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

Chapter 4. Graphical User Interfaces

Connecting Layouts to Java Code

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

Tài liệu tham khảo

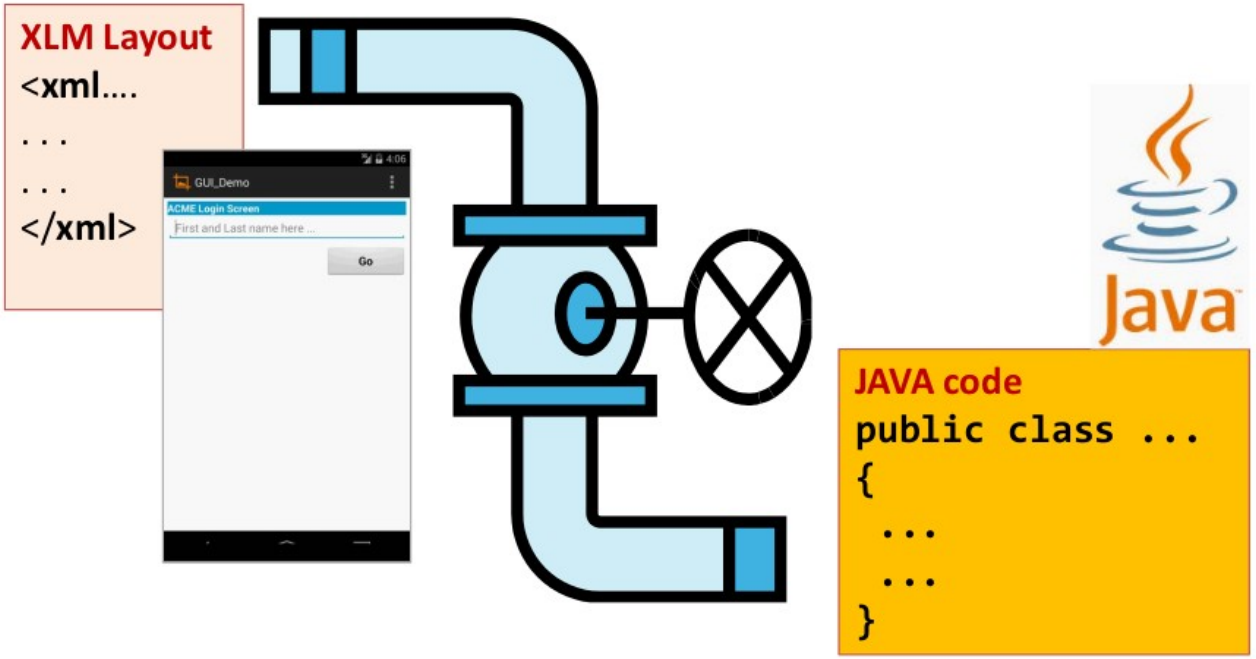
Mobile Application Development – Android OS, Victor Matos, Cleveland State University

Connecting Layouts to Java Code

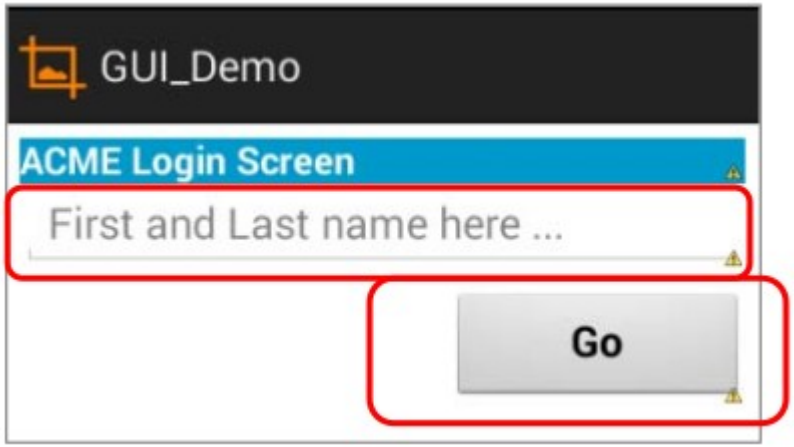
PLUMBING. You must ‘connect’ functional XML elements – such as buttons, text boxes, check boxes- with their equivalent Java objects.

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.



Connecting Layouts to Java Code




```xml
<!– XML LAYOUT -->
<LinearLayout
 android:id="@+id/myLinearLayout"
 ... >
 <TextView
  android:text="ACME Login Screen"
 ... />
 <EditText
  android:id="@+id/edtUserName"
 ... />
 <Button
  android:id="@+id/btnGo"
 ... />
</LinearLayout>
```

```java
package csu.matos.gui_demo;
import android...;
public class MainActivity extends Activity {
  EditText edtUserName;
  EditText edtUserName;
  Button btnGo;
  @Override
  protected void onCreate(Bundle
     savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   edtUserName = (EditText)
     findViewById(R.id.edtUserName);
   btnGo = (Button)
  findViewById(R.id.btnGo);
  }
}
```

What is the meaning of an Android Context?

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

- 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.

Connecting Layouts to Java Code

- 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);
```

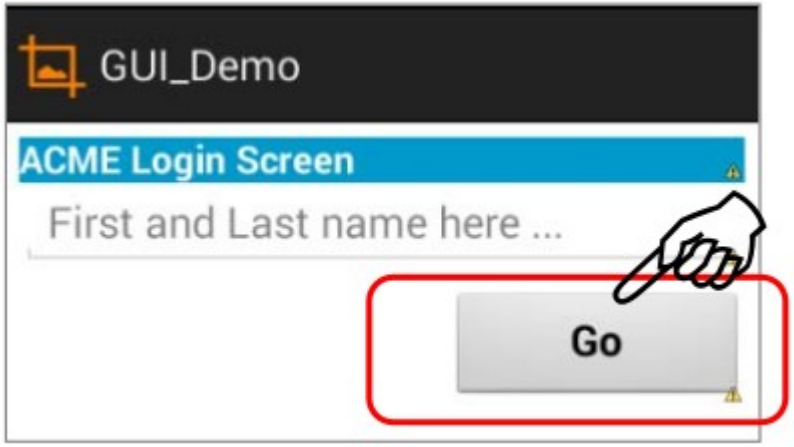
- Where R is a class automatically generated to keep track of resources available to the application. In particular R.id... is the collection of widgets defined in the XML layout (Use Eclipse’s Package Explorer, look at your /gen/package/R.java contents).

Connecting Layouts to Java Code

- 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.

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.



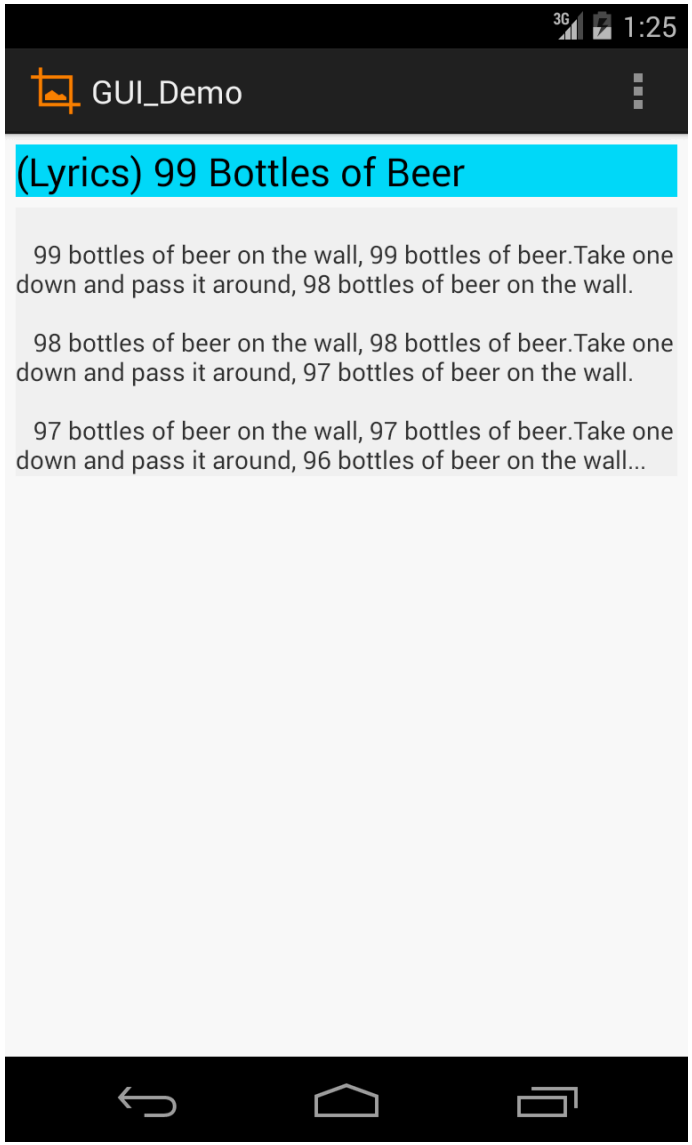

```
Button btnGo = (Button) findViewById(R.id.btnGo);
btnGo.setOnClickListener(new OnClickListener() {
  @Override
  public void onClick(View v) {
    // get userName and validate against some database
    // put some more logic here...
  }
});
```

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

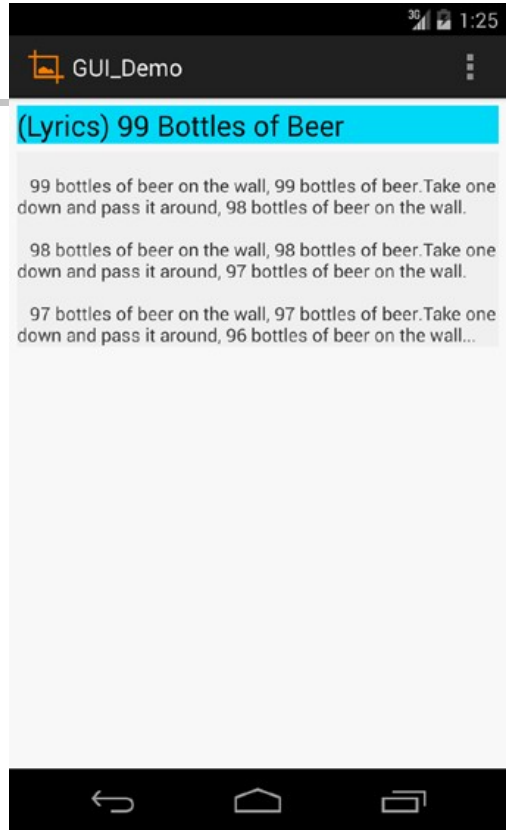
Basic Widgets: TextViews

- 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("<b>bold</b> string")
```



```
<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="\n\t99 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\n\t98 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\n\t97 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>
```



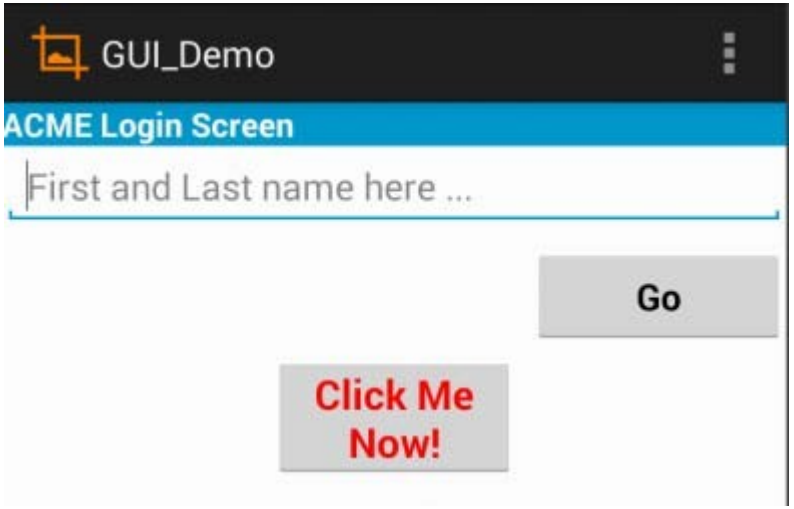
Example 8 - TextViews

Basic Widgets: Buttons

- A Button widget allows the simulation of a GUI clicking action.
- Button is a subclass of TextView. Therefore formatting a button’s face is similar to 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.

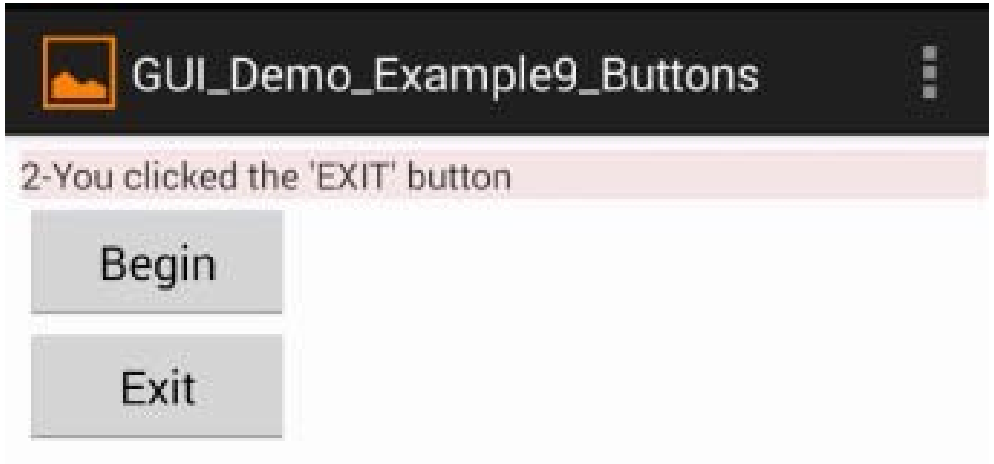
Basic Widgets: Buttons

```
<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" />
```

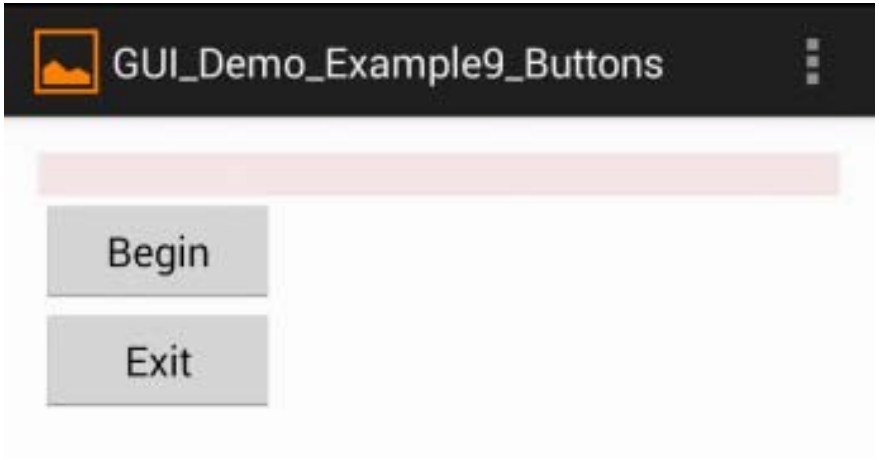


Example 9: Connecting Multiple Buttons

- 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.

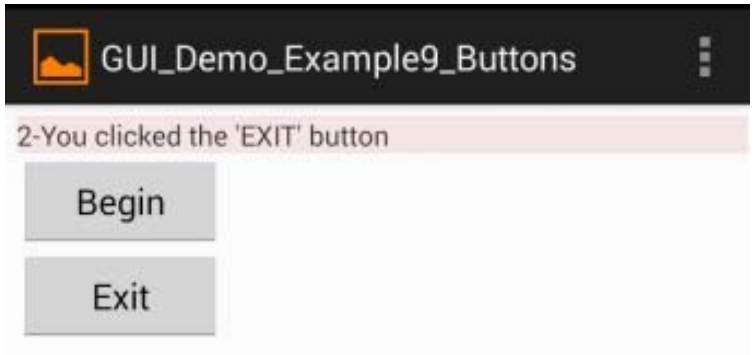


```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.com/tools"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:orientation="vertical"
 android:padding="6dp" >
 <TextView
  android:id="@+id/txtMsg"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:background="#88eed0d0" />
 <Button
  android:id="@+id/btnBegin"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:ems="5"
  android:text="Begin" />
 <Button
  android:id="@+id/btnExit"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:ems="5"
  android:text="Exit" />
</LinearLayout>
```



Layout

```
public class MainActivity extends Activity implements OnClickListener {
  TextView txtMsg;
  Button btnBegin;
  Button 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
```



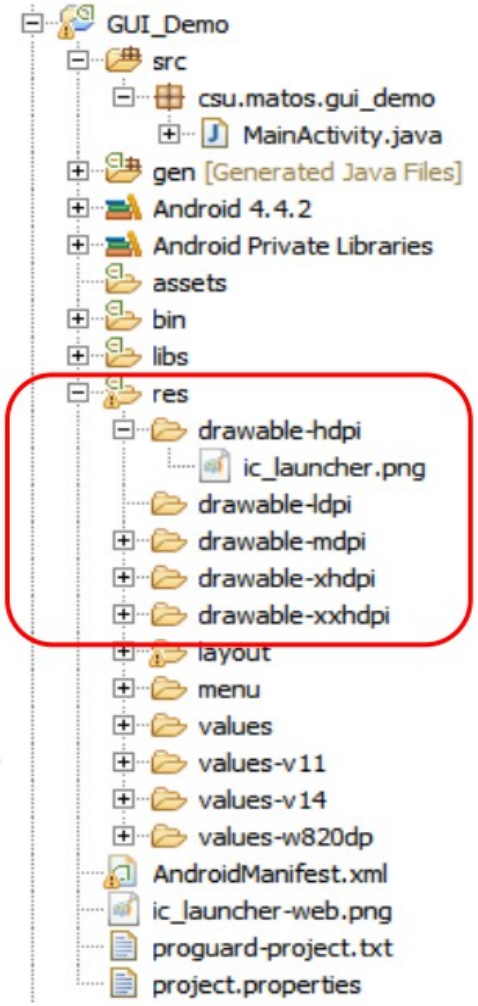
Basic Widgets: ImageView & ImageButton

- ImageView and ImageButton allow the embedding of images in your applications (gif, jpg, png, etc).
- Analogue to TextView and Button controls (respectively).
- Each widget takes an

  android:src or

  android:background

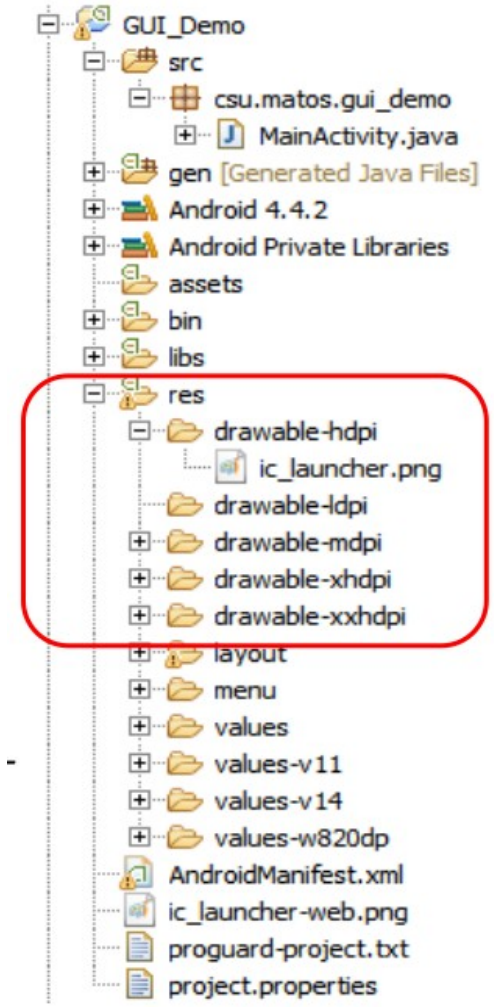
attribute (in an XML layout) to specify what picture to use.



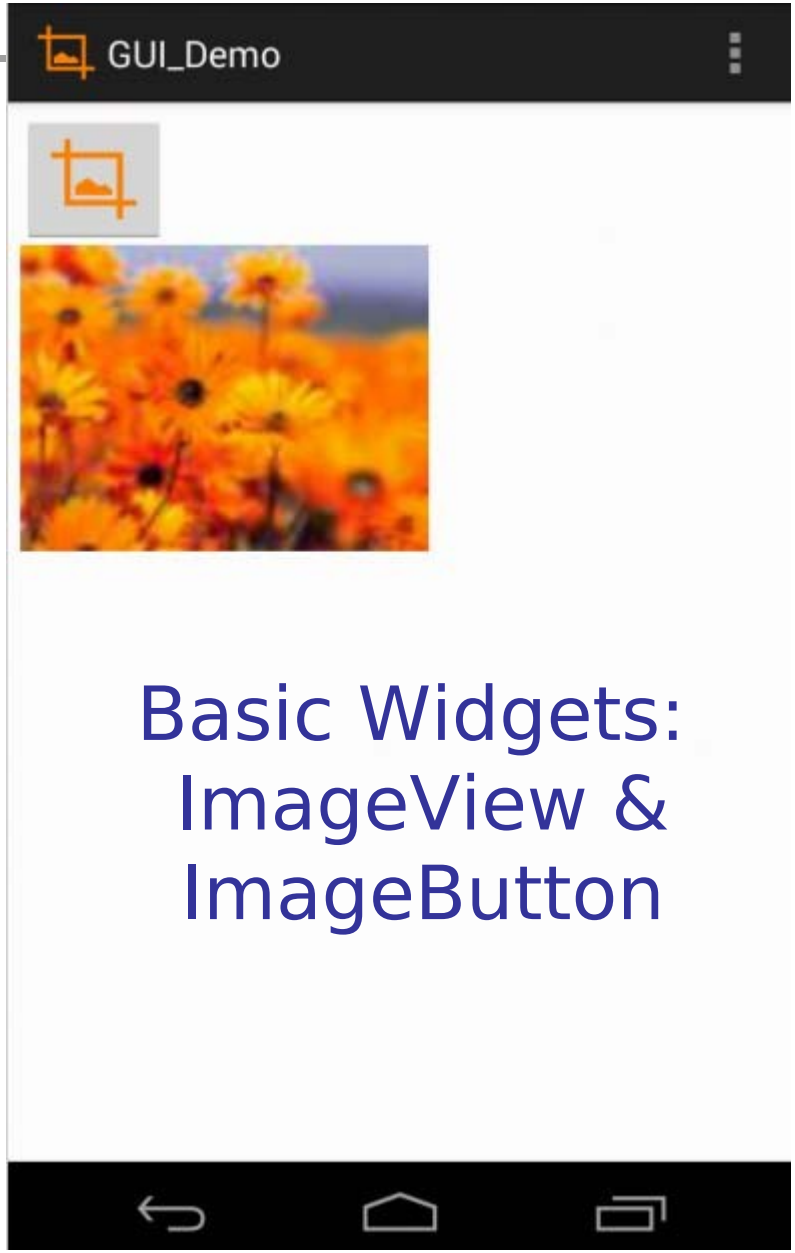
Basic Widgets: ImageView & ImageButton

- Pictures are stored in the res/drawable folder (optionally a 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). Details available at:

http://developer.android.com/design/style/iconography.html



```
<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" >
  </ImageButton>
  <ImageView
    android:id="@+id/imgView1"
    android:layout_width="200dp"
    android:layout_height="150dp"
    android:scaleType="fitXY"
    android:src="@drawable/flowers1" >
  </ImageView>
</LinearLayout>
```

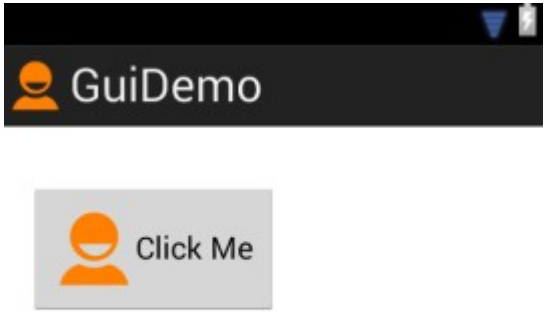


Basic Widgets: ImageView & ImageButton

Basic Widgets: Buttons - Combining Images & Text

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

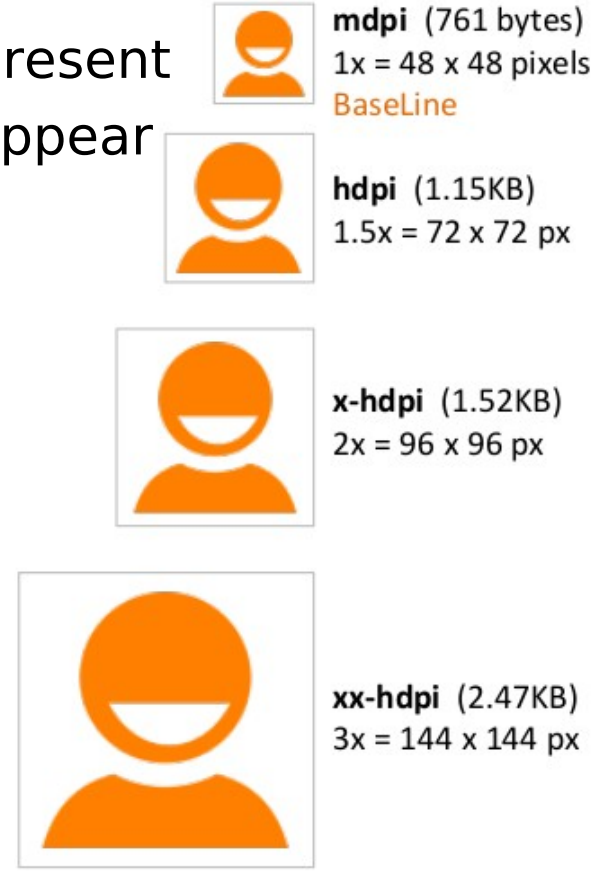
```
<LinearLayout
 . . .
 <Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:drawableLeft="@drawable/ic_launcher"
  android:gravity="left|center_vertical"
  android:padding="15dp"
  android:text="Click me" />

</LinearLayout>
```



Basic Widgets: 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



Detailed information on Android's iconography is available at: http://developer.android.com/design/style/iconography.html

Basic Widgets: EditText Boxes

- 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")

txtBox.getText().toString()

Basic Widgets: EditText Boxes

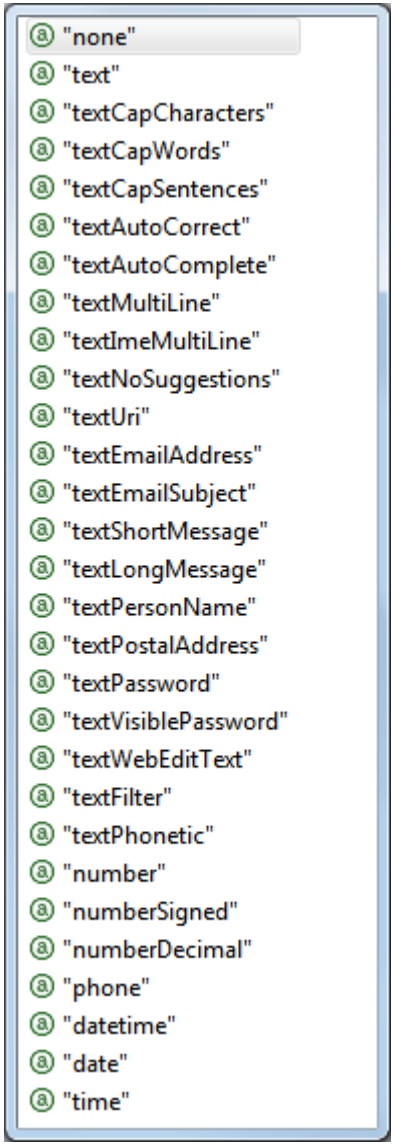
Input Type Formats

- 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"

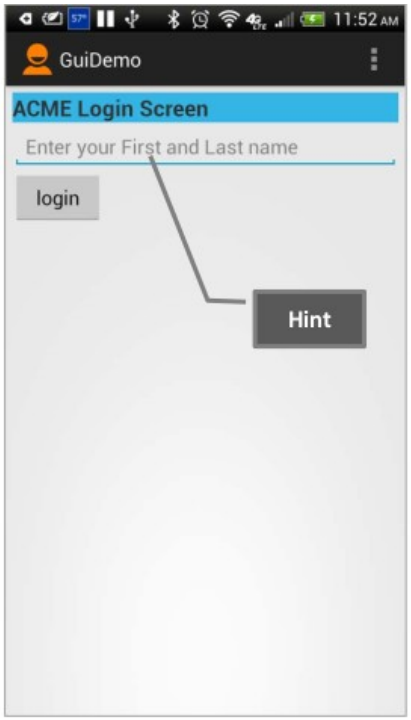
where choices include any of the single values shown in the figure. You may combine types, for instance: textCapWords|textAutoCorrect

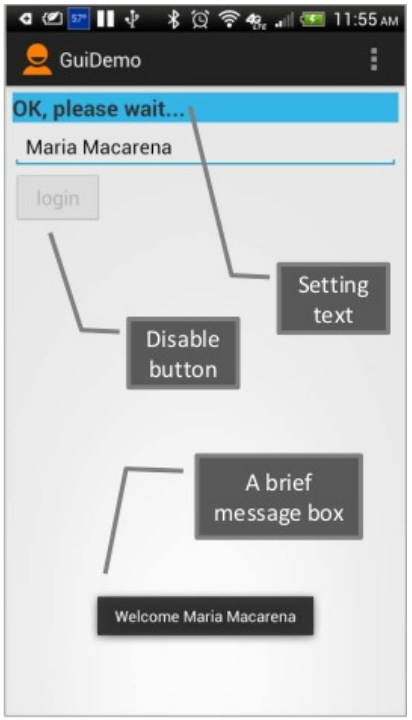
- Accepts text that capitalizes every word, incorrect words are automatically changed (for instance 'teh' is converted into 'the', and so on.



Example10: Login-Screen

- In this example we will create a simple login screen holding a label (TexView), a textBox (EditText), and a Button.
- When the EditTex 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.





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>
```

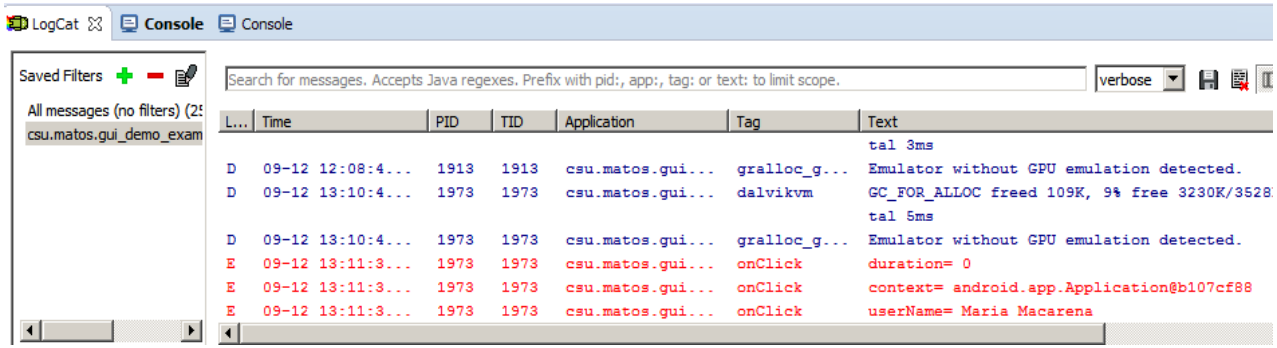
res/values/strings.xml

```
<?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>
```

```
public class MainActivity extends ActionBarActivity {
 // class variables representing UI controls to be controlled from the
 Java program
 TextView txtLogin;
 EditText edtUserName;
 Button btnLogin;
 // variables used with the Toast message class
 private Context context;
 private int duration = Toast.LENGTH_SHORT;
 @Override
 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  // show the login screen
  setContentView(R.layout.activity_main);
  context = getApplicationContext();
  // binding the UI's controls defined in "main.xml" to Java code
  txtLogin = (TextView) findViewById(R.id.txtLogin);
  edtUserName = (EditText) findViewById(R.id.edtUserName);
  btnLogin = (Button) findViewById(R.id.btnLogin);
```

MainActivity.java

```
// LISTENER: allowing the button widget to react to user interaction
btnLogin.setOnClickListener(new OnClickListener() {
  @Override
  public void onClick(View v) {
    String userName = edtUserName.getText().toString();
    Log.e("onClick ", "duration= " + duration);
    Log.e("onClick ", "context= " + context.toString());
    Log.e("onClick ", "userName= " + userName);
    // Log.e used for debugging – remove later!!!
    if (userName.equals("Maria Macarena")) {
      txtLogin.setText("OK, please wait...");
      Toast.makeText(getApplicationContext(),
          "Welcome " + userName, duration).show();
      btnLogin.setEnabled(false);
    } else {
      Toast.makeText(context, userName + " is not a valid USER",
          duration).show();
    }
  }
});// onClick
}// onCreate
```



```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
  // Inflate the menu; this adds items to the action bar if it is present.
  getMenuInflater().inflate(R.menu.main, menu);
  return true;
}
}
@Override
public boolean onOptionsItemSelected(MenuItem item) {
  // Handle action bar item clicks here. The action bar will
  // automatically handle clicks on the Home/Up button, so long
  // as you specify a parent activity in AndroidManifest.xml.
  int id = item.getItemId();
  if (id == R.id.action_settings) {
    return true;
  }
  return super.onOptionsItemSelected(item);
}
```

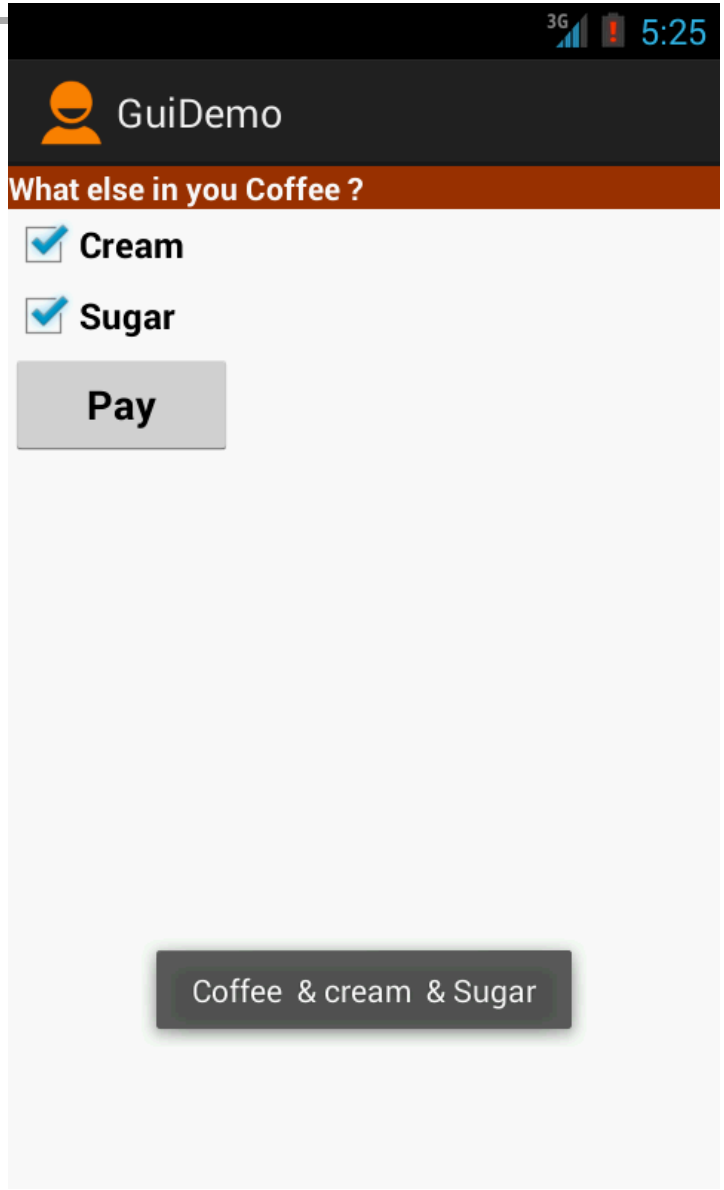
Your turn!
(working as a minimalist developer)

Implement any/all of the following projects using simple UI controls (EditText, TextView, buttons)

1. Currency Exchange calculator
2. Tip Calculator
3. Simple Flashlight

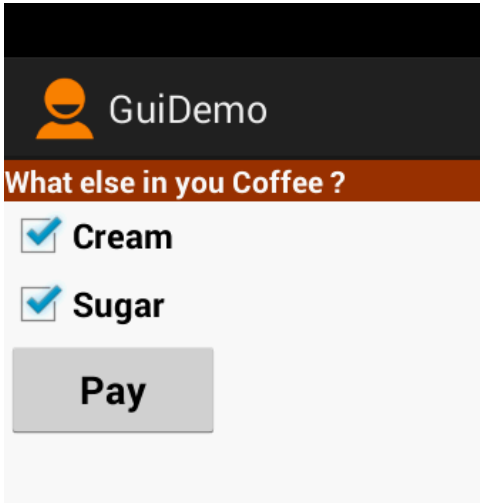
Basic Widgets: CheckBoxes

- 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 any time, 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.

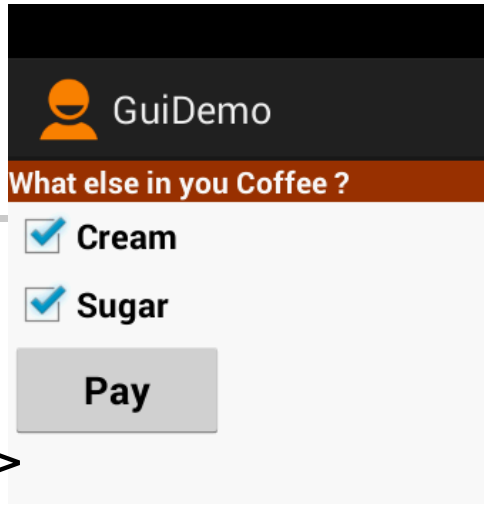


Layout

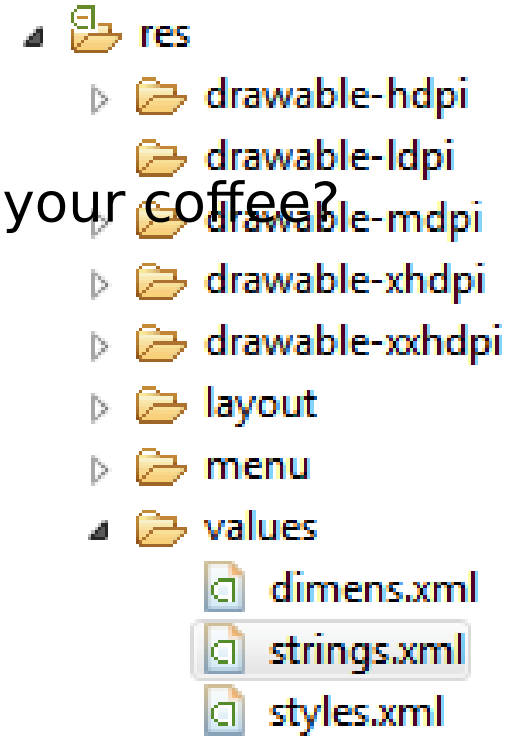
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:padding="6dp"
  android:orientation="vertical" >
  <TextView
    android:id="@+id/labelCoffee"
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



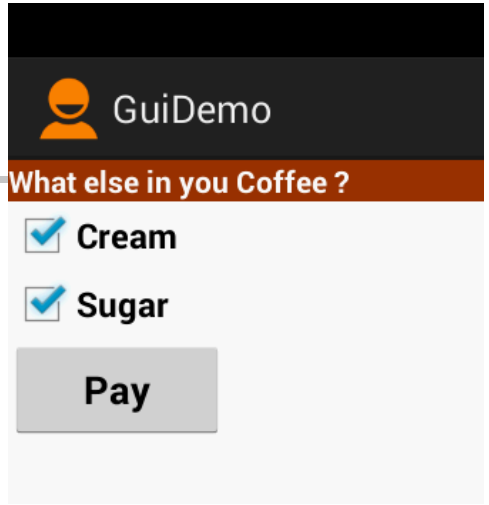
res/values/strings




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

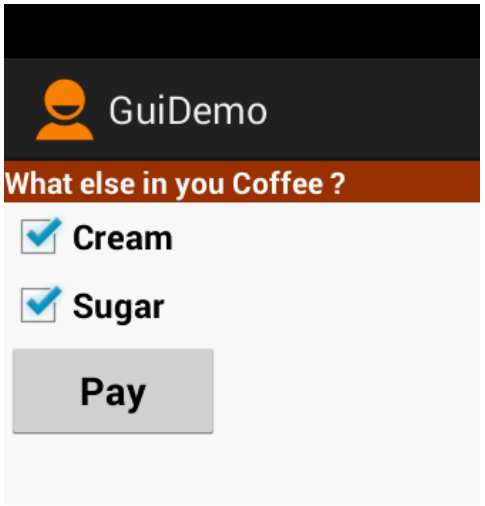


MainActivity.java




```
public class MainActivity extends Activity {
  CheckBox chkCream;
  CheckBox chkSugar;
  Button btnPay;
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // binding XMI 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
}//class
```

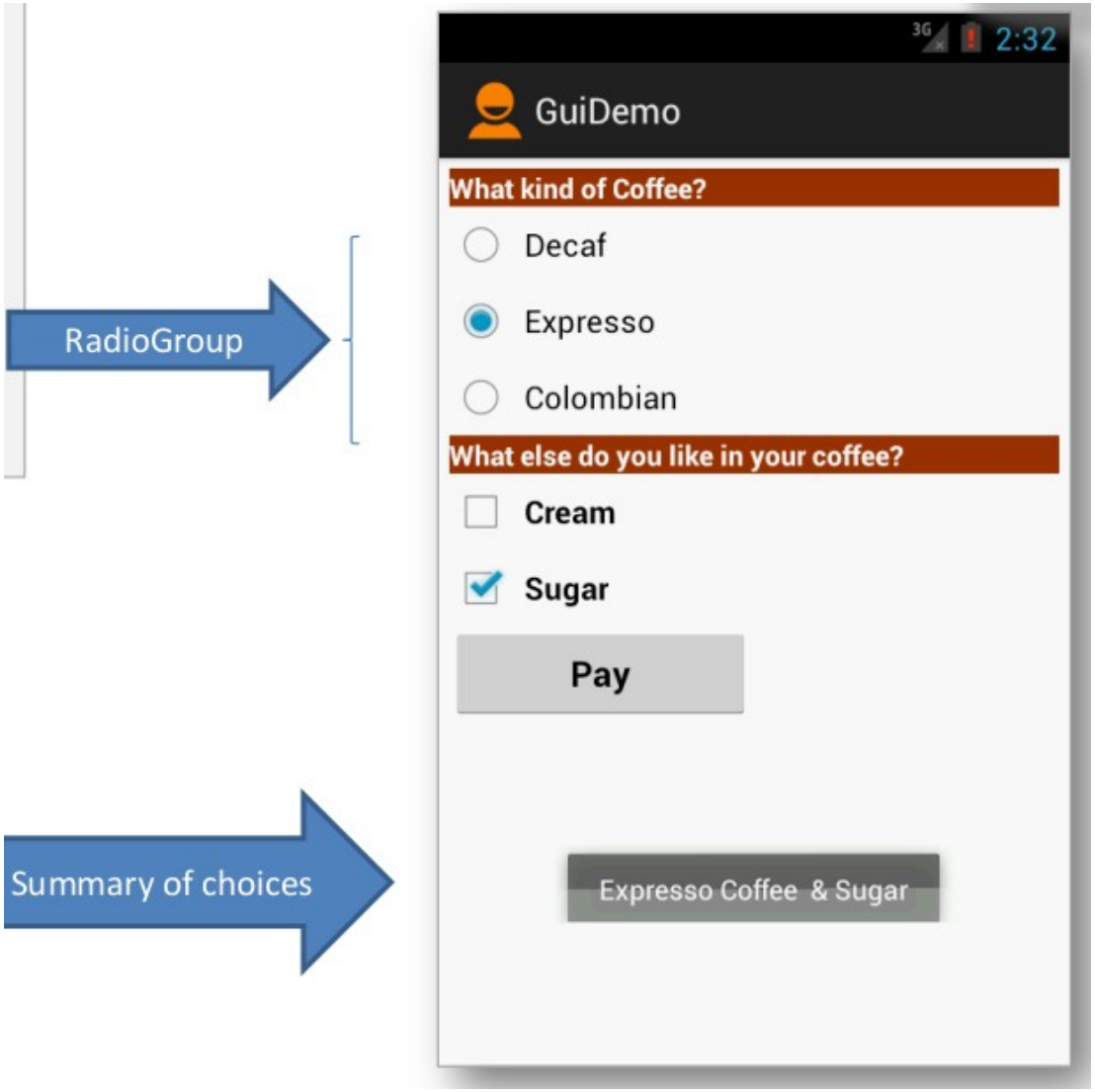


Basic Widgets: CheckBoxes

- 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().

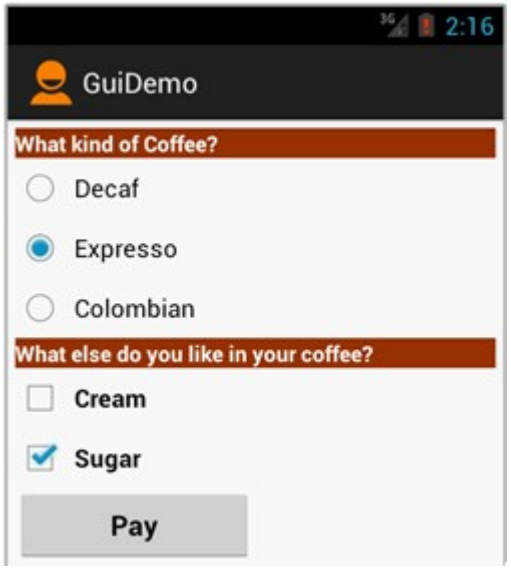
Example

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
  android:id="@+id/radioGroupCoffeeType"
  android:layout_width="match_parent"
  android:layout_height="wrap_content" >
  <RadioButton
    android:id="@+id/radDecaf"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/decaf" />
  <RadioButton
    android:id="@+id/radExpresso"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/expresso" />
  <RadioButton
    android:id="@+id/radColombian"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"
    android:text="@string/colombian" />
</RadioGroup>
```

```
<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" />
```



MainActivity.java

```
public class MainActivity extends Activity {
  CheckBox chkCream;
  CheckBox chkSugar;
  Button btnPay;
  RadioGroup radCoffeeType;
  RadioButton radDecaf;
  RadioButton radExpresso;
  RadioButton radColombian;
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    chkCream = (CheckBox) findViewById(R.id.chkCream);
    chkSugar = (CheckBox) findViewById(R.id.chkSugar);
    btnPay = (Button) findViewById(R.id.btnPay);
    radCoffeeType = (RadioGroup)
        findViewById(R.id.radioGroupCoffeeType);
    radDecaf = (RadioButton) findViewById(R.id.radDecaf);
    radExpresso = (RadioButton) findViewById(R.id.radExpresso);
    radColombian = (RadioButton) findViewById(R.id.radColombian);
```

```
// LISTENER: wiring button-events-&-code
btnPay.setOnClickListener(new OnClickListener() {
@Override
public void onClick(View v) {
  String msg = "Coffee ";
  if (chkCream.isChecked())    msg += " & cream ";
  if (chkSugar.isChecked())     msg += " & Sugar";
  // get 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 (radExpresso.isChecked())   msg = "Expresso " + msg;
  // similarly you may use .isChecked() on each RadioButton
  if (radDecaf.isChecked())    msg = "Decaf " + msg;
  Toast.makeText(getApplicationContext(), msg, 1).show();
  // go now and compute cost...
}// onClick
});
}// onCreate
}// class
```

```
radGroupradioId =
  (RadioGroup)findViewById(R.id.radioGroup1);
int radioId = radGroupradioId.getCheckedRadioButtonId();
switch (radioId) {
  case R.id.radColombian: msg += " Colombian "; break;
  case R.id.radExpresso: msg += " Expresso "; break;
  case R.id.radDecaf: msg += " Decaf "; break;
}
```

Alternative you may also manage a RadioGroup as follows (this is simpler because you don't need to define the individual RadioButtons

Miscellaneous: Useful UI Attributes & Java Methods

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()

User Interfaces

Appendix A. 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)

Appendix A. Using the @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?

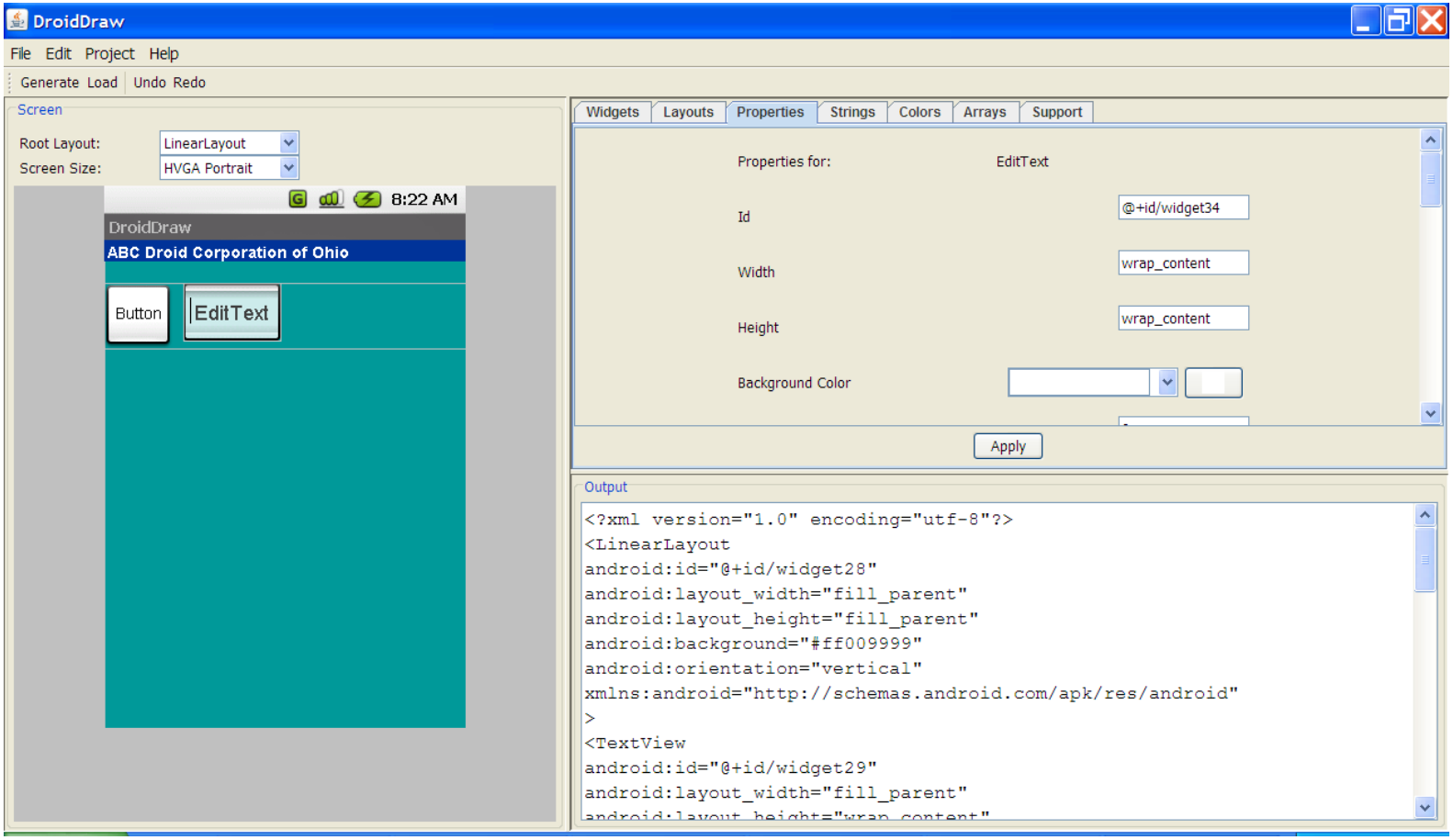
If the string is used in many places and its actual value changes we just update the resource file entry once.

It also provides some support for internationalization - easy to change a resource string from one language to another.

Appendix B. DroidDraw

A simple (but aging) GUI generator

LINK: www.droidDraw.org



Appendix C. Android Asset Studio

LINK: http://romannurik.github.io/AndroidAssetStudio/

This tool offers a number of options to craft high-quality icons and other displayed elements typically

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

Appendix D. 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:

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

G1 (base device 320x480): 155.92 dpi (3.7 in diagonally)

Nexus (480x800): 252.15 dpi

HTC One (1080x1920): 468 dpi (4.7 in)

Samsung S4 (1080x1920): 441 dpi (5.5 in)

Appendix D. Measuring Graphic Elements

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.

Appendix D. Measuring Graphic Elements

How Android deals with screen resolutions?

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 x 720dp

Large: screens are at least 640dp x 480dp

Normal : screens are at least 470dp x 320dp

Small: screens are at least 426dp x 320dp

A set of six generalized densities:

Ldpi: ~120dpi (low)

Mdpi : ~160dpi (medium)

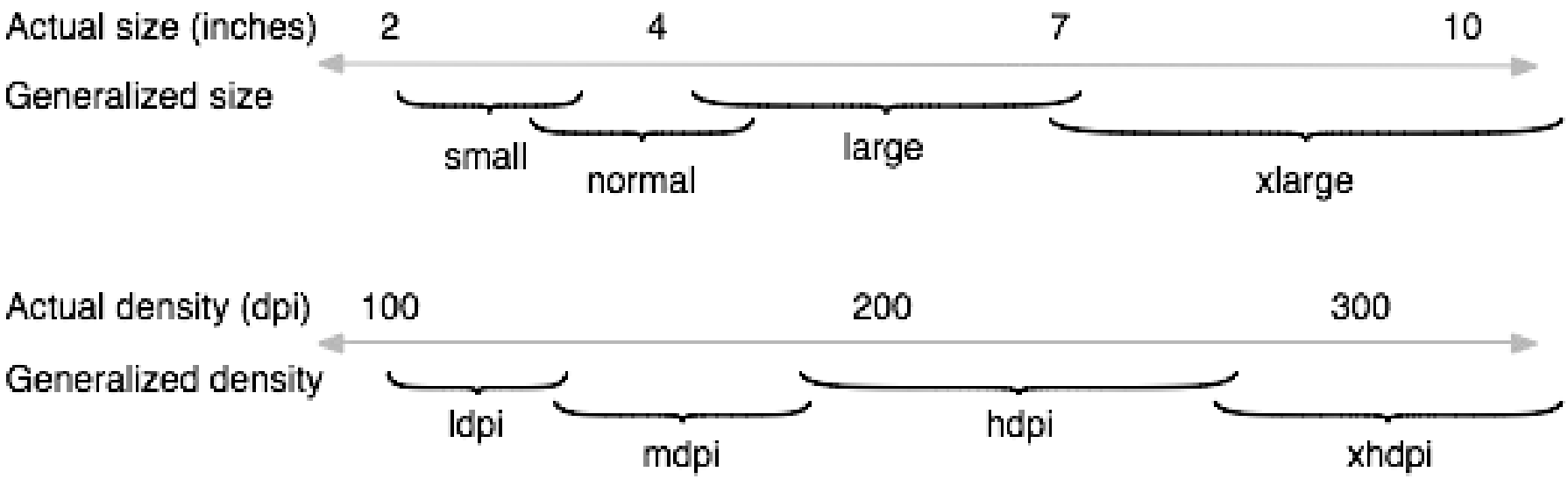
Hdpi: ~240dpi (high)

xhdpi ~320dpi (extra-high)

xxhdpi ~480dpi (extra-extra-high)

Xxxhdpi ~640dpi (extra-extra-extra-high)

Appendix D. Measuring Graphic Elements

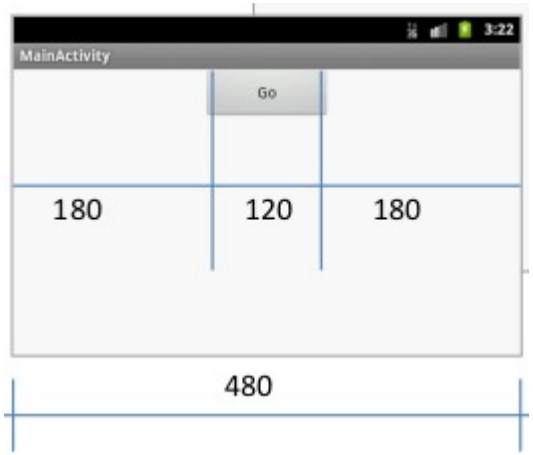


Taken from:
http://developer.android.com/guide/practices/screens_support.html

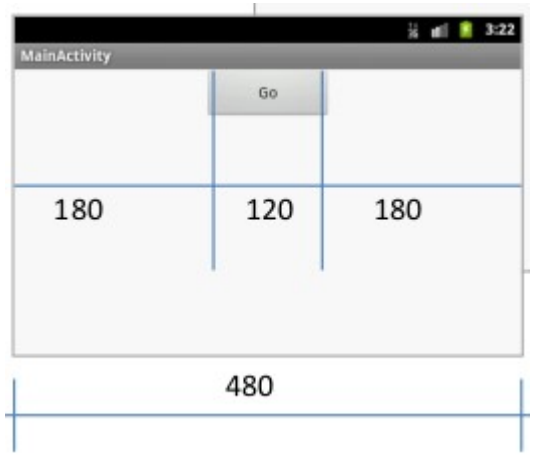
Appendix D. Measuring Graphic Elements

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*160] x [3*160])
- Assume you want a 120dp button to be placed in the middle of the screen.
- On portrait mode you could allocate the 320 horizontal pixels as [100 + 120 + 100].
- On Landscape mode you could allocate 480 pixels as [180 + 120 + 180].



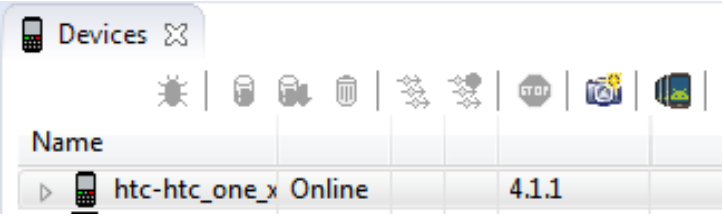
```
<Button
  android:id="@+id/button1"
  android:layout_height="wrap_content"
  android:layout_width="120dp"
  android:layout_gravity="center"
  android:text="@+id/go_caption" />
```

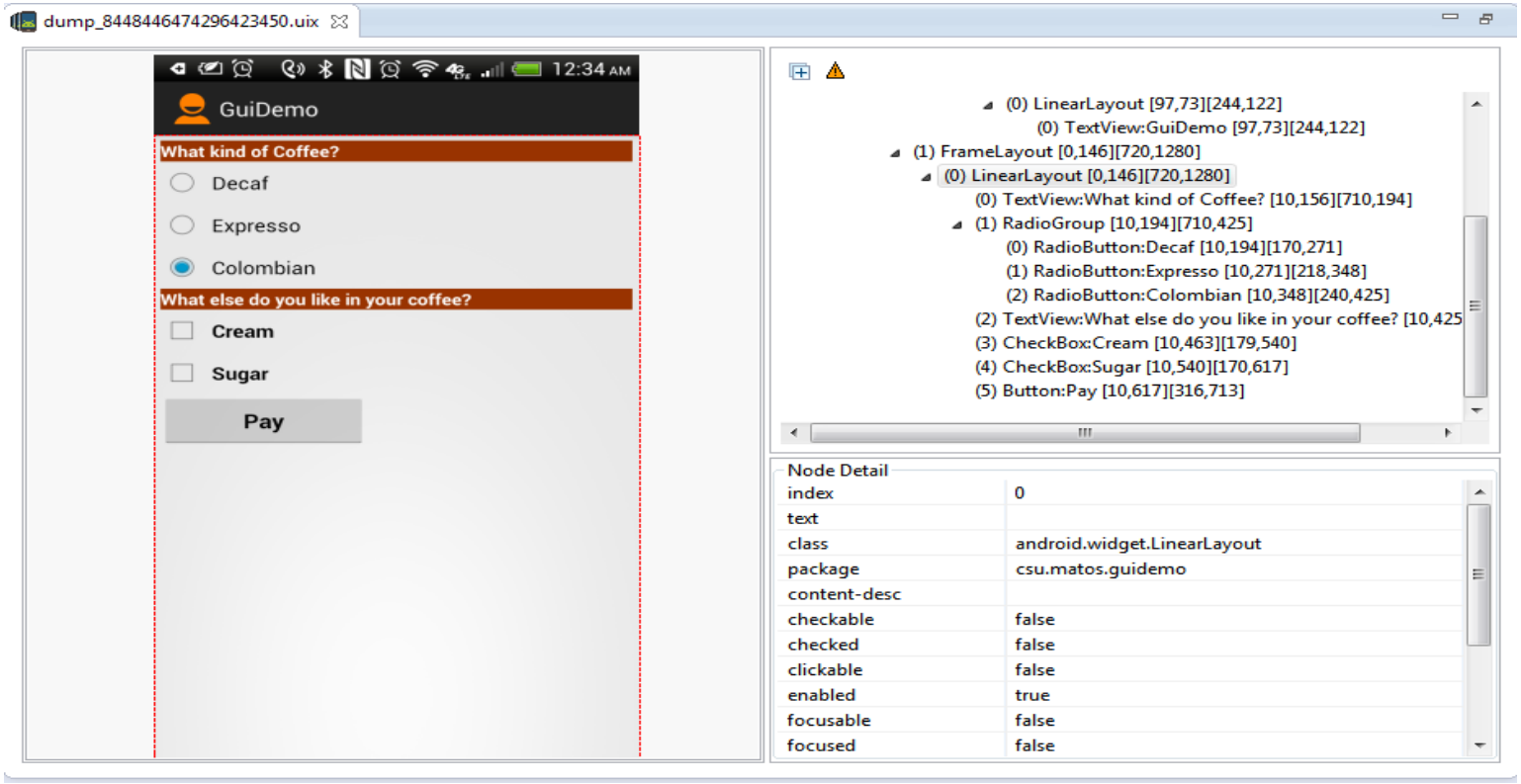


If the application is deployed on devices having a higher resolution the button is still mapped to the middle of the screen.

Appendix E. Hierarchy Viewer Tool

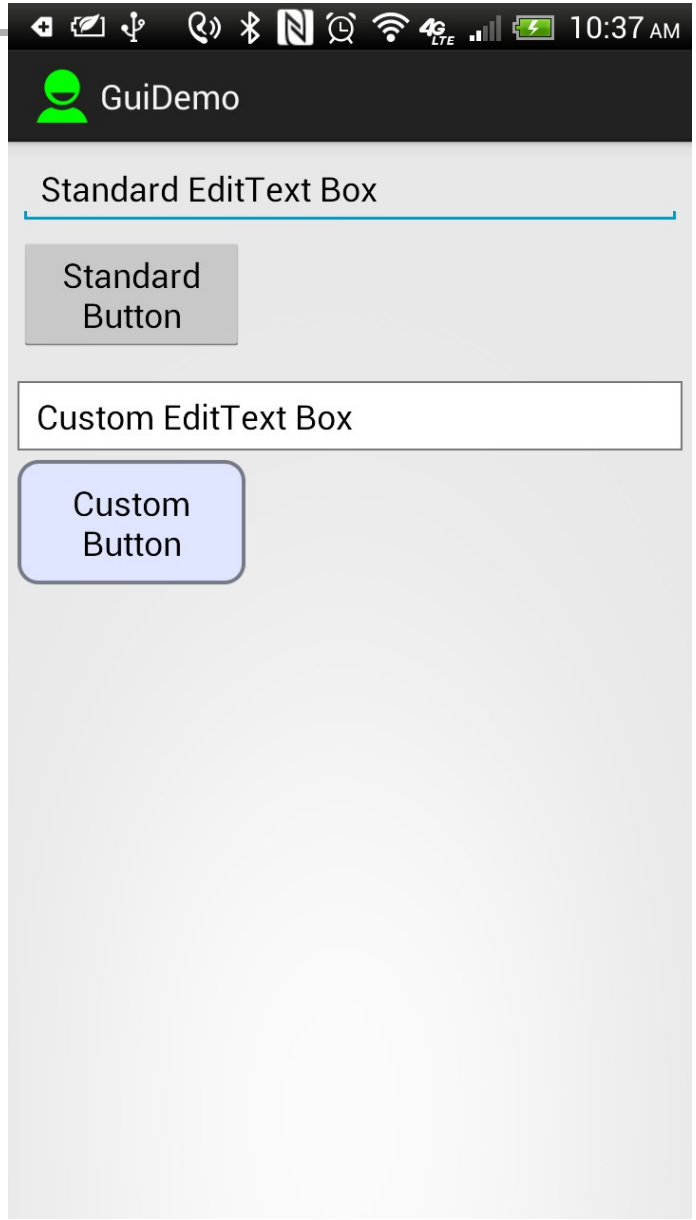
The HierarchyViewer Tool allows exploration of a displayed UI. Use DDMS > Click on Devices > Click on HierarchyViewer icon (next to camera)





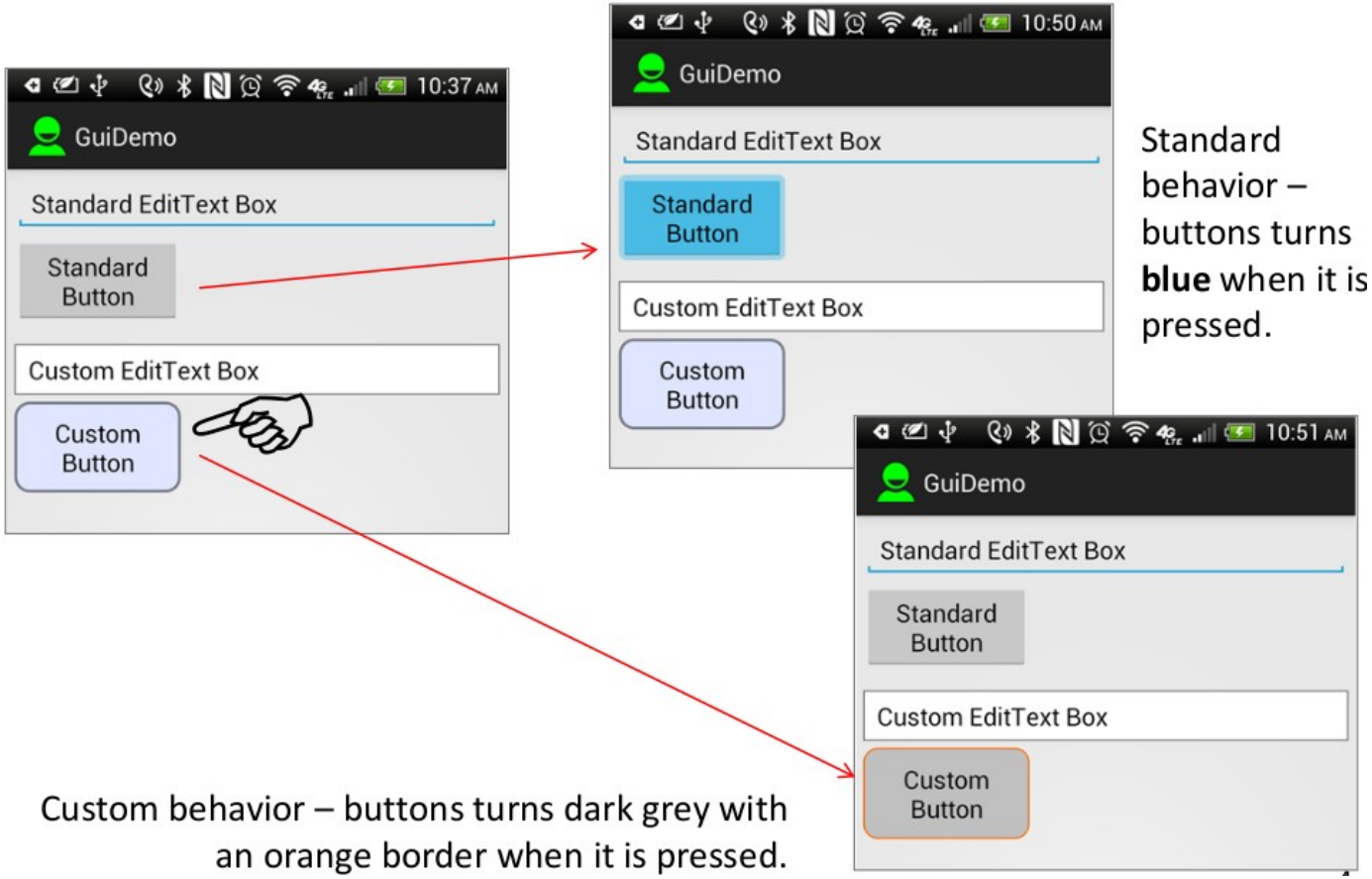
Appendix F. Customizing Widgets

1. The appearance of a widget can be adjusted by the user. For example a button widget could be modified by changing its shape, border, color, margins, etc.
2. Basic shapes include: rectangle, oval, line, and ring.
3. In addition to visual changes, the widget's reaction to user interaction could be adjusted for events such as: Focused, Clicked, etc.
4. The figure shows and EditText and Button widgets as normally displayed by a device running SDK4.3 (Ice Cream). The bottom two widgets (a TextView and a Button) are custom made versions of those two controls respectively.



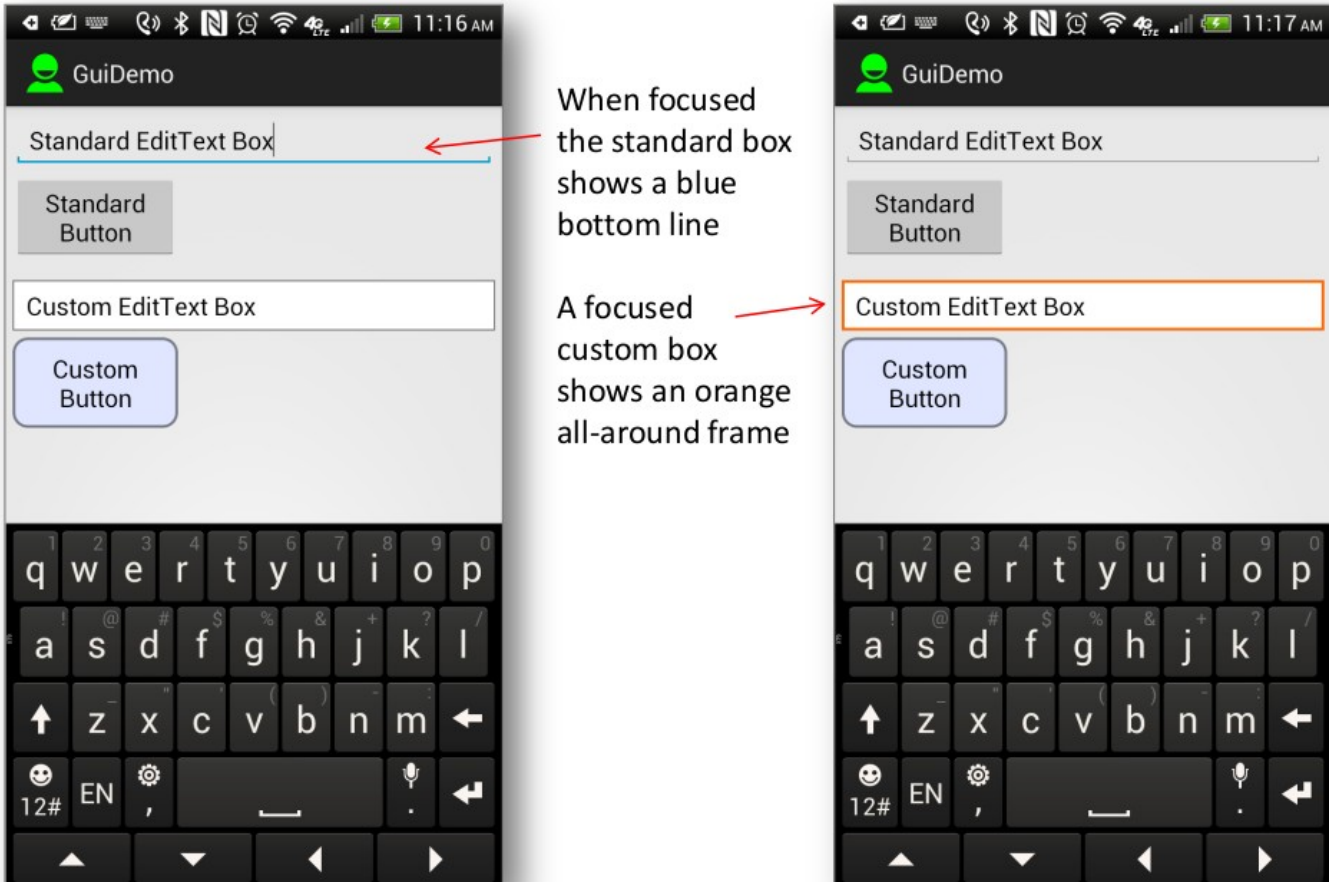
Appendix F. Customizing Widgets

- 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.



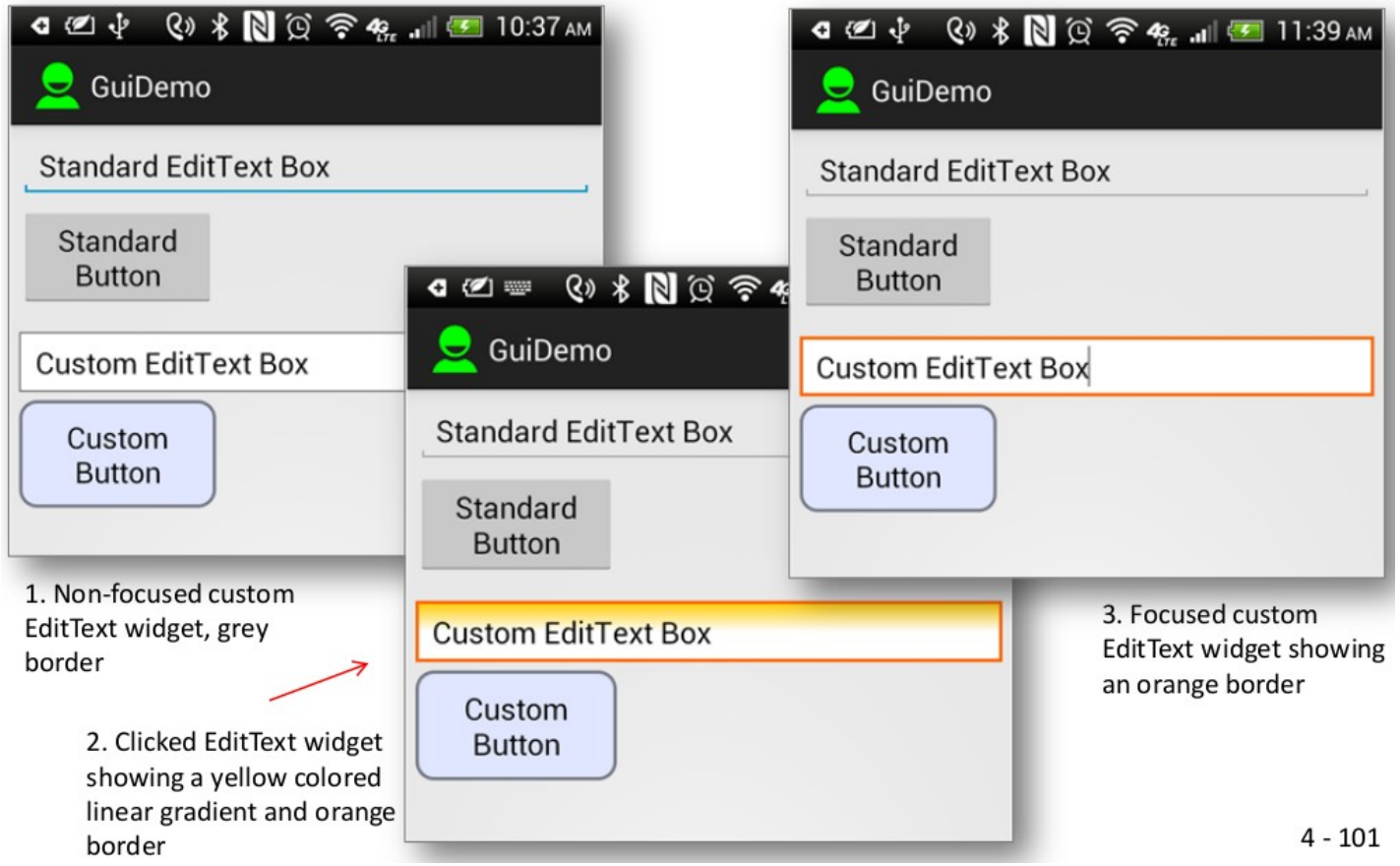
Appendix F. Customizing Widgets

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



Appendix F. 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.

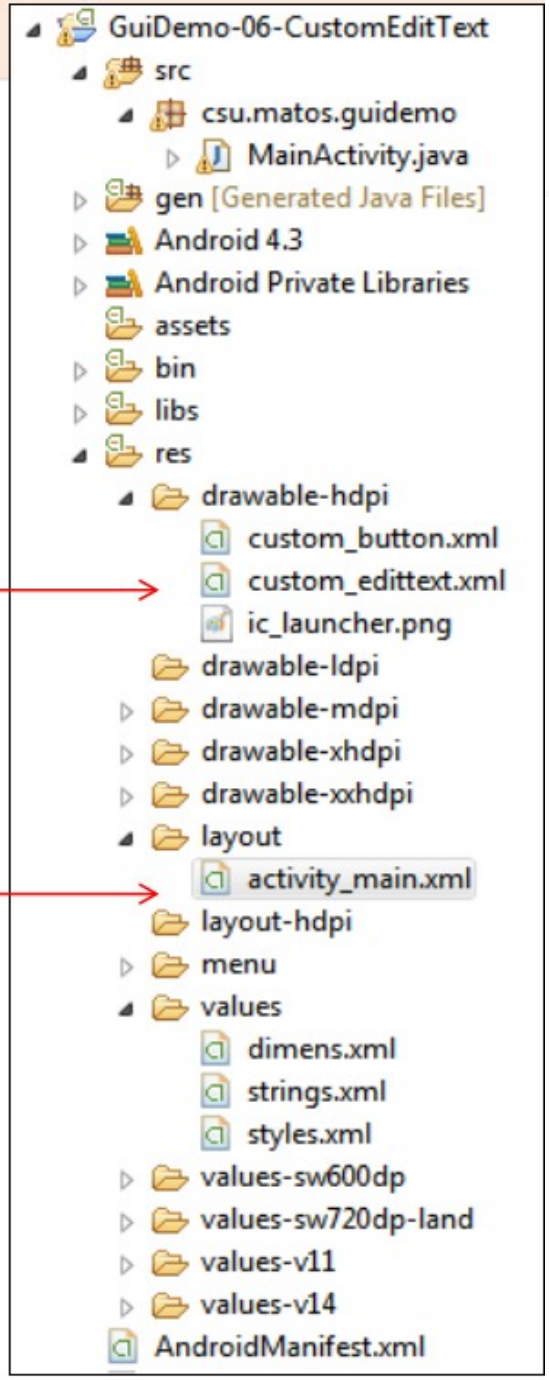


Appendix F. Customizing Widgets

Organizing the application

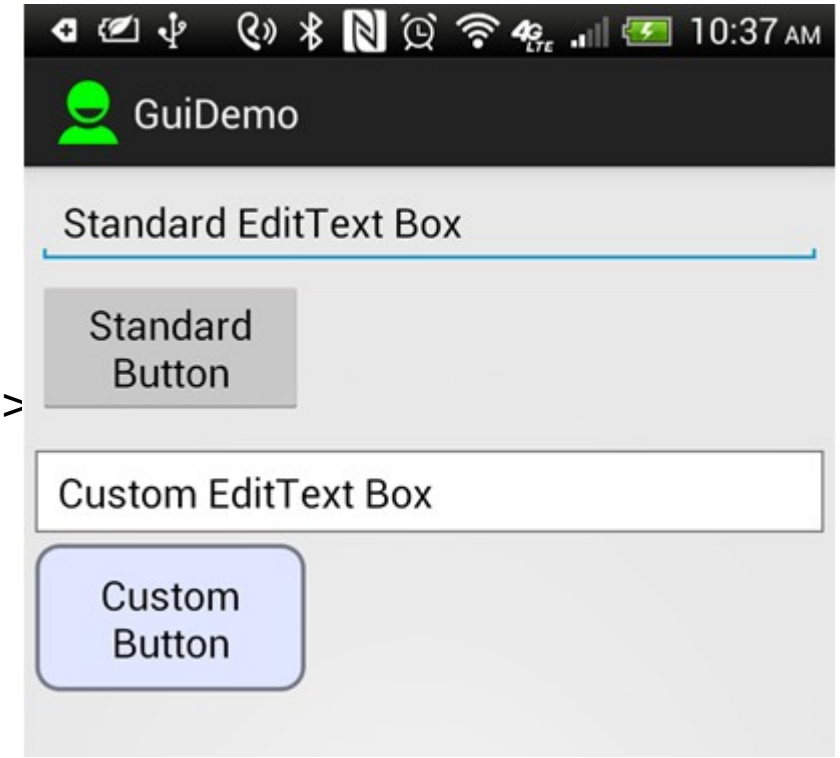
Definition of the custom templates for Button and EditText widgets

Layout referencing standard and custom made widgets



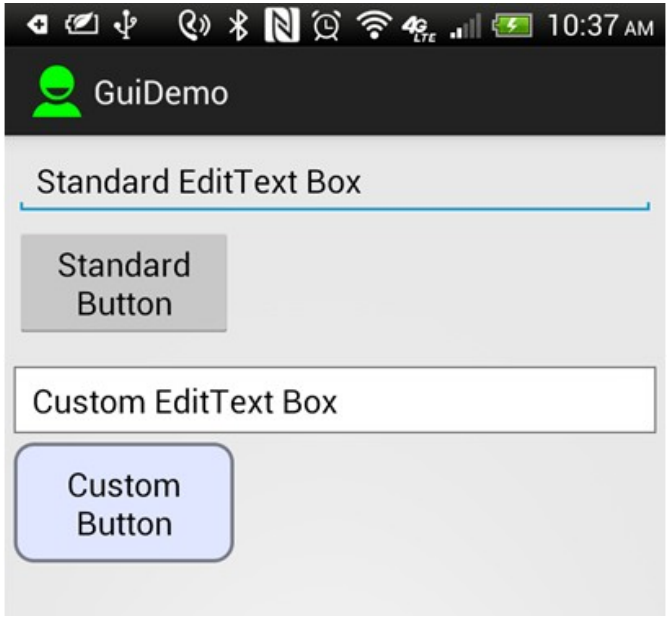
Layout

```
<?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" />
```



Layout

```
<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>
```



Resource: 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="standard_button">Standard
  Button</string>
  <string name="standard_edittext">Standard EditText
  Box</string>
  <string name="custom_button">Custom Button</string>
  <string name="custom_edittext">Custom EditText Box</
  string>
</resources>
```

Resource: res/drawable/custom_button.xml

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

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

Resource: res/drawable/custom_edittext.xml

The rendition of the custom made EditText widget is based on three states:

normal, state_focused, state_pressed.

```xml
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
<item android:state_pressed="true">
  <shape android:shape="rectangle">
    <gradient
      android:angle="90"
      android:centerColor="#FFffffff"
      android:endColor="#FFffcc00"
      android:startColor="#FFffffff"
      android:type="linear" />
  <stroke
    android:width="2dp"
    android:color="#FFff6600" />
  <corners android:radius="0dp" />
  <padding android:left="10dp"
    android:top="6dp"
    android:right="10dp"
    android:bottom="6dp" />
</shape>
</item>
```

Custom EditText Box

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

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

Custom EditText Box

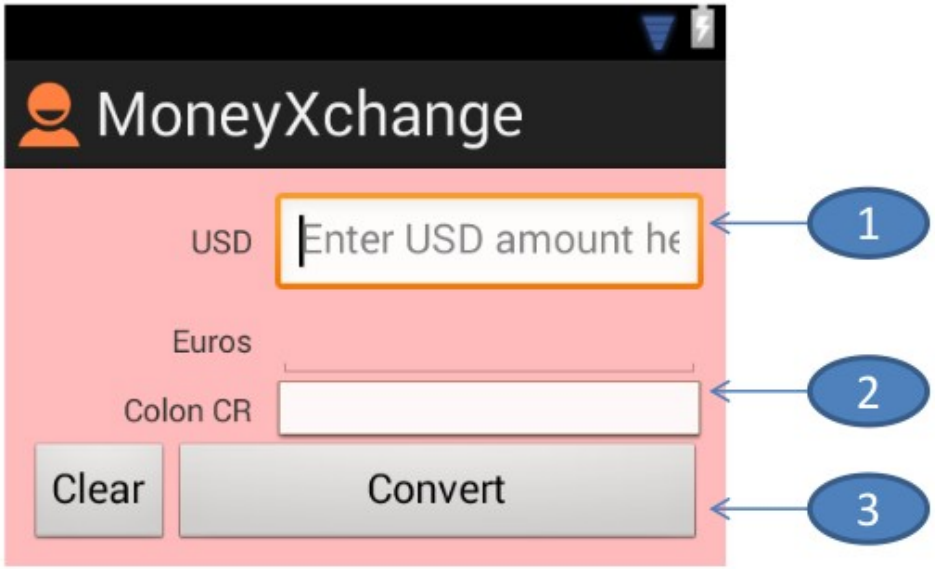
Custom EditText Box

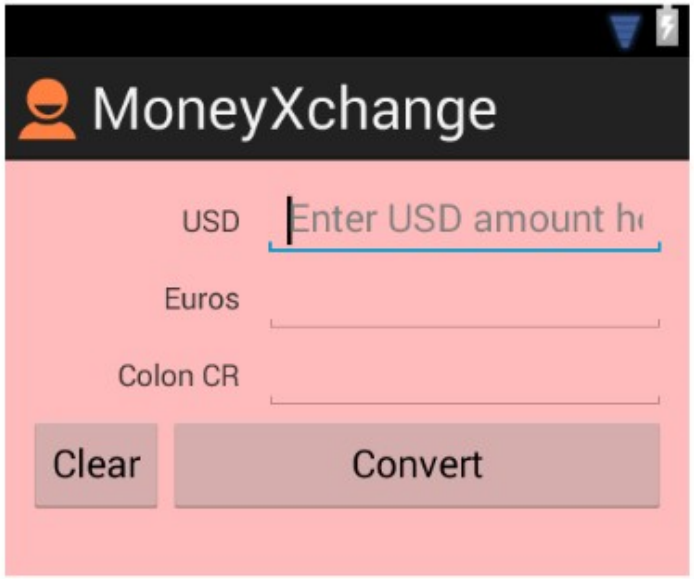
Appendix G: Fixing Bleeding Background Color

- 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.

Appendix G: Fixing Bleeding Background Color

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

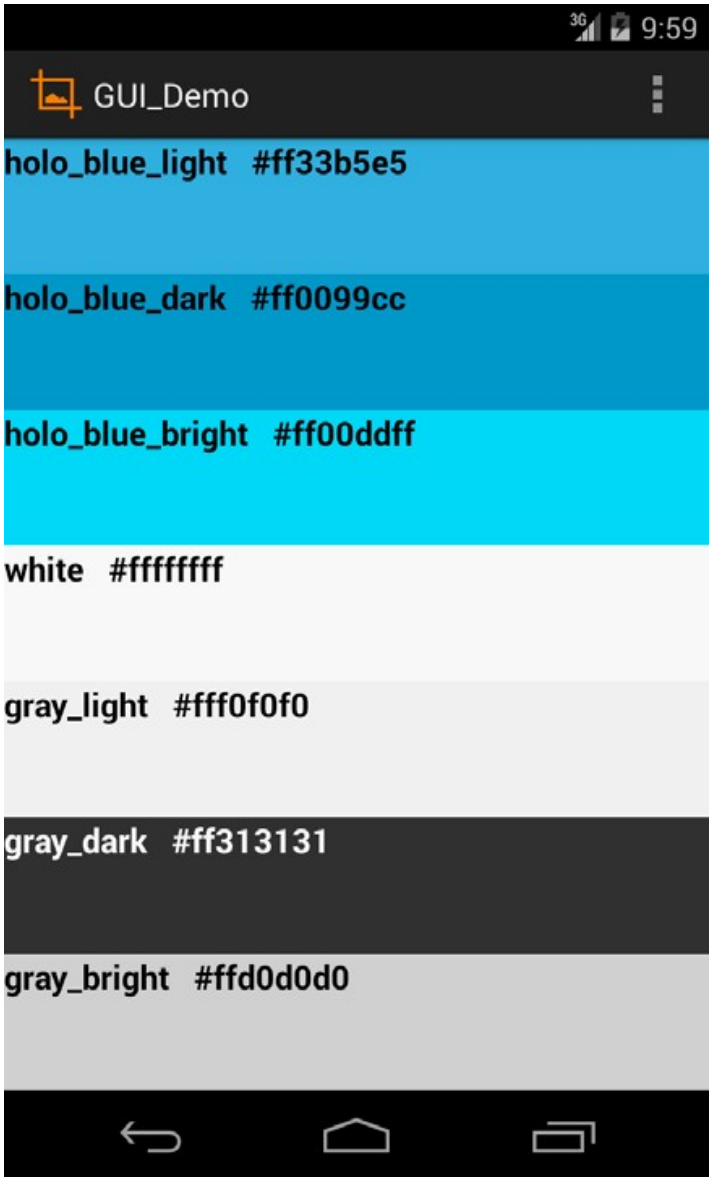


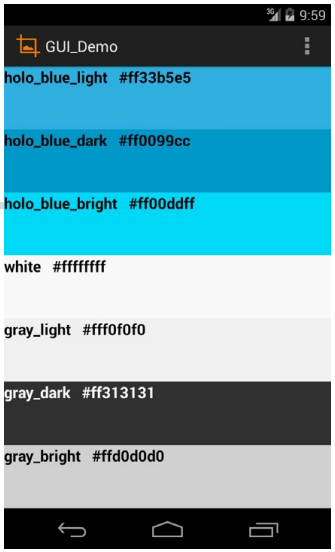


Appendix H: Useful Color Theme (Android Holo)

- 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>
```

Next: Android-Chapter05-ListBased-Widgets.pdf