IT 4785. Phát triển ứng dụng cho thiết bị di động

Chapter 4. Graphical User Interfaces

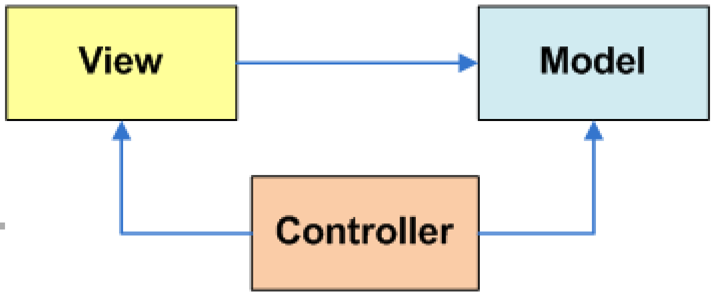
TS. Nguyễn Hồng Quang
Viện Công nghệ thông tin và Truyền thông
Trường Đại học Bách Khoa Hà Nội

- MVC
- LinearLayout
- RelativeLayout
- TableLayout
- ScrollView

Tài liệu tham khảo

Mobile Application Development – Android OS, Victor Matos, Cleveland State University

The Model-View-Control Pattern (MVC)



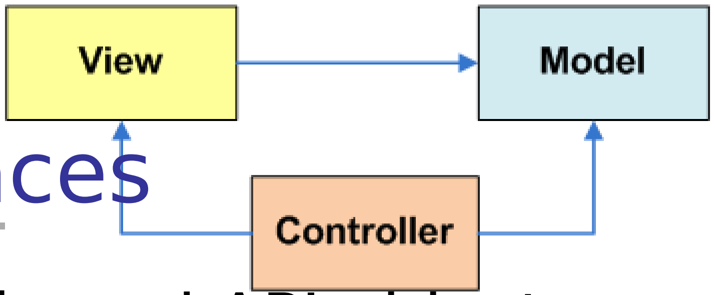
The Model-View-Controller (MVC) is an important software design pattern whose main goal is to separate

(1) user interface,

(2) business,

(3) input logic.

MVC with Android - Graphical User Interfaces



- **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.

MVC Pattern: The View - User Interfaces (GUI s)

- 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

MVC Pattern: The View - User Interfaces (GUI s)

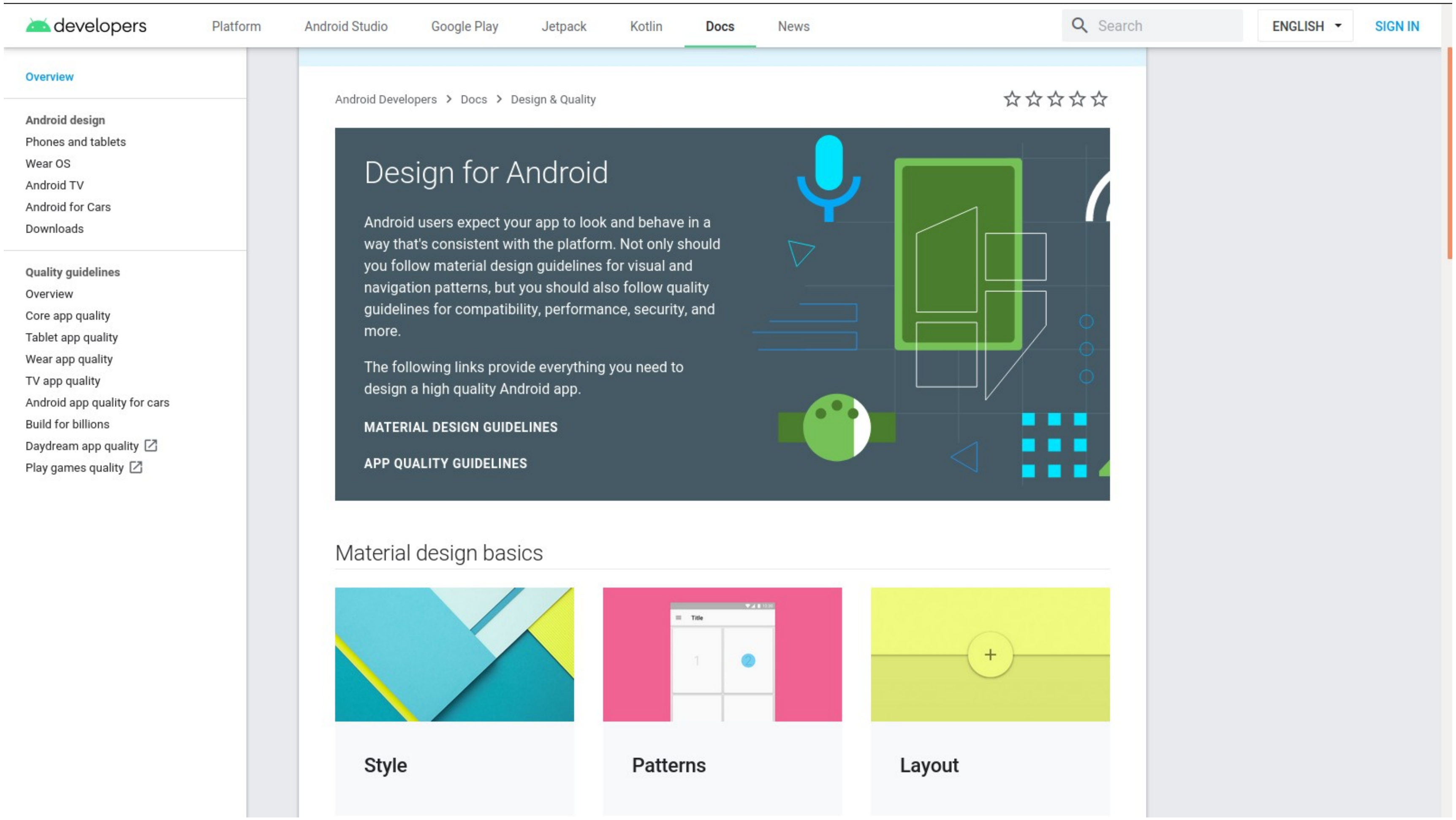
- In a manner similar to a web page interaction, when the Android user touches the screen, the controller interprets the input and determines what specific portion of the screen and gestures were involved.
- Based on this information it tells the model about the interaction in such a way that the appropriate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

MVC Pattern: The View - User Interfaces (GUI s)

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.

Design for Android

https://developer.android.com/design



Design for Android

Android users expect your app to look and behave in a way that's consistent with the platform. Not only should you follow material design guidelines for visual and navigation patterns, but you should also follow quality guidelines for compatibility, performance, security, and more.

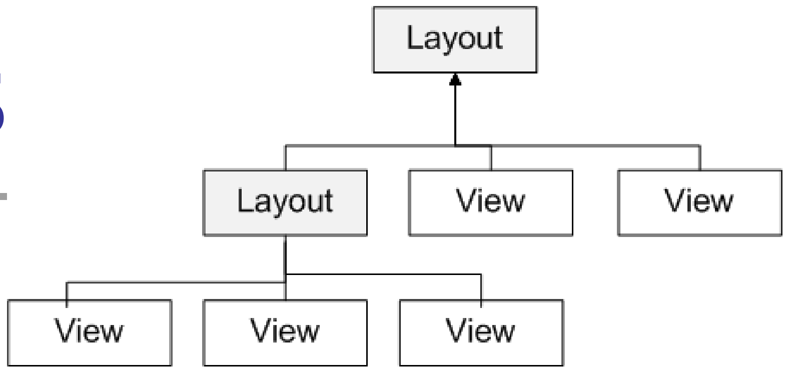
The following links provide everything you need to design a high quality Android app.

MATERIAL DESIGN GUIDELINES

APP QUALITY GUIDELINES

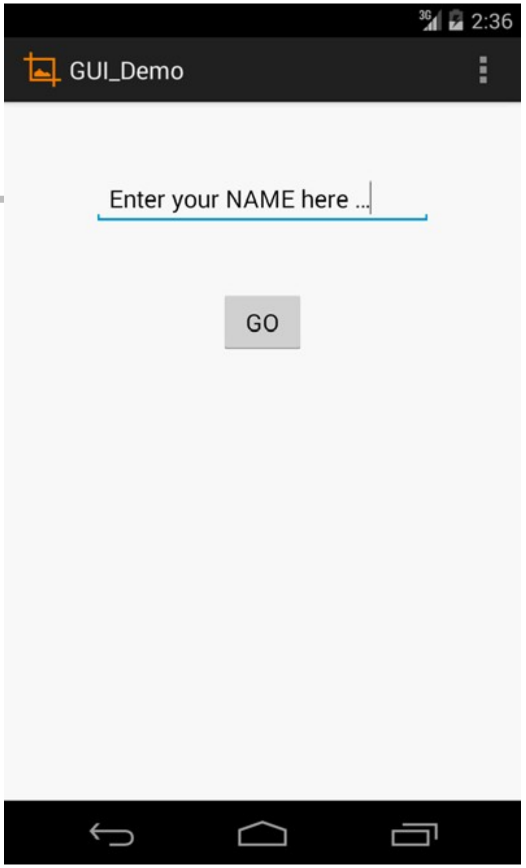
Material design basics

The VIEW Clas



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

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>
```

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.

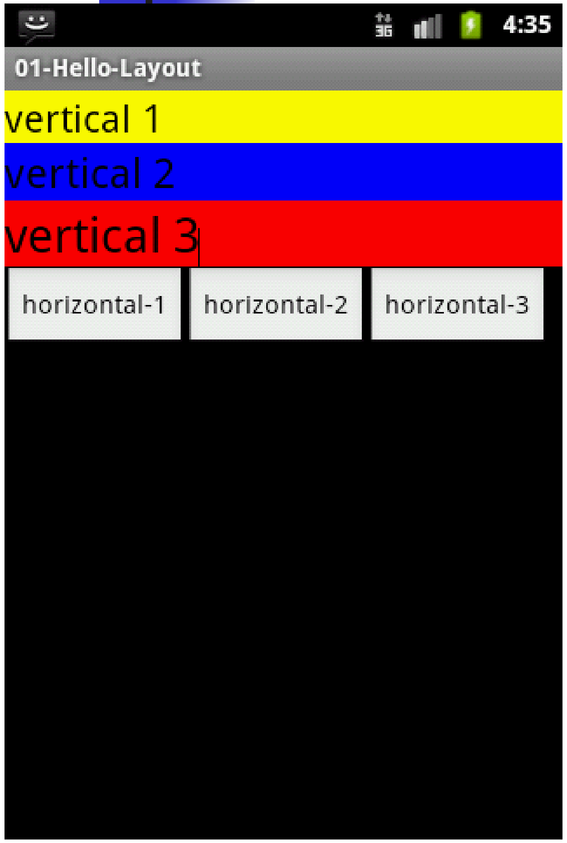
```
<LinearLayout
 xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:orientation="horizontal" >

   Widgets and other nested layouts
</LinearLayout>
```

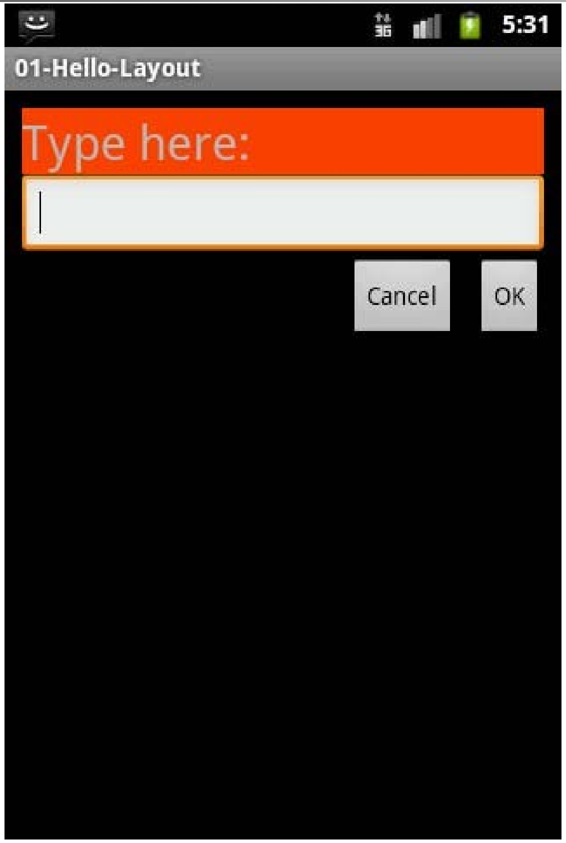
Setting Views to Work

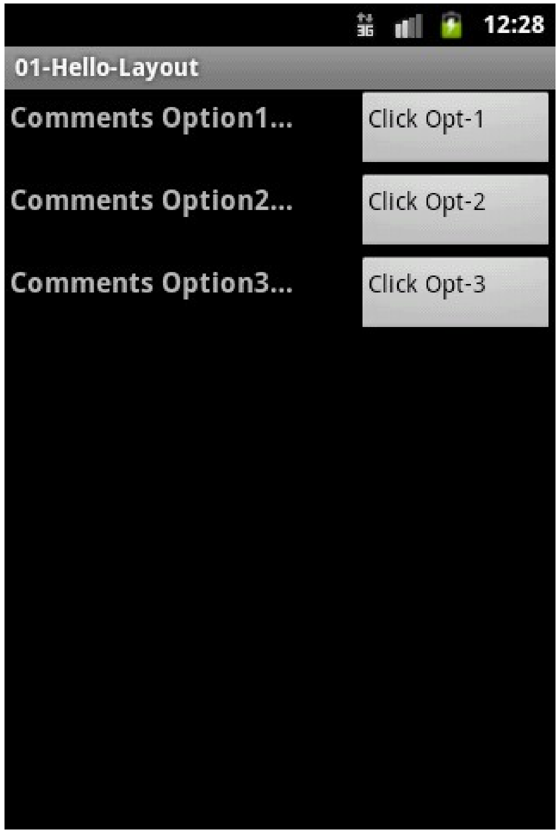
Dealing with widgets & layouts typically involves the following operations

1. Set properties: For instance, when working with a TextView you set the background color, text, font, alignment, size, padding, margin, etc.
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 Sample of Common Android LAYOUTS





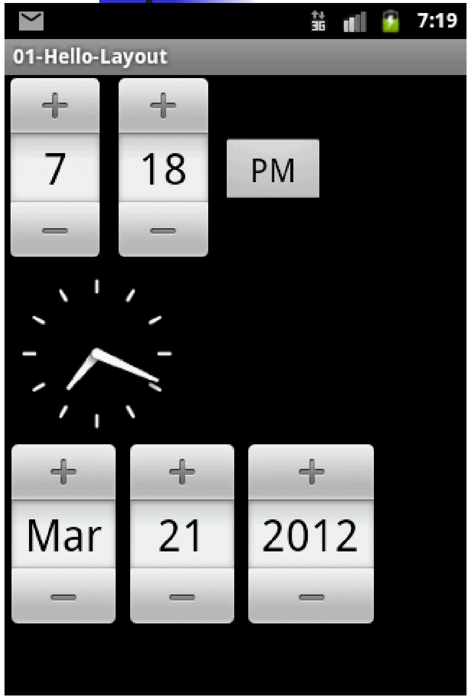


Linear Layout
A LinearLayout places its inner views either in horizontal or vertical disposition.

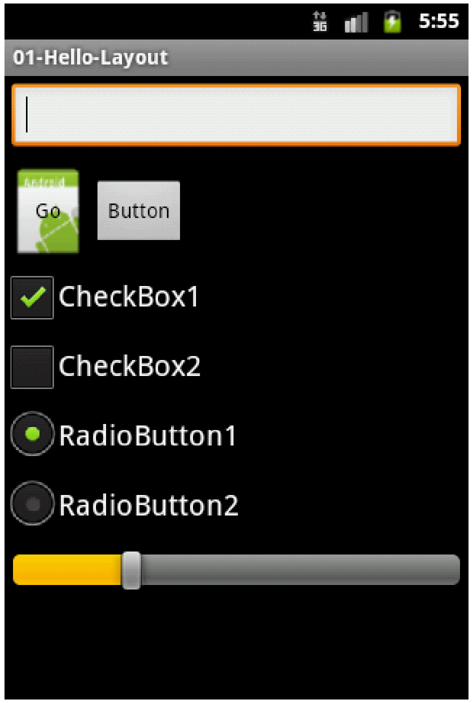
Relative Layout
A RelativeLayout is a ViewGroup
that allows you to position elements relative to each other.

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

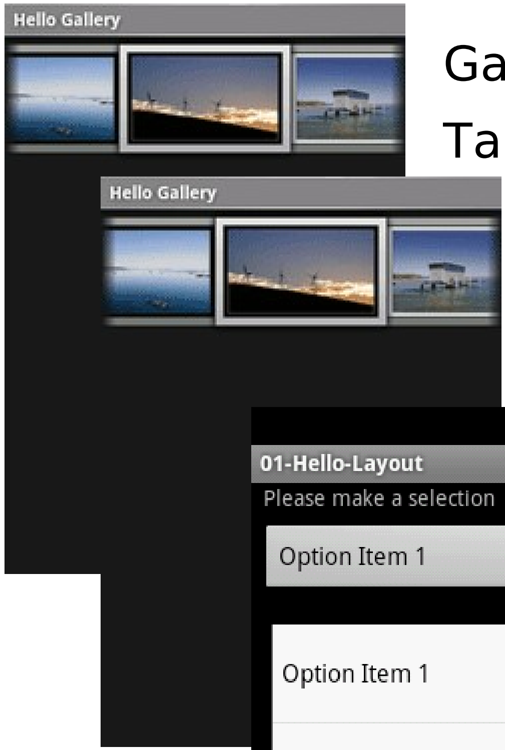
A Sample of Common Android WIDGETS



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

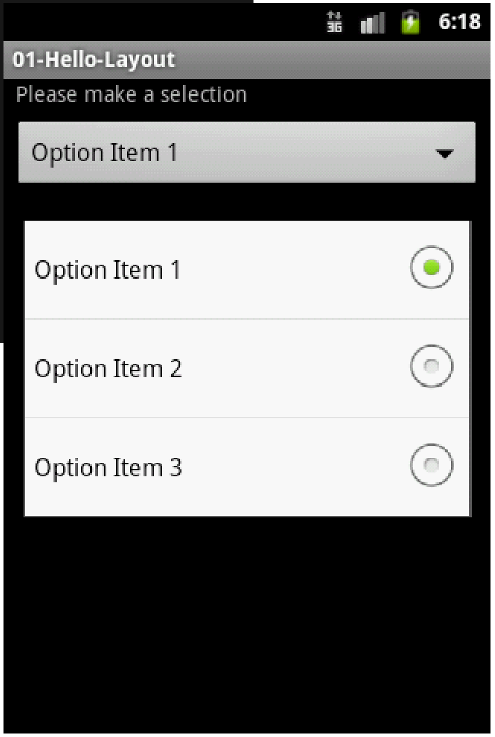


Form Controls
Includes a variety of typical form widgets, like:
image buttons,
text fields,
checkboxes and
radio buttons.

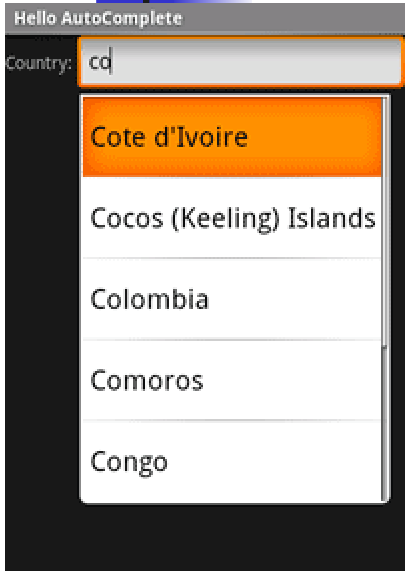


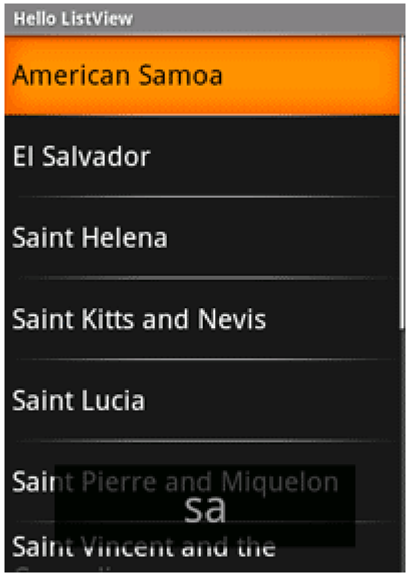
GalleryView
TabWidget

Spinner

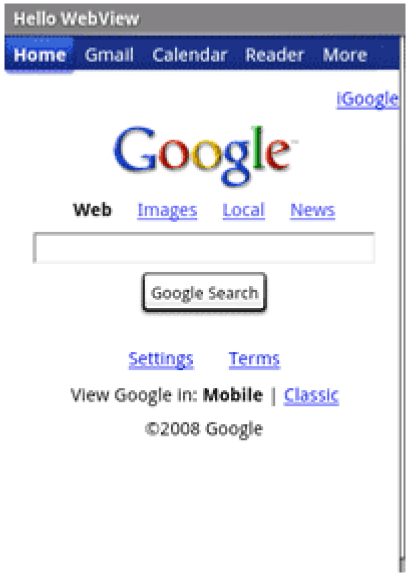


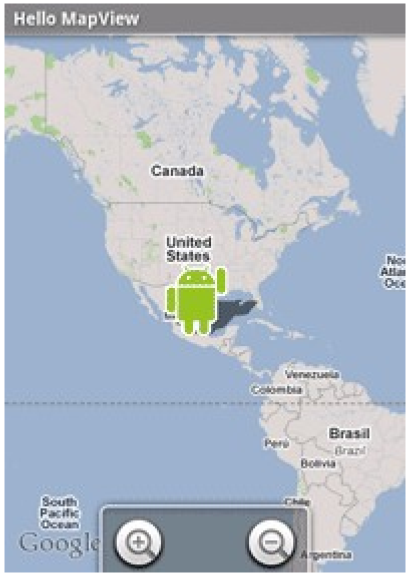
A Sample of Common Android WIDGETS





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





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.

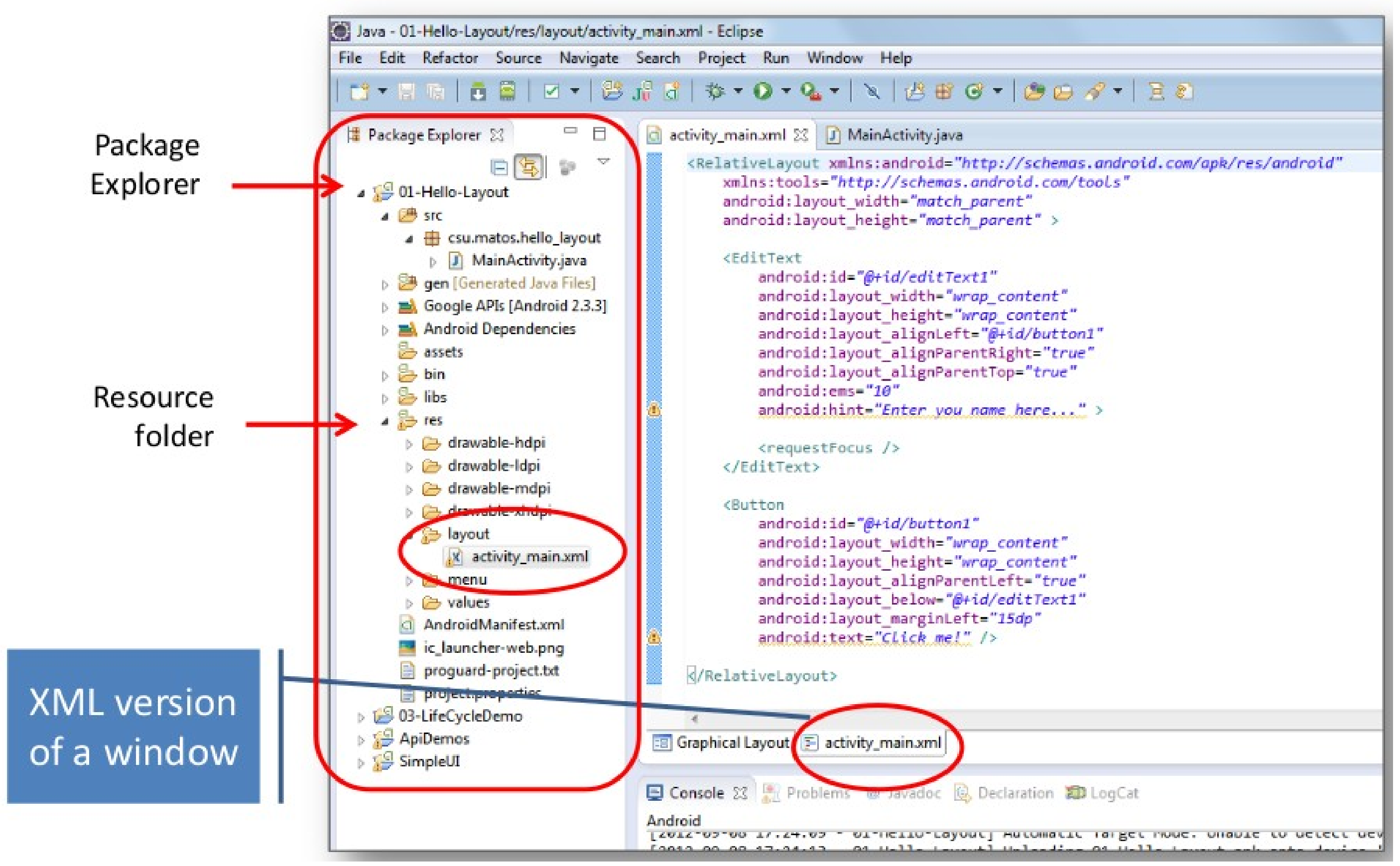
WebView

MapView

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

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.



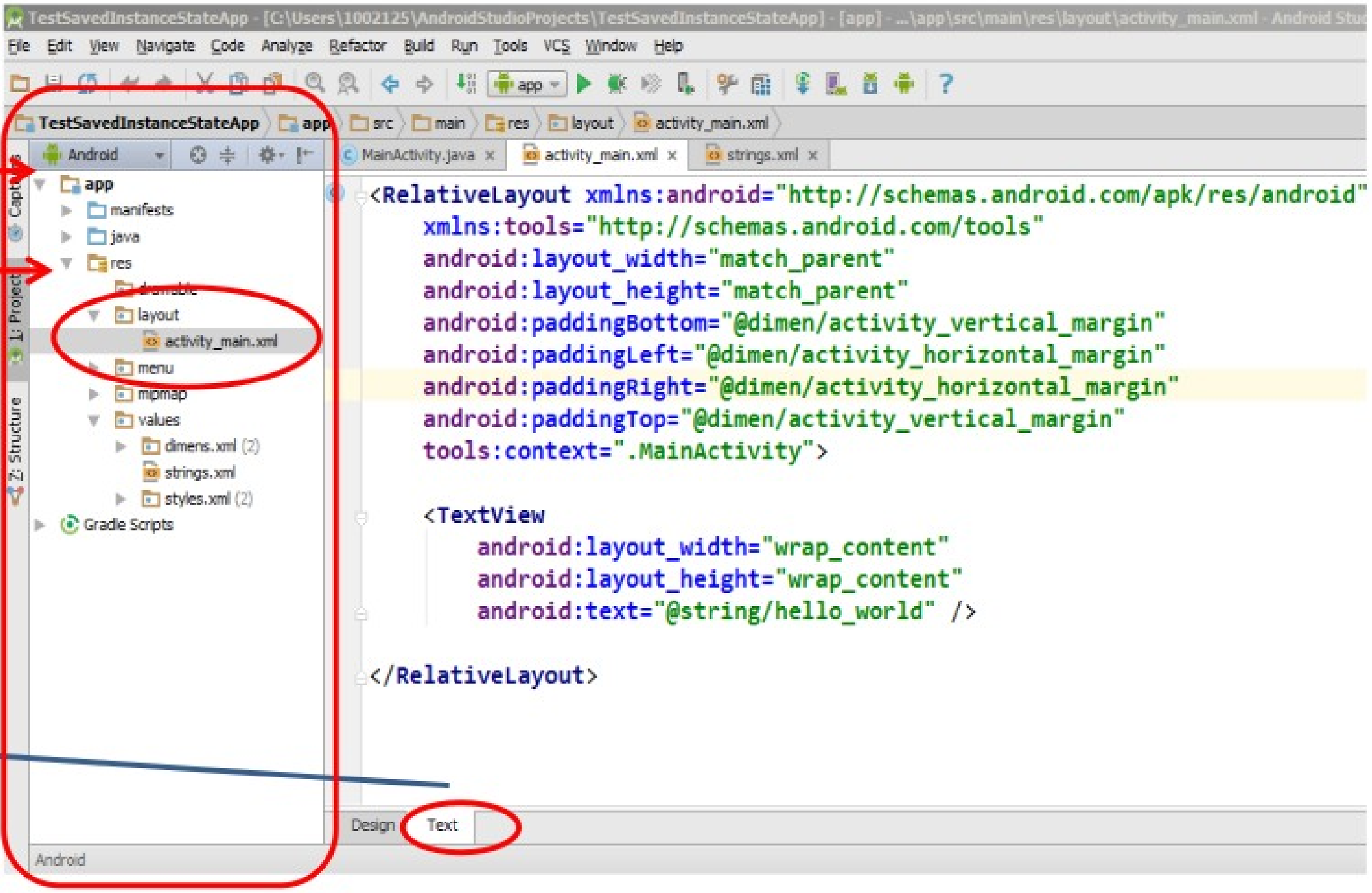
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

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

        <requestFocus />
    </EditText>

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

</RelativeLayout>
```

GUI Editing: XML Version

App explorer

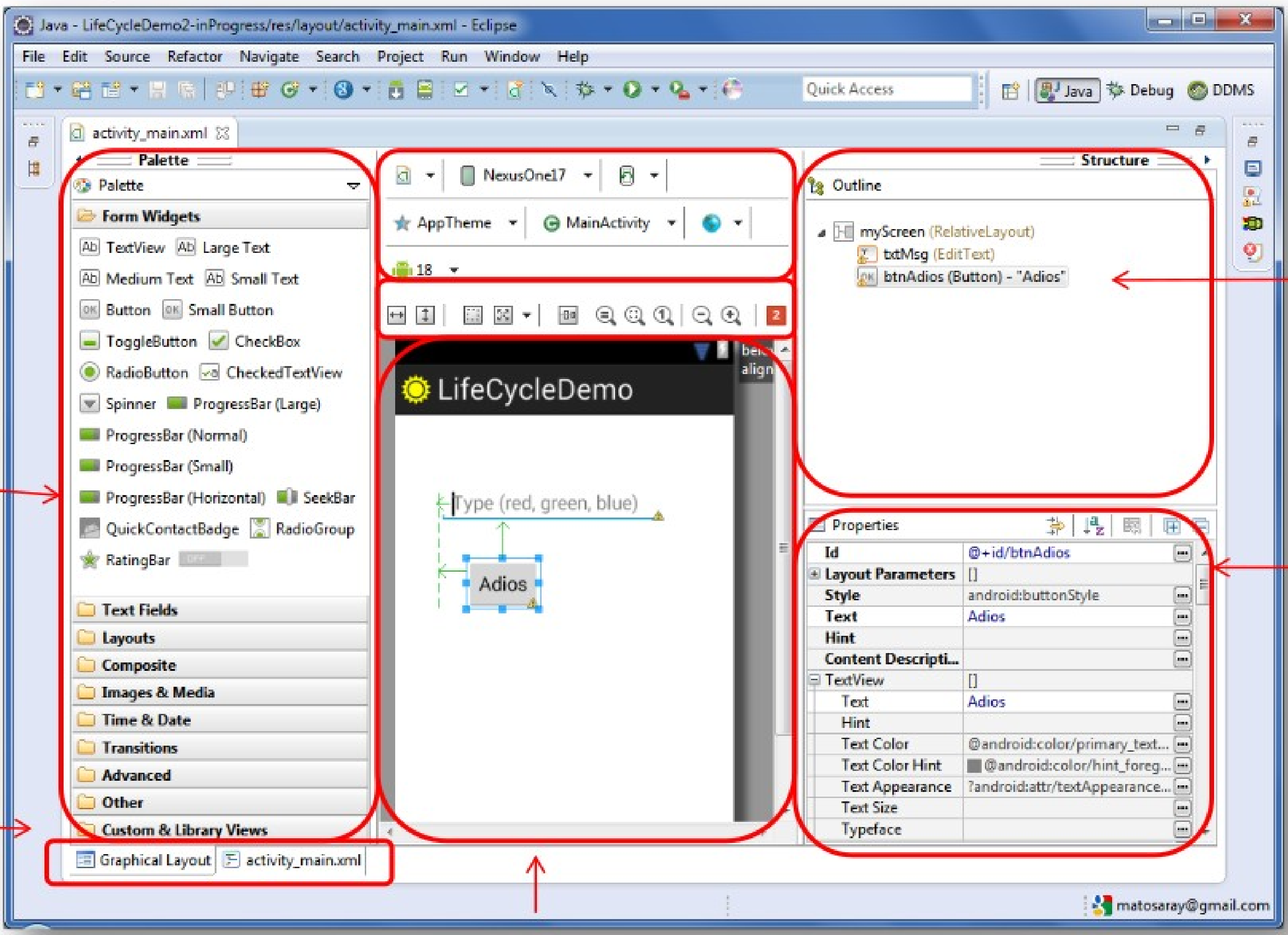
Resource folder

XML version of a window


TestSavedInstanceStateApp - [C:\Users\1002125\AndroidStudioProjects\TestSavedInstanceStateApp] - [app] - ...\app\src\main\res\layout\activity_main.xml - Android Stu
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
TestSavedInstanceStateApp app src main res layout activity_main.xml
Android
MainActivity.java activity_main.xml strings.xml
app
manifests
java
res
drawable
layout
activity_main.xml
menu
mipmap
values
dimens.xml (2)
strings.xml
styles.xml (2)
Gradle Scripts
Design Text
Android

```
<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">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

</RelativeLayout>
```

GUI Editing: WYSIWYG Version



Outline:

- myScreen (RelativeLayout)
  - txtMsg (EditText)
  - btnAdios (Button) - "Adios"

Properties:

Id	@+id/btnAdios
Layout Parameters	[]
Style	android:buttonStyle
Text	Adios
Hint	
Content Descripti...	
TextView	[]
Text	Adios
Hint	
Text Color	@android:color/primary_text...
Text Color Hint	@android:color/hint_foreg...
Text Appearance	?android:attr/textAppearance...
Text Size	
Typeface	

GUI Palette

Screen's Outline

Widget's properties

Select WYSIWYG or XML view

WYSIWYG screen

4 - 15

Tools you can use to create an Android GUI

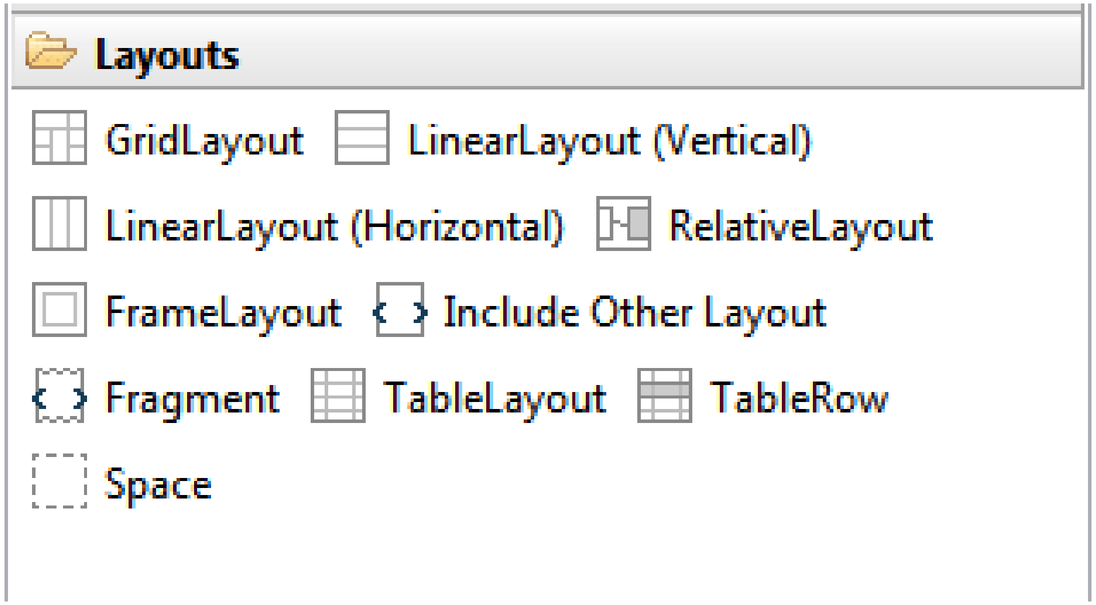
- Android Studio. Based on IntelliJ IDEA IDE. Functionally equivalent to Eclipse with the ADT Plugin.

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- 11012.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/

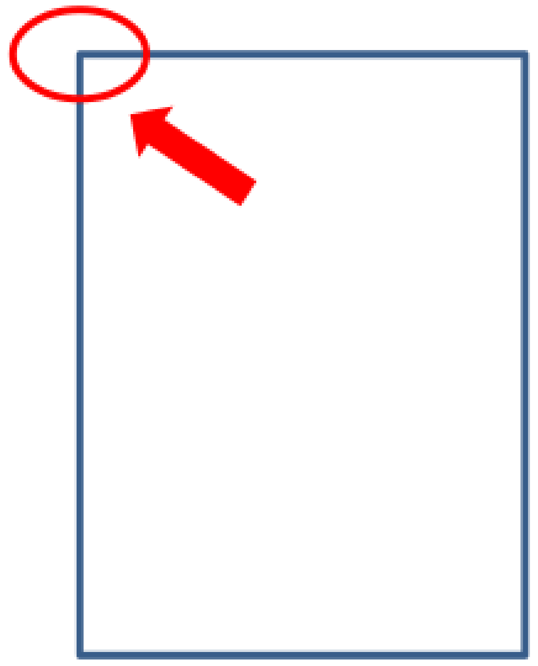
GUI Elements: The LAYOUT

- 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.



FrameLayout

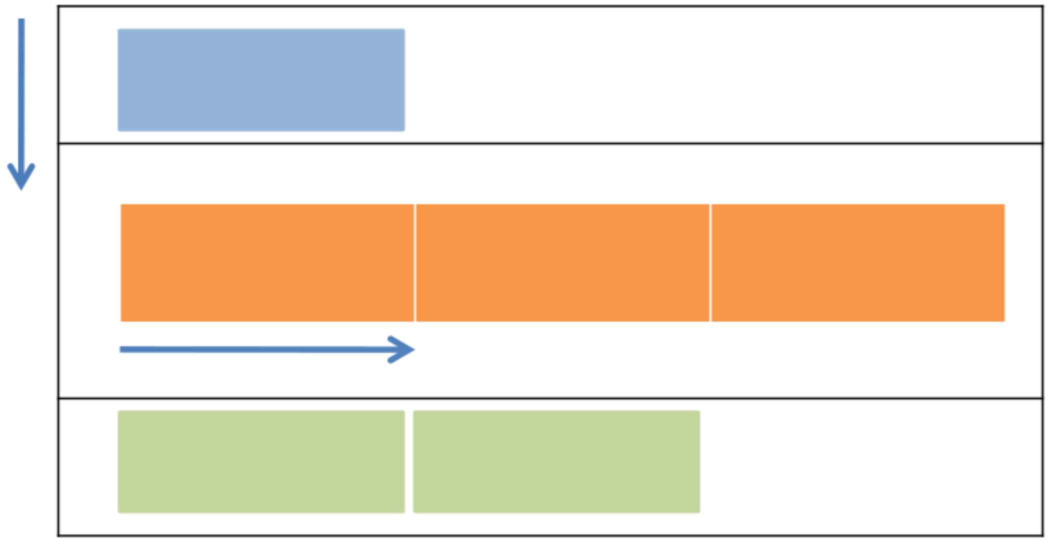
- The FrameLayout is the simplest type of GUI container.
- It is useful as an outermost container holding a window.
- Allows you to define how much of the screen (high, width) is to be used.
- All its children elements are aligned to the top left corner of the screen.;



LinearLayout

LinearLayout

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



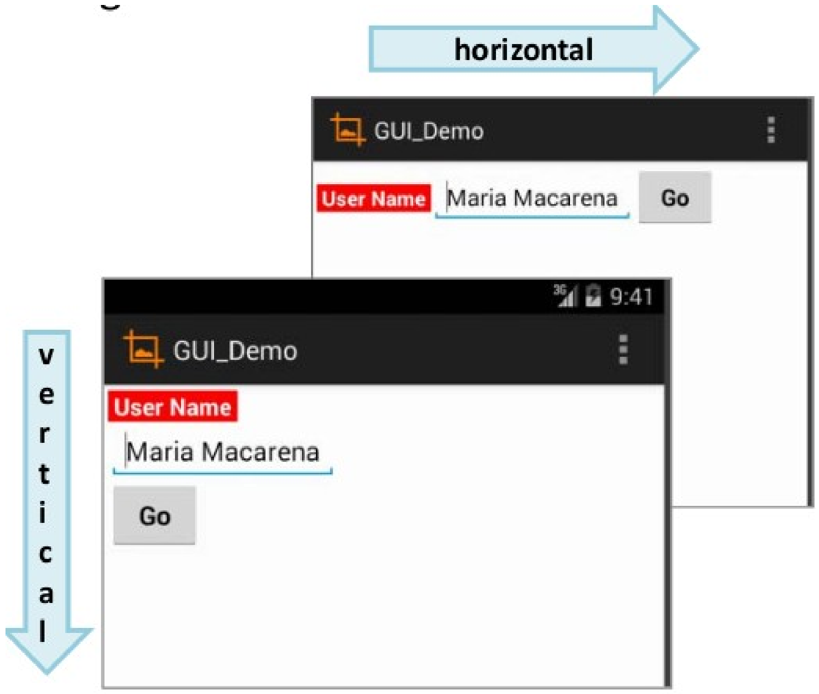
LinearLayout
Setting Attributes

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)

LinearLayout : Orientation

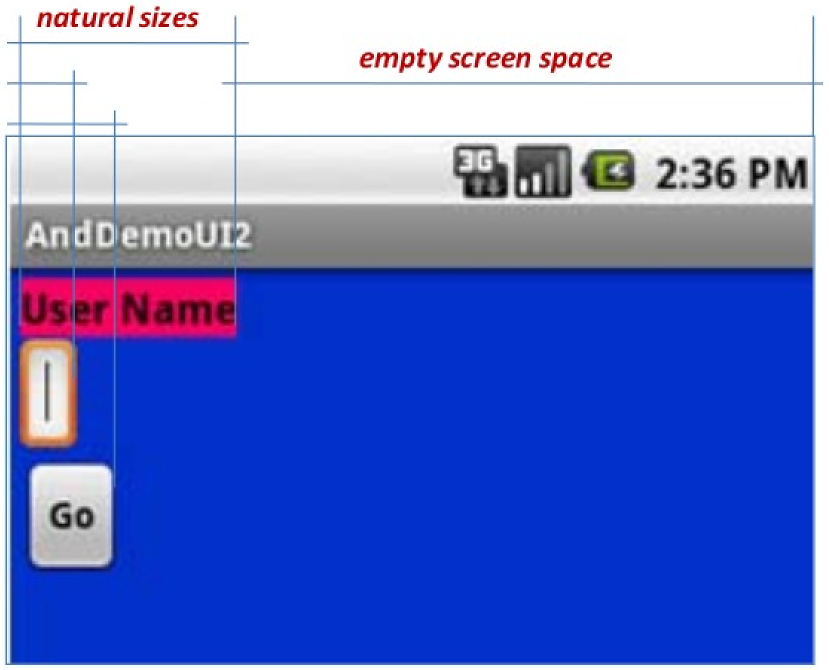
The android:orientation property can be set to: horizontal for columns, or vertical for rows.
Use setOrientation() for runtime changes.



```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/
   android"
  android:id="@+id/myLinearLayout"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="horizontal"
  android:padding="4dp" >
<TextView
  android:id="@+id/labelUserName"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:background="#ffff0000"
  android:text=" User Name "
  android:textColor="#ffffffff"
  android:textSize="16sp"
  android:textStyle="bold" />
<EditText
  android:id="@+id/ediName"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Maria Macarena"
  android:textSize="18sp" />
<Button
  android:id="@+id/btnGo"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Go"
  android:textStyle="bold" />
</LinearLayout>
```

LinearLayout : Fill Model

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



Shown on a Gingerbread device

LinearLayout : Fill Model

All widgets inside a LinearLayout must include ‘width’ and ‘height’ attributes.

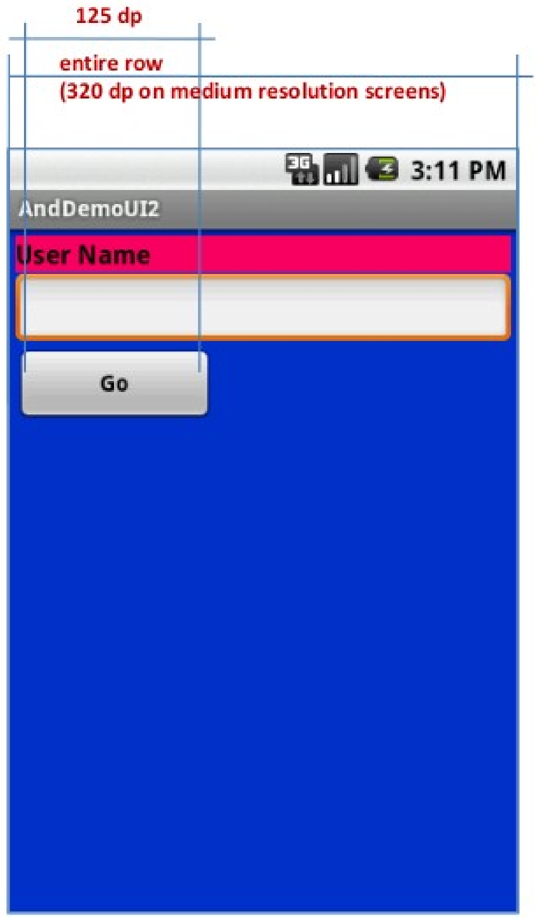
android:layout_width

android:layout_height

Values used in defining height and width can be:

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

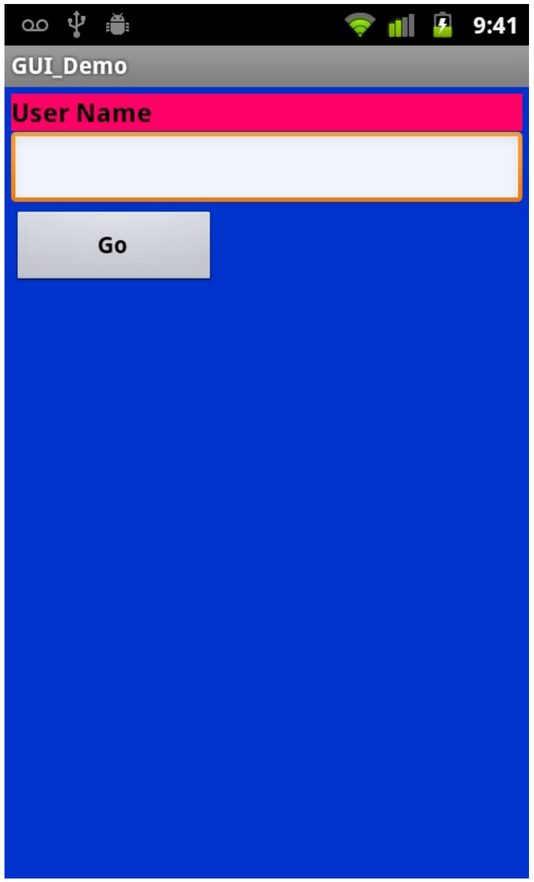
LinearLayout : Fill Model

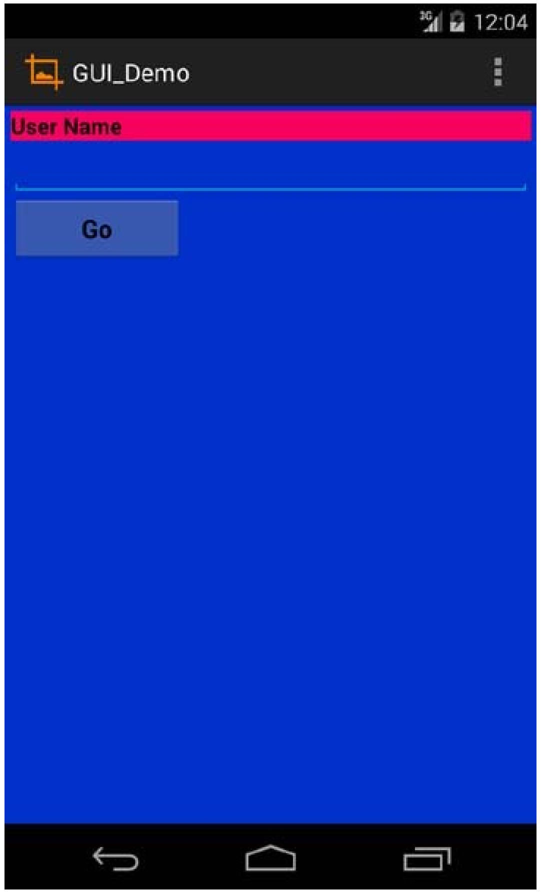


Medium resolution is: 320 x 480 dpi.
Shown on a Gingerbread device

```
<LinearLayout  xmlns:android="http://schemas.android.com/
  apk/res/android"
 android:id="@+id/myLinearLayout"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:background="#ff0033cc"
 android:orientation="vertical"
 android:padding="6dp" >
 <TextView
   android:id="@+id/labelUserName"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:background="#ffff0066"
   android:text="User Name"
   android:textColor="#ff000000"
   android:textSize="16sp"
   android:textStyle="bold" />
 <EditText
   android:id="@+id/ediName"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:textSize="18sp" />
 <Button
   android:id="@+id/btnGo"
   android:layout_width="125dp"
   android:layout_height="wrap_content"
   android:text="Go"
   android:textStyle="bold" />
</LinearLayout>
```

Warning ! Same XML different rendition...





Same XML layout shown on a Gingerbread (left) and Kitkat (right) device.

Warning ! 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)

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:

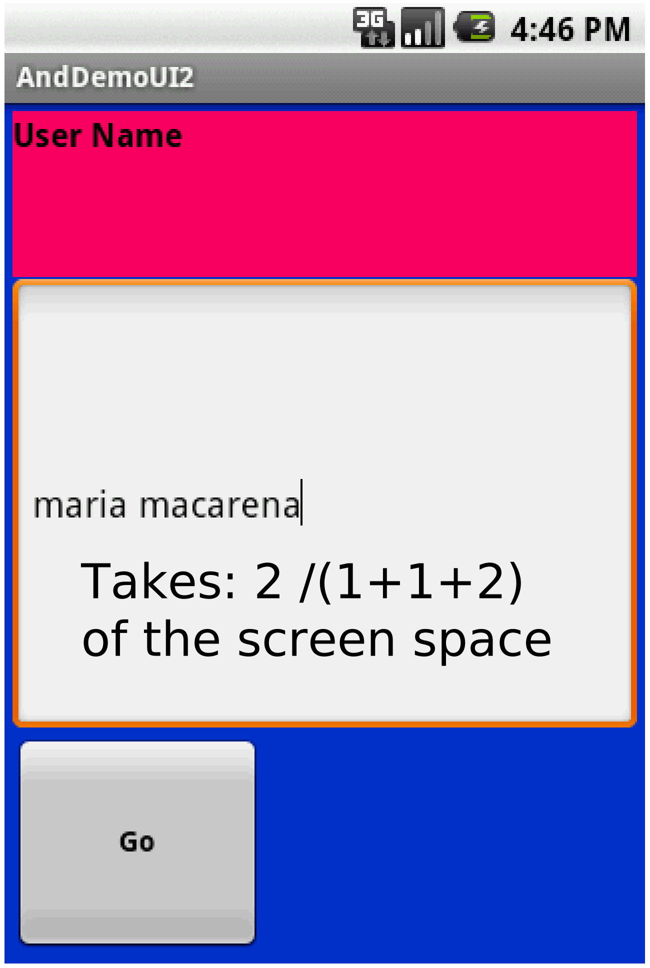
The TextView and Button controls have the additional property

android:layout_weight="1"

whereas the EditText control has

android:layout_weight="2"

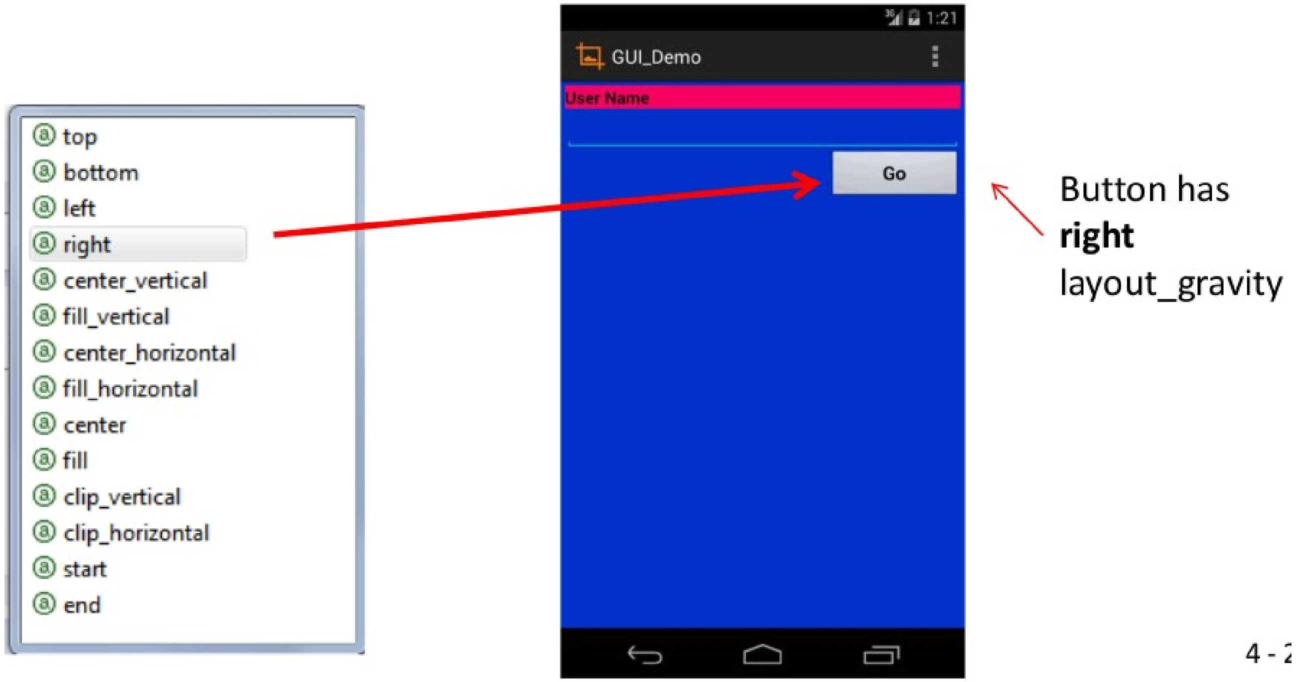
Remember, default value is 0



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

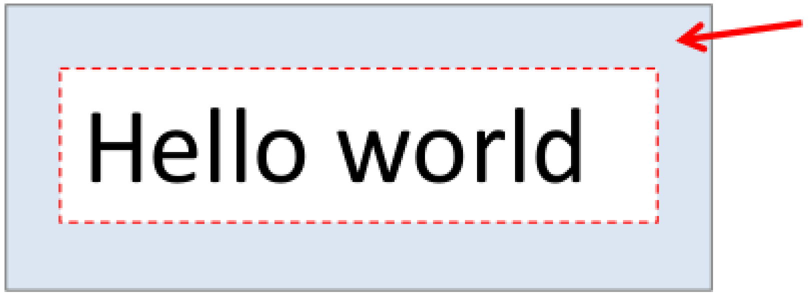
LinearLayout : Gravity

- 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.



LinearLayout : Padding

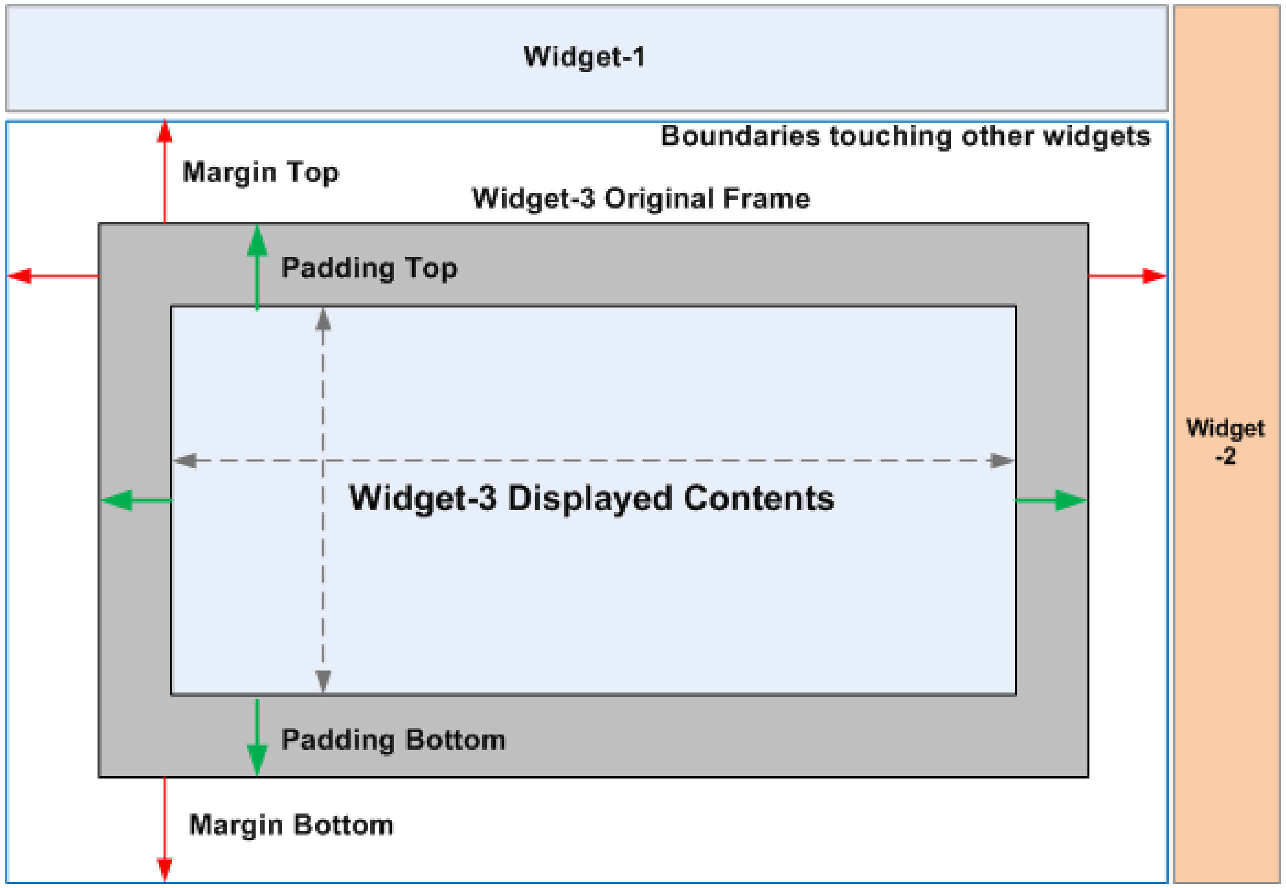
- The padding attribute specifies the widget's internal margin (in dp units).
- The internal margin is the extra space between the borders of the widget's "cell" and the actual widget contents.
- Either use
  - android:padding property
  - or call method setPadding() Hello world at runtime.



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

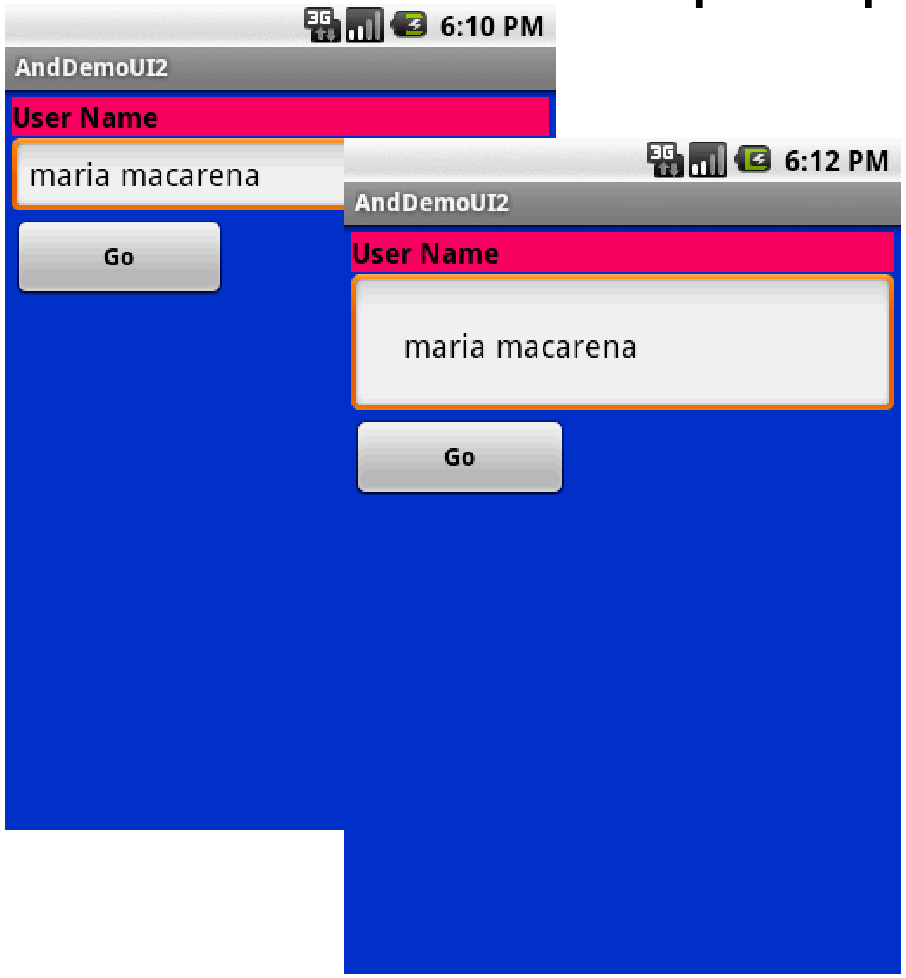
LinearLayout : Padding and Margin

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



LinearLayout : Set Internal Margins Using Padding

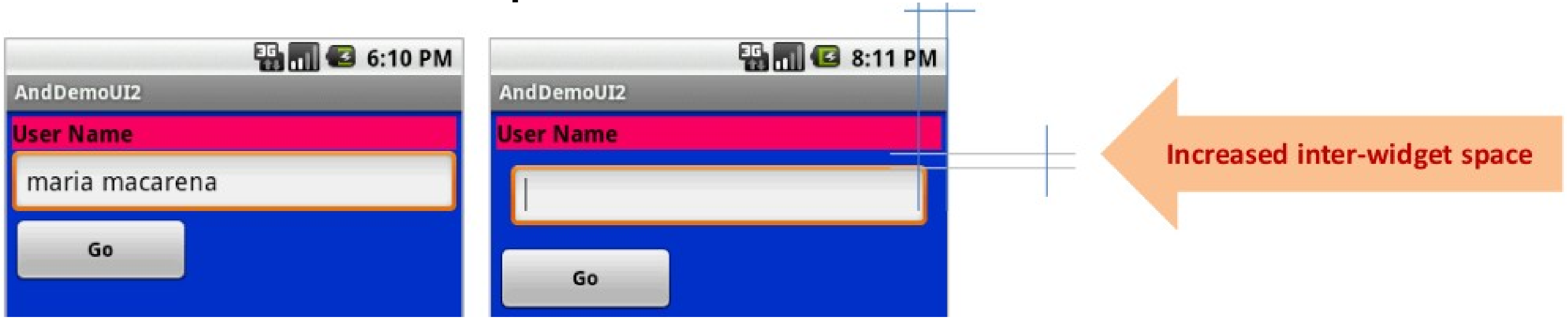
Example: The EditText box has been changed to include 30dp of padding all around



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

LinearLayout : Set External Margins

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

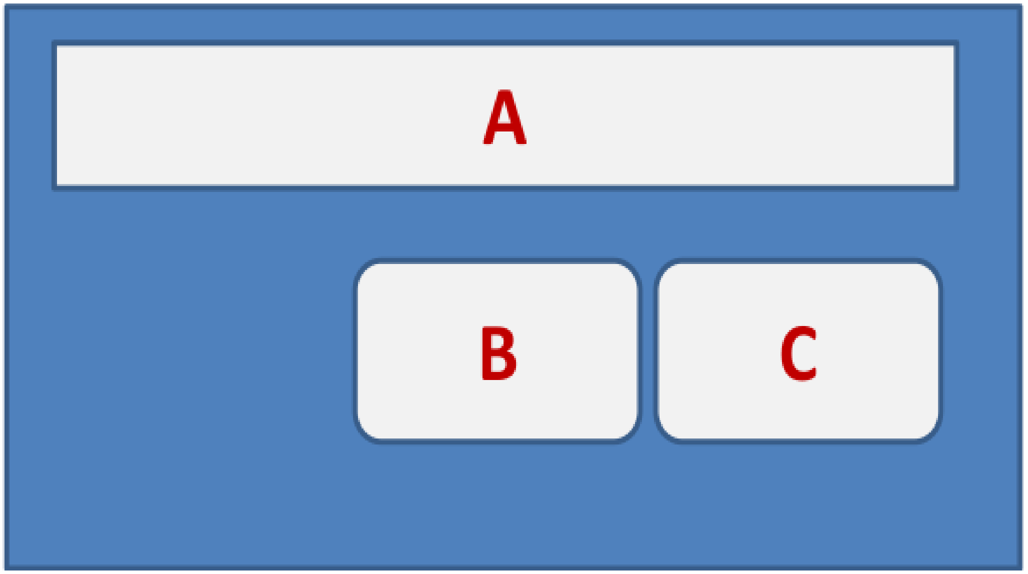


```
<EditText
    android:id="@+id/ediName"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="18sp"
    android:layout_margin="6dp"
>
</EditText>
```

RelativeLayout

Relative Layout

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

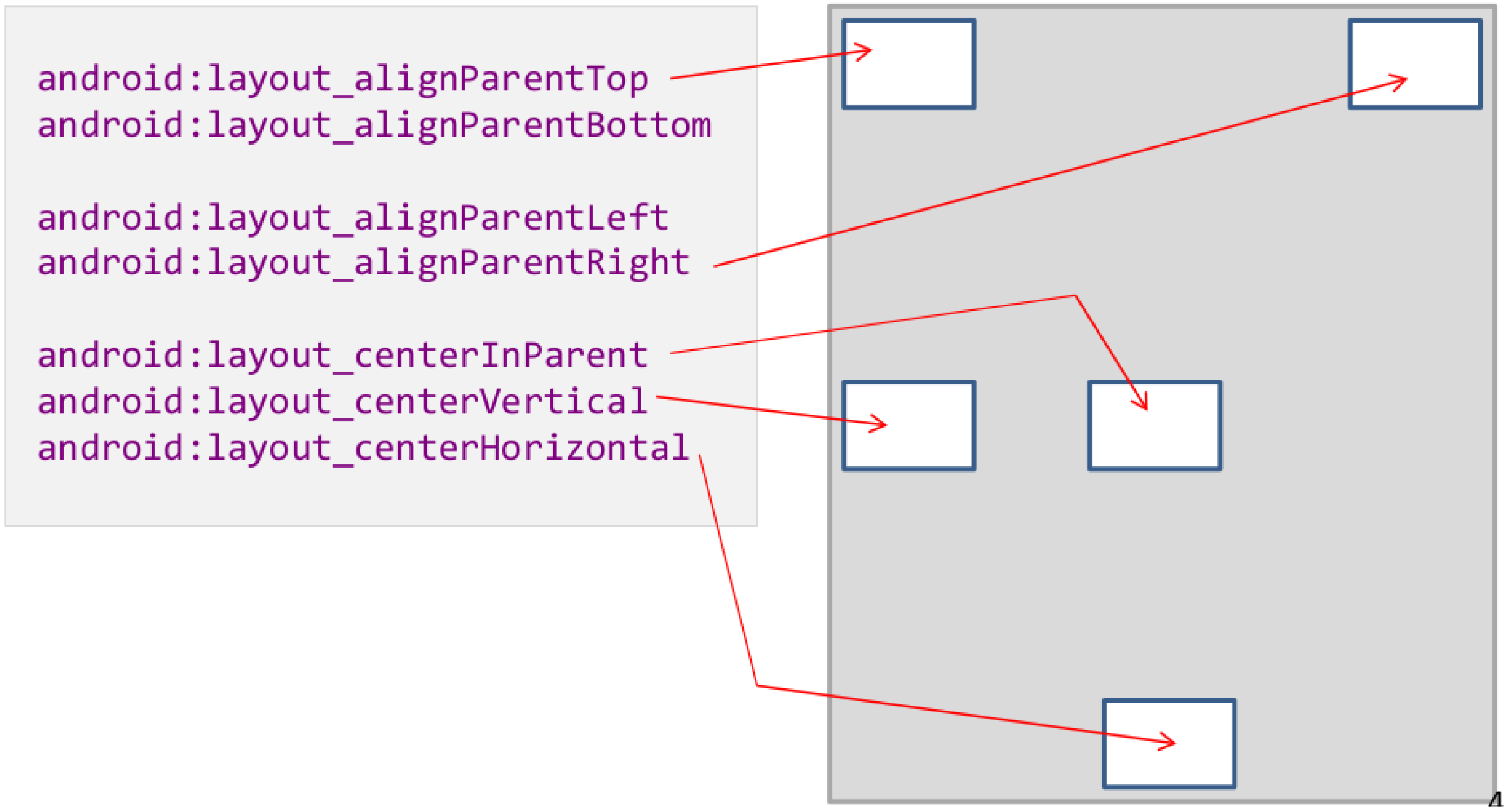


Example:
A is by the parent's top
C is below A, to its right
B is below A, to the left of C

Relative Layout - Referring to the container

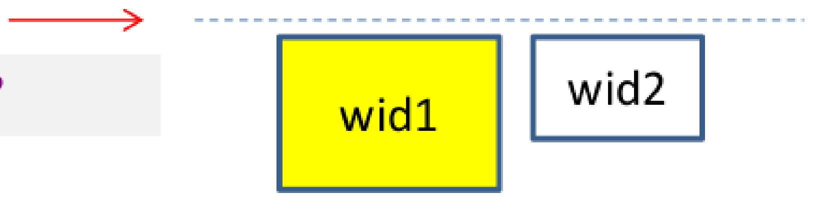
Below there is a sample of various positioning XML boolean properties (true/false) which are useful for collocating a widget based on the location of its parent container.

```
android:layout_alignParentTop
android:layout_alignParentBottom

android:layout_alignParentLeft
android:layout_alignParentRight

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

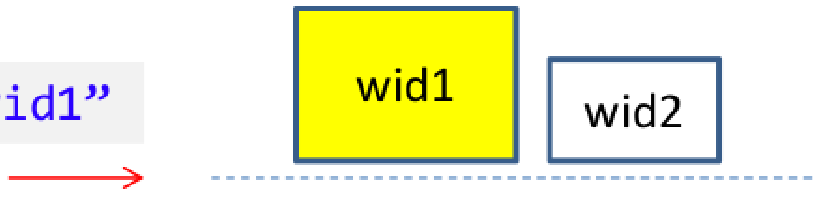


Relative Layout - Referring to Other Widgets

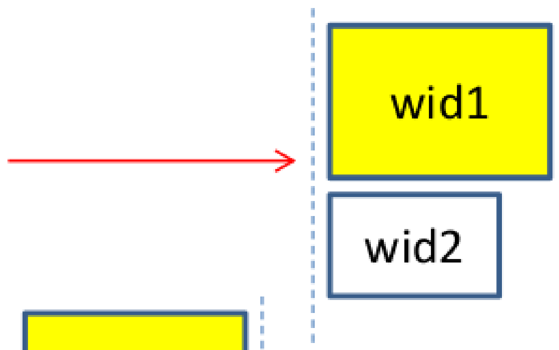
```
android:layout_alignTop=“@+id/wid1”
```



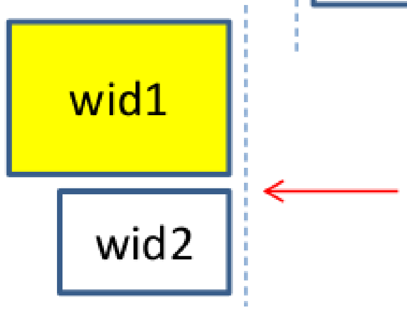
```
android:layout_alignBottom =“@+id/wid1”
```



```
android:layout_alignLeft=“@+id/wid1”
```



```
android:layout_alignRight=“@+id/wid1”
```



Relative Layout - Referring to Other Widgets

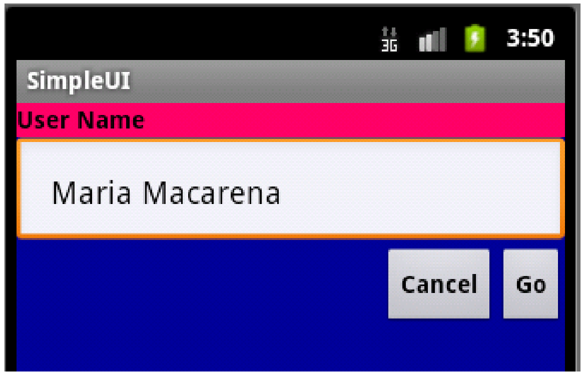
When using relative positioning you need to:

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

android:layout_below="@+id/txtUserName"

Relative Layout - Example

```
<RelativeLayout
xmlns:android="http://schemas.android.com/
apk/res/android"
android:id="@+id/myRelativeLayout"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:background="#ff000099" >
<TextView
android:id="@+id/lblUserName"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_alignParentLeft="true"
android:layout_alignParentTop="true"
android:background="#ffff0066"
android:text="User Name"
android:textColor="#ff000000"
android:textStyle="bold" >
</TextView>
```



```
<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>
```

TableLayout

Table Layout

1. Android's TableLayout uses a grid template to position your widgets.
2. Like in a 2D matrix, cells in the grid are identified by rows and columns.
3. Columns are flexible, they could shrink or stretch to accommodate their contents.
4. The element TableRow is used to define a new row in which widgets can be allocated.
5. The number of columns in a TableRow is determined by the total of side-by-side widgets placed on the row.

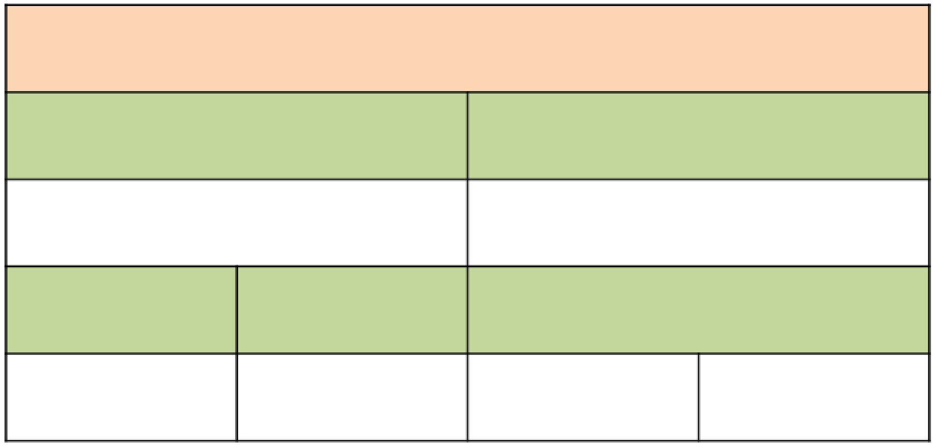


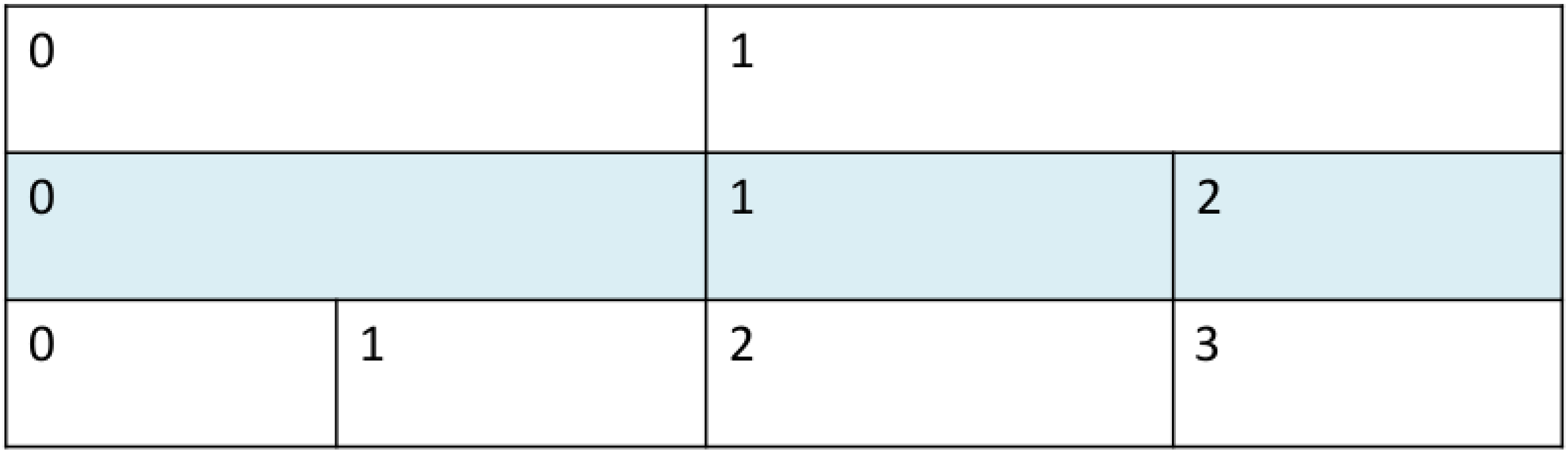
Table Layout – Setting Number of Columns

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

Example: If your TableLayout have three rows

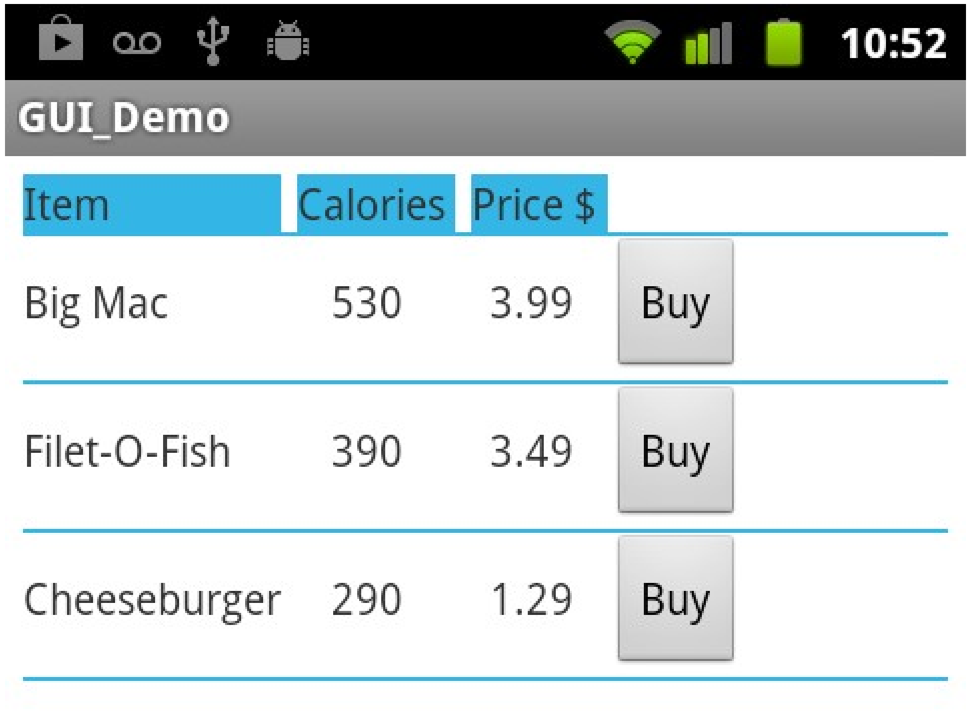
- one row with two widgets,
- one with three widgets, and
- one final row with four widgets,

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

Table Layout – Example 3



GUI_Demo

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 [*].

The TableLayout has four TableRows, with three columns in the first row (labels) and four cells in each of the other three rows (item, Calories, Price, and Buy button).

```
<TableLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/myTableLayout"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  android:padding="6dp" >
<TableRow>
<TextView
  android:background="#FF33B5E5"
  android:text="Item " />
<TextView
  android:layout_marginLeft="5dp"
  android:background="#FF33B5E5"
  android:text="Calories " />
<TextView
  android:layout_marginLeft="5dp"
  android:background="#FF33B5E5"
  android:text="Price $ " />
</TableRow>
<View
  android:layout_height="1dp"
  android:background="#FF33B5E5" />
<TableRow>
<TextView  android:text="Big Mac" />
<TextView
  android:gravity="center"
  android:text="530" />
<TextView
  android:gravity="center"
  android:text="3.99" />
<Button
  android:id="@+id/btnBuyBigMac"
  android:gravity="center"
  android:text="Buy" />
</TableRow>
<View
  android:layout_height="1dp"
  android:background="#FF33B5E5" />
<!-- other TableRows ommitted --!>

</TableLayout>
```

Table Layout – Stretching a Column

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

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

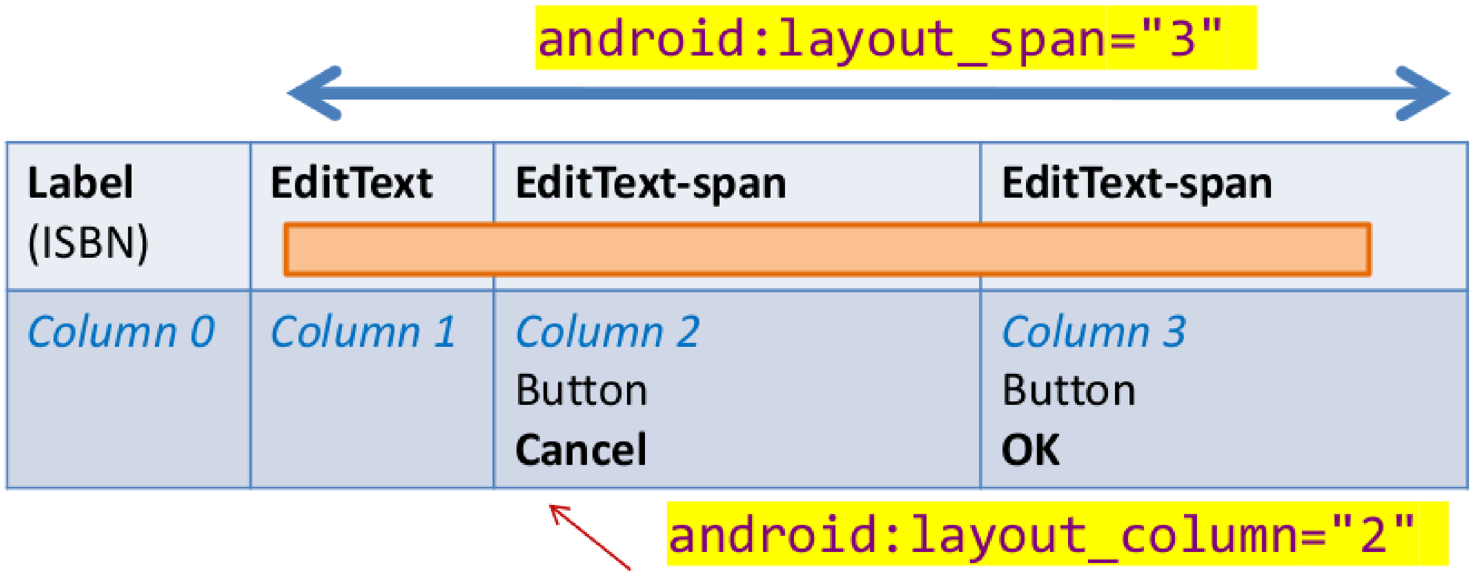
Table Layout – Stretching a Column

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.

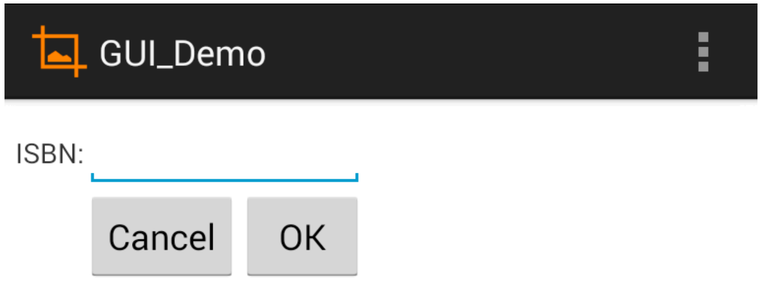
Example 4

- 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).



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

Example 4 continuation



```
<?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/ediISBN"
         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

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

Table Layout – Stretching the Entire Table

- By default, a column is as wide as the "natural' size of the widest widget collocated in this column (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"

Table Layout – Stretching the Entire Table

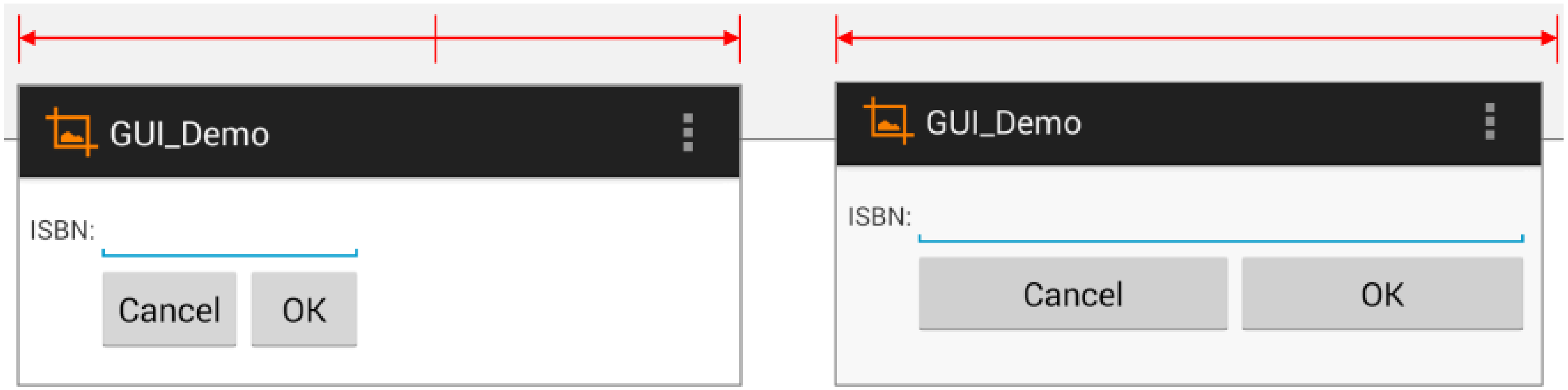
In Example 4 we created a table with four columns.

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 ‘:strechColumns’

```
...
<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"
>
...
```

Table Layout – Stretching the Entire Table

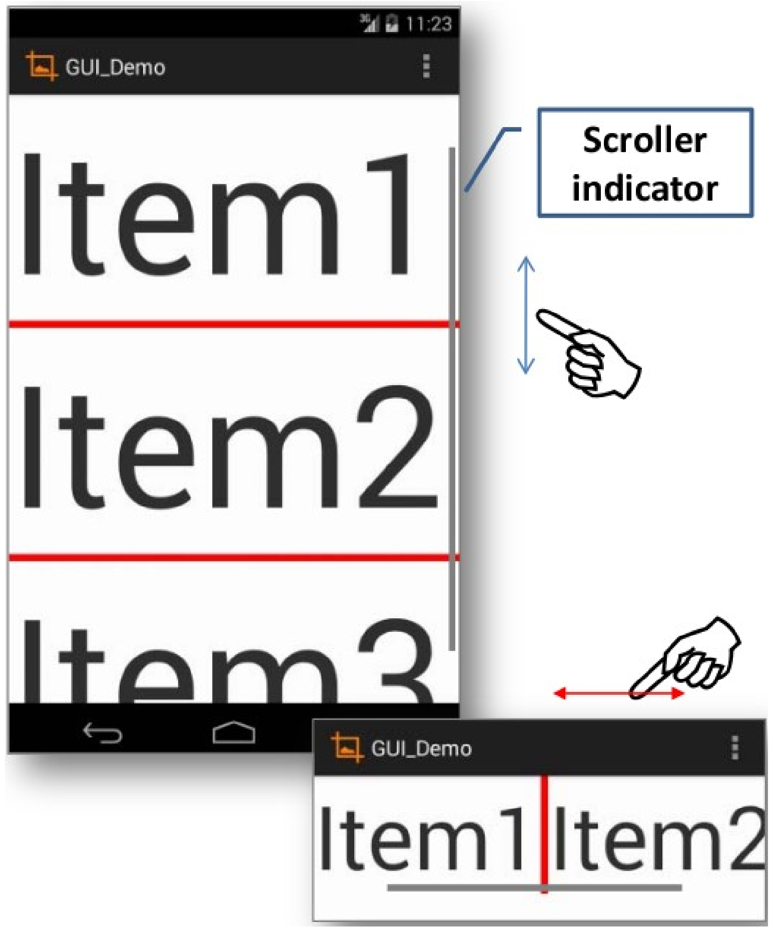
Screens shown before and after using the android:stretchColumns clause.



ScrollView

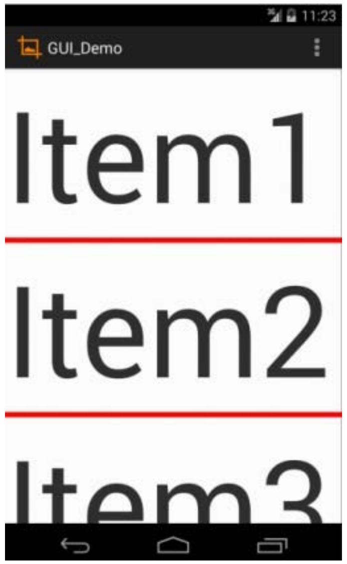
ScrollView Layout (Vertical & Horizontal)

- 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.

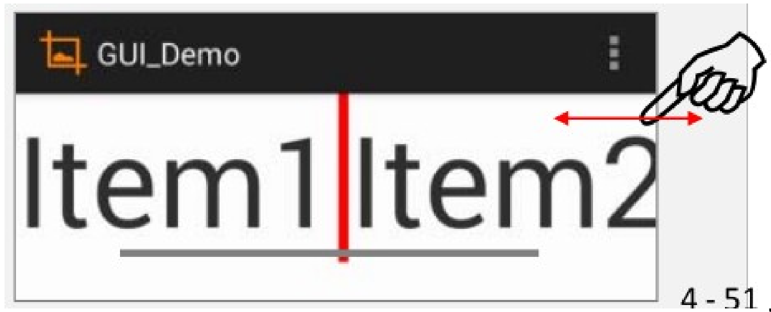


Vertical ScrollView Layout

```
<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>
```

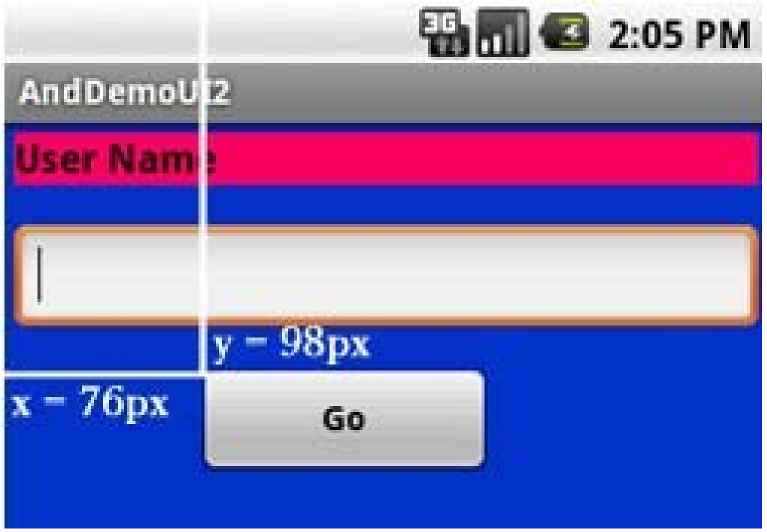


```
<HorizontalScrollView
xmlns:android="http://schemas.android.com/apk/r
es/android"
 android:id="@+id/myHorizontalScrollView1"
 android:layout_width="match_parent"
 android:layout_height="wrap_content" >
 <LinearLayout
  android:id="@+id/myLinearLayoutVertical"
  android:layout_width="match_parent"
  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="75sp" />
 <View
  android:layout_width="6dp"
  android:layout_height="match_parent"
  android:background="#ffff0000" />
<TextView
  android:id="@+id/textView2"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:text="Item2"
  android:textSize="75sp" />
<View
  android:layout_width="6dp"
  android:layout_height="match_parent"
  android:background="#ffff0000" />
<TextView
  android:id="@+id/textView3"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:text="Item3"
  android:textSize="75sp" />
</LinearLayout>
</HorizontalScrollView>
```



Miscellaneous: Absolute Layout (Deprecated)

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



```xml
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
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:padding="4dp"
>
  <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>
```

Next: Connecting Layouts to Java Code