screen densities and sizes to generalized density and size configurations.



Appendix C: Measuring Graphic Elements

Q. What do I gain by using screen densities?

More homogeneous results as shown below

Examples of density independence on WVGA high density (left), HVGA medium density (center), and QVGA low density (right).

Q. How to set different density/size screens in my application?

The following manifest's code declares support for small, normal, large, and xlarge screens in any density.

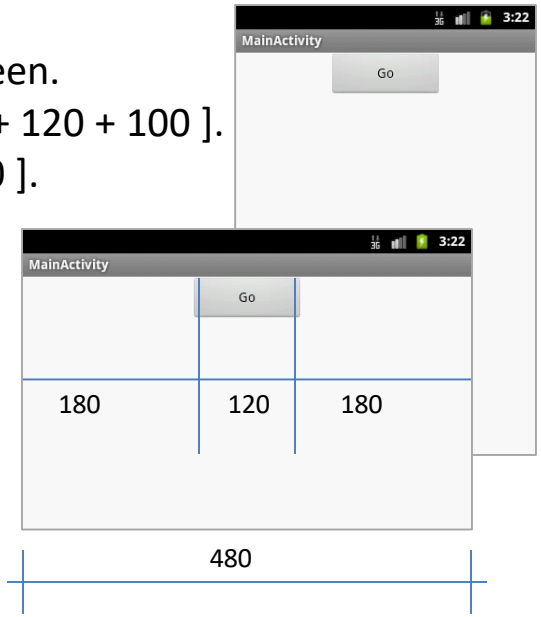
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    <supports-screens
        android:anyDensity="true"
        android:largeScreens="true"
        android:normalScreens="true"
        android:smallScreens="true"
        android:xlargeScreens="true" />
    ...
</manifest>
```

Appendix C: Measuring Graphic Elements

Q. Give me an example on how to use dp units.

Assume you design your interface for a G1 phone having 320x480 pixels (Abstracted LCD density is **160** – See your AVD entry the actual pixeling is a: 2***160** x 3***160**)

Assume you want a 120dp button to be placed in the middle of the screen.
On portrait mode you could allocate the 320 horizontal pixels as [100 + 120 + 100].
On Landscape mode you could allocate 480 pixels as [180 + 120 + 180].



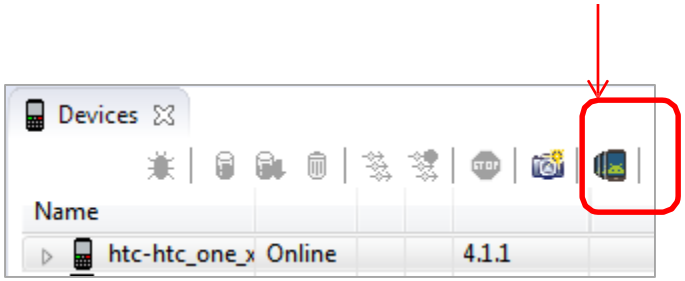
The XML would be

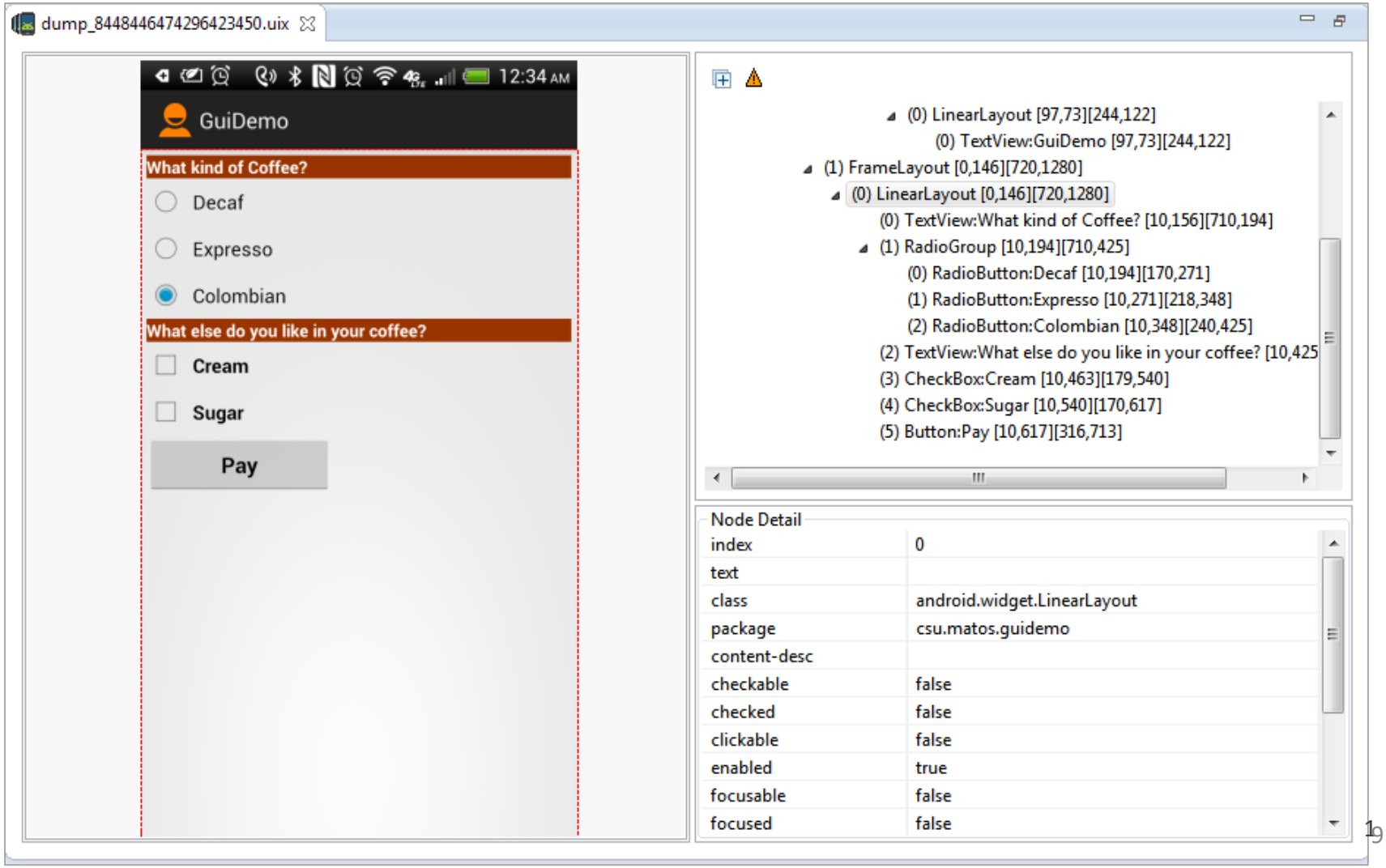
```
<Button
    android:id="@+id/button1"
    android:layout_height="wrap_content"
    android:layout_width="120dp"
    android:layout_gravity="center"
    android:text="@+id/go_caption" />
```

If the application is deployed on devices having a higher resolution the button is still mapped to the middle of the screen.

Appendix D: Hierarchy Viewer Tools

The HierarchyViewer Tool allows exploration of a displayed UI. Use **DDMS** > Click on Devices > Click on HierarchyViewer (next to camera)





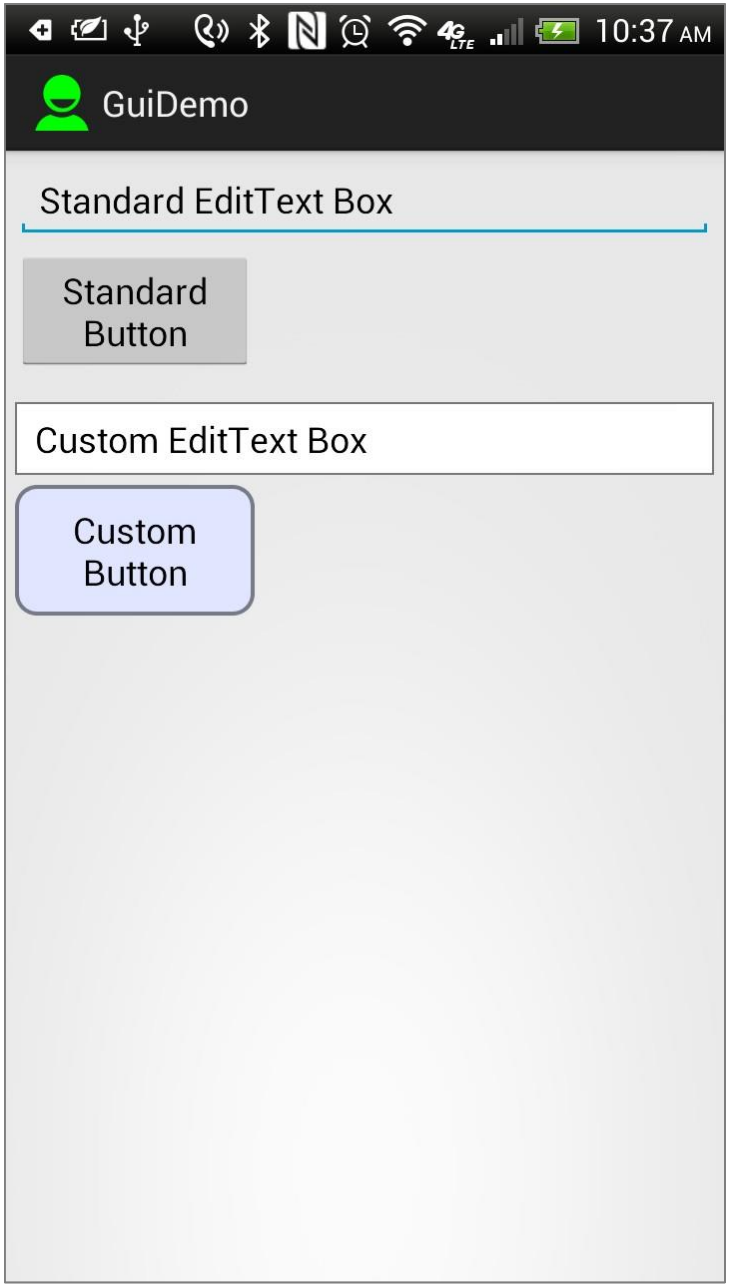
Appendix E: Customizing Widgets

1. The appearance of a widget can be adjusted by the user. For example a button widget could be modified by changing its shape, border, color, margins, etc.

2. Basic shapes include: rectangle, oval, line, and ring.

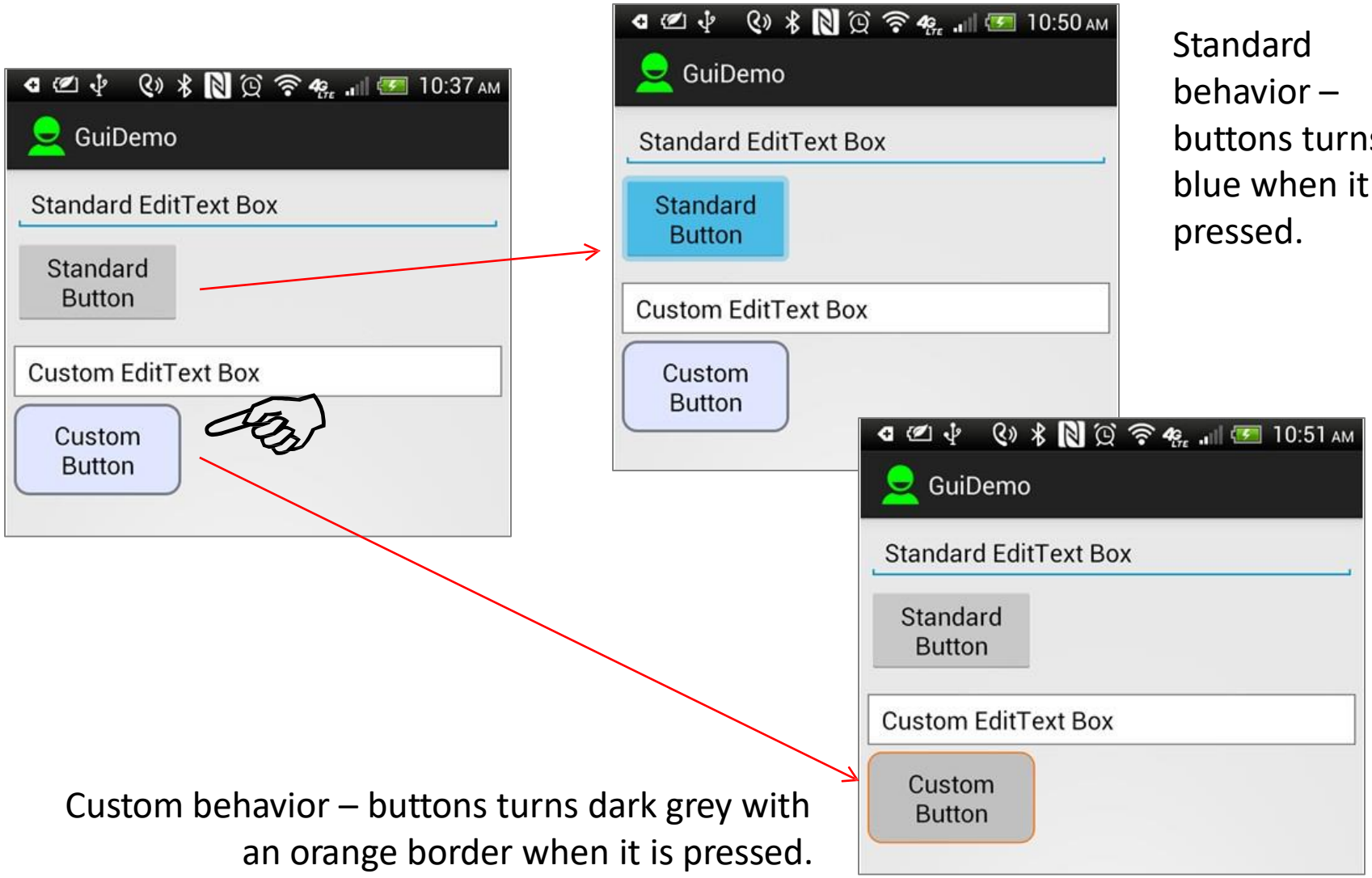
3. In addition to visual changes, the widget's reaction to user interaction could be adjusted for events such as: Focused, Clicked, etc.

4. The figure shows and EditText and Button widgets as *normally* displayed by a device running SDK4.3 (Ice Cream). The bottom two widgets are custom made versions of those two controls respectively.



Appendix E: Customizing Widgets

The image shows visual feedback provided to the user during the clicking of a standard and a *custom* Button widget. Assume the device runs under SDK4.3

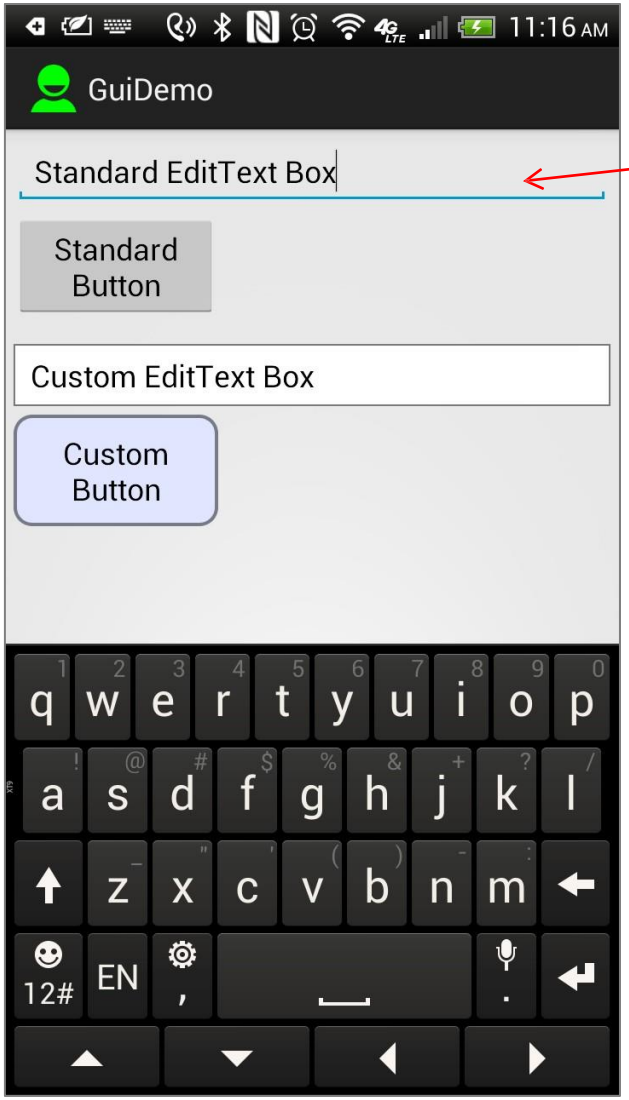


Standard behavior – buttons turns blue when it is pressed.

Custom behavior – buttons turns dark grey with an orange border when it is pressed.

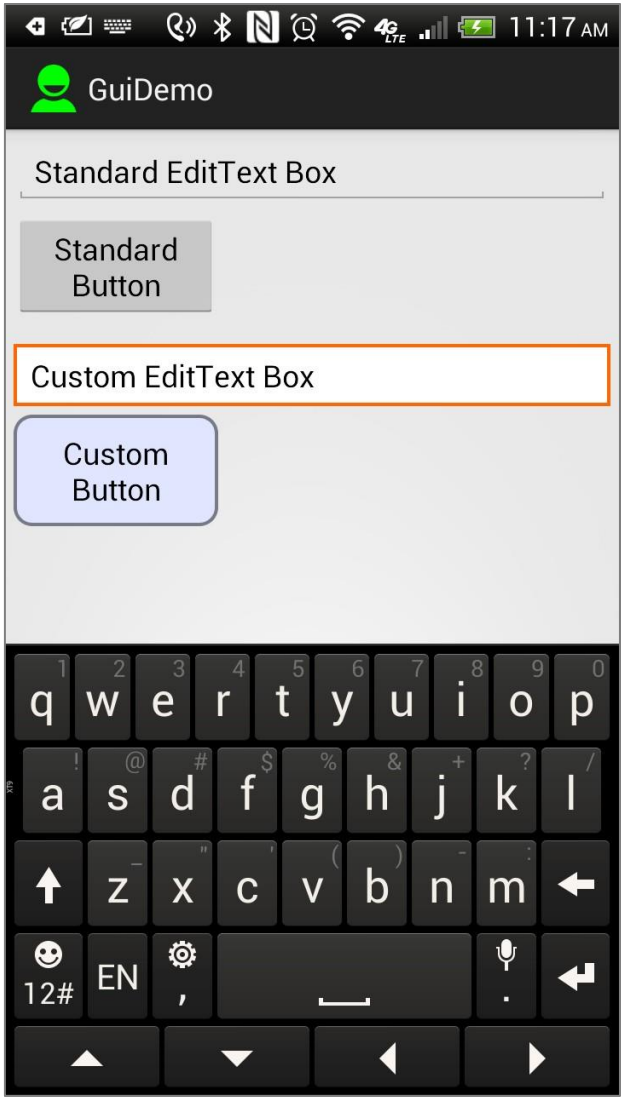
Appendix E: Customizing Widgets

Observe the transient response of the standard and custom made EditText boxes when the user touches the widgets provoking the 'Focused' event.



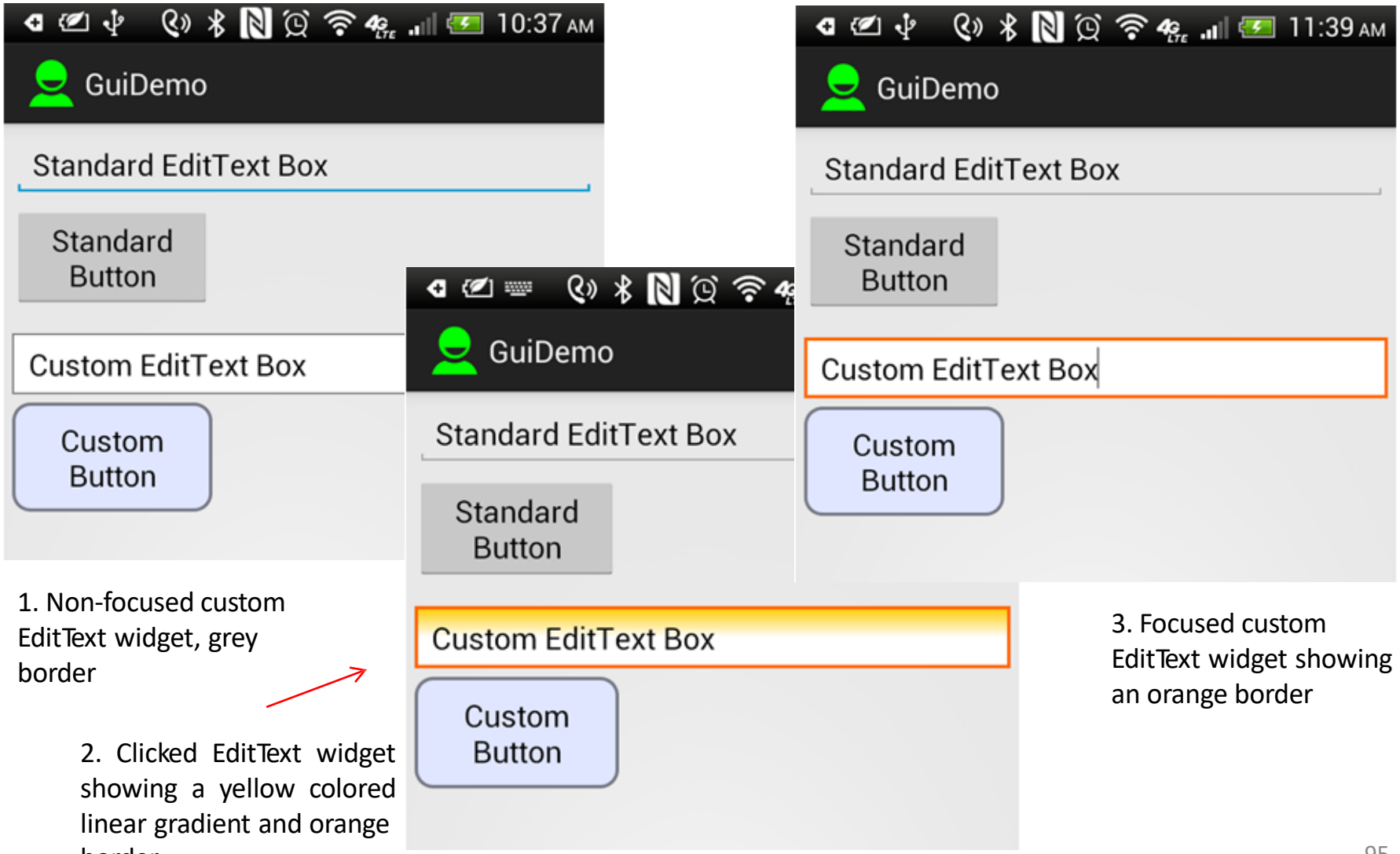
When focused the standard box shows a blue bottom line

A focused custom box shows an orange all-around frame



Appendix E: Customizing Widgets

When the user taps on the custom made EditText box a gradient is applied to the box to flash a visual feedback reassuring the user of her selection.



1. Non-focused custom EditText widget, grey border

2. Clicked EditText widget showing a yellow colored linear gradient and orange border

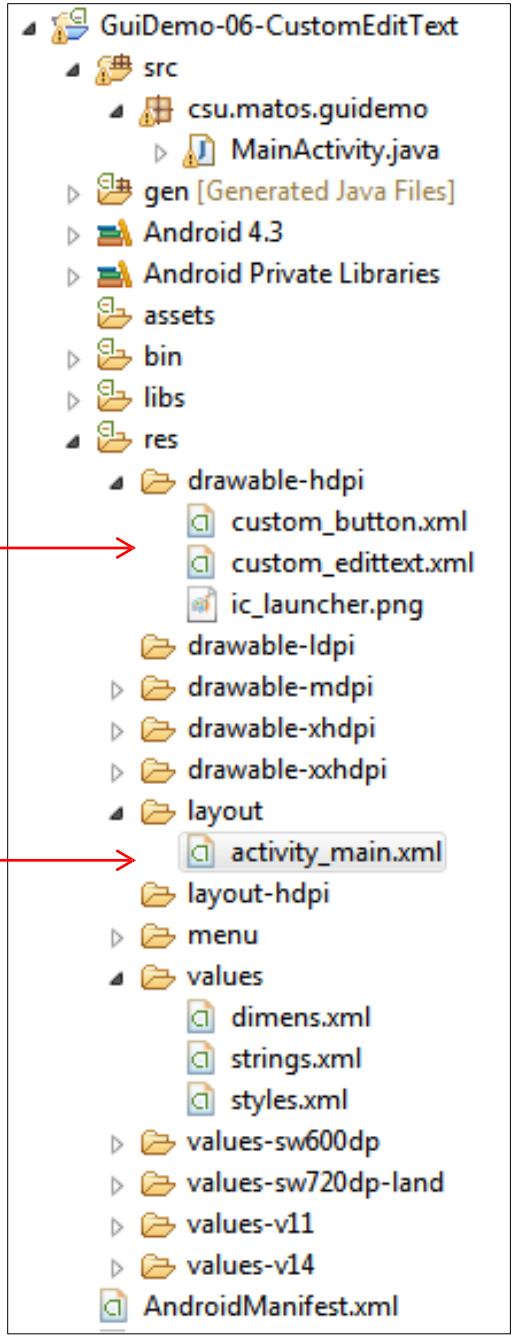
3. Focused custom EditText widget showing an orange border

Appendix E: Customizing Widgets

Organizing the application

Definition of the custom templates for Button and EditText widgets

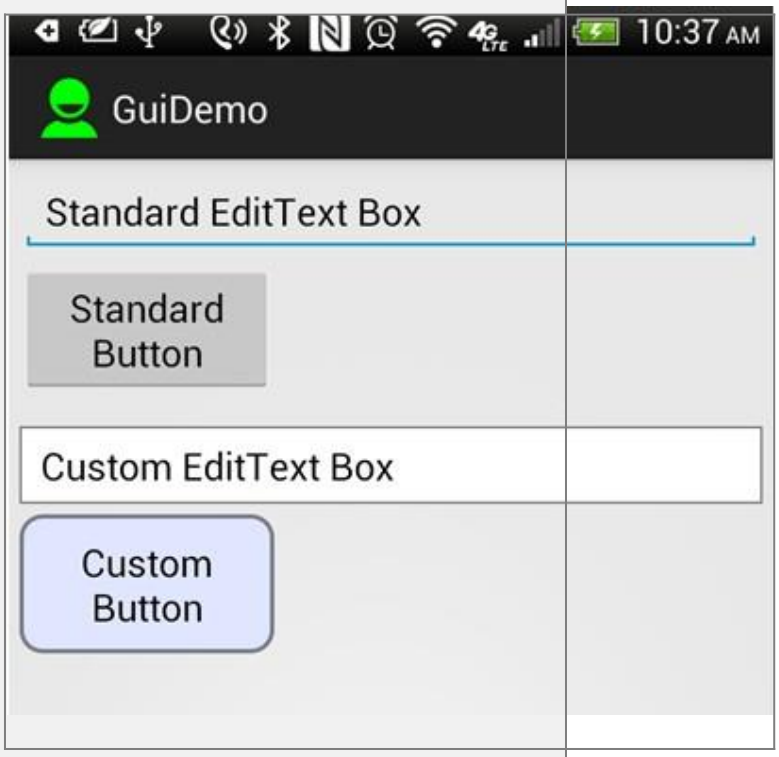
Layout referencing standard and custom made widgets



Appendix E: Customizing Widgets

Activity Layout 1 of 2

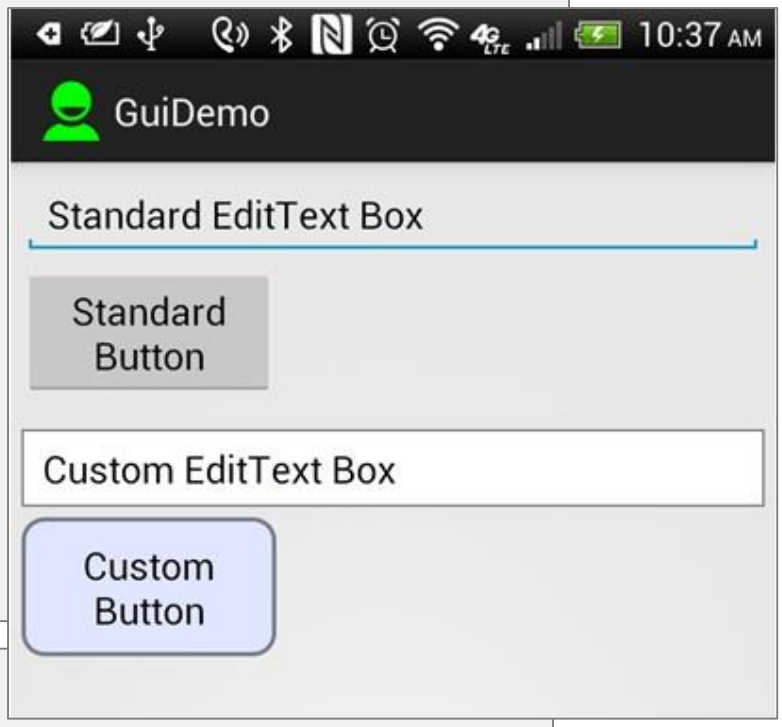
```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="5dp" >

    <EditText
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="5dp"
        android:ems="10"
        android:inputType="text"
        android:text="@string/standard_edittext" >

        <requestFocus />
    </EditText>

    <Button
        android:id="@+id/button1"
        android:layout_width="120dp"
        android:layout_height="wrap_content"
        android:layout_marginBottom="15dp"
        android:text="@string/standard_button" />
```



Appendix E: Customizing Widgets

Activity Layout (2 of 2) and Resource: res/values/strings

```
    <EditText
        android:id="@+id/editText2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="5dp"
        android:background="@drawable/custom_edittext"
        android:ems="10"
        android:inputType="text"
        android:text="@string/custom_edittext" />

    <Button
        android:id="@+id/button2"
        android:layout_width="120dp"
        android:layout_height="wrap_content"
        android:background="@drawable/custom_button"
        android:text="@string/custom_button" />

</LinearLayout>
```




```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">GuiDemo</string>
    <string name="action_settings">Settings</string>
    <string name="standard_button">Standard Button</string>
    <string name="standard_edittext">Standard EditText Box</string>
    <string name="custom_button">Custom Button</string>
    <string name="custom_edittext">Custom EditText Box</string>
</resources>
```

Appendix E: Customizing Widgets

Resource: res/drawable/custom_button.xml

The custom Button widget has two faces based on the event **state_pressed** (true, false). The Shape attribute specifies its solid color, padding, border (stroke) and corners (rounded corners have radius > 0)

```xml
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:state_pressed="true">
        <shape android:shape="rectangle">
            <corners android:radius="10dp"/>
            <solid android:color="#ffc0c0c0" />
            <padding android:left="10dp"
                android:top="10dp"
                android:right="10dp"
                android:bottom="10dp"/>
            <stroke android:width="1dp" android:color="#ffFF6600"/>
        </shape>
    </item>
    <item android:state_pressed="false">
        <shape android:shape="rectangle">
            <corners android:radius="10dp"/>
            <solid android:color="#ffE0E6FF"/>
            <padding android:left="10dp"
                android:top="10dp"
                android:right="10dp"
                android:bottom="10dp"/>
            <stroke android:width="2dp" android:color="#ff777B88"/>
        </shape>
    </item>
</selector>
```

Appendix E: Customizing Widgets

Resource: res/drawable/custom_edittext.xml

The rendition of the custom made EditText widget is based on three states: normal, state_focused, state_pressed.

```xml
<?xml version="1.0" encoding="utf-8"?>
<selector  xmlns:android="http://schemas.android.com/apk/res/android">

<item android:state_pressed="true">
    <shape android:shape="rectangle">
          <gradient
           android:angle="90"
           android:centerColor="#FFffffff"
           android:endColor="#FFffcc00"
           android:startColor="#FFffffff"
           android:type="linear" />

        <stroke android:width="2dp" android:color="#FFff6600" />
        <corners android:radius="0dp" />
        <padding android:left="10dp"
            android:top="6dp"
            android:right="10dp"
            android:bottom="6dp" />
    </shape>
</item>
```

Custom EditText Box

Appendix E: Customizing Widgets

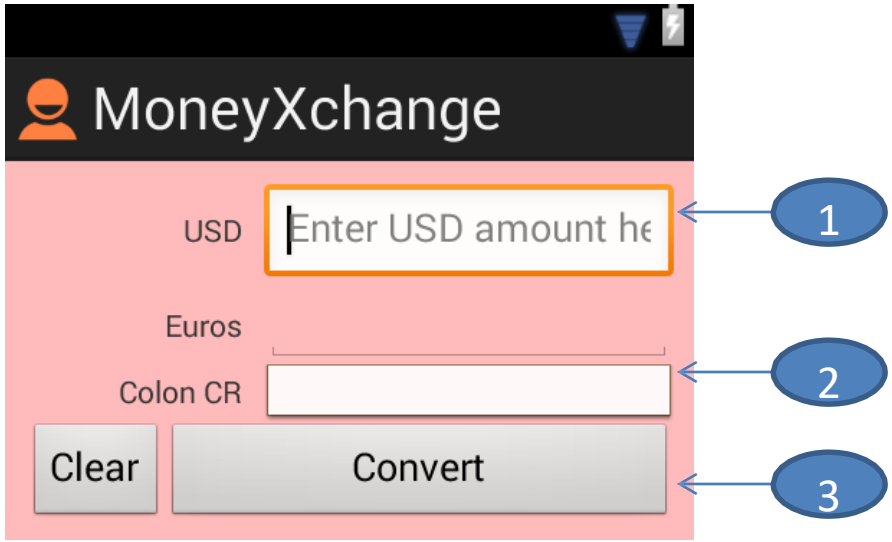
Resource: res/drawable/custom_edittext.xml

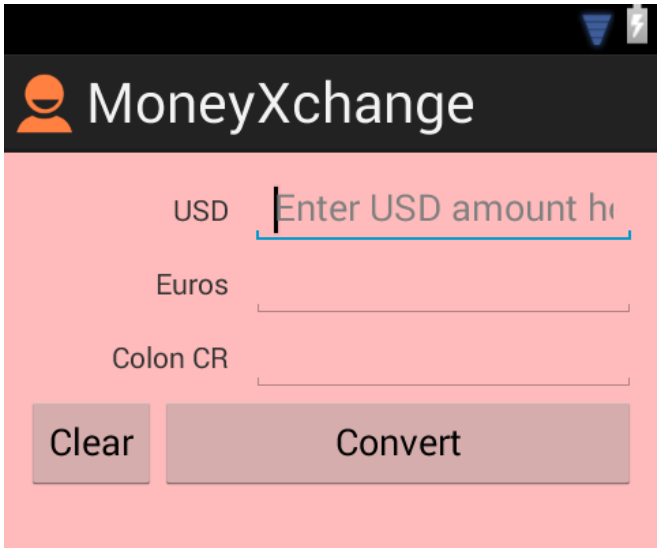
The rendition of the custom made EditText widget is based on three states: normal, state focused, state_pressed.

```
<item android:state_focused="true">
    <shape>
        <solid android:color="#FFffffff" />
        <stroke android:width="2dp" android:color="#FFff6600" />
        <corners android:radius="0dp" />
        <padding android:left="10dp"
            android:top="6dp"
            android:right="10dp"
            android:bottom="6dp" />
    </shape>
</item>

<item>
    <!-- state: "normal" not-pressed & not-focused  -->
    <shape>
        <stroke android:width="1dp" android:color="#ff777777" />
        <solid android:color="#ffffffff" />
        <corners android:radius="0dp" />
        <padding android:left="10dp"
            android:top="6dp"
            android:right="10dp"
            android:bottom="6dp" />
    </shape>
</item>
</selector>
```

Appendix F: Fixing Bleeding Background Color

You may change a layout's color by simply adding in the XML layout the clause `android:background="#44ff0000"` (color is set to semi-transparent red).
The problem is that the layout color appears to be placed on top of the other controls making them look 'smeared' as show in the figure below (right).
A solution is to reassert the smeared widgets' appearance by explicitly setting a value in their corresponding `android:background` XML attributes.
The figure on the left include explicit assignments to the widgets' background.






```
1. android:background="@android:drawable/edit_text"
2. android:background="@android:drawable/editbox_dropdown_light_frame"
3. android:background="@android:drawable/btn_default"
```