

MOBILE DEVELOPMENT

USER INTERFACE

CONTENTS

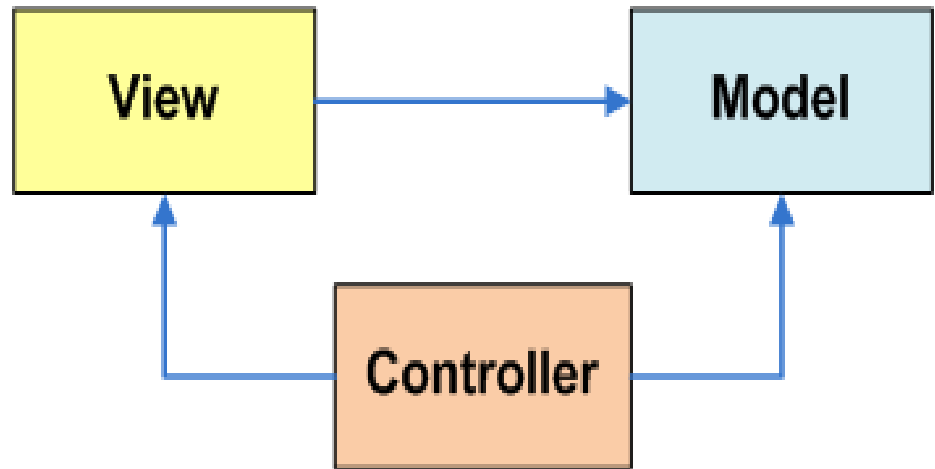
Graphical user interfaces

Widgets and layouts

Some examples

GRAPHICAL USER INTERFACES

(The model-view-control pattern - MVC)



The Model-View-Controller (MVC) is an important software design pattern first introduced with the Xerox-Smalltalk80 system whose main goal is to separate the (1) user interface, (2) business, and (3) input logic.

How is this pattern seen by the Android developer?

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

GRAPHICAL USER INTERFACES

(Create MVC conforming solutions)

The Android developer should be aware of

- Inputs could be sent to the application from various physical/logical components. Reacting to those signals is typically handled by callback methods. Usually there are many of them, you want to learn how to choose the appropriate one.
- Moving to states in the lifecycle is tied to logic in the model. For instance, if forced to Pause you may want to save uncommitted data.
- A notification mechanism is used to inform the user of important events happening outside the current application (such as arrival of a text message or email, low battery, fluctuations of the stock market, etc) and consequently choose how to proceed.
- Views are unlimited in terms of aesthetic and functionality. However physical constraints such as size, and hardware acceleration (or lack of) may affect how graphical components are managed.

GRAPHICAL USER INTERFACES

(MVC Pattern: the view - user interfaces)

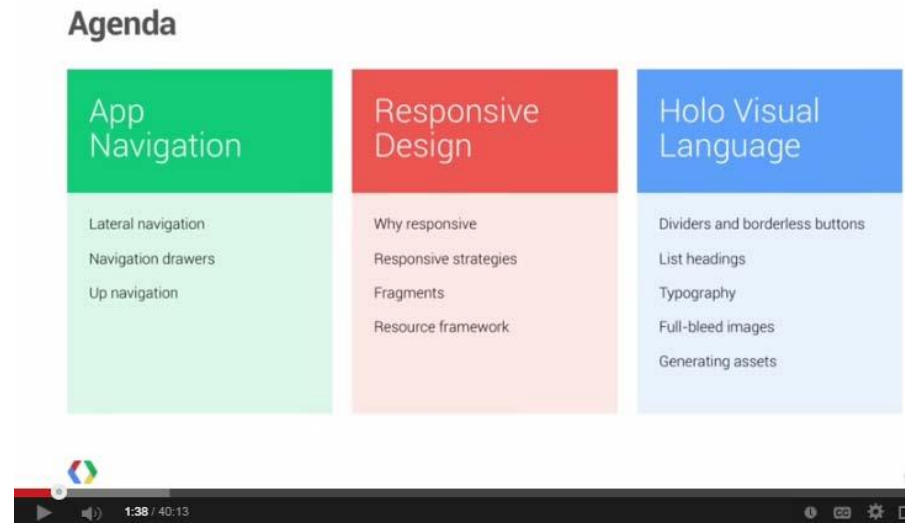
Android graphical interfaces are usually implemented as XML files (although they could also be dynamically created from Java code)

An Android UI is conceptually similar to a common HTML page

- In a manner similar to a web page interaction, when the Android user touches the screen, the controller interprets the input and determines what specific portion of the screen and gestures were involved. Based on this information it tells the model about the interaction in such a way that the appropriate “callback listener” or lifecycle state could be called into action.
- Unlike a web application (which refreshes its pages after explicit requests from the user) an asynchronous Android background service could quietly notify the controller about some change of state (such as reaching a given coordinate on a map) and in turn a change of the view’s state could be triggered; all of these without user intervention.

GRAPHICAL USER INTERFACES (Android UI design patterns)

For a discussion of the newest Android UI Design Patterns (2013) see video:
<https://www.youtube.com/watch?v=Jl3-lzlzOJI>



A collection of weekly instructional videos made by the same presenters can be obtained from the page (visited on Sept 6, 2014)

https://www.youtube.com/results?search_query=android+design+in+action

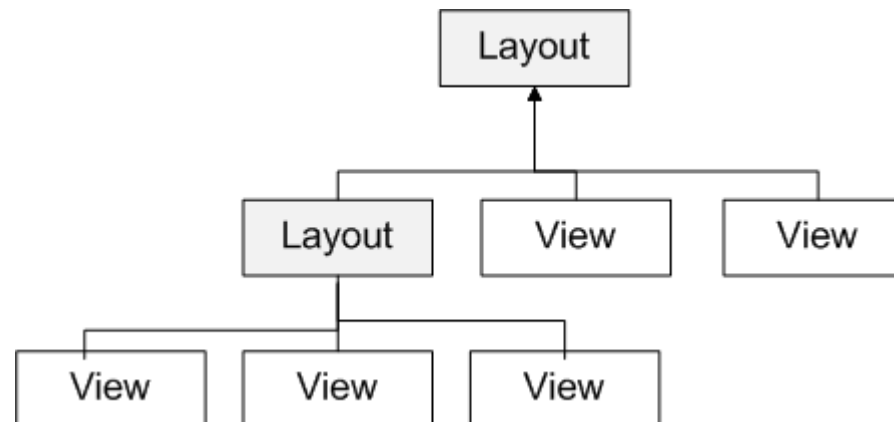
GRAPHICAL USER INTERFACES (The VIEW class)

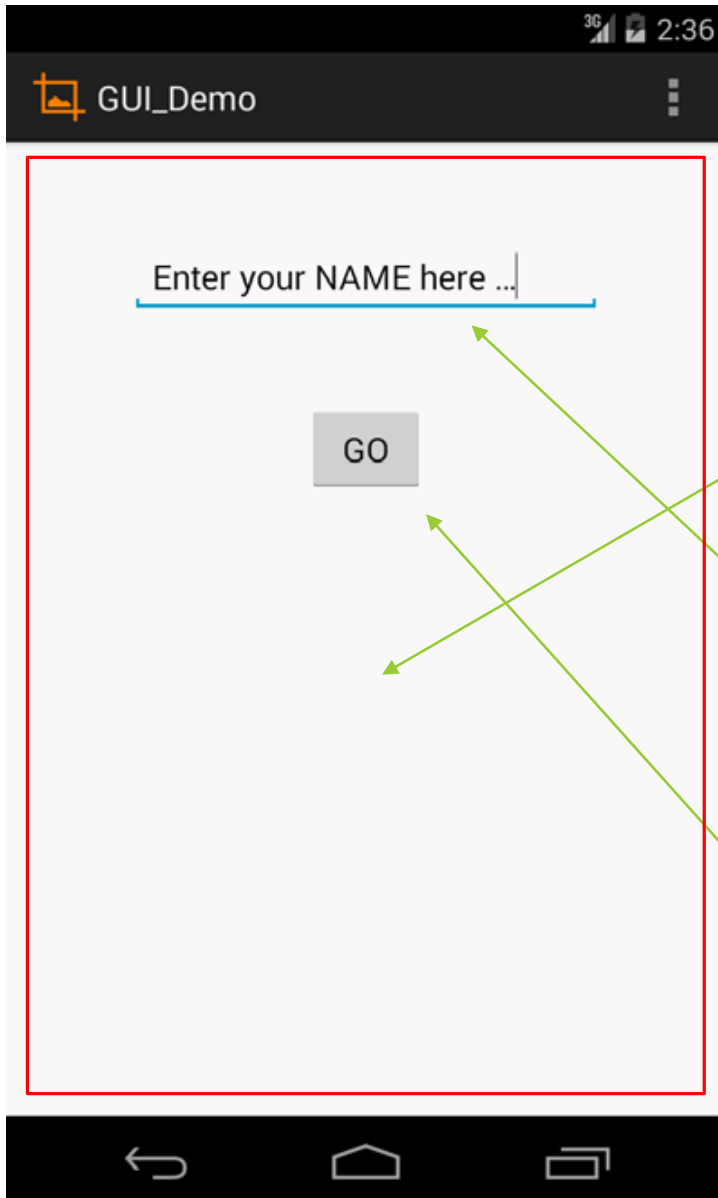
The View class is the Android's most basic component from which user interfaces can be created. It acts as a container of displayable elements.

A View occupies a rectangular area on screen and is responsible for drawing & 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 structured containers used for holding other Views and nested layouts.





GRAPHICAL USER INTERFACES (Using XML to represent UIs)

```
<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>
```



GRAPHICAL USER INTERFACES

(Nesting XML layouts)

An Android's XML view file consists of a layout design holding a hierarchical arrangement of its contained elements.

The inner elements could be basic widgets or user-defined nested layouts holding their own viewgroups.

An Activity uses the `setContentView(R.layout.xmlfilename)` method to render a view on the device's screen.

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent" android:layout_height="wrap_content"
  android:orientation="horizontal" >
     Widgets and other nested layouts
</LinearLayout>
```

GRAPHICAL USER INTERFACES

(Setting views to work)

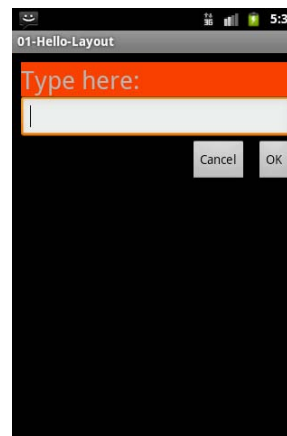
Dealing with widgets & layouts typically involves the following operations

- Set properties: For instance, when working with a TextView you set the background color, text, font, alignment, size, padding, margin, etc.
- Set up listeners: For example, an image could be programmed to respond to various events such as: click, long-tap, mouse-over, etc.
- Set focus: To set focus on a specific view, you call the method `.requestFocus()` or use XML tag `<requestFocus />`
- Set visibility: You can hide or show views using `setVisibility(...)`



Linear Layout

Places its inner views either in horizontal or vertical disposition.



Relative Layout

A ViewGroup allowing you to position elements relative to each other.

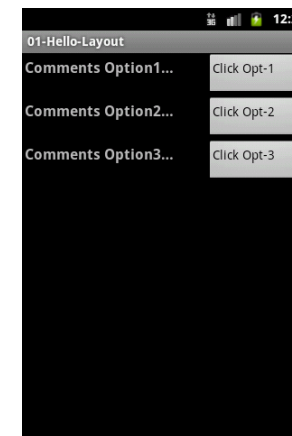


Table Layout

A ViewGroup placing elements using a row & column disposition.

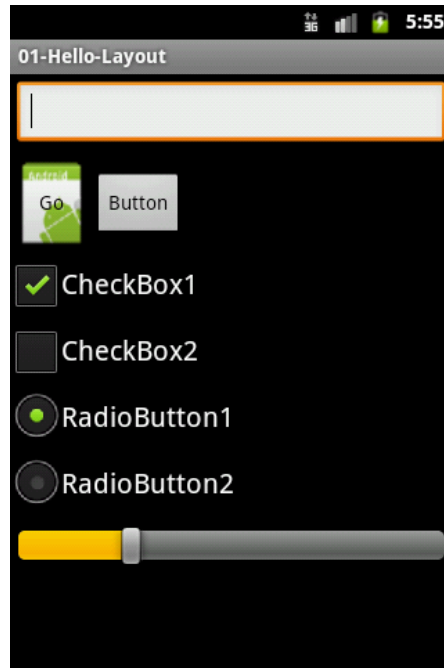
GRAPHICAL USER INTERFACES

(A sample of common Android WIDGETS)



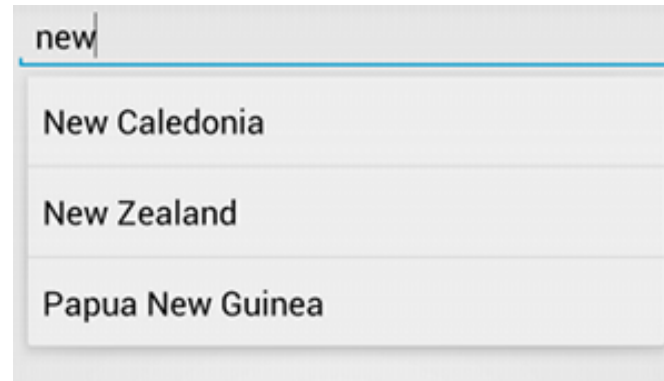
TimePicker-AnalogClock-DatePicker

A DatePicker is a widget allowing to select a month, day and year



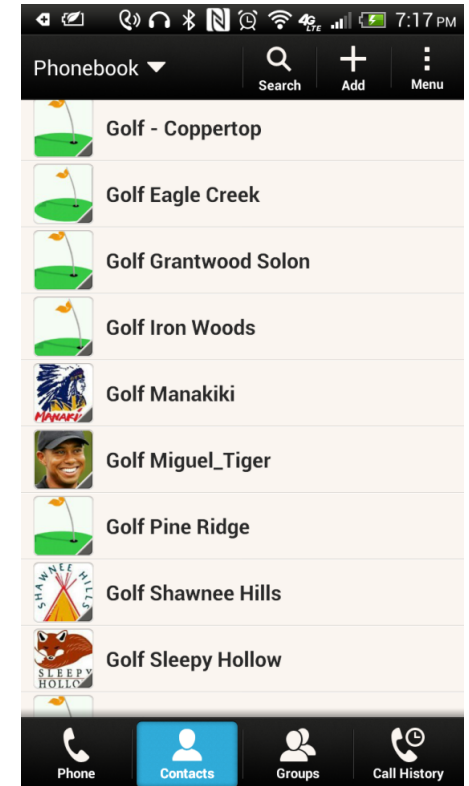
Form Controls

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



AutoCompleteTextView

A version of EditText widget providing autocomplete suggestions as typing

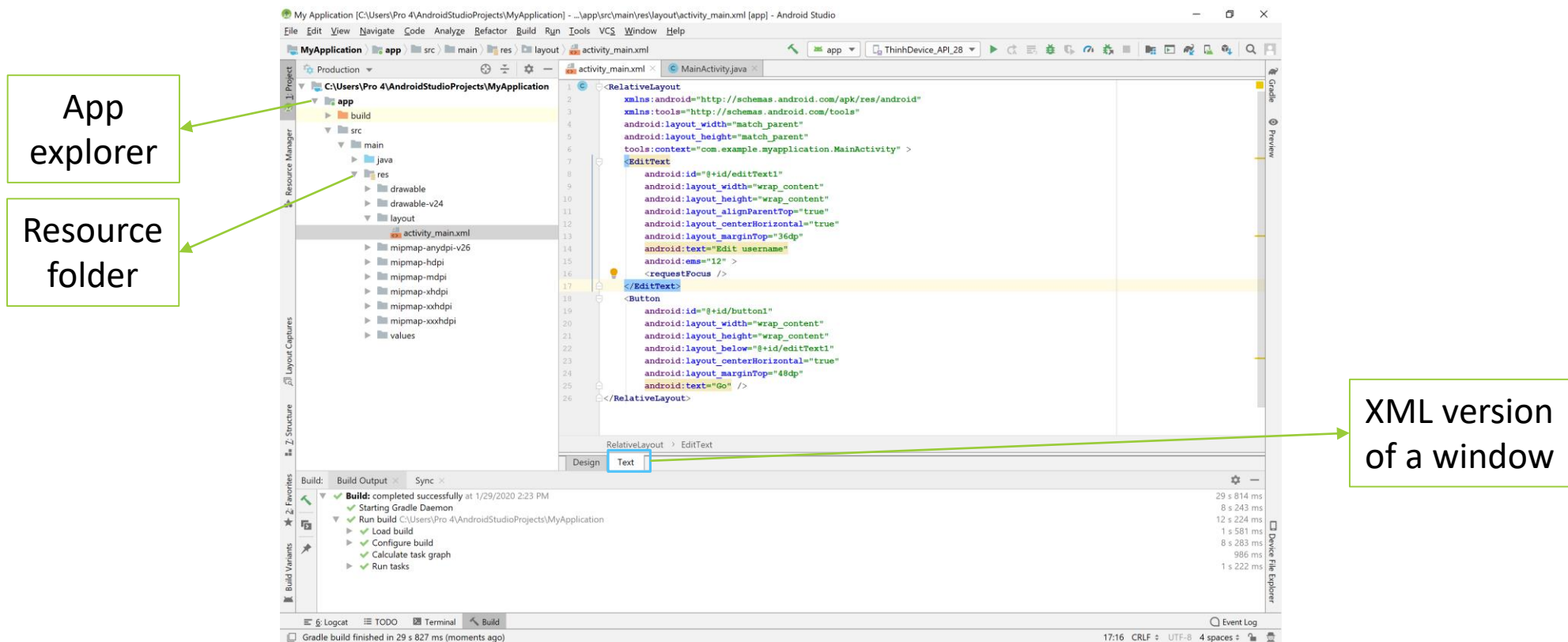


ListView

A View showing items in a vertically scrolling list. The items are acquired from a ListAdapter.

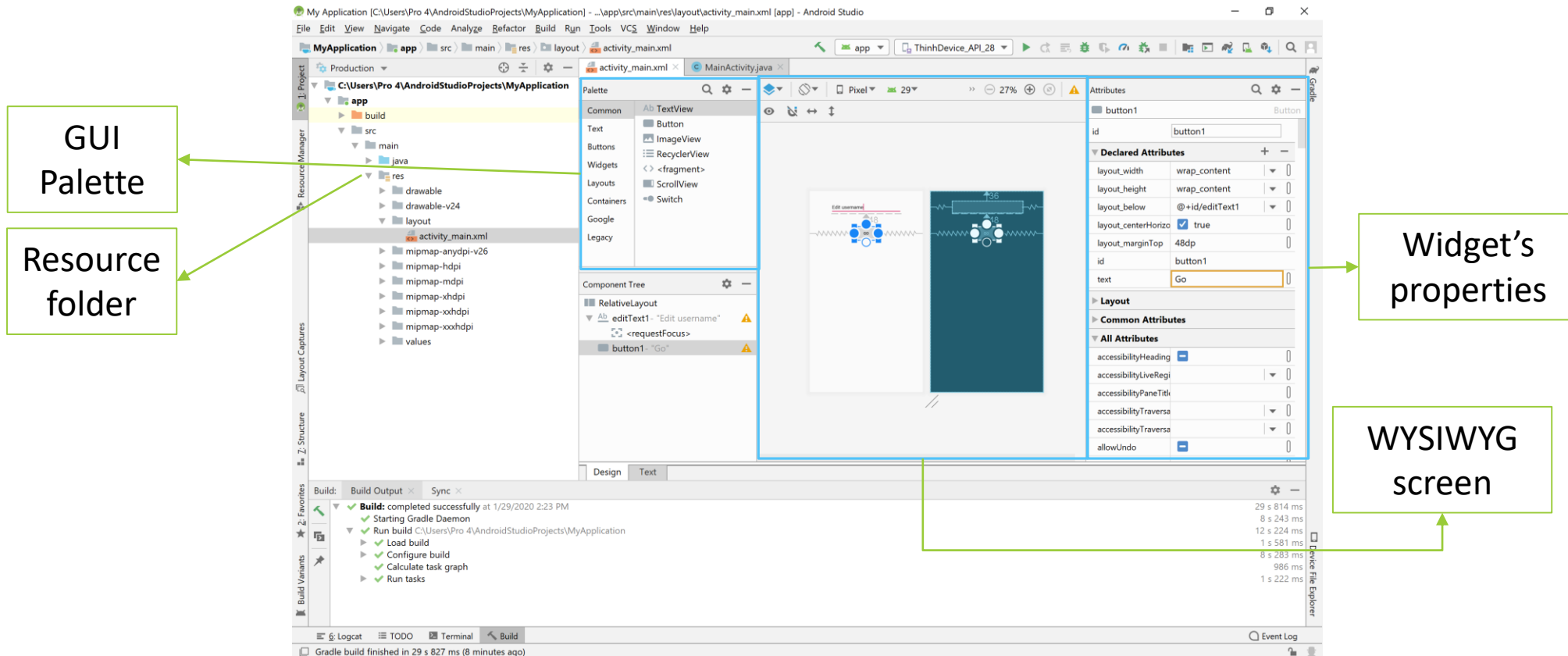
GRAPHICAL USER INTERFACES (GUI editing: XML version)

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



GRAPHICAL USER INTERFACES (GUI editing: WYSIWYG version)

The Screen Designer Tool allows you to operate each screen using either a WYSIWIG or XML editor



GRAPHICAL USER INTERFACES

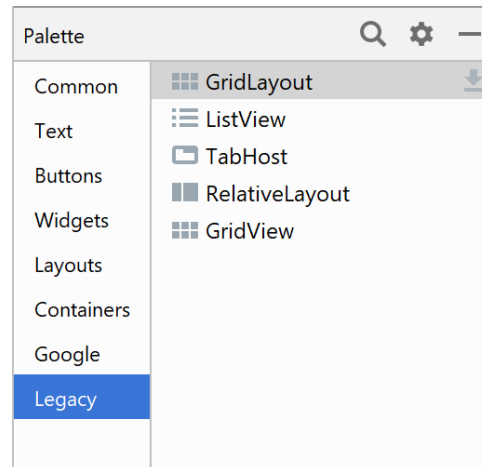
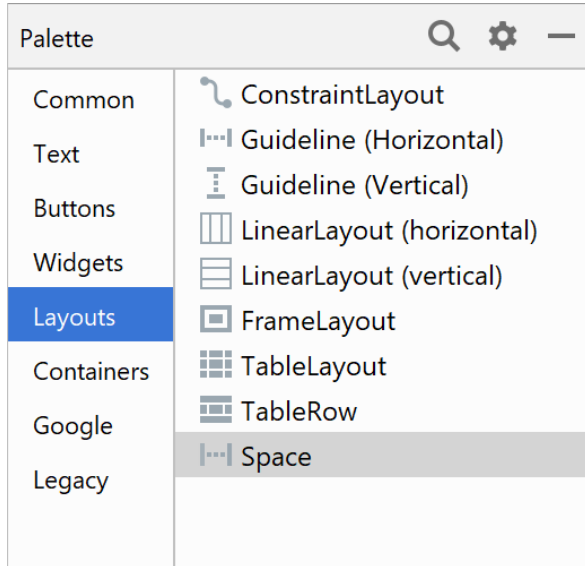
(Tools used to create an Android GUI)

Alternative tools for creating Android apps and GUIs:

- Android Studio. Based on IntelliJ IDEA IDE. Functionally equivalent to Eclipse with the ADT Plugin.
<http://developer.android.com/sdk/installing/studio.html>
- Android SDK. Streamlined workbench based on Eclipse+ADT in a simpler to install package.
<http://developer.android.com/sdk/index.html>
- NBAndroid. Workbench based on NetBeans+ADT.
<http://www.nbandroid.org/2014/07/android-plugin-for-gradle-011012.html>
- DroidDraw Very simple GUI designer, incomplete, not integrated to the Eclipse IDE, aging! <http://www.droiddraw.org/>
- App Inventor (educational, very promising & ambitious, 'hides' coding ...) <http://appinventor.mit.edu/>

The LAYOUT

- Android GUI Layouts are containers having a predefined structure and placement policy such as relative, linear horizontal, grid-like, etc.
- Layouts can be nested, therefore a cell, row, or column of a given layout could be another layout.



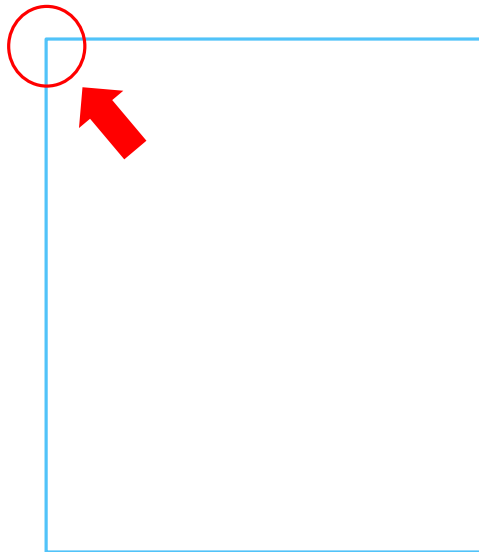
WIDGETS AND LAYOUTS (FrameLayout)

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

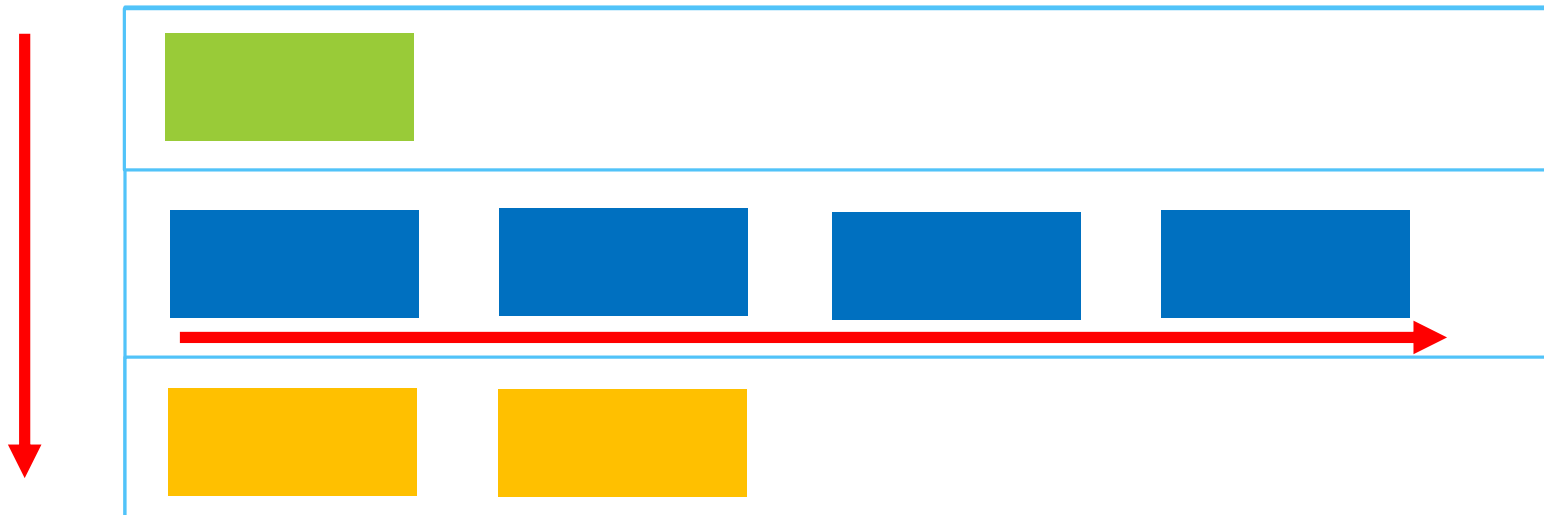


WIDGETS AND LAYOUTS (LinearLayout)

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.



WIDGETS AND LAYOUTS

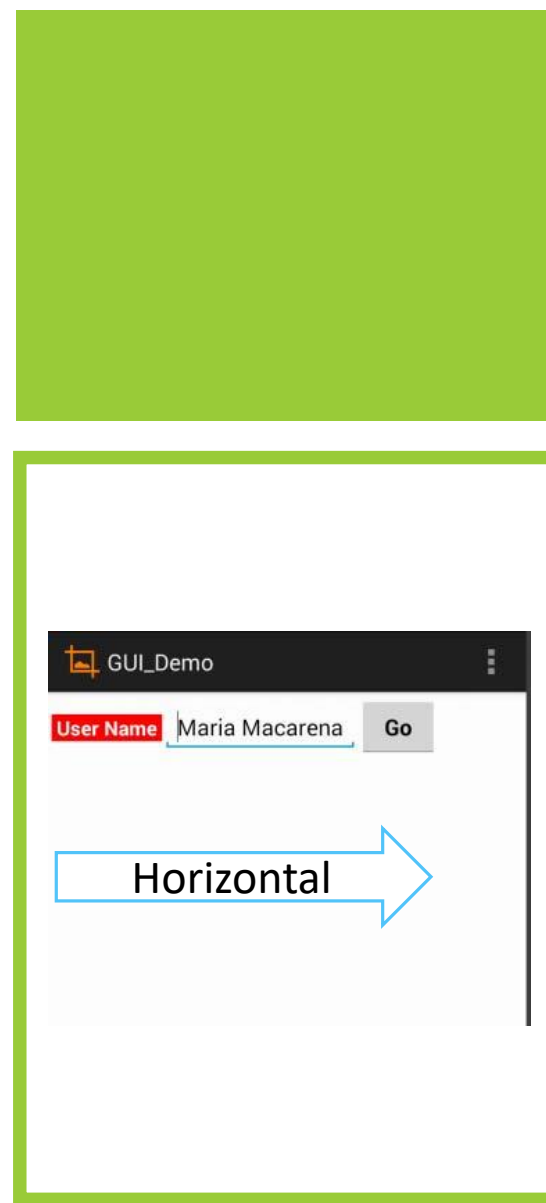
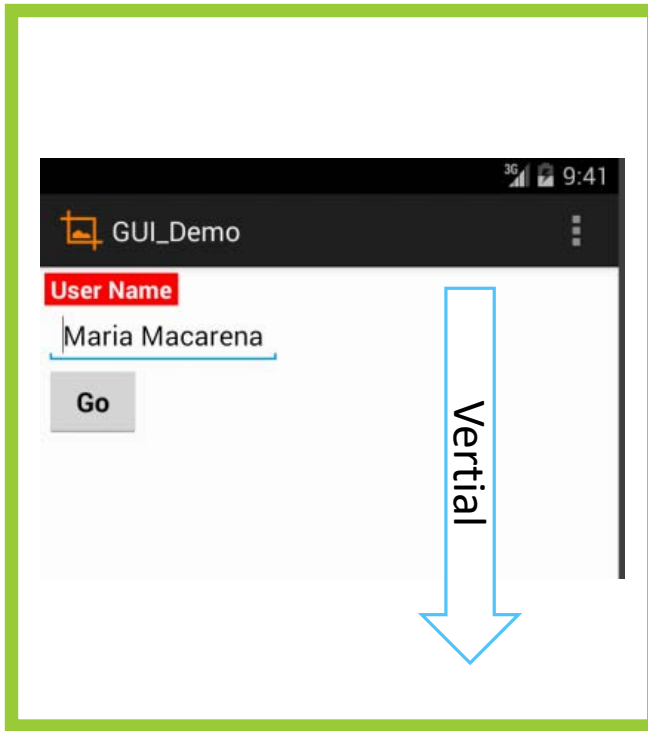
(LinearLayout – setting attributes)

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 `android:orientation` property can be set to horizontal for columns, or vertical for rows.

Use `setOrientation()` for runtime changes.

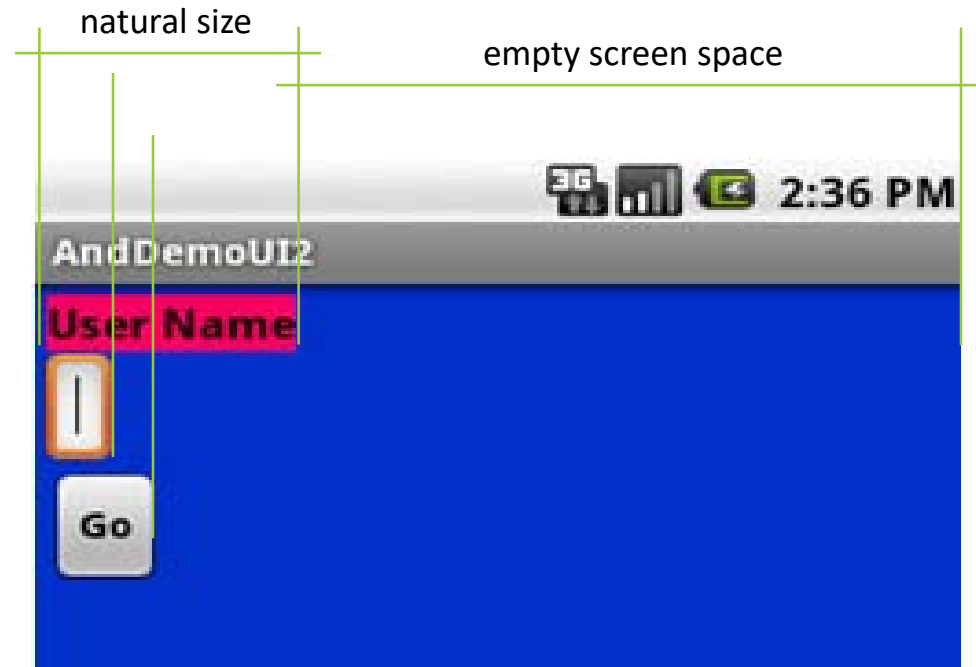


WIDGETS AND LAYOUTS

(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).



WIDGETS AND LAYOUTS

(LinearLayout - Fill model)

All widgets inside a LinearLayout include 'width' and 'height' attributes:

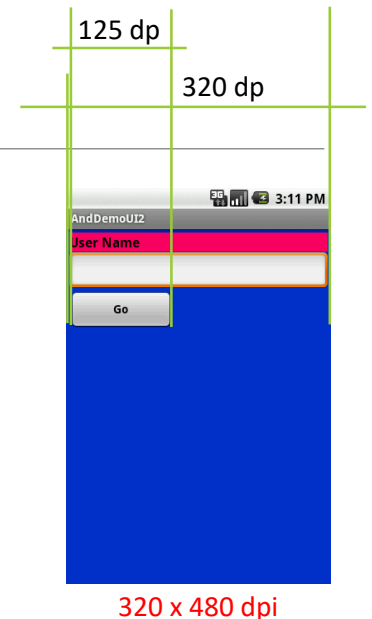
- android:layout_width
- android:layout_height

Values used in defining height and width can be:

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

Code:

```
<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>
```



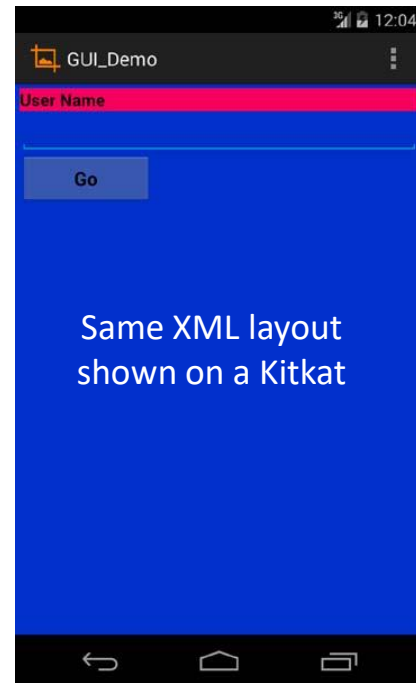
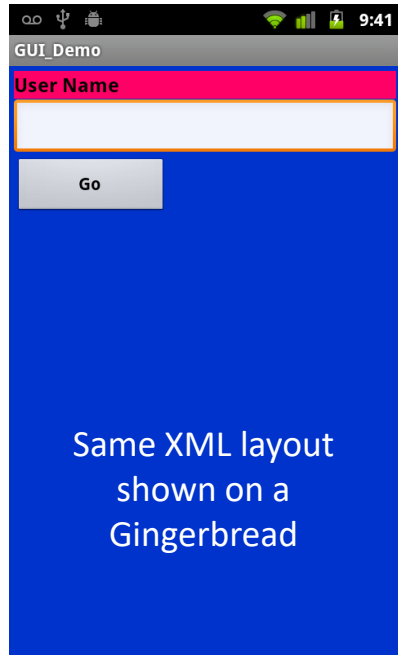
320 x 480 dpi

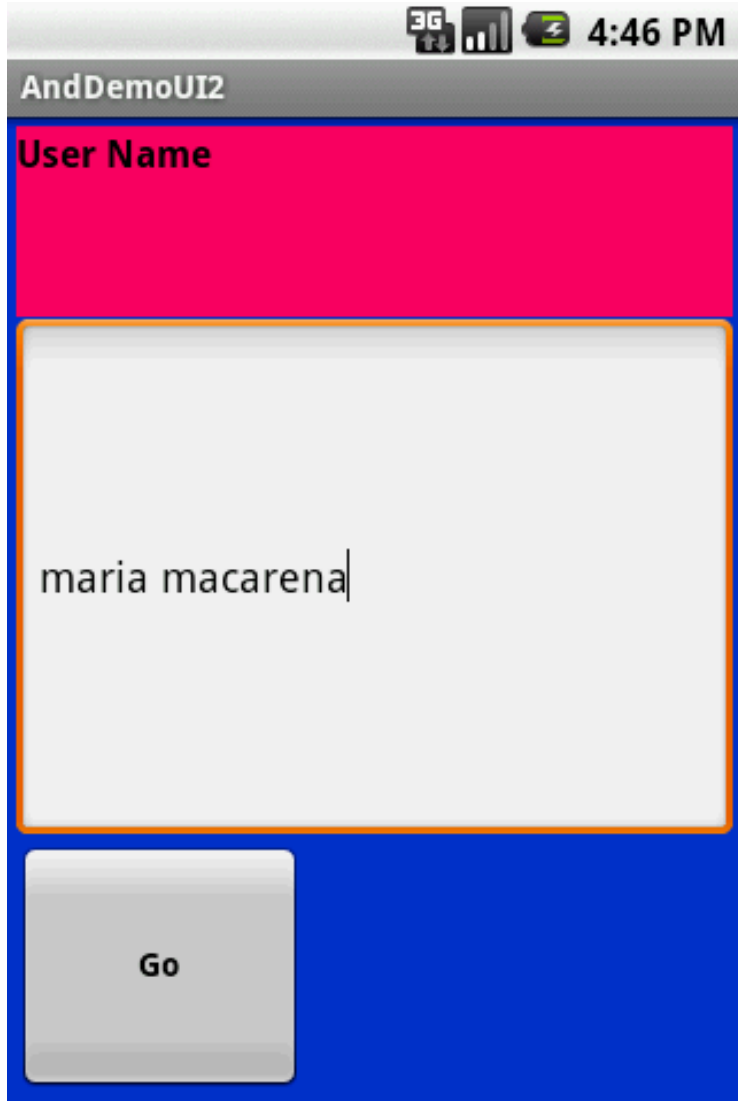
WIDGETS AND LAYOUTS

(Same XML different rendition)

Since the introduction of Android 4.x, changes in the SDK make layouts to be more uniformly displayed in all 4.x and newer devices (the intention is to provide a seamless Android experience independent from provider, hardware, and developer).

The XML spec used in the previous example looks different when displayed on a 4.x and older devices (see figures on the right, please also notice the color bleeding occurring on top of the GO button, more on this issue in the Appendix)





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

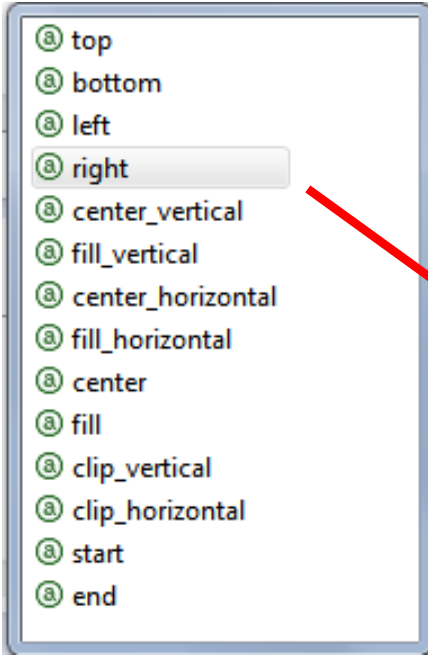
WIDGETS AND LAYOUTS (LinearLayout - Weight)

The extra space left unclaimed in a layout could be assigned to any of its inner components by setting its Weight attribute. 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"`

Remember, default value is 0



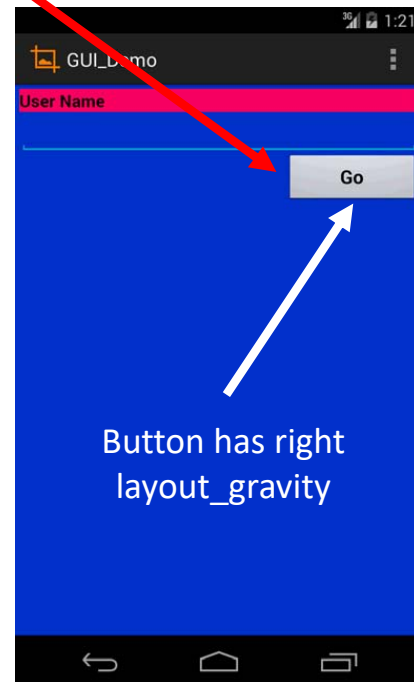
- Ⓐ top
- Ⓐ bottom
- Ⓐ left
- Ⓐ right
- Ⓐ center_vertical
- Ⓐ fill_vertical
- Ⓐ center_horizontal
- Ⓐ fill_horizontal
- Ⓐ center
- Ⓐ fill
- Ⓐ clip_vertical
- Ⓐ clip_horizontal
- Ⓐ start
- Ⓐ end

WIDGETS AND LAYOUTS (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.



WIDGETS AND LAYOUTS

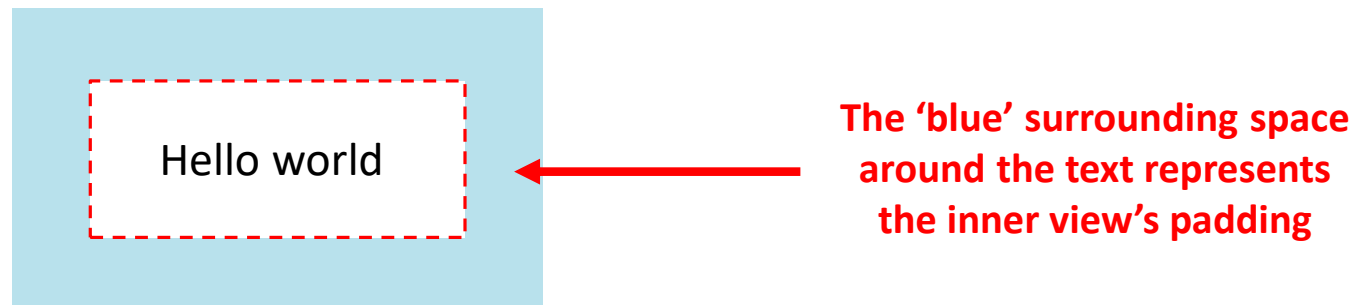
(LinearLayout - Padding)

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

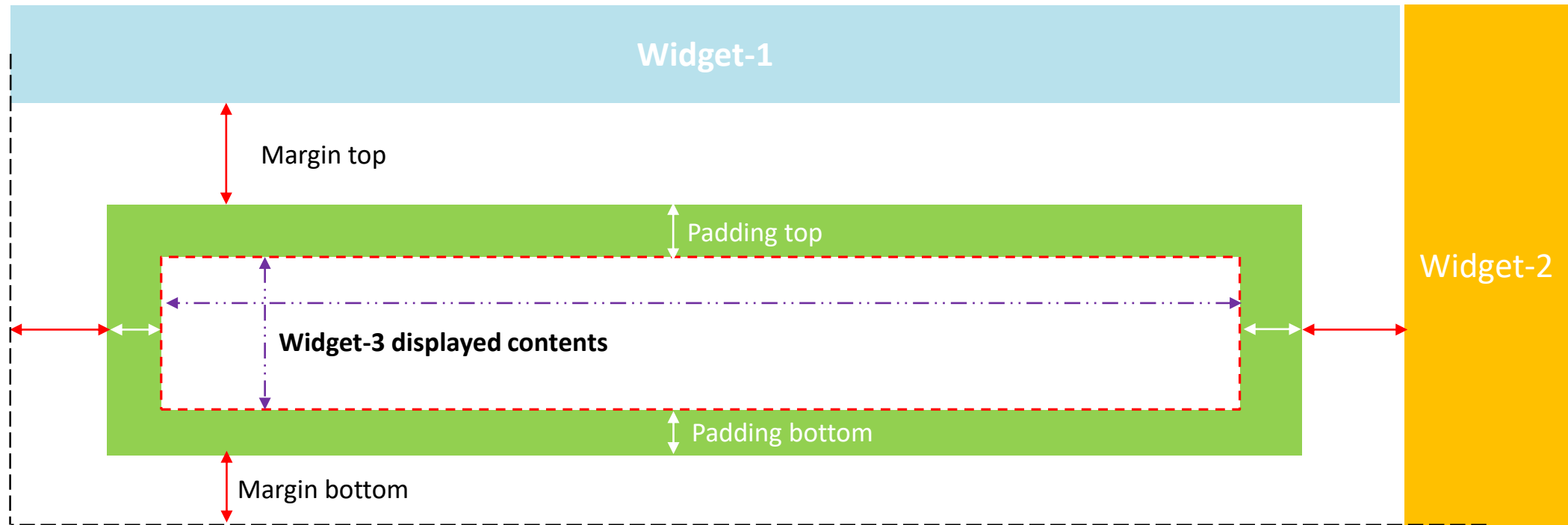
- `android:padding` property
- or call method `setPadding()` at runtime.

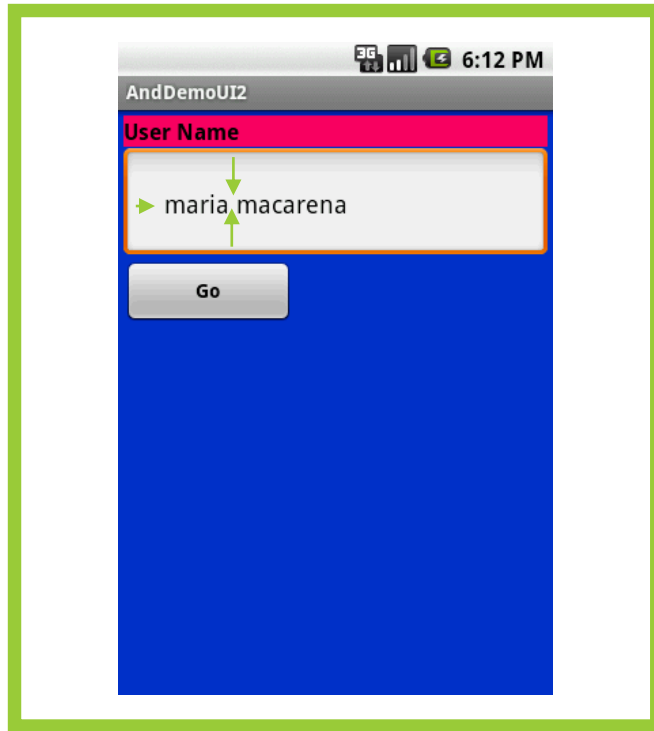


WIDGETS AND LAYOUTS

(LinearLayout – Padding and margin)

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





WIDGETS AND LAYOUTS

(LinearLayout - Set internal margins with padding)

Example: EditText box has been changed to include 30dp of padding all around

<EditText

android:id="@+id/ediName"

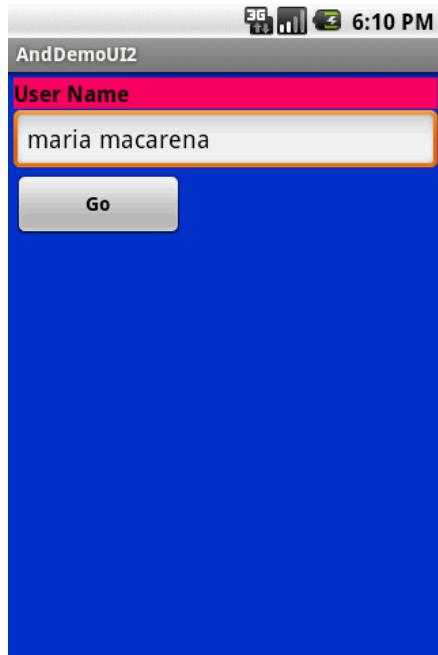
android:layout_width="match_parent"

android:layout_height="wrap_content"

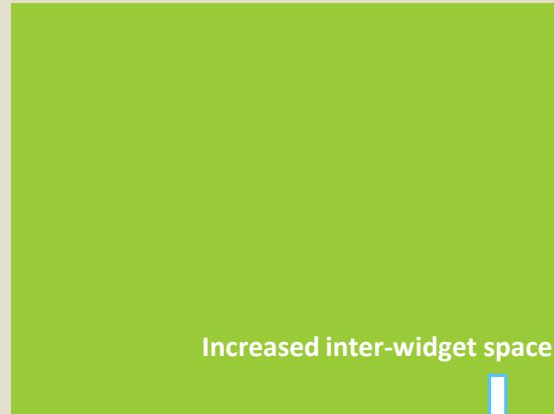
android:textSize="18sp"

android:padding="30dp"/>

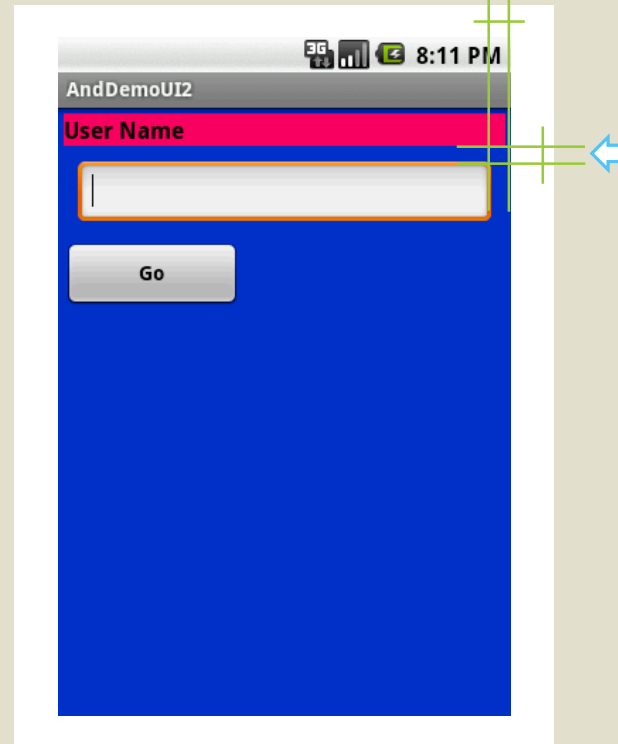
...



Using default spacing between widgets



Increased inter-widget space



WIDGETS AND LAYOUTS

(LinearLayout - Set external margins)

Widgets –by default– are closely displayed next to each other.

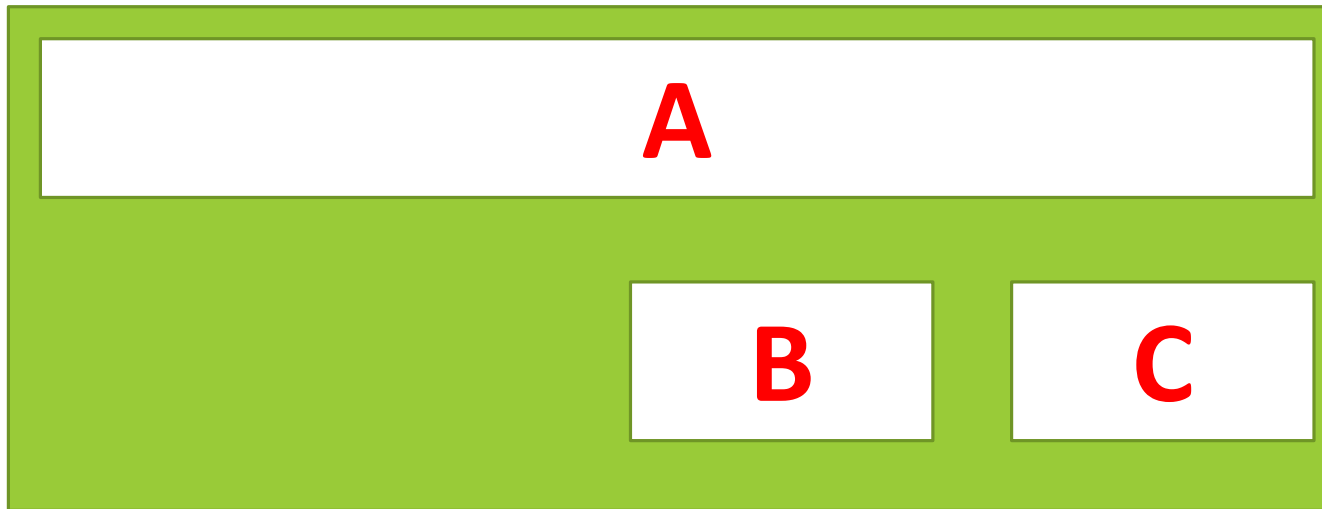
To increase space between them use the `android:layout_margin` attribute

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

...

WIDGETS AND LAYOUTS (RelativeLayout)

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

WIDGETS AND LAYOUTS

(RelativeLayout - Referring to the **container**)

Below there is a sample of various positioning XML boolean properties (true/false) which are useful for collocating a widget based on the location of its parent container.

`android:layout_alignParentTop`

`android:layout_alignParentBottom`

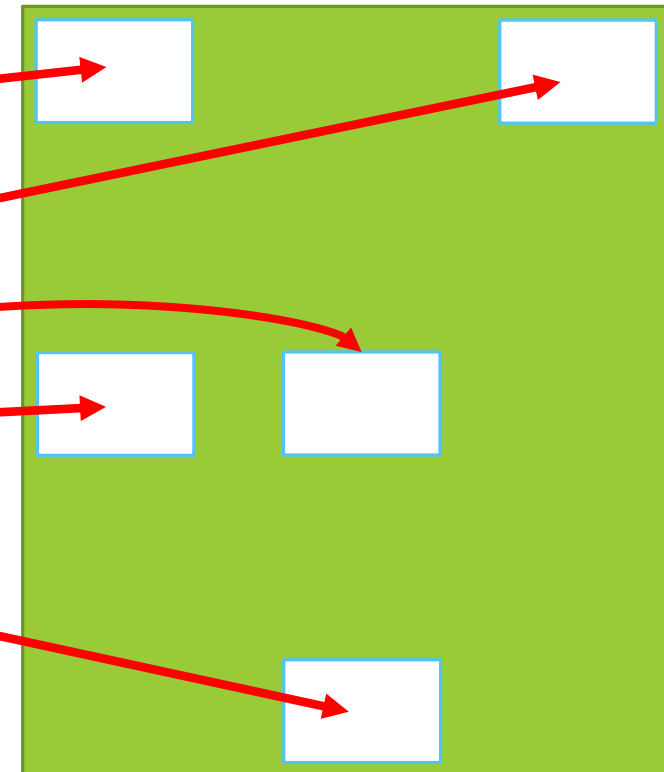
`android:layout_alignParentLeft`

`android:layout_alignParentRight`

`android:layout_centerInParent`

`android:layout_centerVertical`

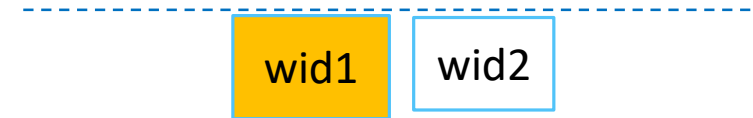
`android:layout_centerHorizontal`



WIDGETS AND LAYOUTS

(RelativeLayout - Referring to other **wid**gets)

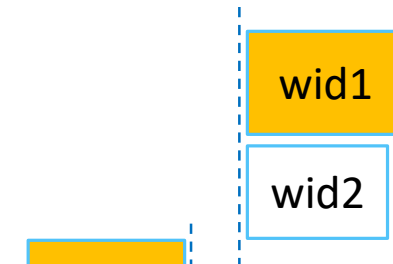
`android:layout_alignTop="@+id/wid1" →`



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

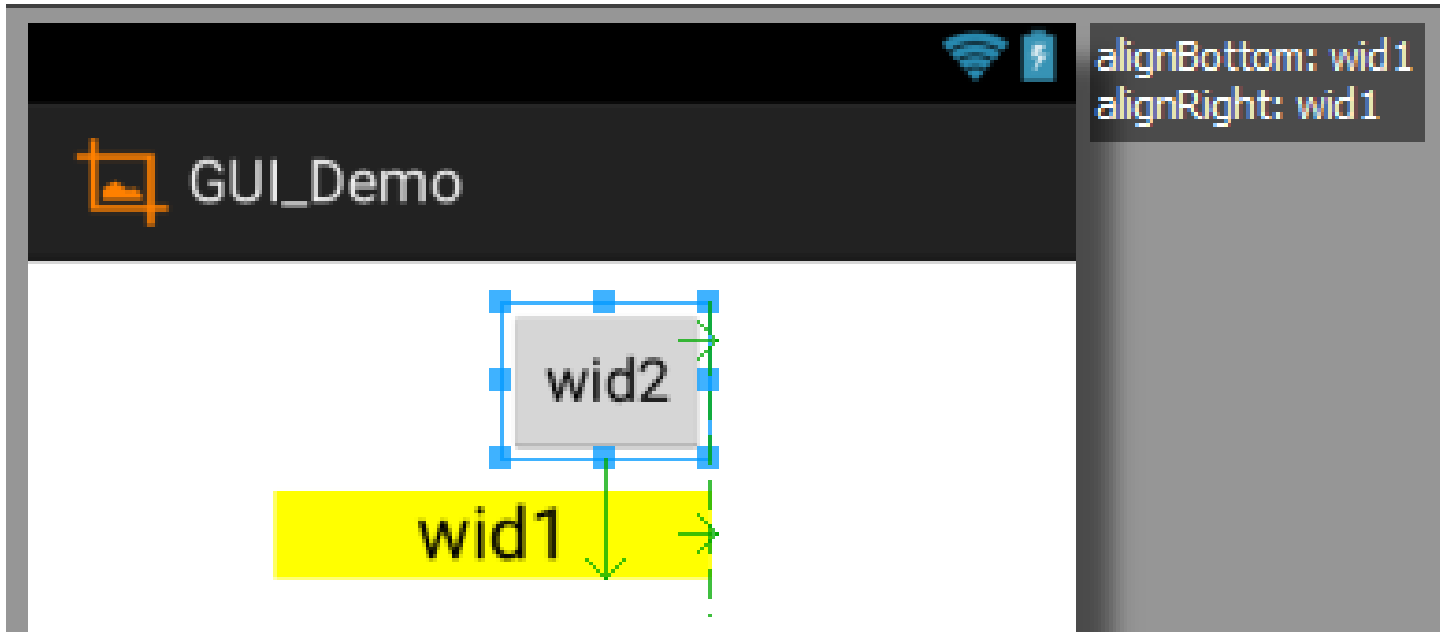


`android:layout_alignLeft="@+id/wid1" →`



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





WIDGETS AND LAYOUTS

(RelativeLayout - Referring to other widgets)

Example1: Image shows a screen designed with WYSIWYG Editor. We try to collocate the button identified as wid2. Observe its placement is visually described with (green) lines referencing the already drawn wid1 view. Both views have same bottom, same right, but wig2 has an elevation of 36dps respect wid1.

```
<Button android:id="@+id/wid2"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_alignBottom="@+id/wid1"  
        android:layout_alignRight="@+id/wid1"  
        android:layout_marginBottom="36dp"  
        android:text="@string/wid2"/>
```

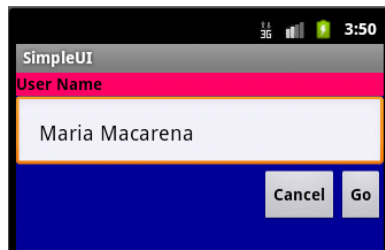
WIDGETS AND LAYOUTS

(RelativeLayout - Referring to other **wid**gets)

When using relative positioning you need to:

- Use identifiers (android:id attributes) on all elements that you will be referring to.
- XML elements are named using @+id/... E.g., an EditText could be called: android:id="@+id/txtUserName"
- Must refer only to widgets having been already defined. E.g., a new control to be positioned below txtUserName EditText could refer to it using android:layout_below="@+id/txtUserName"

```
<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"/>
```



```
<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"/>
<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 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"/>
</RelativeLayout>
```

WIDGETS AND LAYOUTS (TableLayout)

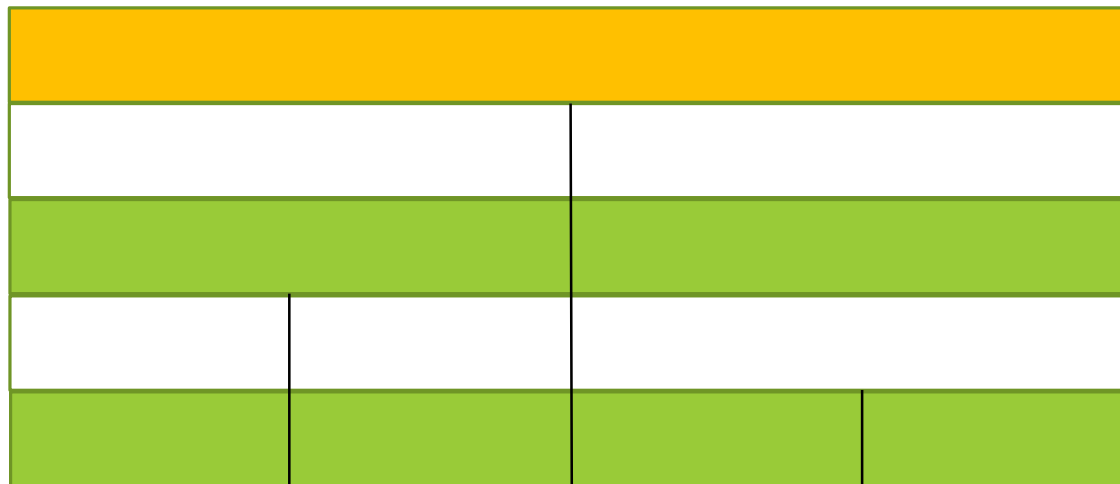
Android's TableLayout uses a grid template to position your widgets.

Like in a 2D matrix, cells in the grid are identified by rows and columns.

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 total of side-by-side widgets placed on the row.



The diagram illustrates a TableLayout grid with 5 rows and 4 columns. The first row is a single yellow cell spanning all 4 columns. The second row has two white cells: the first spans 2 columns, and the second spans 2 columns. The third row has two green cells, each spanning 2 columns. The fourth row has two white cells, each spanning 2 columns. The fifth row has four green cells, each spanning 1 column. A solid green bar is at the bottom of the slide.

WIDGETS AND LAYOUTS

(TableLayout - Setting number of columns)

The final number of columns in a table is determined by Android

Example: if your TableLayout have three rows

- one row with two widgets,
- one with three widgets, and
- 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

WIDGETS AND LAYOUTS (TableLayout – example)



Item	Calories	Price \$	
Big Mac	530	3.99	Buy
Filet-O-Fish	390	3.49	Buy
Cheeseburger	290	1.29	Buy

The screen shows various items from a McDonald's restaurant menu [*]. TableLayout has 4 TableRows, with 3 columns in the first row (labels) and 4 cells in each of the other 3 rows (item, Calories, Price, and Buy button)

```
<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 omitted --!>
</TableLayout>
```

[*] Reference: Pages visited on Sept 8, 2014

WIDGETS AND LAYOUTS

(TableLayout - 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>
```

WIDGETS AND LAYOUTS

(TableLayout - Stretching a column)

Widgets on a table's row are placed lexicographically from left to right, beginning with the first available column. Each column in the table stretches as needed to accommodate its occupants.

Example 4:

- The table shown below has four columns (indices: 0,1,2,3).
- The label ("ISBN") goes in the first column (index 0).
- 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).

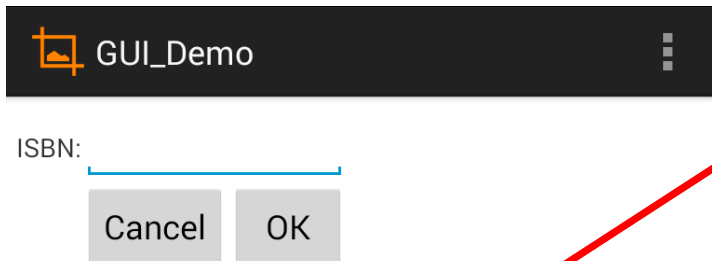
`android:layout_span="3"`

Label (ISBN)	EditText	EditText-span	EditText-span
<i>Column 0</i>	<i>Column 1</i>	<i>Column 2</i> Button Cancel	<i>Column 3</i> Button OK

`android:layout_column="2"`

WIDGETS AND LAYOUTS

(TableLayout - Stretching a column)



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

```
<?xml version="1.0" encoding="utf-8"?>
<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:padding="6dp" android:orientation="vertical" >
    <TableRow>
        <TextView android:text="ISBN:" />
        <EditText android:id="@+id/editISBN" android: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>
```

Occupy 3 columns

Skip columns 0, 1

WIDGETS AND LAYOUTS

(TableLayout - Stretching the entire table)

By default, a column is as wide as the “natural” size of the widest widget collocated in this column (e.g. a column holding a button showing the caption “Go” is narrower than other column holding a button with the caption “Cancel”).

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 (or comma-separated column indices) 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"
```

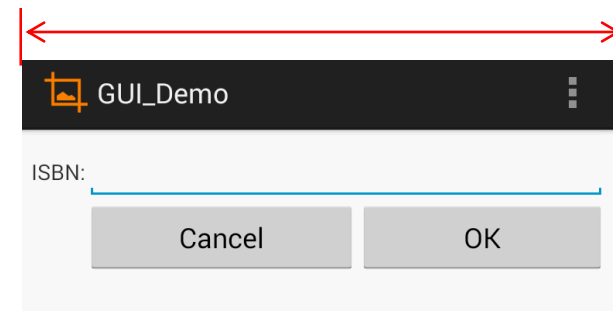
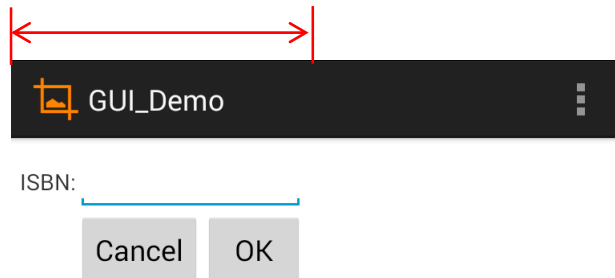
WIDGETS AND LAYOUTS

(TableLayout - Stretching the entire table)

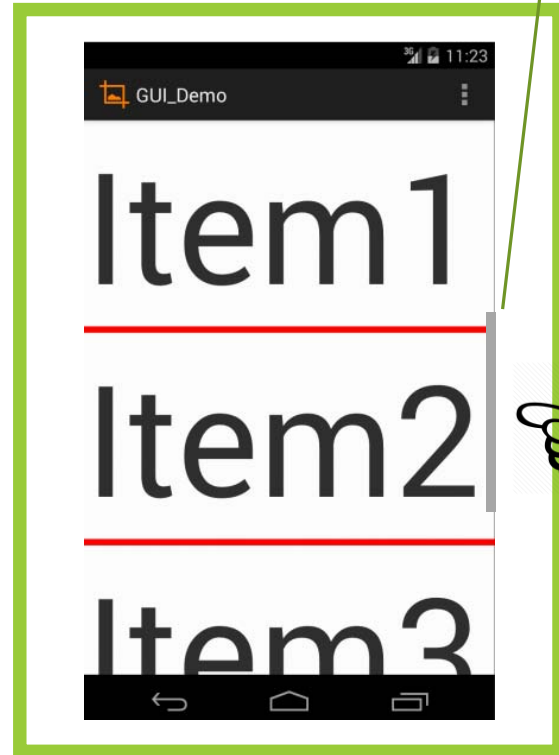
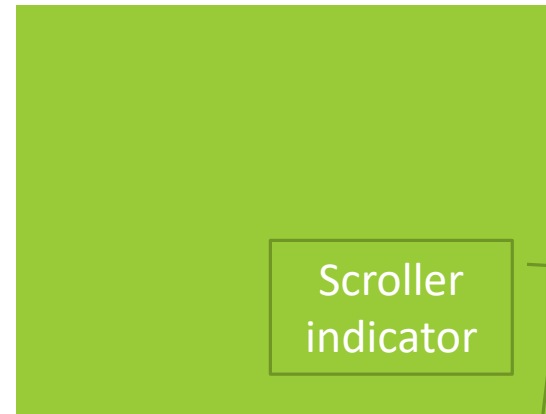
Example 4 created a table with four columns. We may elongate its columns 2, 3 to force the TableLayout to horizontally occupy the empty rest of the screen. Observe the use of the clause 'stretchColumns'

```
...  
<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



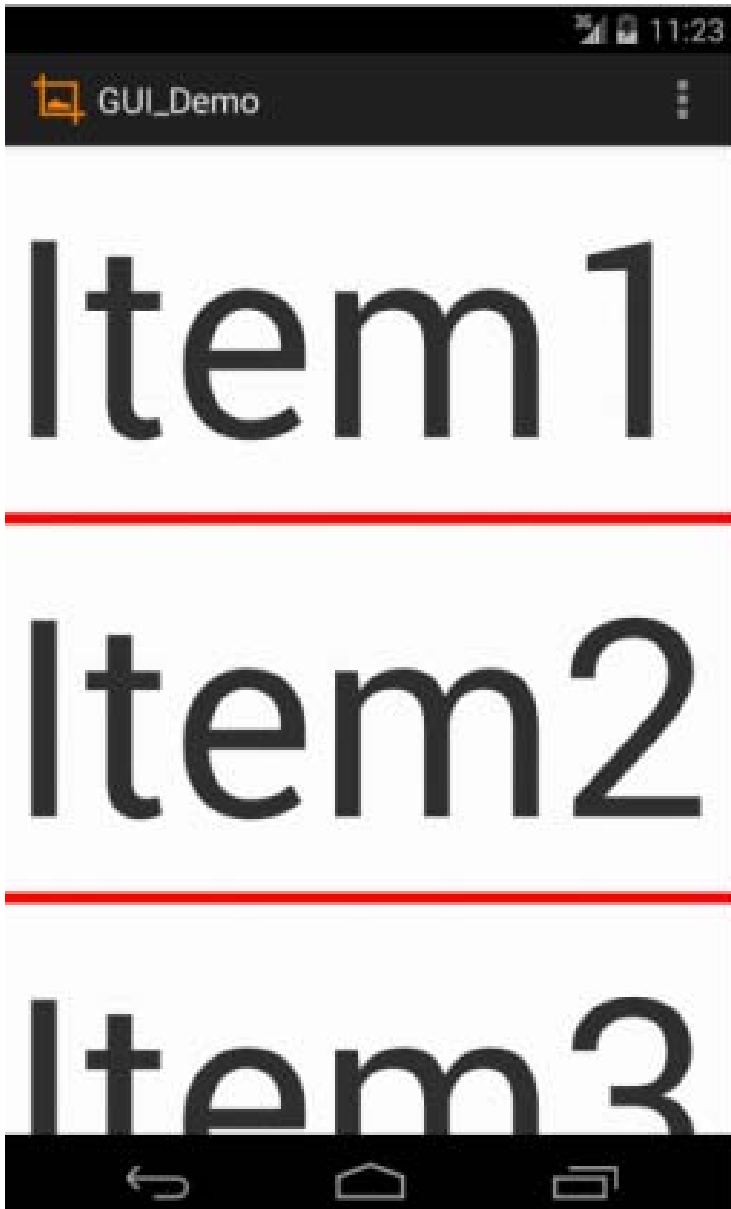
WIDGETS AND LAYOUTS (ScrollView layout - Vertical & horizontal)

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.



WIDGETS AND LAYOUTS

(ScrollView layout - Vertical & horizontal)

Vertical scrollView layout:

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myVerticalScrollView1" 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="Item1" android:textSize="150sp"/>
        <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="Item2" android:textSize="150sp"/>
        <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="Item3" android:textSize="150sp"/>
    </LinearLayout>
</ScrollView>
```

WIDGETS AND LAYOUTS

(ScrollView layout - Vertical & horizontal)

HorizontalScrollView layout:

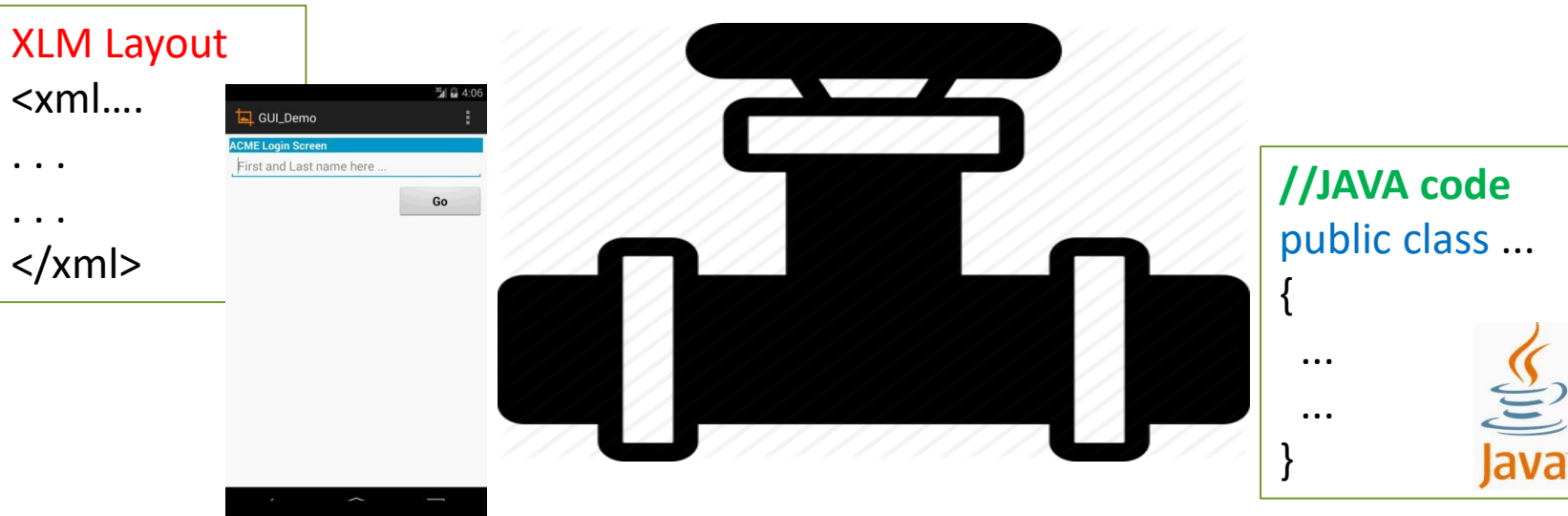


```
<HorizontalScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myVerticalScrollView1" android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout android:id="@+id/myLinearLayoutVertical" android:layout_width="match_parent"
        android:layout_height="match_parent" android:orientation="horizontal" >
        <TextView android:id="@+id/textView1" android:layout_width="match_parent"
            android:layout_height="wrap_content" android:text="Item1" android:textSize="150sp"/>
        <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="Item2" android:textSize="150sp"/>
        <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="Item3" android:textSize="150sp"/>
    </LinearLayout>
</HorizontalScrollView>
```

WIDGETS AND LAYOUTS

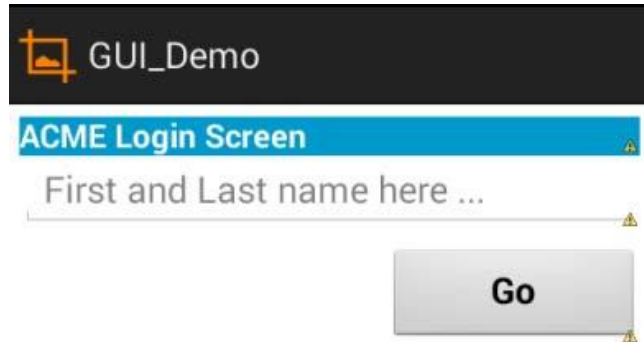
(Connecting layouts to java code)

You must 'connect' functional XML elements –such as buttons, text boxes, check boxes- with their equivalent Java objects. This is typically done in the onCreate(...) method of your main activity. After all the connections are made and programmed, your app should be ready to interact with the user.



WIDGETS AND LAYOUTS

(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>
```

```
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);
        ...
    }
    ...
}
```

WIDGETS AND LAYOUTS (Android context)

On Android, a Context defines a logical workspace on which an app can load and access resources.

- When a widget is created, it is attached to a particular Context. By means of its affiliation to that environment, it then could access other members of the hierarchy on which it has been collocated.
- For a simple 'one activity app' -say MainActivity- the method `getApplicationContext()` and the reference `MainActivity.this` return the same result.
- 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.

Assume the UI in `res/layout/activity_main.xml` has been created. This layout could be called by an application using the statement: `setContentView(R.layout.activity_main);`

Individual XML defined widgets, such as `btnGo` is later associated to the Java application using the statement `findViewById(...)` as in: `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).

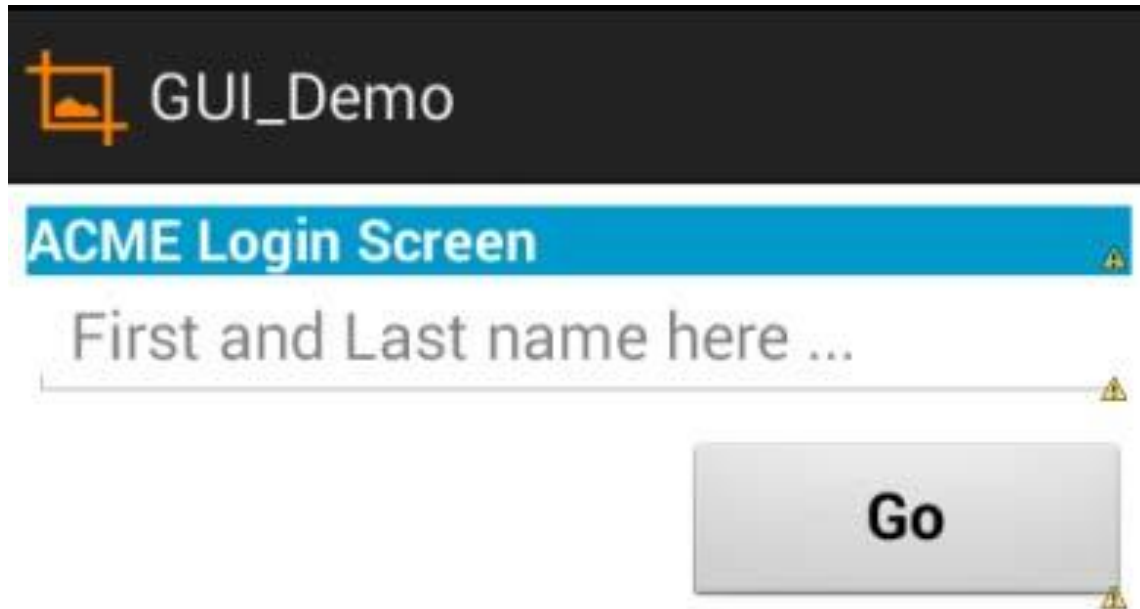
A suggestion: The widget's identifiers used in the XML layout and Java code could be the same. It is convenient to add a prefix to each identifier indicating its nature. Some options are `txt`, `btn`, `edt`, `rad`, `chk`, etc. Try to be consistent.

WIDGETS AND LAYOUTS

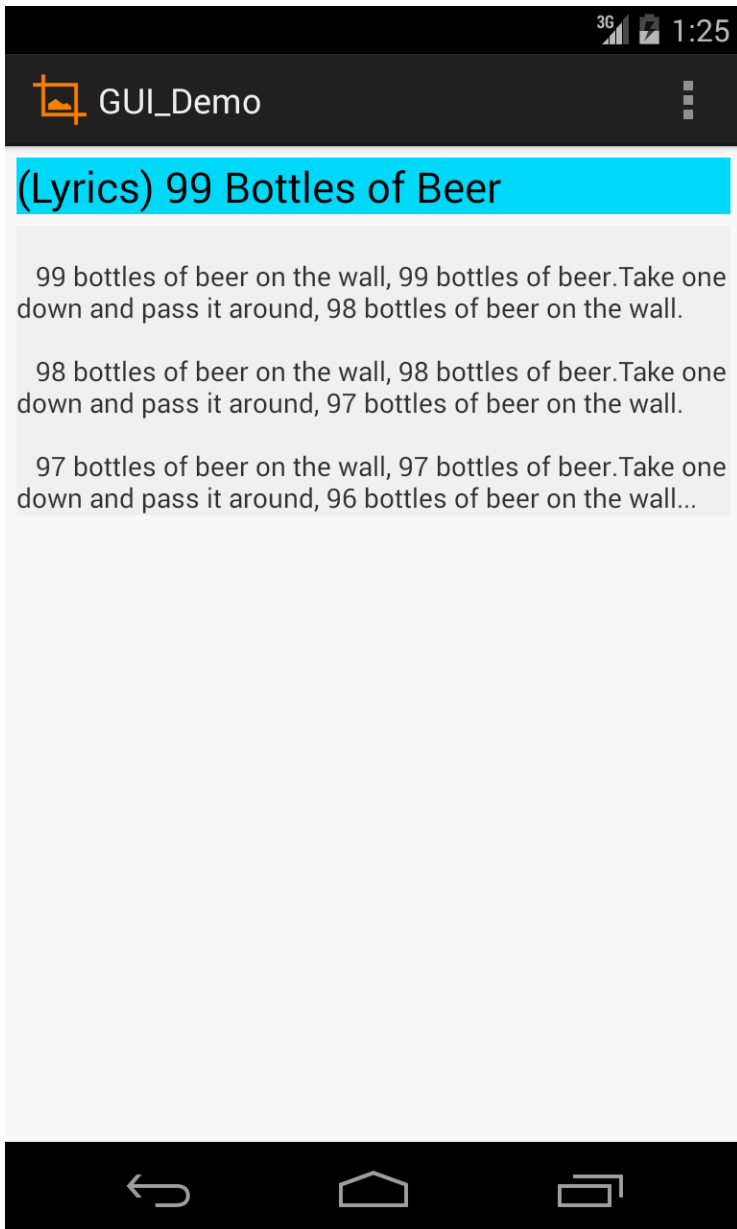
(Attaching listeners to widgets)

Consider the screen on the right. To make its 'Go' button widget be responsive to the user's pushing of that button, we may add a listener for the click event.

```
Button btnGo = (Button) findViewById(R.id.btnGo);
btnGo.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        // get userName and validate against some database
        // put some more logic here...
    }
});
```



Note: Other common 'listeners' watch for events such as: textChanged, tap, long-press, select, focus, etc.



WIDGETS AND LAYOUTS (TextView)

In Android a label or text-box is called a TextView.

A TextView is typically used for showing a caption or a text message.

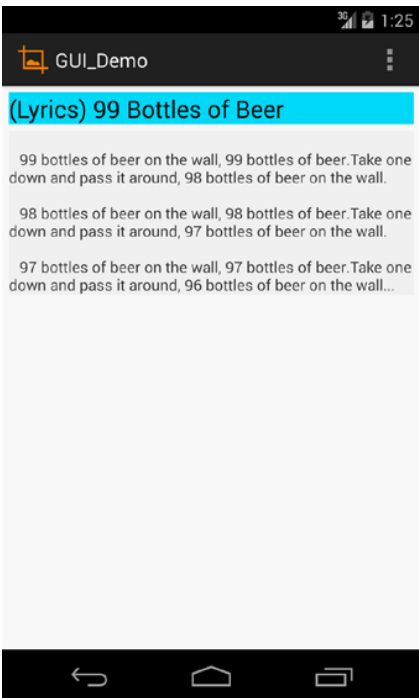
TextViews are not editable, therefore they take no input.

The text to be shown may include the `\n` formatting character (newline)

You may also use HTML formatting by setting the text to: `Html.fromHtml("bold string")`

WIDGETS AND LAYOUTS (Textview)

Example:



```
<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="6dp">
    <TextView android:id="@+id/textView1" android:layout_width="match_parent"
        android:layout_height="wrap_content" android:background="@color/holo_blue_bright"
        android:text="(Lyrics) 99 Bottles of Beer" android:textAppearance="?android:attr/textAppearanceLarge"/>
    <TextView android:id="@+id/textView2" android:layout_width="match_parent"
        android:layout_height="wrap_content" android:layout_marginTop="6dp"
        android:background="@color/gray_light"
        android:text="\n99 bottles of beer on the wall, 99 bottles of beer.Take one down and pass it around, 98 bottles of beer on the wall.\n\n98 bottles of beer on the wall, 98 bottles of beer.Take one down and pass it around, 97 bottles of beer on the wall.\n\n97 bottles of beer on the wall, 97 bottles of beer.Take one down and pass it around, 96 bottles of beer on the wall..."
        android:textSize="14sp"/>
</LinearLayout>
```

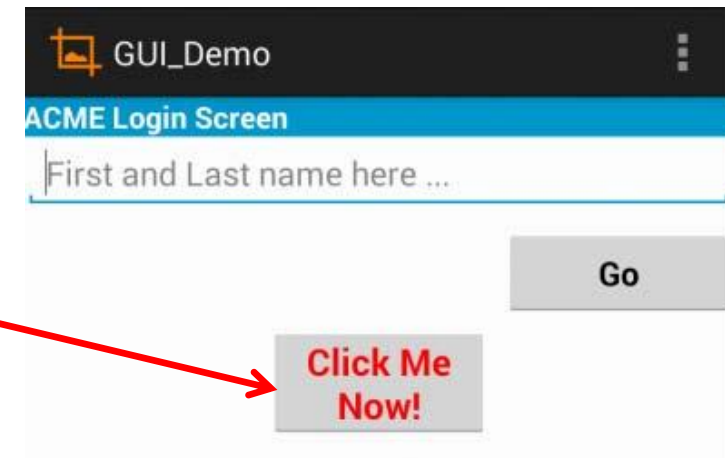

WIDGETS AND LAYOUTS (Button)

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

Button is a subclass of TextView. Therefore formatting a button's face is like the setting of a TextView.

You may alter the default behavior of a button by providing a custom drawable.xml specification to be applied as background. In those specs you indicate the shape, color, border, corners, gradient, and behavior based on states (pressed, focused). More on this issue in the appendix.

```
<Button android:id="@+id/btnClickMeNow"  
    android:layout_width="120dp" android:layout_height="wrap_content"  
    android:layout_gravity="center" android:layout_marginTop="5dp"  
    android:gravity="center" android:padding="5dp"  
    android:text="Click Me Now!" android:textColor="#ffff0000"  
    android:textSize="20sp" android:textStyle="bold"/>
```

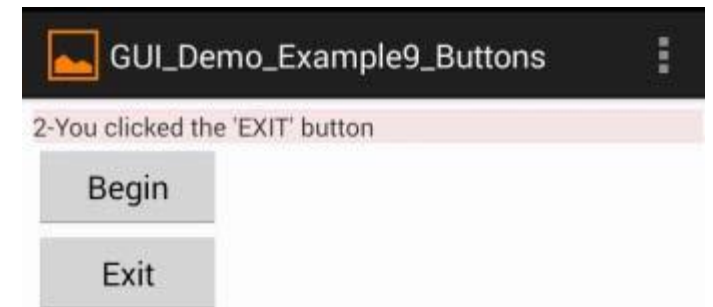


WIDGETS AND LAYOUTS

(Connecting multiple buttons)

```
public class MainActivity extends Activity implements OnClickListener {
    TextView txtMsg;
    Button btnBegin, btnExit;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main );
        txtMsg = (TextView) findViewById(R.id.txtMsg);
        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()) txtMsg.setText("1-You clicked the 'BEGIN' button");
        if (v.getId() == btnExit.getId()) txtMsg.setText("2-You clicked the 'EXIT' button");
    } //onClick
}
```

This example shows an alternative way of wiring-up multiple buttons. Observe how the main activity implements the `OnClickListener` interface. The mandatory `onClick` method checks which of the many buttons sent the signal and proceeds from there.



WIDGETS AND LAYOUTS

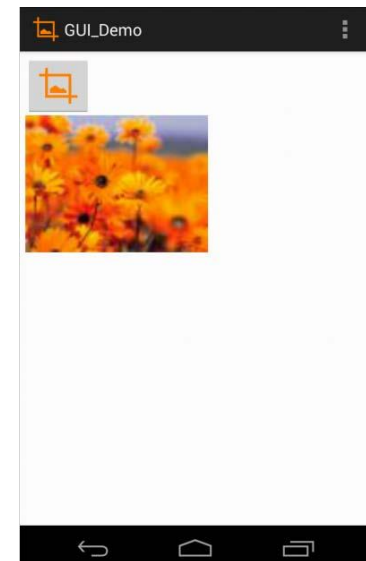
(ImageView and ImageButton)

ImageView and ImageButton allow the embedding of images in your applications (gif, jpg, png, etc).

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

Pictures are stored in “res/drawable” (medium, high, x-high, xx-high, and xxx-high respectively definition version of the same image could be stored for later usage with different types of screens).

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent"
    android:padding="6dp" android:orientation="vertical">
    <ImageButton android:id="@+id/imgButton1"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:src="@drawable/ic_launcher"/>
    <ImageView android:id="@+id/imgView1"
        android:layout_width="200dp" android:layout_height="150dp"
        android:scaleType="fitXY" android:src="@drawable/flowers1"/>
</LinearLayout>
```

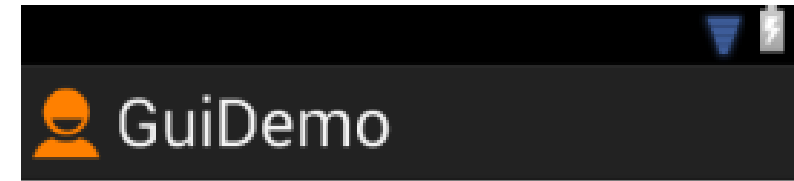


WIDGETS AND LAYOUTS

(Buttons - combining images & text)

A common Button widget 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_launcher"
    android:gravity="left|center_vertical"
    android:padding="15dp"
    android:text="Click me"/>
</LinearLayout>
```

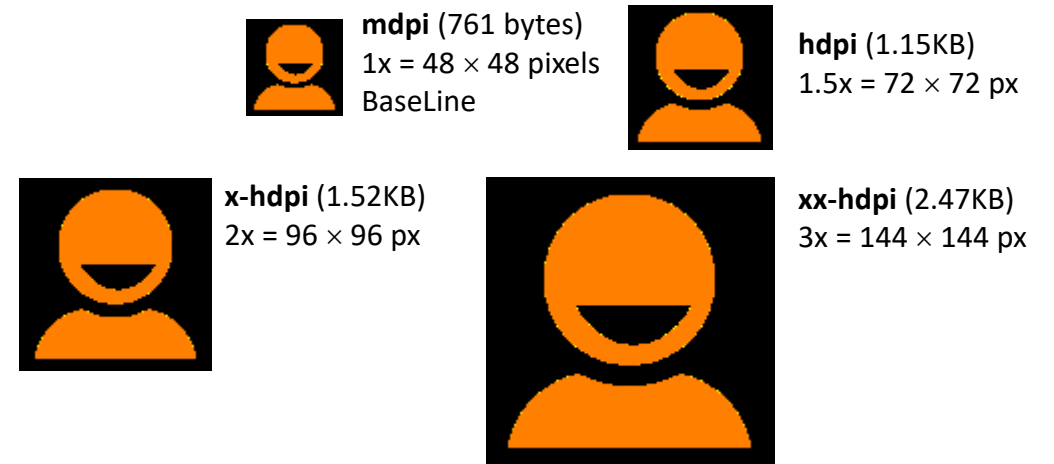


WIDGETS AND LAYOUTS

(How icons are used in Android?)

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

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



HINT: Several websites allow you to convert for free your pictures to image-files under a variety of formats and sizes such as png, .jpg, .gif, etc. For instance try:
<http://www.prodraw.net/favicon/index.php>



WIDGETS AND LAYOUTS

(EditText boxe)

The EditText widget is an extension of TextView that allows user's input.

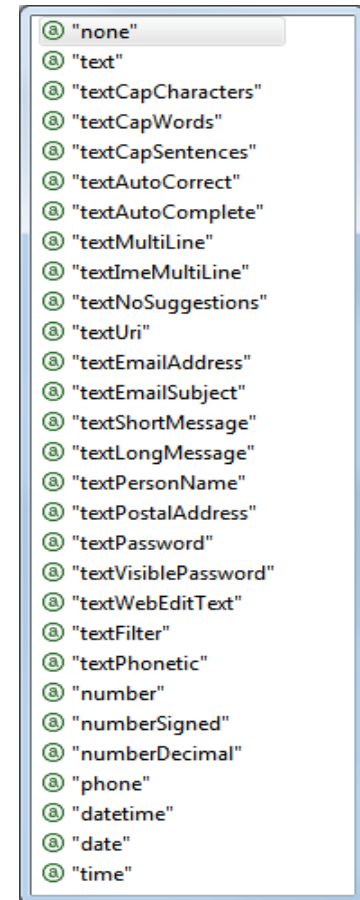
In addition to plain text, this widget can display editable text formatted with HTML-styles such as bold, italics, underline, etc). This is done with `Html.fromHtml(html_text)`

Moving data in and out of an EditText box is usually done in Java through the following methods: `txtBox.setText("someValue")` and `txtBox.getText().toString()`

WIDGETS AND LAYOUTS (EditText boxe)

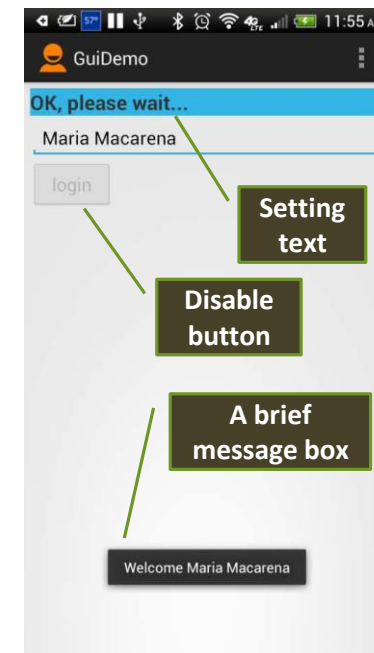
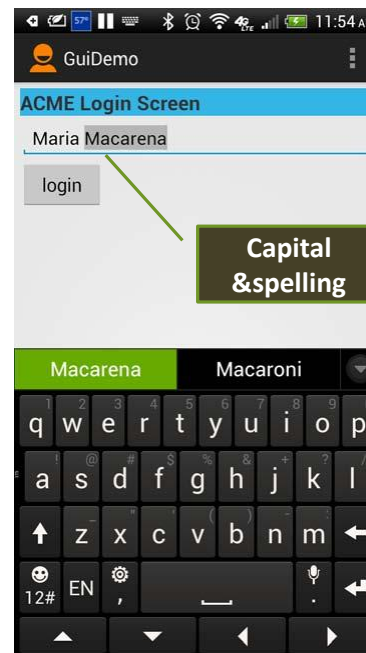
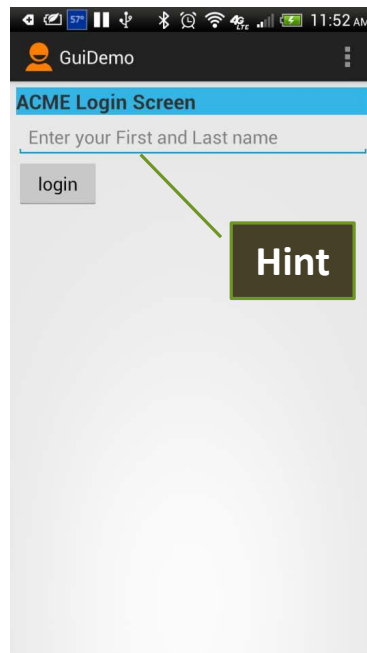
An EditText box could be set to accept input strings satisfying a particular pattern such as: numbers (with and without decimals or sign), phones, dates, times, uris, etc.

Setting the EditText box to accept a particular choice of data-type, is done through the XML clause `android:inputType="choices"`, where choices include any of the single values shown in the figure. You may combine types, for instance: `textCapWords|textAutoCorrect` Accepts text that capitalizes every word, incorrect words are automatically changed (for instance 'the' is converted into 'The', and so on)



SOME EXAMPLES (Login-screen)

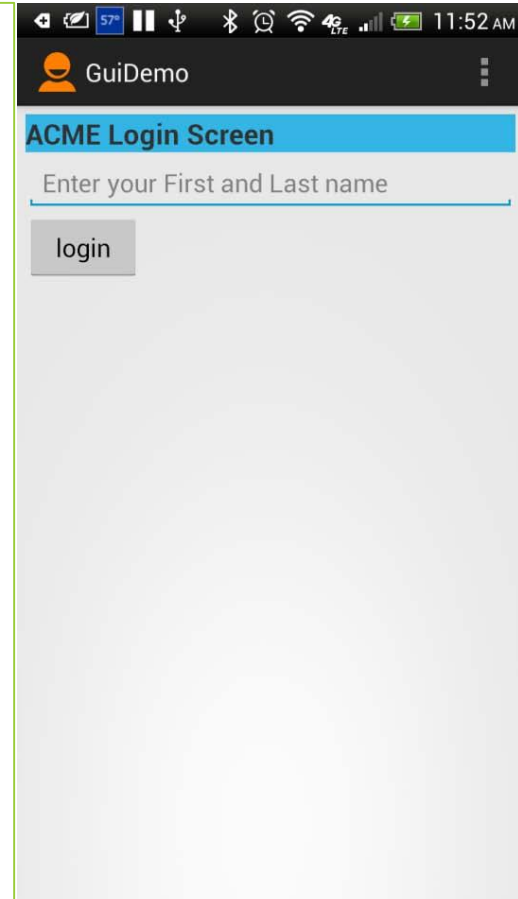
In this example we will create a simple login screen holding a label (TextView), a text box (EditText), and a Button. When the EditText box gains focus, the system provides a virtual keyboard customized to the input-type given to the entry box (capitals & spelling). Clicking the button displays a Toast-message that echoes the supplied user-name.



SOME EXAMPLES

(Login-screen layout)

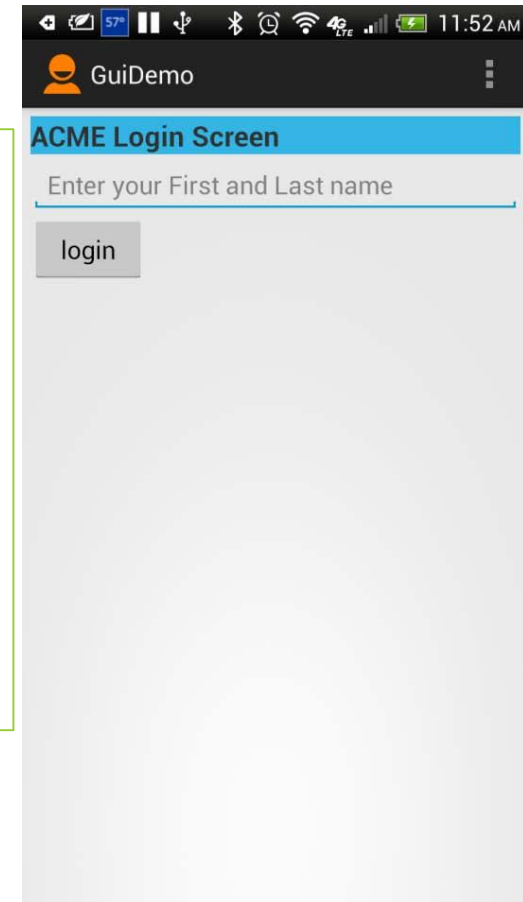
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent"
    android:orientation="vertical" android:padding="6dp">
    <TextView android:id="@+id/txtLogin"
        android:layout_width="match_parent" android:layout_height="wrap_content"
        android:background="@android:color/holo_blue_light" android:text="@string/ACME_Login_Screen"
        android:textSize="20sp" android:textStyle="bold"/>
    <EditText android:id="@+id/edtUserName"
        android:layout_width="match_parent" android:layout_height="wrap_content"
        android:layout_marginTop="2dp" android:hint="@string/Enter_your_First_and_Last_name"
        android:inputType="textCapWords|textAutoCorrect" android:textSize="18sp">
        <requestFocus/>
    </EditText>
    <Button android:id="@+id/btnLogin"
        android:layout_width="82dp" android:layout_height="wrap_content"
        android:layout_marginTop="2dp" android:text="@string/login"/>
</LinearLayout>
```



SOME EXAMPLES

(Login-screen layout - res/values/strings.xml)

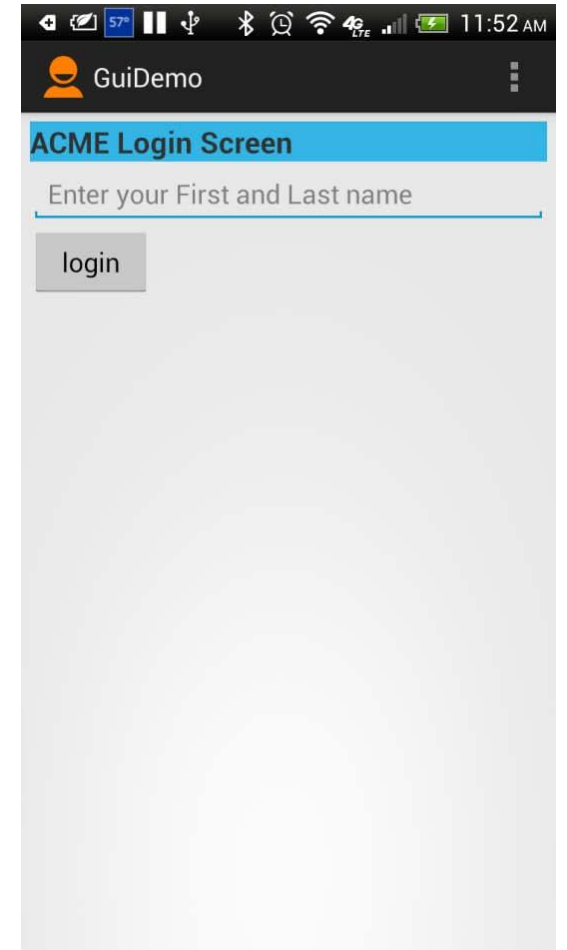
```
<?xml version="1.0" encoding="utf-8"?>
<!-- this is the res/values/strings.xml file -->
<resources>
  <string name="app_name">GuiDemo</string>
  <string name="action_settings">Settings</string>
  <string name="login">login</string>
  <string name="ACME_Login_Screen">ACME Login Screen</string>
  <string name="Enter_your_First_and_Last_name">Enter your First and Last name</string>
</resources>
```



SOME EXAMPLES

(Login-screen MainActivity)

```
public class MainActivity extends ActionBarActivity {
    TextView txtLogin; EditText edtUserName; Button btnLogin;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        txtLogin = (TextView) findViewById(R.id.txtLogin);
        edtUserName = (EditText) findViewById(R.id.edtUserName);
        btnLogin = (Button) findViewById(R.id.btnLogin);
        btnLogin.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                String userName = edtUserName.getText().toString();
                if (userName.equals("Maria Macarena")) {
                    txtLogin.setText("OK, please wait...");
                    Toast.makeText(getApplicationContext(), "Welcome " + userName, Toast.LENGTH_SHORT).show();
                    btnLogin.setEnabled(false);
                }
                else Toast.makeText(getApplicationContext(), userName + " is not a valid USER", Toast.LENGTH_SHORT).show();
            }
        }); // onClick
    } // onCreate
    ...
}
```



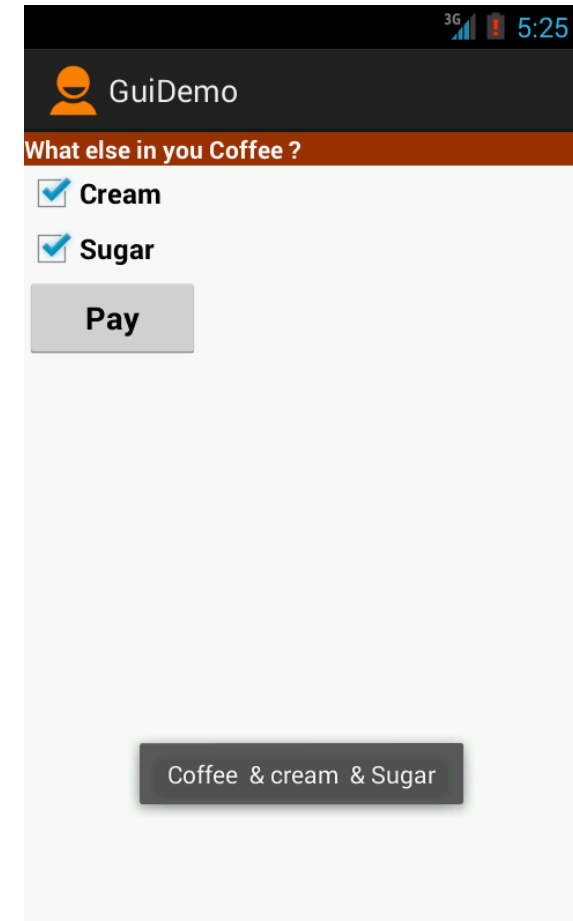
WIDGETS AND LAYOUTS (CheckBoxes)

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

A screen may include any number of mutually inclusive (independent) CheckBoxes. At anytime, more than one CheckBox in the GUI could be checked.

In our “CaféApp” example, the screen on the right displays two CheckBox controls, they are used for selecting ‘Cream’ and ‘Sugar’ options. In this image both boxes are ‘checked’.

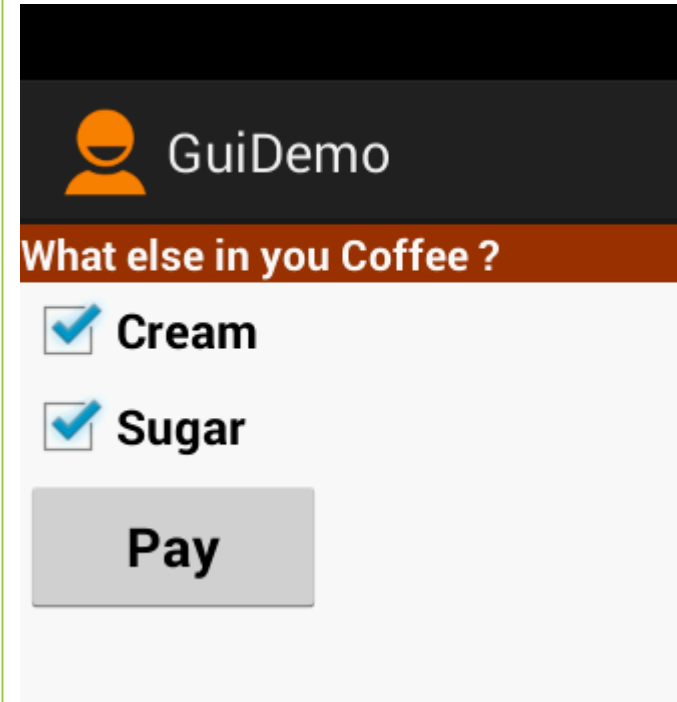
When the user pushes the ‘Pay’ button a Toast-message is issue echoing the current combination of choices held by the checkboxes.



SOME EXAMPLES

(CaféApp - layout)

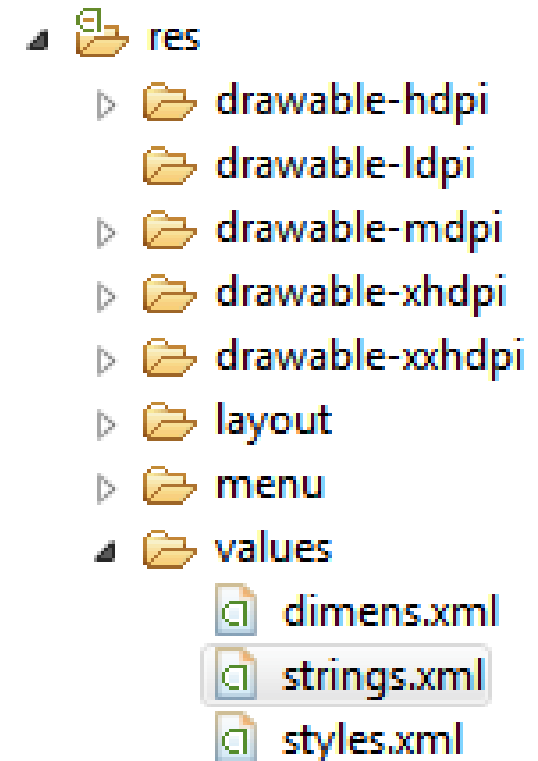
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent"
    android:padding="6dp" android:orientation="vertical">
    <TextView android:id="@+id/labelCoffee"
        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" />
    <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>
```



SOME EXAMPLES

(CaféApp - res/values/strings)

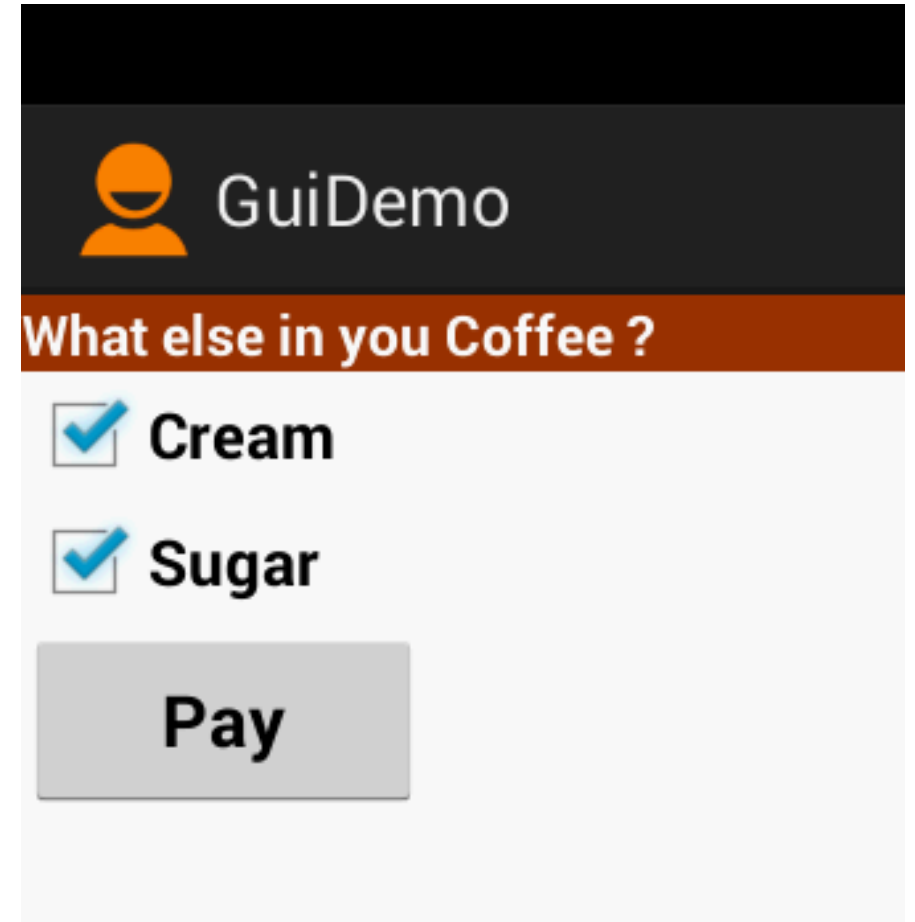
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">GuiDemo</string>
  <string name="action_settings">Settings</string>
  <string name="click_me">Click Me</string>
  <string name="sugar">Sugar</string>
  <string name="cream">Cream</string>
  <string name="coffee_addons">What else in your coffee?</string>
  <string name="pay">Pay</string>
</resources>
```



SOME EXAMPLES

(CaféApp MainActivity)

```
public class MainActivity extends Activity {
    CheckBox chkCream, 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);
        //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
    ...
}
```



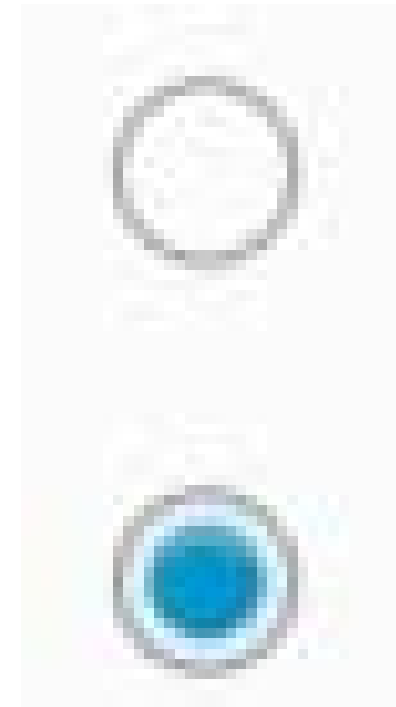
WIDGETS AND LAYOUTS (RadioButtons)

A radio button (like a CheckBox) is a two-states button that can be either checked or unchecked.

Logically related radio buttons are normally put together in a RadioGroup container. The container forces the enclosed radio buttons to behave as mutually exclusive selectors. That is, the checking of one radio button unchecks all the others.

Properties for font face, style, color, etc. are managed in a way similar to setting a TextView.

You may call the method `isChecked()` to see if a specific RadioButton is selected or change its state by calling `toggle()`.



SOME EXAMPLES

(CaféApp - layout)

Example

We extend the previous CaféApp example by adding a RadioGroup control that allows the user to pick one type of coffee from three available options.

RadioGroup

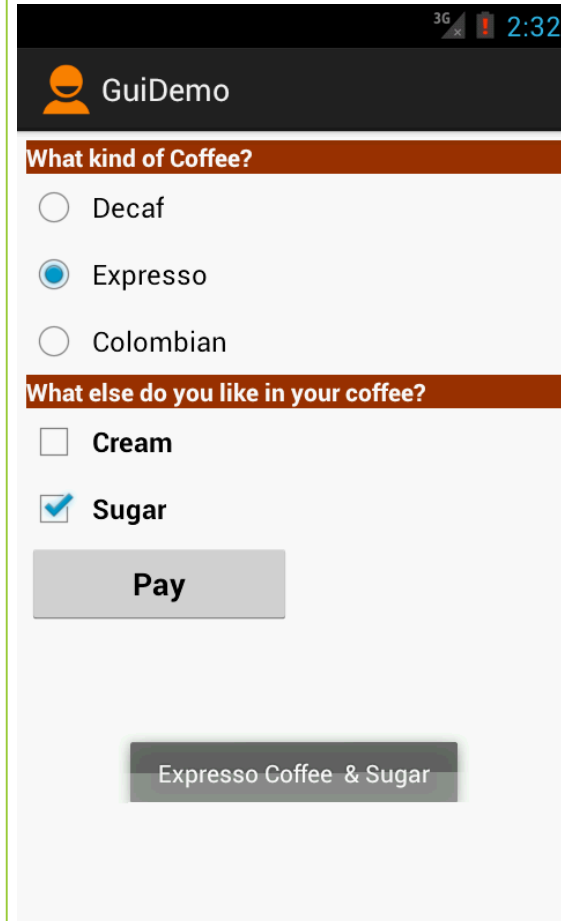
Summary of choices

The screenshot shows a mobile application interface titled "GuiDemo". It features two sections for user input. The first section, titled "What kind of Coffee?", contains three radio button options: "Decaf", "Espresso" (which is selected), and "Colombian". The second section, titled "What else do you like in your coffee?", contains two checkbox options: "Cream" (unchecked) and "Sugar" (checked). Below these sections is a "Pay" button. At the bottom of the screen, a summary bar displays the selected options: "Espresso Coffee & Sugar".

SOME EXAMPLES

(CaféApp - layout)

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



SOME EXAMPLES (CaféApp - MainActivity)

```
public class MainActivity extends Activity {
    CheckBox chkCream, chkSugar; Button btnPay;
    RadioGroup radCoffeeType; RadioButton radDecaf, radEspresso, 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);
        radEspresso = (RadioButton) findViewById(R.id.radEspresso);
        radColombian = (RadioButton) findViewById(R.id.radColombian);
```



```
        btnPay.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                String msg = "Coffee ";
                if (chkCream.isChecked()) msg += " & cream ";
                if (chkSugar.isChecked()) msg += " & Sugar ";
                // get selected radio button 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 (radEspresso.isChecked()) msg = "Espresso " + 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
```

GuiDemo

What kind of Coffee?

☐ Decaf

☒ Espresso

☐ Colombian

What else do you like in your coffee?

☐ Cream

☒ Sugar

Pay

Espresso Coffee & Sugar

WIDGETS AND LAYOUTS (Miscellaneous)

XML Controls the focus sequence:

- `android:visibility` `true/false` set visibility
- `android:background` `color, image, drawable`
- `<requestFocus/>` react to user's interaction

Java methods:

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

WIDGETS AND LAYOUTS

(Using the @string resource)

A good programming practice in Android is NOT to directly enter literal strings as immediate values for attribute inside xml files.

For example, if you are defining a TextView to show a company headquarter's location, a clause such as `android:text="Cleveland"` should not be used (observe it produces a Warning [I18N] Hardcoded string "Cleveland", should use @string resource)

Instead you should apply a two steps procedure in which

1. You write the literal string –say *headquarter* – in `res/values/string.xml`. Enter `<string name="headquarter">Cleveland</string>`
2. Whenever the string is needed provide a reference to the string using the notation `@string/headquarter`. For instance in our example you should enter `android:text="@string/headquarter"`

WHY?

If the string is used in many places and its actual value changes we just update the resource file entry once. It also provides some support for internationalization -easy to change a resource string from one language to another.

WIDGETS AND LAYOUTS (Android asset studio)

Link: <http://romannurik.github.io/AndroidAssetStudio/> [Visited on 9/14/2014]

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

WIDGETS AND LAYOUTS

(Measuring graphic elements)

Q. What is dpi (also know as dp and ppi)?: Stands for dots per inch. It suggests a measure of screen quality. You can compute it using the following formula:

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

Example:

- G1 (base device 320 × 480) 155.92 dpi (3.7 in diagonally)
- Nexus (480 × 800) 252.15 dpi
- HTC One (1080 × 1920) 468 dpi (4.7 in)
- Samsung S4 (1080 × 1920) 441 dpi (5.5 in)

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

- dp: 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 on a 160 dpi screen. Use it for measuring anything but fonts.
- sp: Scale-independent Pixels – similar to the relative density dp unit but used for font size preference.

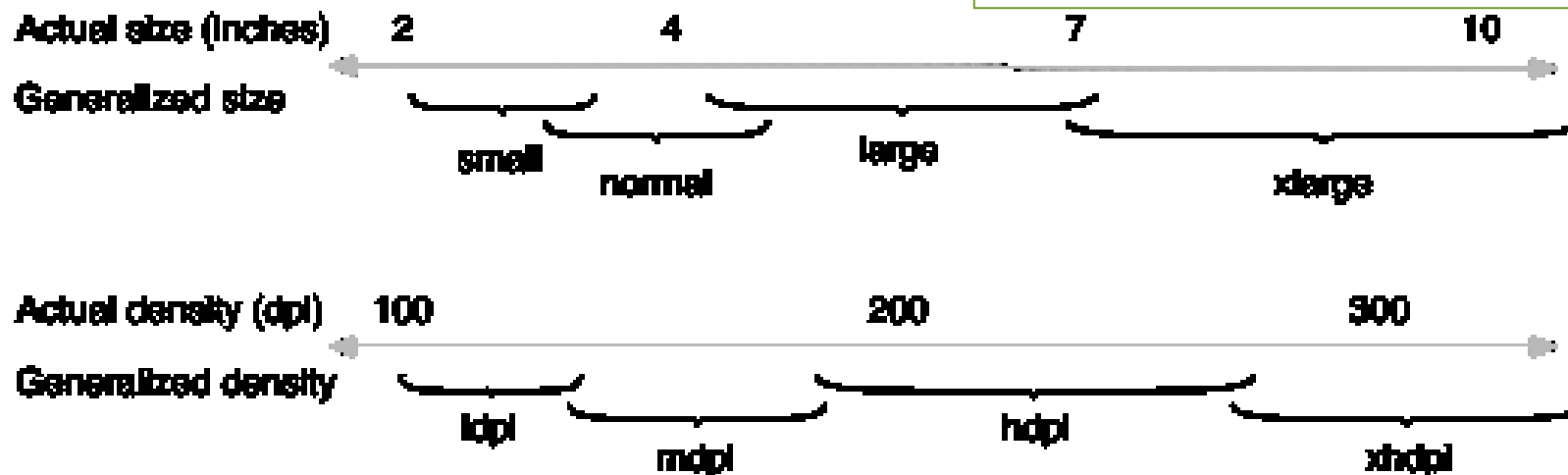
WIDGETS AND LAYOUTS

(Measuring graphic elements)

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

A set of four generalized screen sizes
xlarge screens are at least 960dp × 720dp
large screens are at least 640dp × 480dp
normal screens are at least 470dp × 320dp
small screens are at least 426dp × 320dp

A set of six generalized densities:
ldpi ~120dpi (low)
mdpi ~160dpi (medium)
hdpi ~240dpi (high)
xhdpi ~320dpi (extra-high)
xxhdpi ~480dpi (extra-extra-high)
xxxhdpi ~640dpi (extra-extra-extra-high)



WIDGETS AND LAYOUTS

(Measuring graphic elements)

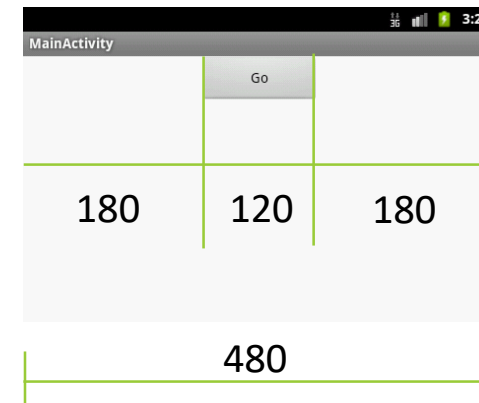
Q. Give me an example on how to use dp units.

Assume you design your interface for a G1 phone having 320x480 pixels (Abstracted density is 160 – See your AVD entry, the actual pixeling is defined as: $[2 \times 160] \times [3 \times 160]$)

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

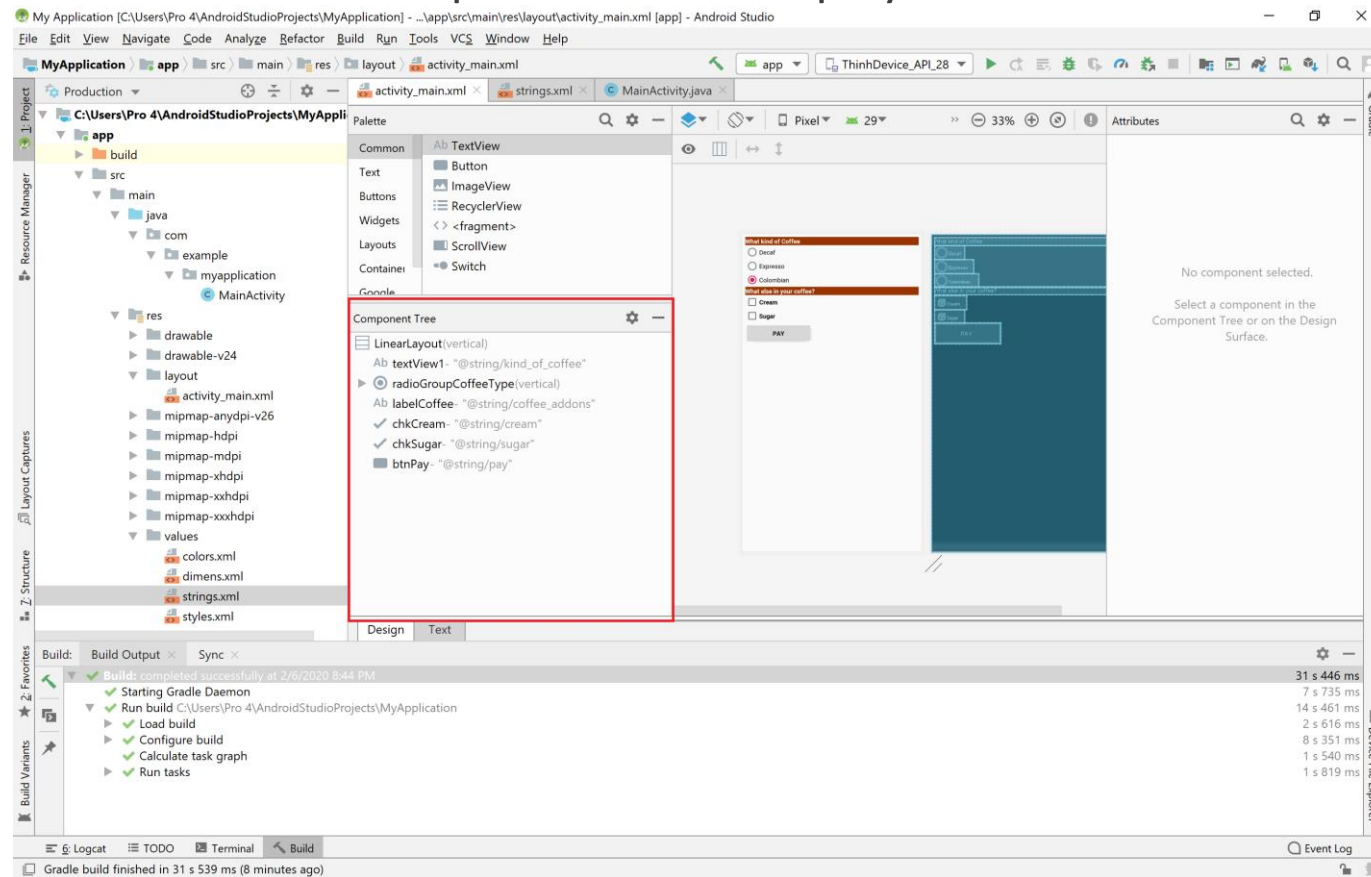
- `<Button`
 - `android:id="@+id/button1"`
 - `android:layout_height="wrap_content"`
 - `android:layout_width="120dp"`
 - `android:layout_gravity="center"`
 - `android:text="@+id/go_caption"/>`

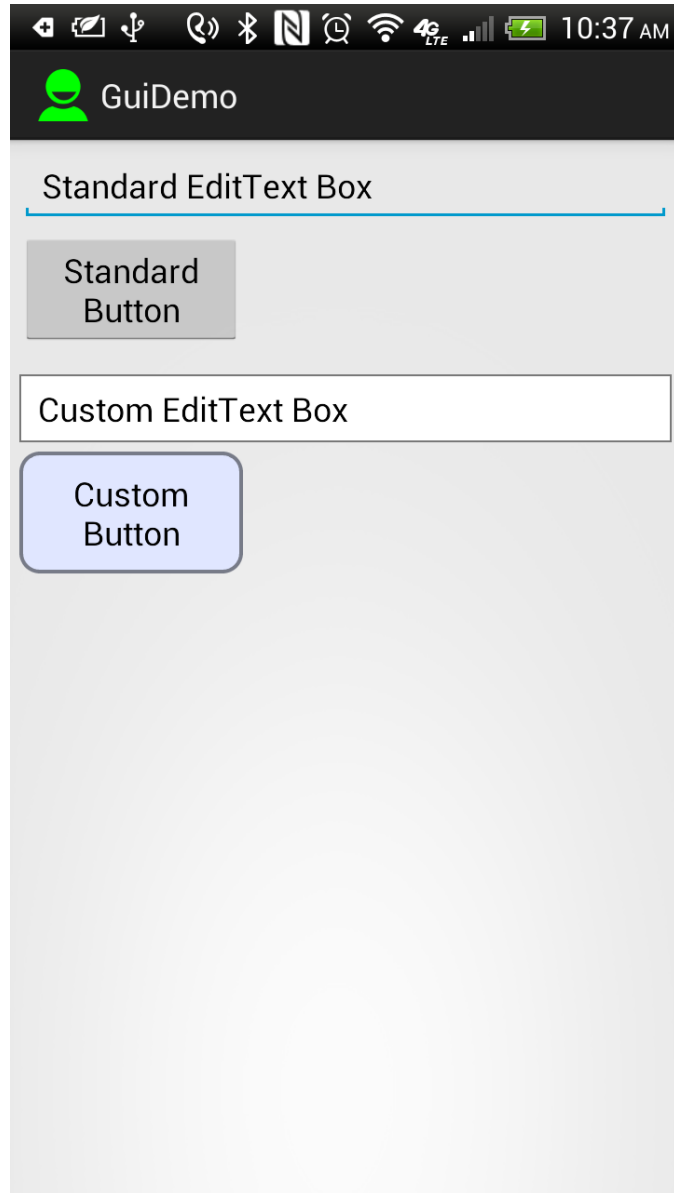


If the application is deployed on devices having a higher resolution the button is still mapped to the middle of the screen.

WIDGETS AND LAYOUTS (Hierarchy viewer tool)

The Component Tree section allows exploration of a displayed UI





WIDGETS AND LAYOUTS

(Customizing widgets)

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.

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

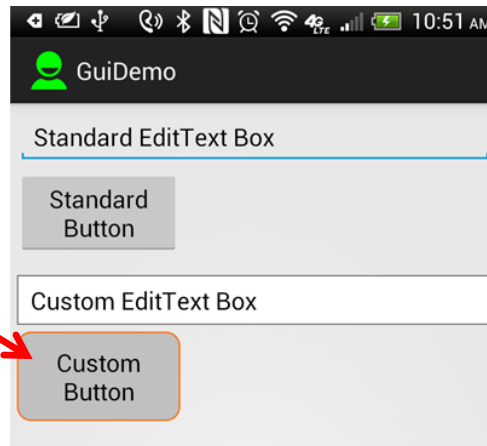
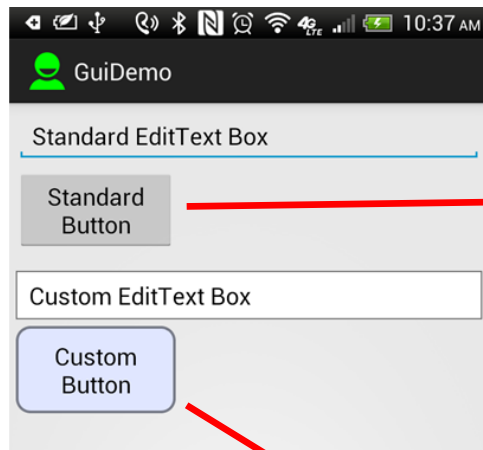
In addition to visual changes, the widget's reaction to user interaction could be adjusted for events such as: Focused, Clicked, etc.

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

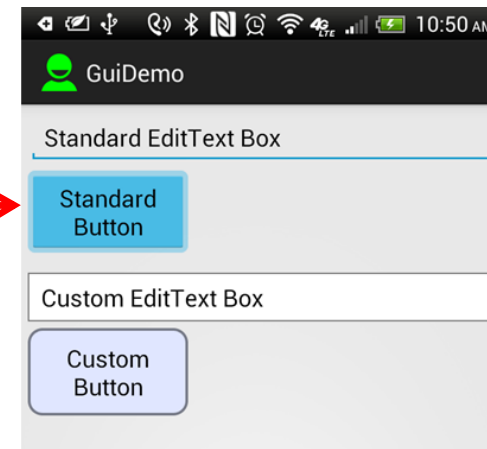
WIDGETS AND LAYOUTS

(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



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

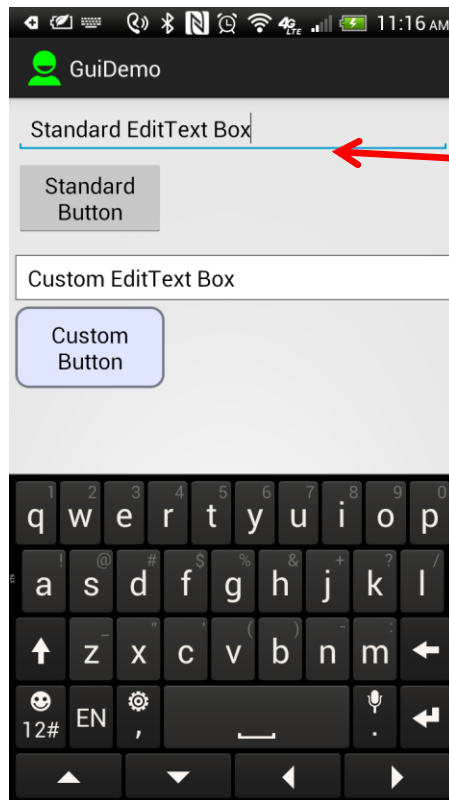


Standard behavior – buttons turns blue when it is pressed.

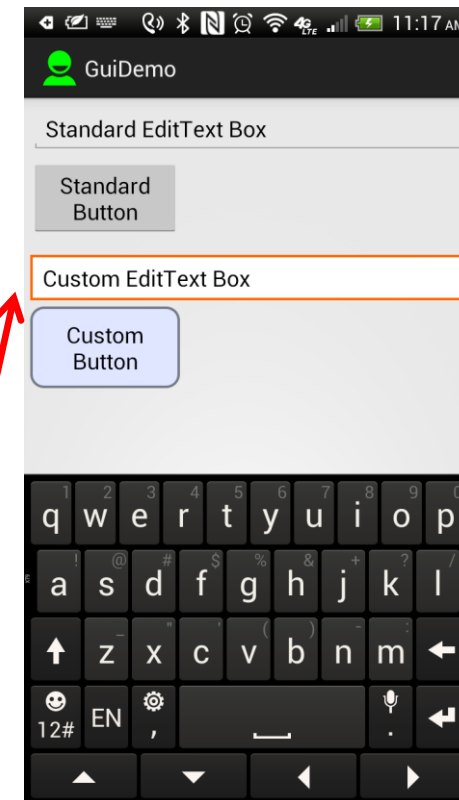
WIDGETS AND LAYOUTS

(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

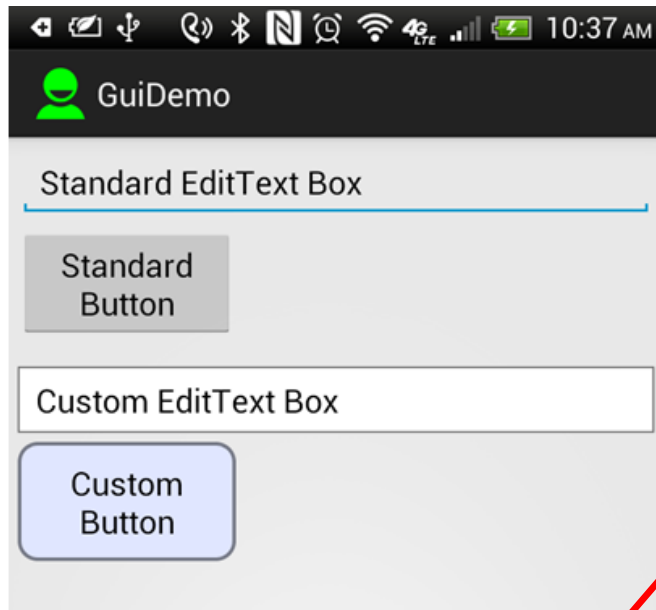


A focused
custom box
shows an orange
all-around frame

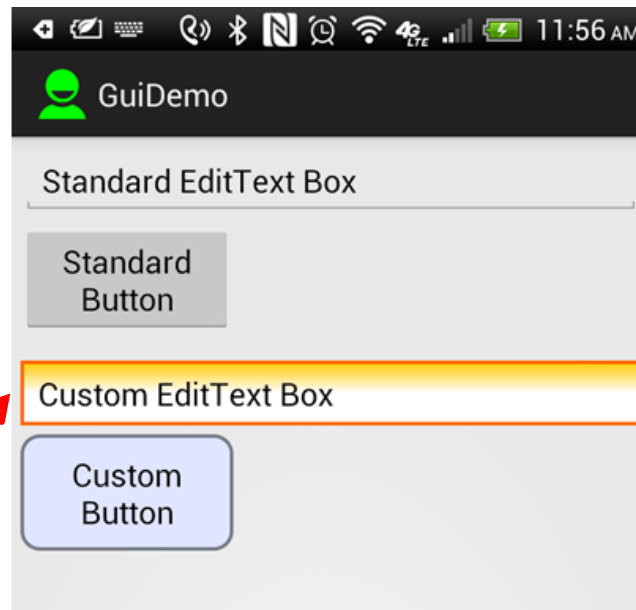
WIDGETS AND LAYOUTS

(Customizing widgets)

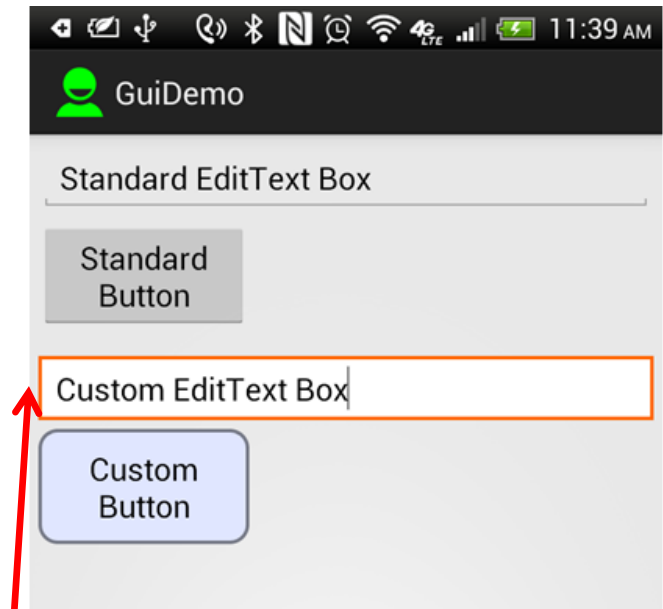
When the user taps on the custom made EditText box a gradient is applied to the box to flash a visual feedback reassuring the user of her selection.



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

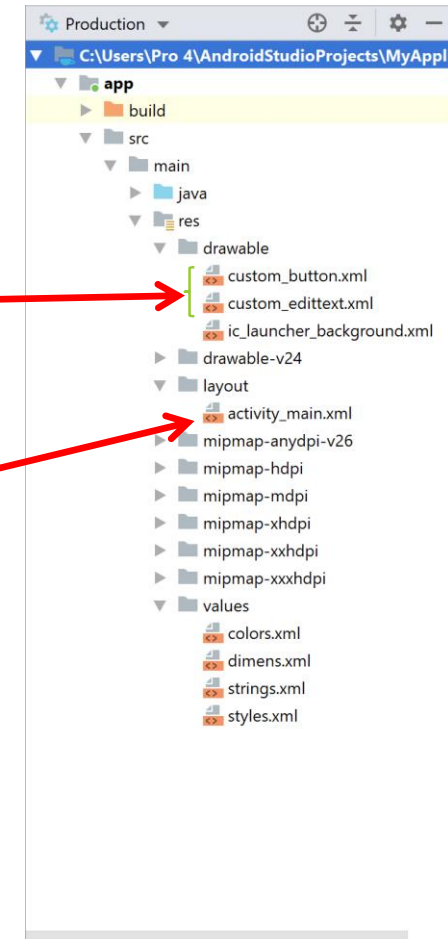
WIDGETS AND LAYOUTS

(Customizing widgets)

Organizing the application

Definition of the custom templates for
Button and EditText widgets

Layout referencing standard and custom
made widgets

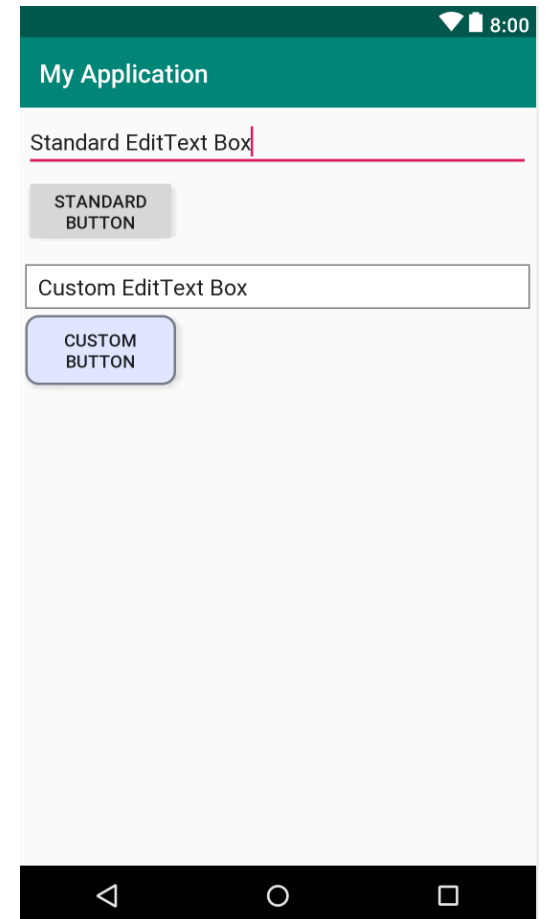


WIDGETS AND LAYOUTS

(Customizing widgets)

Activity Layout

```
<?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"/>
    <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>
```

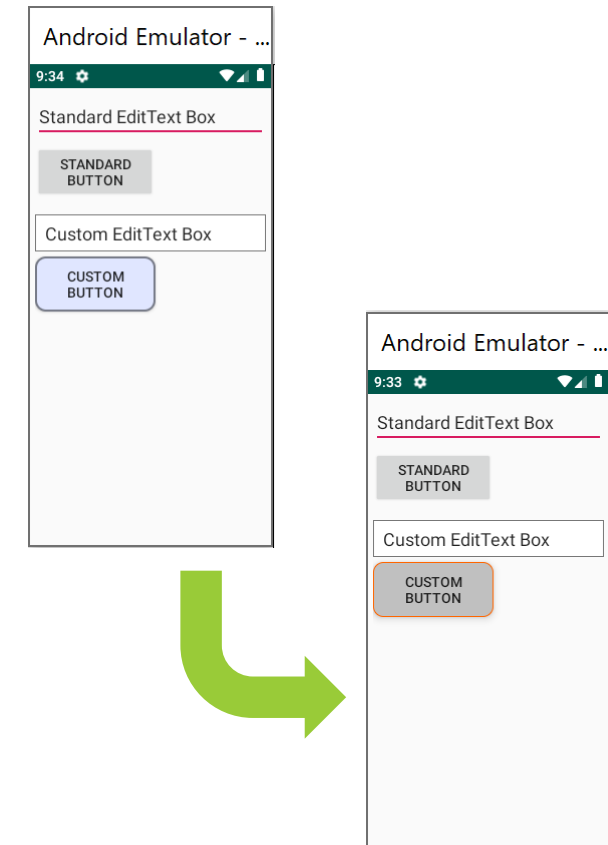


WIDGETS AND LAYOUTS

(Customizing widgets)

Resource: res/drawable/custom_button.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="#ffc0c0"/>
      <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>
```

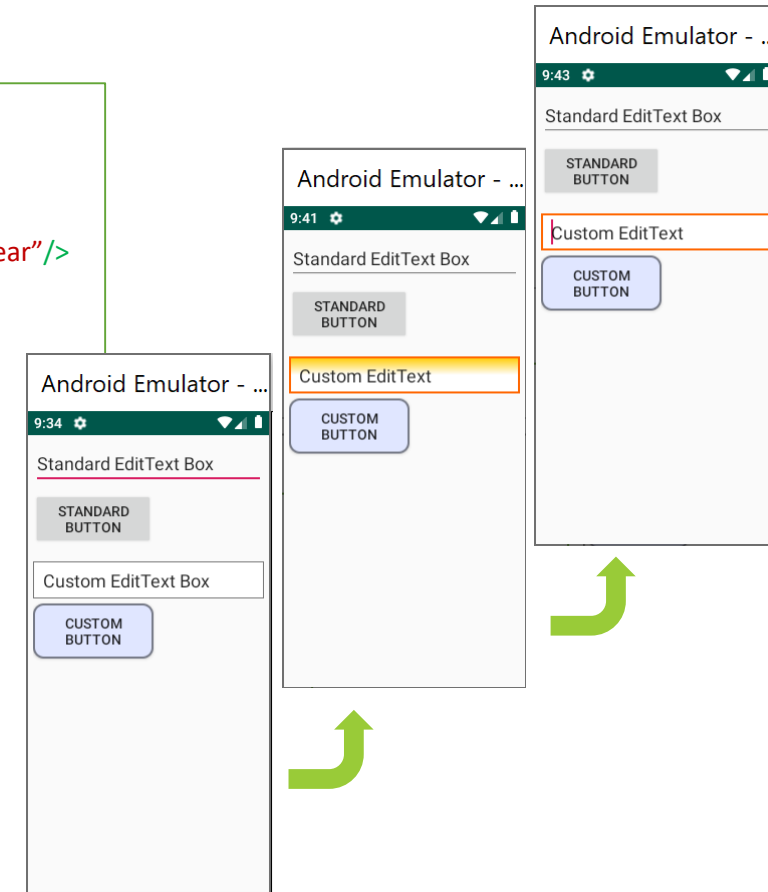


WIDGETS AND LAYOUTS

(Customizing widgets)

Resource: res/drawable/custom_edittext.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="#FFFFFF" android:endColor="#FFffcc00" android:startColor="#FFFFFF" 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>
  <item android:state_focused="true">
    <shape>
      <solid android:color="#FFFFFF" /><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="#ffffff" />
      <corners android:radius="0dp" /><padding android:left="10dp" android:top="6dp" android:right="10dp" android:bottom="6dp" />
    </shape>
  </item>
</selector>
```



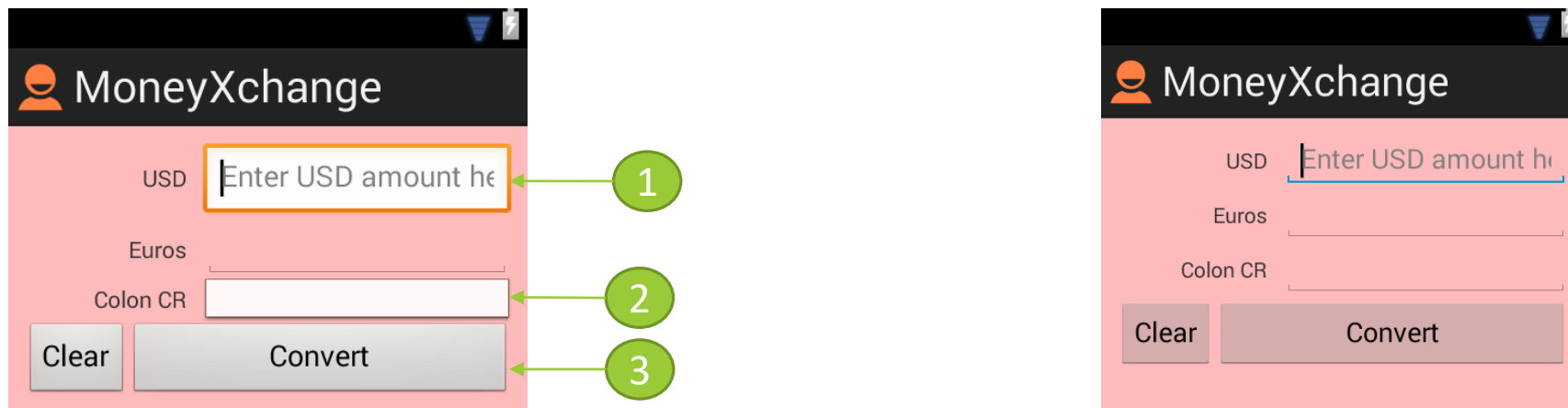
WIDGETS AND LAYOUTS

(Fixing bleeding background color)

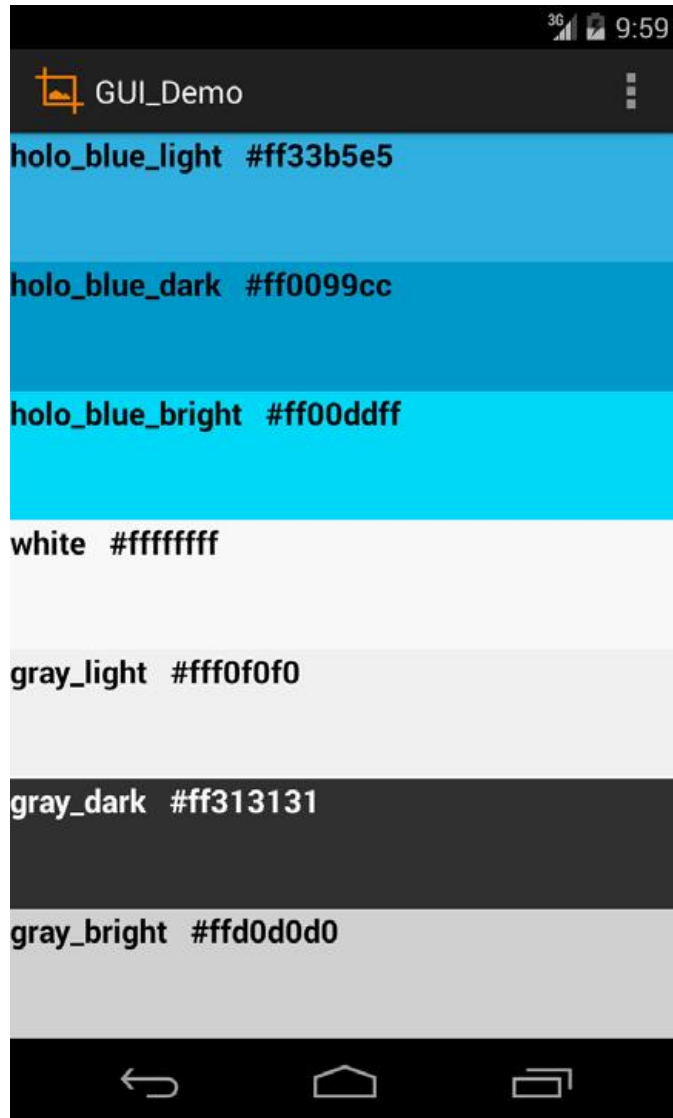
You may change a layout's color by simply adding in the XML layout the clause `android:background="#44ff0000"` (color is set to semi-transparent red).

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).

Although tedious, 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 includes 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"`



WIDGETS AND LAYOUTS (Useful color theme)

The screen shows color included in Android's Holo-Theme. The Holo-Theme color set provides a palette of harmonious colors recommended for all your applications. Benefits: uniform design, homogeneous user-experience, beauty(?)...

You may want to add the following entries to your `res/values/colors.xml` file. Example of usage: `android:background="@color/holo_blue_light"`

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="holo_blue_light">#ff33b5e5</color>
  <color name="holo_blue_dark">#ff0099cc</color>
  <color name="holo_blue_bright">#ff00ddff</color>
  <color name="gray_light">#fff0f0f0</color>
  <color name="gray_dark">#ff313131</color>
  <color name="gray_bright">#ffd0d0d0</color>
</resources>
```