User Interface - Components and Event Listener

UI Components

Palette

- Layouts
- Widgets
- Text Fields
- Containers
- Date & Time
- Expert
- Custom

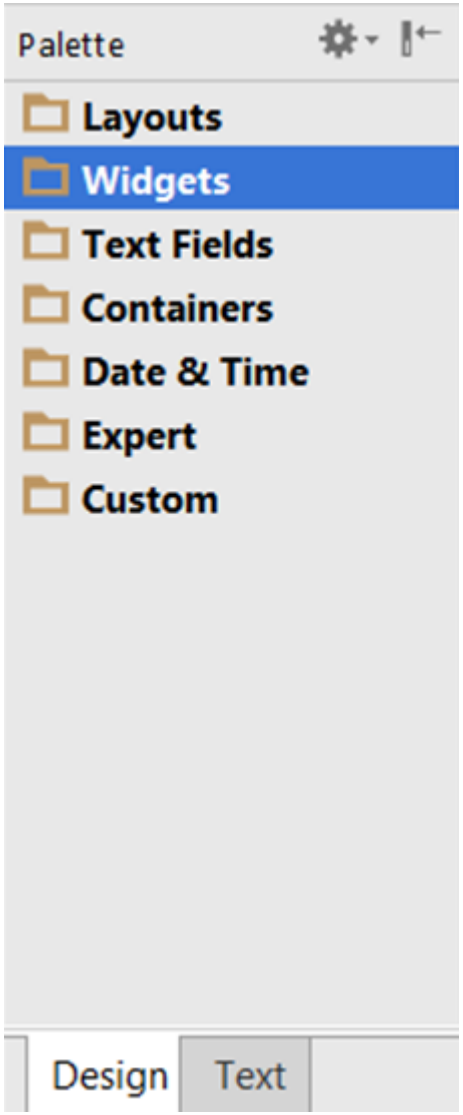
Design | Text

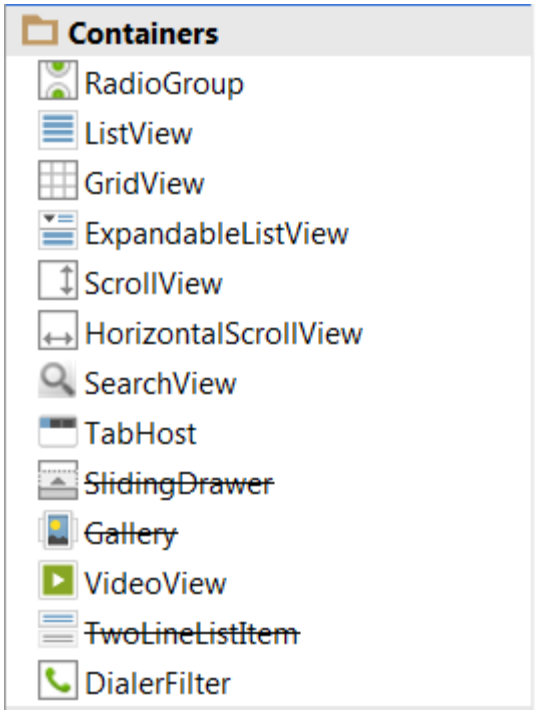
- Plain TextView
- Large Text
- Medium Text
- Small Text
- Button
- Small Button
- RadioButton
- CheckBox
- Switch
- ToggleButton
- ImageButton
- ImageView
- ProgressBar (Large)
- ProgressBar (Normal)
- ProgressBar (Small)
- ProgressBar (Horizontal)
- SeekBar
- RatingBar
- Spinner
- WebView

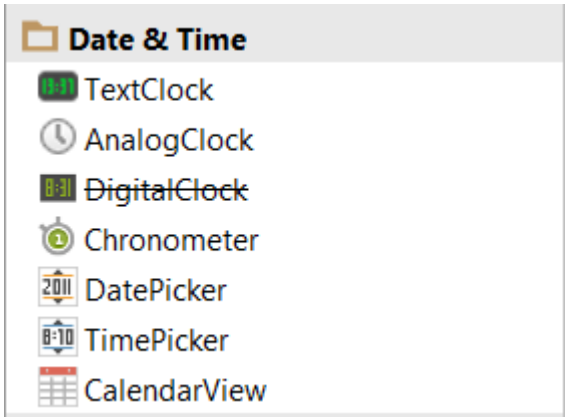
Text Fields

- Plain Text
- Person Name
- Password
- Password (Numeric)
- E-mail
- Phone
- Postal Address
- Multiline Text
- Time
- Date
- Number
- Number (Signed)
- Number (Decimal)

UI Components



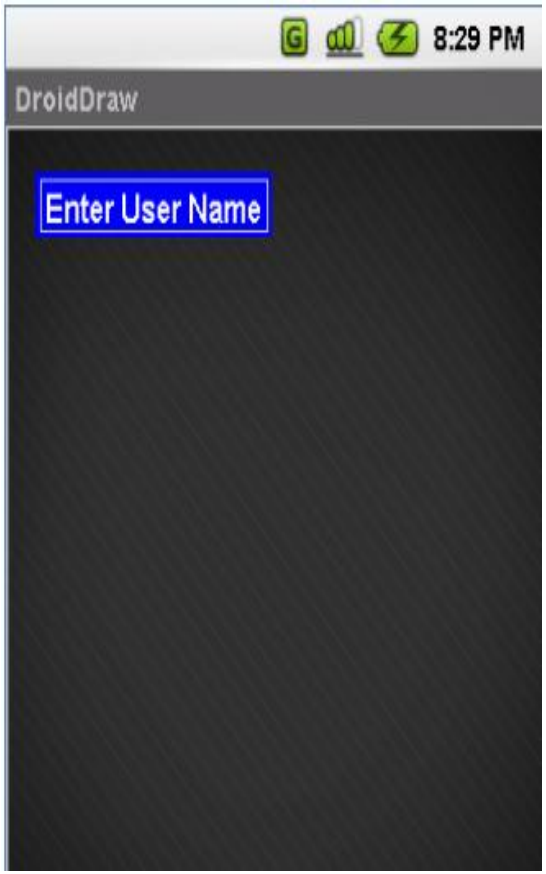




TextView

A label is called a **TextView.**

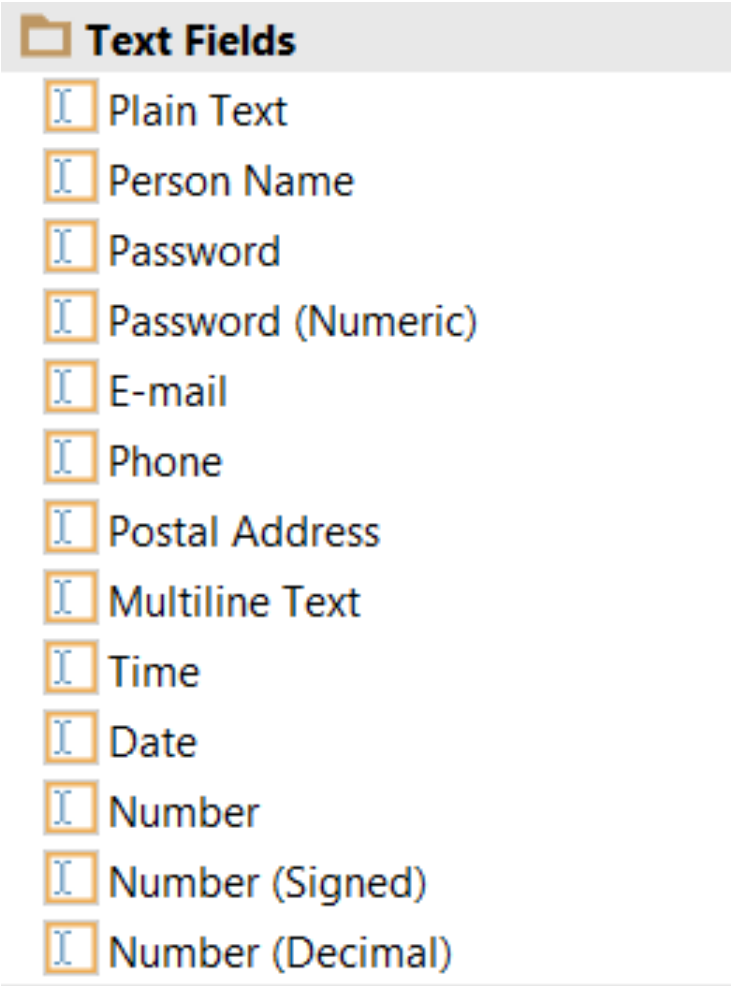
- TextViews are typically used to display a caption
- TextViews are *not editable, therefore they take no input*




```
-------
<TextView
        android:id="@+id/myTextView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#ff0000ff"
        android:padding="3px"
        android:text="Enter User Name"
        android:textSize="16sp"
        android:textStyle="bold"
        android:gravity="center"
        android:layout_x="20px"
        android:layout_y="22px“ >
</TextView>
-------
```

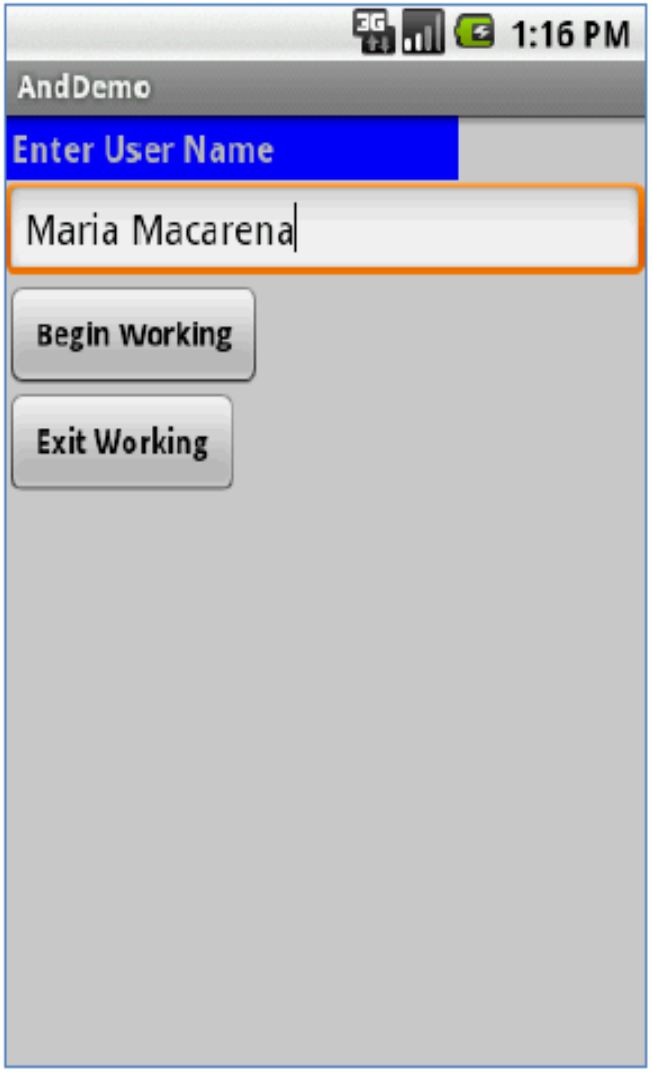
Text Fields

a) EditText



Text Fields (EditText)

- The **EditText** (or *textBox*) widget is an extension of *TextView* that allows updates.
- The control configures itself to be *editable.*
- Important Java methods are:
  txtBox.setText("someValue")
  and
  txtBox.getText().toString()

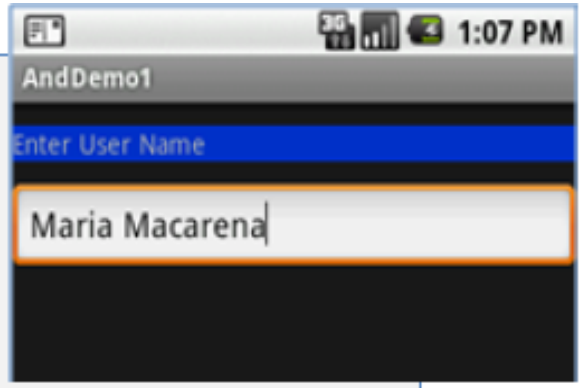


Text Fields (EditText)

Example

```
...
<EditText
    android:id="@+id/txtUserName"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="18sp"
    android:autoText="true"
    android:capitalize="words"
    android:hint="First Last Name"
>
</EditText>
...
```

Upper case words



Enter "teh"
It will be changed to: "The"

Suggestion (grey)

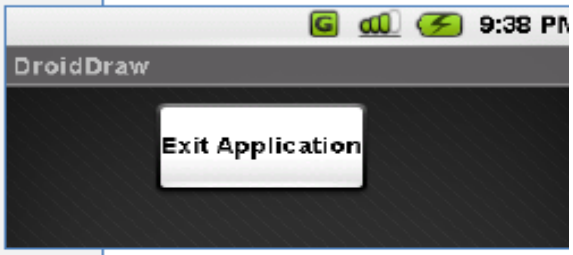
Text Fields (EditText)

android:hint	This is a default string to display when the text field is empty.
android:password	(true/false) Controls field visibility
android:singleLine	Is the field for Single line or Multiple line
android:inputType	The type determines what kind of characters are allowed inside the field, and may prompt the virtual keyboard to optimize its layout for frequently used characters.
android:imeOptions	Specify an action to be made when users have completed their input

Button

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

```
...
<Button
android:id="@+id/btnExitApp"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:padding="10px"
android:layout_marginLeft="5px"
android:text="Exit Application"
android:textSize="16sp"
android:textStyle="bold"
android:gravity="center"
android:layout_gravity="center_horizontal"
>
</Button>
```



Event Listener

- An event listener is an interface in the View class that contains a single callback method
- These methods are invoked by the Android framework to handle the event eg onClick, onLongClick, onCheckedChange, onTouch etc
- UI item has to register the class that is going to implement the user interaction event with the item in the UI.

Events and listener interfaces

- onClick() From View.OnClickListener.

  This is called when the user either touches the item (when in touch mode), or focuses upon the item with the navigation-keys or trackball and presses the suitable "enter" key or presses down on the trackball
- onCheckedChange() From CompoundButton.OnCheckedChangeListener

  Interface definition for a callback to be invoked when the checked state of a compound button changed. Compound button is button having two states. When button is pressed or clicked, the state changes automatically. The compound buttons are CheckBox, Radio Button, Switch and Toggle Button.
- onLongClick() From View.OnLongClickListener.

  This is called when the user either touches and holds the item (when in touch mode), or focuses upon the item with the navigation-keys or trackball and presses and holds the suitable "enter" key or presses and holds down on the trackball (for one second).
- onFocusChange() From View.OnFocusChangeListener.

  This is called when the user navigates onto or away from the item, using the navigation-keys or trackball.

Event & listener interfaces

- onKey() From View.OnKeyListener.

  This is called when the user is focused on the item and presses or releases a hardware key on the device.

- onTouch() From View.OnTouchListener

  This is called when the user performs an action qualified as a touch event, including a press, a release, or any movement gesture on the screen (within the bounds of the item).

- onCreateContextMenu() From View.OnCreateContextMenuListener

  This is called when a Context Menu is being built (as the result of a sustained "long click"). See the discussion on context menus in the Menus developer guide.

Event Listener - Registration

- In Activity class, UI object must register the class that is going to implementation the respective listener interface using method

  View.set.. Listener(reference to the class that implements the callback method)

(e.g., call setOnClickListener() and pass it reference of class that is implementing the OnClick method)

  - Callback method to be implemented by self only (this)
  - Callback method to be implemented by another class(className)
  - Create an **anonymous** implementation of OnClickListener

- OnClick method name may be defined in XML file. The method should be implemented in corresponding activity class. Eg

  android:onClick = “btnClicked”
  Method Signature in Class: public void btnClicked(View v) { }

Event Listener - Implementation

The example below shows **how to register an onClickListener for a Button with reference to same class.** It is handled by activity itself. This will avoid extra class load and object allocation.

```
 public class MainActivity  extends  Activity  implements   OnClickListener {
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
Button button1 = (Button) findViewById(R.id.button1);
button1.setOnClickListener(this);

}

 public void onClick(View v) {
Toast.makeText(this,"Button Clicked", Toast.Length_Long).show() ;

 }
```

Event Listener -Implementation

The example below shows **how to register an onClickListener for a Button with reference to another class that implements OnClick**

```
//Now inside your Activity class
    protected void onCreate(Bundle savedValues) {
        ...
        //  Capture our button from layout
        Button button = (Button)findViewById(R.id.button1);
        // Register the onClick listener with the implementation above
        button.setOnClickListener(mButtonListener);
        ... }

// Create an anonymous implementation of OnClickListener
    private OnClickListener mButtonListener = new OnClickListener()      {
        public void onClick(View v)          {
         // do something when the button is clicked
        }   };
```

Event Listener

Notice that the onClick() callback in the above example has no return value, but some other event listener methods must return a boolean. The reason depends on the event. For the few that do, here's why:

- onLongClick()- This returns a boolean to indicate whether you have consumed the event and it should not be carried further. That is, return *true* to indicate that you have handled the event and it should stop here; return *false* if you have not handled it and/or the event should continue to any other on-click listeners.

Event Listener

- onKey() - This returns a boolean to indicate whether you have consumed the event and it should not be carried further. That is, return *true* to indicate that you have handled the event and it should stop here; return *false* if you have not handled it and/or the event should continue to any other on-key listeners.

- onTouch()- This returns a boolean to indicate whether your listener consumes this event. The important thing is that this event can have multiple actions that follow each other. So, if you return *false* when the down action event is received, you indicate that you have not consumed the event and are also not interested in subsequent actions from this event. Thus, you will not be called for any other actions within the event, such as a finger gesture, or the eventual up action event.

Event Handling

- If you're building a custom component from View, then you'll be able to define several callback methods used as default event handlers.
- onKeyDown(int, KeyEvent) - Called when a new key event occurs.
- onKeyUp(int, KeyEvent) - Called when a key up event occurs.
- onTrackballEvent(MotionEvent) - Called when a trackball motion event occurs.
- onTouchEvent(MotionEvent) - Called when a touch screen motion event occurs.
- onFocusChanged(boolean, int, Rect) - Called when the view gains or loses focus.

Event Handling

There are some other methods that you should be aware of, which are not part of the View class, but can directly impact the way you're able to handle events. So, when managing more complex events inside a layout, consider these other methods:

- Activity.dispatchTouchEvent(MotionEvent)-
  This allows your Activity to intercept all touch events before they are dispatched to the window.

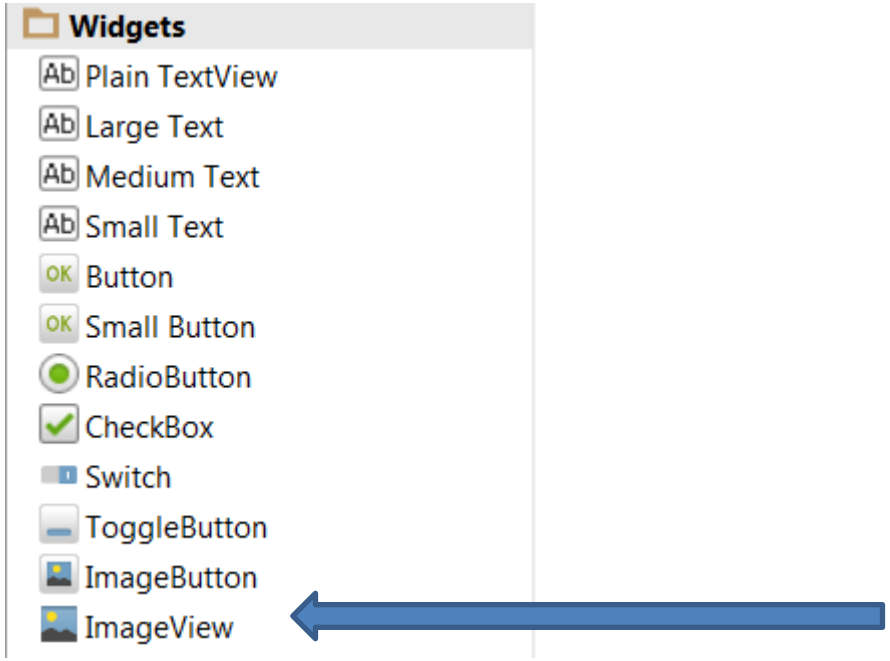
- ViewGroup.onInterceptTouchEvent(MotionEvent)-
  This allows a ViewGroup to watch events as they are dispatched to child Views

- ViewParent.requestDisallowInterceptTouchEvent(boolean)-
  Call this upon a parent View to indicate that it should not intercept touch events with onInterceptTouchEvent(MotionEvent).

Image

a) ImageView

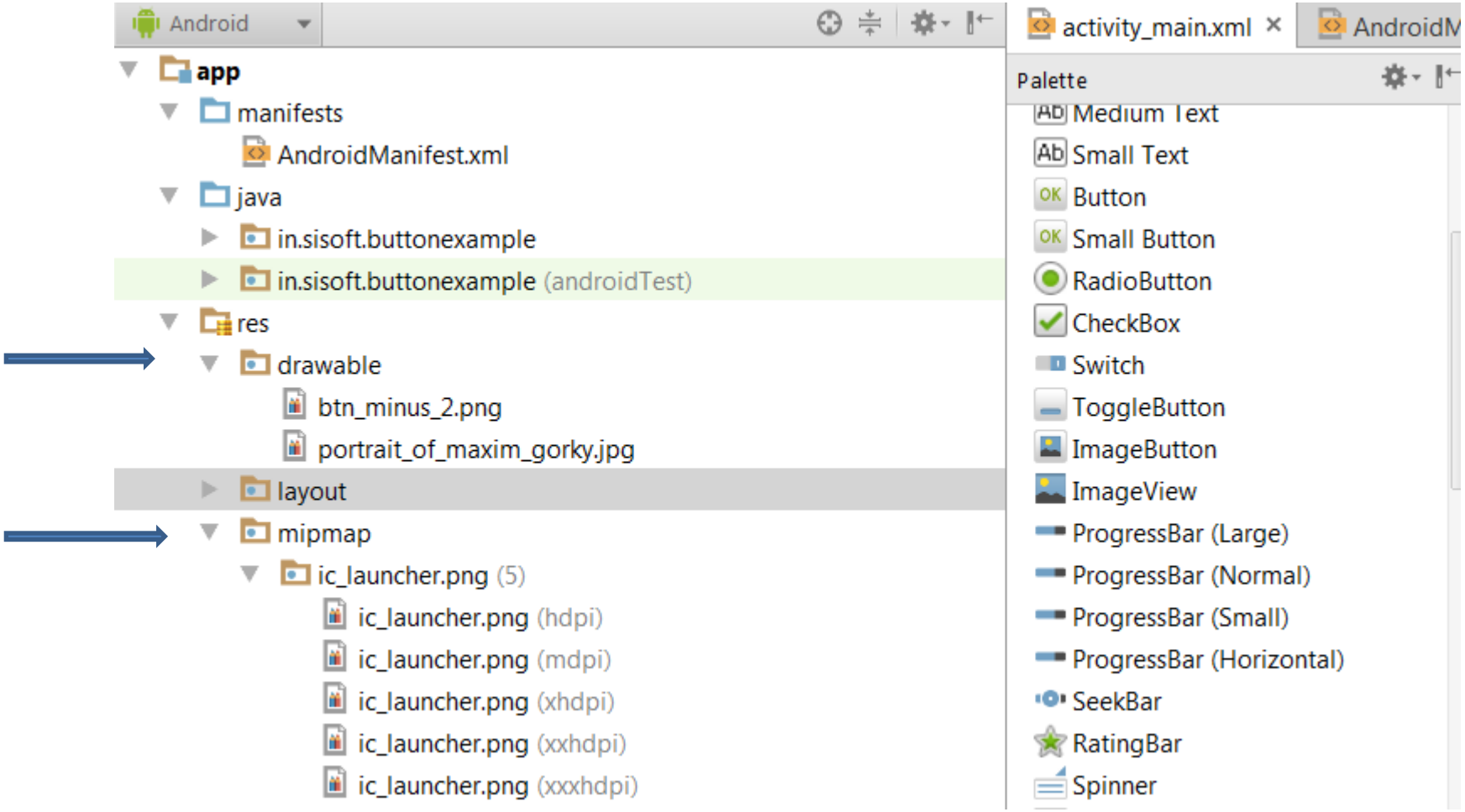
b) ImageButton



Image

- **ImageView and ImageButton are two Android widgets that allow embedding of images in your applications.**
- Both are *image-based widgets analogue to TextView and Button, respectively.*
- Each image widget takes an **android:src.** This exists for Imageviews and its subclasses
- **android:background attribute is used to specify what picture to use as background.**
- src is a foreground image and background is a background image
- Pictures usually reference a *drawable resource.*

Location of image files



All images are kept in drawable folder, icons should be kept in mipmap.

Image Button

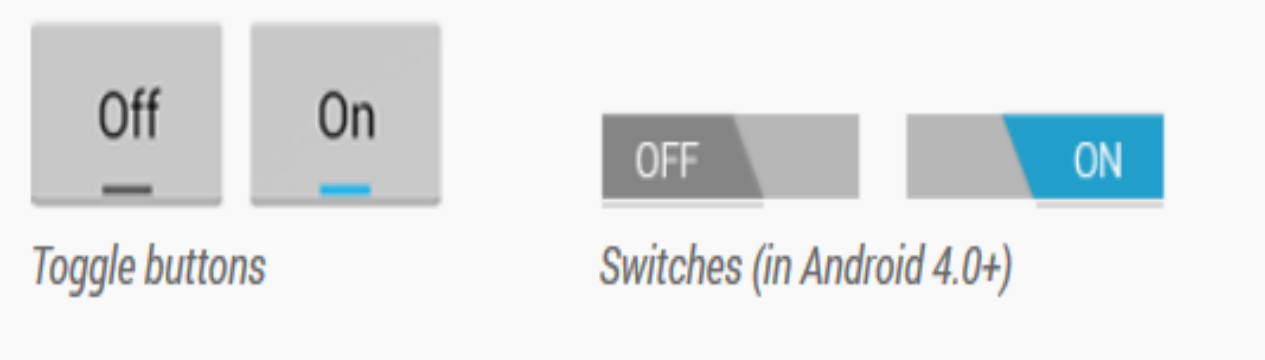
- **ImageButton, is a subclass of ImageView. It adds the standard *Button behavior for responding to click events***
- Displays a button with an image (instead of text) that can be pressed or clicked by the user.
- An ImageButton looks like a regular Button, with the standard button background that changes color during different button states.
- The image on the surface of the button is defined either by the android:src attribute in the XML element or by the setImageResource(int) method

Compound Buttons

- A button with two states, checked and unchecked. When the button is pressed or clicked, the state changes automatically.
- SubClasses: ToggleButton, Switch, CheckBox, Radio Button
- Callback method: OnCheckedChangeListener
- Useful Methods: isChecked(), toggle(), performClick(), setOnCheckedChangeListener

ToggleButton

- A toggle button allows the user to change a setting between two states.
- You can add a basic toggle button to your layout with the **ToggleButton** object. Android 4.0 (API level 14) introduces another kind of toggle button called a switch that provides a slider control, which you can add with a **Switch** object.



ToggleButton

```
<ToggleButton
    android:id="@+id/togglebutton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="Vibrate on"
    android:textOff="Vibrate off"
    android:onClick="onToggleClicked"/>
```

Within the `Activity` that hosts this layout, the following method handles the click event:

```
public void onToggleClicked(View view) {
    // Is the toggle on?
    boolean on = ((ToggleButton) view).isChecked();

    if (on) {
        // Enable vibrate
    } else {
        // Disable vibrate
    }
}
```

ToggleButton

```
ToggleButton toggle = (ToggleButton) findViewById(R.id.togglebutton);
toggle.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
            // The toggle is enabled
        } else {
            // The toggle is disabled
        }
    }
});
```

CheckBox

- A checkbox is a specific type of two-states button that can be either ***checked or unchecked***
- In Check Box you can select either one or more check boxes. To find the CheckBox status isChecked() is method is used
- To change the CheckBox state, setChecked(boolean) or toggle() methods are used
- Listeners:
  - View.OnClickListener
  - CompoundButton.OnCheckedChangedListener

Using CheckBox In Java Coding

checkbox inside your activity would be

```
public class MyActivity extends Activity {

    protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);

        setContentView(R.layout.content_layout_id);

        final CheckBox checkBox = (CheckBox) findViewById(R.id.checkbox_id);
        if (checkBox.isChecked()) {
            checkBox.setChecked(false);
        }
    }
}
```

RadioGroup/RadioButton

- Radio buttons are used to select one item from a small group of items
- Android class **android.widget.RadioButton** is used to render radio button and these radio buttons grouped by **android.widget.RadioGroup**
- A radio button is a two-states button that can be either *checked or unchecked*
- Radio buttons are normally used together in a **RadioGroup.**
- When several radio buttons live inside a radio group, checking one radio button *uncheck all the others.*
- RadioButton inherits from … TextView. Hence, all the standard TextView properties for *font face, style, color, etc. are available for controlling the look of radio buttons.*
- •Similarly, you can call ***isChecked() on a RadioButton to see if it is selected, toggle() to select it, and so on, like you can with a CheckBox.***

RadioGroup/RadioButton: Responding to Click Events

- RadioButton object receives an on-click event where as Radio Group object receives On Checked Change event
- It will have to implement RadioGroup.OnCheckedChangeListener
- **onCheckedChanged (RadioGroup group, int checkedId)**
  - ***group*:** the group in which the checked radio button has changed
  - **checkedId:** the unique identifier of the newly checked radio button
- getCheckedRadioButtonId(): Returns the identifier of the selected radio button in this group

RadioGroup/RadioButton

Example

We extend the previous example by adding a *RadioGroup* and three *RadioButtons*. Only new XML and Java code is shown:

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

<RadioGroup
android:id="@+id/radGroupCoffeeType"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:orientation="vertical"
>
<TextView
android:id="@+id/labelCoffeeType"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:background="#ff993300"
android:text="What type of coffee?"
android:textStyle="bold"
>
</TextView>
<RadioButton
android:id="@+id/radDecaf"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Decaf"
>
</RadioButton>
<RadioButton
android:id="@+id/radExpresso"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Expresso"
>
</RadioButton>
<RadioButton
android:id="@+id/radColombian"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Colombian"
>
</RadioButton>
</RadioGroup>

...

</LinearLayout>
```

RadioGroup/RadioButton

Java code :-

```
public class RadioButtonDemoUI extends Activity {

@Override
Public  void  onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);
   //binding XMlcontrols to Java code
    CheckBox chkCream= (CheckBox)findViewById(R.id.chkCream);
   CheckBox chkSugar= (CheckBox)findViewById(R.id.chkSugar);
   Button btnPay= (Button) findViewById(R.id.btnPay);

   RadioGroup radCoffeeType= (RadioGroup)findViewById(R.id.radGroupCoffeeType);
   RadioButton radDecaf= (RadioButton)findViewById(R.id.radDecaf);
   RadioButton radExpresso= (RadioButton)findViewById(R.id.radExpresso);
   RadioButton radColombian= (RadioButton)findViewById(R.id.radColombian);
```

```
//LISTENER: wiring button-events-&-code
btnPay.setOnClickListener(newOnClickListener() {
@Override
Public  void  onClick(View v) {
        String msg= "Coffee ";
       if(chkCream.isChecked())
              msg+= " & cream ";
       if(chkSugar.isChecked())
             msg+= " & Sugar";
      // get radio buttons ID number
       int radioId= radCoffeeType.getCheckedRadioButtonId();
// compare selected'sId with individual RadioButtonsID
      if(radColombian.getId()== radioId)
            msg= “Colombian " + msg;
// similarly you may use .isChecked() on each RadioButton
     if(radExpresso.isChecked())
             msg= "Expresso" + msg;
        Toast.makeText(getApplicationContext(), msg, Toast.LENGTH_SHORT).show();
// go now and compute cost...
    }// onClick
});
}// onCreate
}// class
```

RadioGroup/RadioButton

Example

This UI uses *RadioButtons* and *CheckBoxes* to define choices