# Buttons and Similar Clickable Widgets

Originals of Slides and Source Code for Examples:
http://www.coreservlets.com/android-tutorial/

more SERVLETS and JAVASERVER PAGES
MARTY HALL
core SERVLETS and JAVASERVER PAGES
2ND EDITION
Volume 1: Core Technologies
MARTY HALL · LARRY BROWN

## Topics in This Section

- Buttons
- ImageButtons each with single image
- ImageButtons each with 3 (normal/focused/pressed) images
- RadioButtons with OnClickListener on each
- RadioButtons with OnCheckedChangeListener on RadioGroup
- CheckBoxes
- ToggleButtons

## General Approach for Widget Examples

## Widget Lectures Combined in Single Project

- **Main screen**
  - Lets user choose screens on various Widget topics
- **Other screens**
  - Correspond to separate lectures.
    - One screen for lecture on Buttons, another for lecture on Spinners, another for number input, etc.
- **Separate layout files**
  - main.xml, buttons.xml, spinners.xml, etc. See next slide.
- **Separate Java classes**
  - WidgetActivity.java, ButtonActivity.java, SpinnerActivity.java, etc.
- **Shared strings file**
  - strings.xml has separate sections for each lecture, but same file

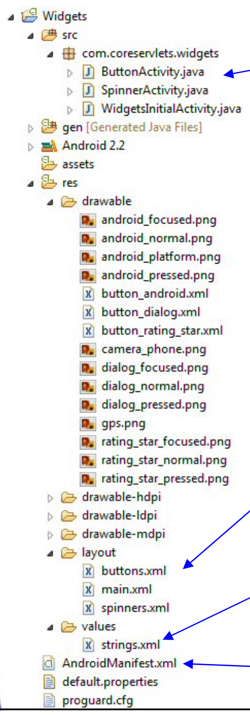
## Layout Files for Widget Lectures

- **Separate layout files for each Activity**
  - res/layout/main.xml
    - Gives layout for main screen. Loaded with setContentView(R.layout.main);
  - res/layout/buttons.xml
    - Gives layout for screen on Button and related Widgets. Loaded with setContentView(R.layout.buttons);
  - res/layout/spinners.xml
    - Gives layout for screen on Spinners (i.e., combo boxes). Loaded with setContentView(R.layout.spinners);
- **Two common layout attributes**
  - android:layout_width, android:layout_height
    - match_parent (fill up space in enclosing View)
    - wrap_content (use natural size)

## Switching Activities: Summary

- **Switches Activities with Intents**
  - Main screen has buttons to navigate to other Activities
  - Return to original screen with phone’s “back” button
- **Syntax required to start new Activity**
  - Java
    - Intent newActivity = new Intent(this, NewActivity.class);
    - startActivity(newActivity);
  - XML
    - Requires entry in AndroidManifest.xml (which is part of downloadable Eclipse project for Widgets)
  - More details
    - Code and some information given in Spinner lecture
    - Even more information given in later lecture on Intents

## Overall Widget Project Layout



```
Widgets
  src
    com.coreservlets.widgets
      ButtonActivity.java
      SpinnerActivity.java
      WidgetsInitialActivity.java
  gen [Generated Java Files]
  Android 2.2
  assets
  res
    drawable
      android_focused.png
      android_normal.png
      android_platform.png
      android_pressed.png
      button_android.xml
      button_dialog.xml
      button_rating_star.xml
      camera_phone.png
      dialog_focused.png
      dialog_normal.png
      dialog_pressed.png
      gps.png
      rating_star_focused.png
      rating_star_normal.png
      rating_star_pressed.png
    drawable-hdpi
    drawable-ldpi
    drawable-mdpi
    layout
      buttons.xml
      main.xml
      spinners.xml
    values
      strings.xml
  AndroidManifest.xml
  default.properties
  proguard.cfg
```

Java code

Images and XML files that refer to sets of images. The layout files will refer to these images via @drawable/*base_file_name* (e.g., @drawable/gps). The first ImageButton example will use an image file, and the second ImageButton example will use an XML file containing references to image files.

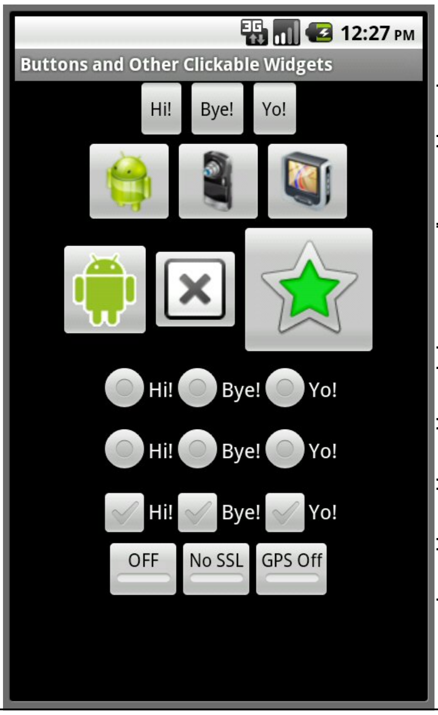
Layout files. The Java code will refer to the overall layouts via R.layout.*base_file_name* (R.layout.main, R.layout.buttons, etc.). The Java code will refer to specific GUI elements with findViewById(R.id.*element_id*).

Strings. The Java code will refer to these via getString(R.string.*string_name*). The layout files will refer to these with @string/*string_name*. You can also define arrays of strings here, or put the arrays in a separate file typically called arrays.xml. Arrays are used in the next lecture on Spinners.

In order for one Activity to start another Activity in the same project, you need some entries in here. See Spinner lecture.

# Approach for Button-Related Examples

## Summary of Layout



Horizontal LinearLayout (with 3 Buttons)

Horizontal LinearLayout (with 3 ImageButtons)

Horizontal LinearLayout (with 3 ImageButtons)

Horizontal RadioGroup
(with 3 RadioButtons)

Horizontal RadioGroup
(with 3 RadioButtons)

Horizontal LinearLayout (with 3 CheckBoxes)

Horizontal LinearLayout (with 3 ToggleButtons)

Vertical LinearLayout

An upcoming tutorial section gives details on using layouts. However, you can do a pretty lot now by knowing just two simple things:

1) You can make some pretty complex layouts by nesting horizontal and vertical layouts inside each other.
2) You can experiment interactively with the visual layout editor in Eclipse. Edit layout file and click on Graphical Layout.

## XML: Layout File (res/layout/buttons.xml)

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

    <!--
         One entry for each row in previous slide.
         These entries are shown in upcoming slides.
     -->

</LinearLayout>
```

## XML: Strings File (res/values/strings.xml)

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

    <!--  Initial screen -->
    <string name="app_name">...</string>
    <string name="show_buttons_button_label">...</string>
    <string name="show_spinners_button_label">...</string>

    <!-- Buttons example -->
    <!-- Shown in this lecture -->

    <!-- Spinners example -->
    <!-- Shown in next lecture -->

</resources>
```

## Java (ButtonActivity.java)

```
public class ButtonActivity extends Activity {
    ...

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.buttons);
        ...
    }

    private void showToast(String text) {
        Toast.makeText(this, text, Toast.LENGTH_LONG).show();
    }

    ...
}
```

## Button

## Button

- **Idea**
  - A push button displaying text
- **Main Listener type**
  - View.OnClickListener
    - If you specify the handler method in the XML file, you never explicitly refer to this Listener class.
- **Key XML attributes**
  - android:text
    - The label of the button. Can also be manipulated in Java with setText and getText
  - android:onClick
    - The event handler method. As shown in event-handling lecture, you can also use android:id and then have Java code programmatically assign event handler.

## XML: Layout File Entry (Part of res/layout/buttons.xml)

```
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_horizontal">
    <Button
        android:text="@string/hi_label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="showButtonText"/>
    <Button
        android:text="@string/bye_label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="showButtonText"/>
    <Button
        android:text="@string/yo_label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="showButtonText"/>
</LinearLayout>
```

## XML: Strings File Entries (Part of res/values/strings.xml)

```
<string name="hi_label">Hi!</string>
<string name="bye_label">Bye!</string>
<string name="yo_label">Yo!</string>
<string name="button_message_template">
   You clicked the \'%s\' Widget.
</string>
```

These are the labels referred to in previous slide

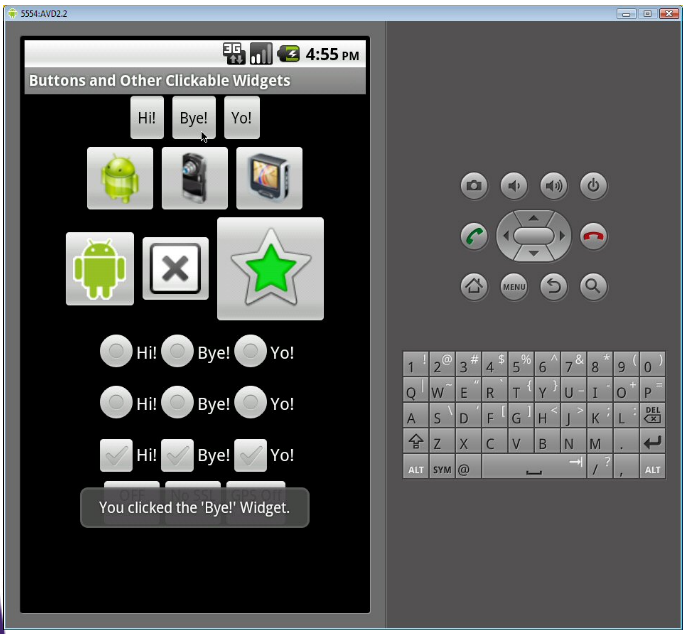
The event handler method will use String.format and this template to produce a message that will be shown in a Toast (short-lived popup message) when a Button is clicked.

## Java (Relevant Parts)

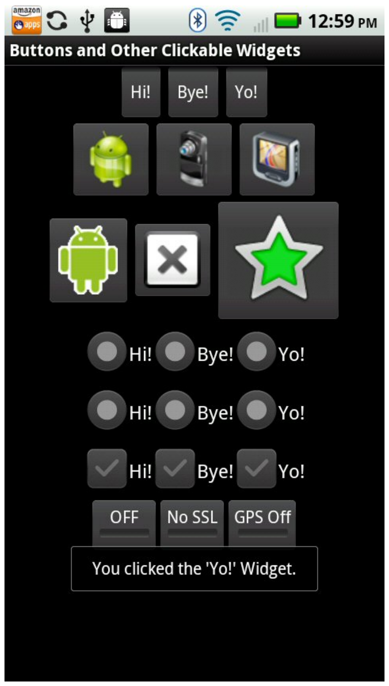
```
public class ButtonActivity extends Activity {
    private String mButtonMessageTemplate;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.buttons);
        mButtonMessageTemplate =
                getString(R.string.button_message_template);
    }

    public void showButtonText(View clickedButton) {
        Button button = (Button)clickedButton;
        CharSequence text = button.getText();
        String message =
                String.format(mButtonMessageTemplate, text);
        showToast(message);
    }
}
```

This is the method specified for each Button via the android:onClick attribute in the layout file.

## Results


5554:AVD2.2
4:55 PM
Buttons and Other Clickable Widgets
Hi!
Bye!
Yo!
Hi! Bye! Yo!
Hi! Bye! Yo!
Hi! Bye! Yo!
You clicked the 'Bye!' Widget.

Emulator



Phone

## ImageButton (Each with Single Image)

## ImageButton, Variation 1

- **Idea**
  - A push button displaying an image
- **Main Listener type**
  - View.OnClickListener
- **Key XML attributes**
  - android:src
    - The image for the button. Refers to the base name (minus the extension) of an image file in the res/drawable folder
      - Supported formats are png, jpeg, gif, and bmp. You can also refer to a drawable XML file as in next example.
      - The localization lecture will talk about drawable-xdpi folders
    - Can also be set in Java with setImageDrawable
  - android:onClick
    - The event handler method

If you just want to display an image, but not take action when it is clicked, see the ImageView class.

## XML: Layout File Entry (Part of res/layout/buttons.xml)

```
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_horizontal">
    <ImageButton
        android:src="@drawable/android_platform"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="showImageButton1Info"/>
    <ImageButton
        android:src="@drawable/camera_phone"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="showImageButton2Info"/>
    <ImageButton
        android:src="@drawable/gps"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="showImageButton3Info"/>
</LinearLayout>
```

Refers to res/drawable/android_platform.png

Refers to res/drawable/camera_phone.png

Refers to res/drawable/gps.png

## XML: Strings File Entries (Part of res/values/strings.xml)

```
<string name="image_button_message_template">
    You clicked the ImageButton that displays %s.
 </string>
 <string name="image_button_1_image">
     the android_platform.png image
 </string>
 <string name="image_button_2_image">
      the camera_phone.png image
 </string>
 <string name="image_button_3_image">
     the gps.png image
 </string>
```

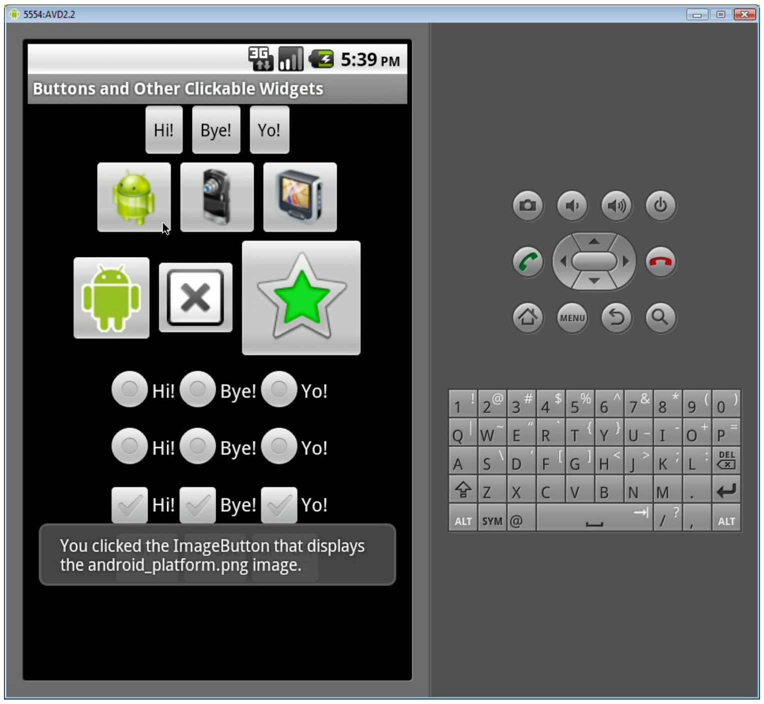
The event handler method will use String.format, this template, and the descriptions below to produce a message that will be shown in a Toast when an ImageButton is clicked.

## Java (Relevant Parts)

```
public class ButtonActivity extends Activity {
    private String mImageButtonMessageTemplate;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
        mImageButtonMessageTemplate =
                getString(R.string.image_button_message_template);
    }

    public void showImageButton1Info(View clickedImageButton) {
        showImageButtonInfo(R.string.image_button_1_image);
    }
    ...
    private void showImageButtonInfo(int imageId) {
        String image = getString(imageId);
        String message =
                String.format(mImageButtonMessageTemplate, image);
        showToast(message);
    }
}
```

This is the method specified for the first ImageButton via the android:onClick attribute in the layout file. Methods for the other ImageButtons are similar.

## Results (Emulator)


5554:AVD2.2

5:39 PM

Buttons and Other Clickable Widgets

You clicked the ImageButton that displays the android_platform.png image.

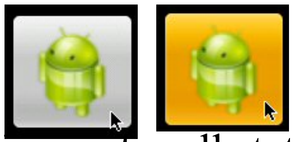
## ImageButton (Each with 3 Images)

## ImageButton, Variation 2

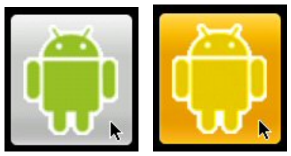
- **Idea**
  - A push button displaying one of three images, depending upon the situation
- **Main Listener type**
  - View.OnClickListener
- **Key XML attributes**
  - android:src
    - The image descriptor file for the button. Refers to the base name (minus the .xml extension) of an XML file in the res/drawable folder
      - The file, in turn, refers to three regular images in drawable folder
    - Can also be set in Java with setImageDrawable
  - android:onClick
    - The event handler method

## Individual Image Files vs. XML Files

- **Individual image files**



  - Android will use the same image for all states of the button (normal, focused, pressed)
  - Android will change the background color when focused or pressed. This affects the transparent pixels.



- **XML files**
  - Android will use a different image for each state of the button (normal, focused, pressed)
  - The different images can have different foreground colors, not just different backgrounds.

To get images for practicing, look in *android-sdk-install-dir*/platform-*x*/data/res/drawable-xdpi. Or, do a Google search for free icons. Also, see http://developer.android.com/guide/developing/tools/draw9patch.html for building your own images.

## Image Descriptor File (res/drawable/button_android.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<selector
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
          android:drawable="@drawable/android_pressed" />
    <item android:state_focused="true"
          android:drawable="@drawable/android_focused" />
    <item android:drawable="@drawable/android_normal" />
</selector>
```

These are the actual image files for each of the three possible states of the ImageButton.

The order of the three files matters. For more detail, see http://developer.android.com/reference/android/widget/ImageButton.html

## XML: Layout File Entry (Part of res/layout/buttons.xml)

```
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center">
    <ImageButton
        android:src="@drawable/button_android"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="showImageButton4Info"/>
    <ImageButton
        android:src="@drawable/button_dialog"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="showImageButton5Info"/>
    <ImageButton
        android:src="@drawable/button_rating_star"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="showImageButton6Info"/>
</LinearLayout>
```

Refers to res/drawable/button_android.xml. This, in turn, refers to three regular image files. Code on previous slide.

Refers to res/drawable/button_dialog.xml. This, in turn, refers to three regular image files.

Refers to res/drawable/button_rating_star.xml. This, in turn, refers to three regular image files.

## XML: Strings File Entries (Part of res/values/strings.xml)

```
<string name="image_button_message_template">
   You clicked the ImageButton that displays %s.
</string>


<string name="image_button_4_image">
    the Drawable defined in button_android.xml
</string>
<string name="image_button_5_image">
    the Drawable defined in button_dialog.xml
</string>
<string name="image_button_6_image">
    the Drawable defined in button_rating_star.xml
</string>
```

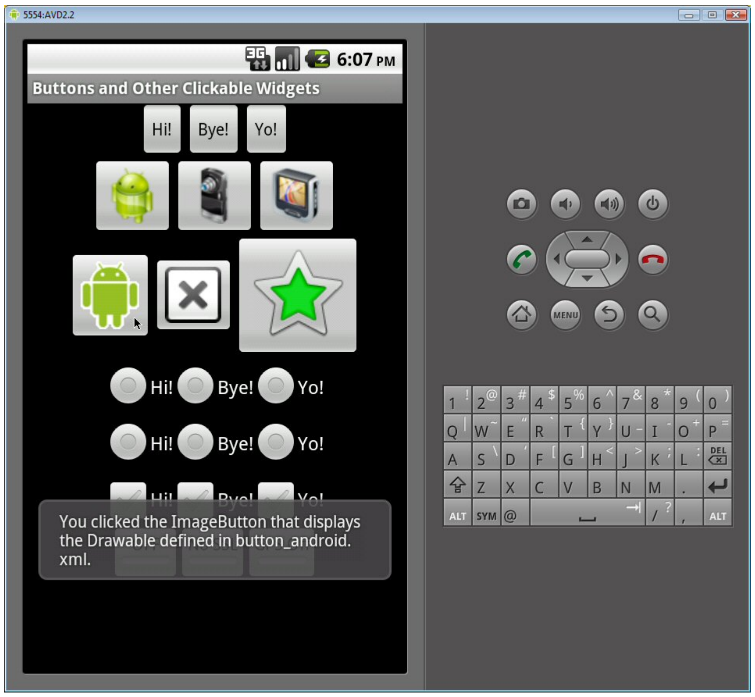
The event handler method will use String.format, this template, and the descriptions below to produce a message that will be shown in a Toast when an ImageButton is clicked. This is just a copy of entry already shown in previous ImageButton example.

## Java (Relevant Parts)

```
public class ButtonActivity extends Activity {
    private String mImageButtonMessageTemplate;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
        mImageButtonMessageTemplate =
                getString(R.string.image_button_message_template);
    }

    public void showImageButton4Info(View clickedImageButton) {
        showImageButtonInfo(R.string.image_button_4_image);
    }
    ...
    private void showImageButtonInfo(int imageId) {
        String image = getString(imageId);
        String message =
                String.format(mImageButtonMessageTemplate, image);
        showToast(message);
    }
}
```

This is the method specified for the first of these 3 ImageButtons via the android:onClick attribute in the layout file. Methods for the other ImageButtons are similar.

## Results (Emulator)



You clicked the ImageButton that displays the Drawable defined in button_android.xml.

# RadioButton
## (with Event Handler Attached to Each)

## RadioButton

- **Idea**
  – A button for choosing a single option among alternatives
- **Main Listener types**
  – View.OnClickListener
    - Assign to each RadioButton if you only care about which has been pressed most recently. But also see upcoming example for Listener attached to the RadioGroup.
      – No need to explicitly refer to Listener when using android:onClick
  – No Listener at all
    - Some apps take no action when RadioButton is clicked, but instead query the RadioGroup later to find selection
- **Key XML attributes**
  – android:text, android:onClick
    - Same as in previous examples.

## RadioGroup

- **Idea**
  – Similar to LinearLayout, but specifically for organizing RadioButtons.
  – Makes the RadioButtons exclusive (checking one causes previous selection to become unchecked)
- **Main Listener types**
  – RadioGroup.OnCheckedChangeListener
    - Assign to RadioGroup if you want to keep track of both current and previous selections
    - You can also call getCheckedRadioButtonId, if you don't need to respond immediately, but want to find selection later
- **Key XML attributes**
  – Mostly same as for LinearLayout
  – Use android:id if you want to programmatically set an OnCheckedChangeListener
    - No android:on*Blah* to set RadioGroup Listener in XML

## First Example: Event Handlers Attached to Each RadioButton

- **Idea**
  - Respond to clicks on each RadioButton by showing Toast saying which one was pressed.
- **Approach**
  - Put RadioButtons inside RadioGroup so that they are mutually exclusive.
  - To assign event handlers, use android:onClick for each RadioButton
  - No id for RadioGroup. No Listener for RadioGroup

## XML: Layout File Entry (Part of res/layout/buttons.xml)

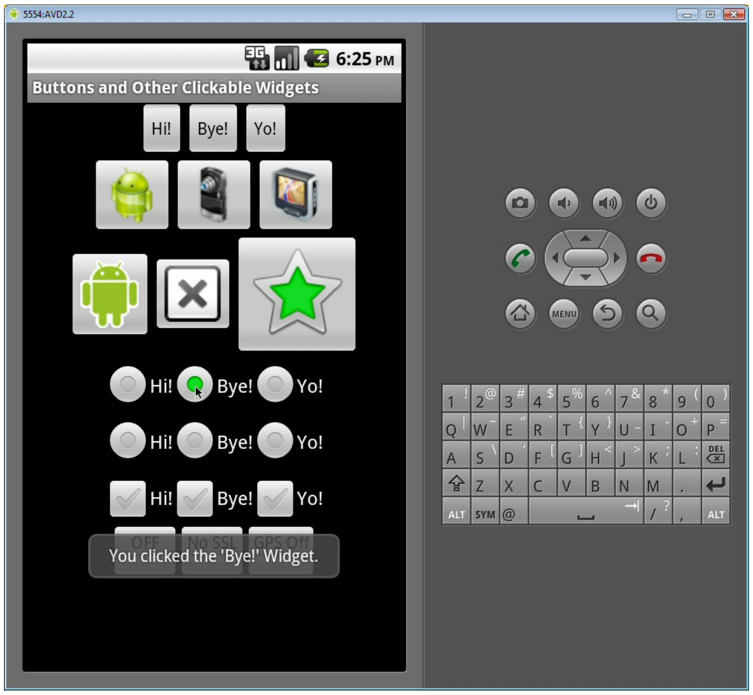
```
<RadioGroup
    android:gravity="center_horizontal"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:orientation="horizontal">
    <RadioButton
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="@string/hi_label"
        android:onClick="showButtonText"/>
    <RadioButton
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="@string/bye_label"
        android:onClick="showButtonText"/>
    <RadioButton
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="@string/yo_label"
        android:onClick="showButtonText"/>
</RadioGroup>
```

This first example uses click handlers attached to each RadioButton.

## Strings File and Java Code

- **Nothing new for this example**
  - Strings file
    - Already showed button labels and button_message_template
  - Java code
    - Already showed makeToast and showButtonText

## Results (Emulator)



You clicked the 'Bye!' Widget.

# RadioButton (with Event Handler Attached to RadioGroup)

## Second Example: Event Handler Attached to RadioGroup

- **Idea**
  - Respond to clicks by showing Toast saying which one was pressed *and* which one was previously selected.
- **Approach**
  - Put RadioButtons inside RadioGroup so that they are mutually exclusive.
    - Same as last example
  - In XML, give id to RadioGroup.
  - In Java, find RadioGroup and call setOnCheckedChangeListener

## XML: Layout File Entry (Part of res/layout/buttons.xml)

```
<RadioGroup
    android:id="@+id/radio_group"
    android:gravity="center_horizontal"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:orientation="horizontal">
    <RadioButton
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="@string/hi_label"/>
    <RadioButton
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="@string/bye_label"/>
    <RadioButton
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="@string/yo_label"/>
</RadioGroup>
```

The id is needed so that Java can get a reference and programmatically set the OnCheckedChangeListener.

RadioButtons do *not* have android:onClick entries

## XML: Strings File Entries (Part of res/values/strings.xml)

```
<string name="new_selection_message_template">
    You selected the \'%s\' RadioButton.
    There was no previous selection.
</string>
<string name="changed_selection_message_template"
        formatted="false">
    You selected the \'%s\' RadioButton.
    Previous selection was \'%s\'.
</string>
```

The event handler method will use String.format, one of these templates, the current selection, and the previous selection to produce a message that will be shown in a Toast when a RadioButton is clicked.

Use formatted="false" if a string has more than one %s placeholder.

## Java (Relevant Parts)

```
public class ButtonActivity extends Activity {
    ...

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.buttons);
        ...
        RadioGroup radioGroup =
                (RadioGroup)findViewById(R.id.radio_group);
        radioGroup.setOnCheckedChangeListener(new RadioGroupInfo());
    }
```

Continued on next page.
RadioGroupInfo is an inner class inside ButtonActivity.

## Java (Relevant Parts, Continued)

```
private class RadioGroupInfo implements OnCheckedChangeListener {
    private RadioButton mLastChecked;
    private String mNewSelectionMessageTemplate;
    private String mChangedSelectionMessageTemplate;

    public RadioGroupInfo() {
        mNewSelectionMessageTemplate =
            getString(R.string.new_selection_message_template);
        mChangedSelectionMessageTemplate =
            getString(R.string.changed_selection_message_template);
    }
```

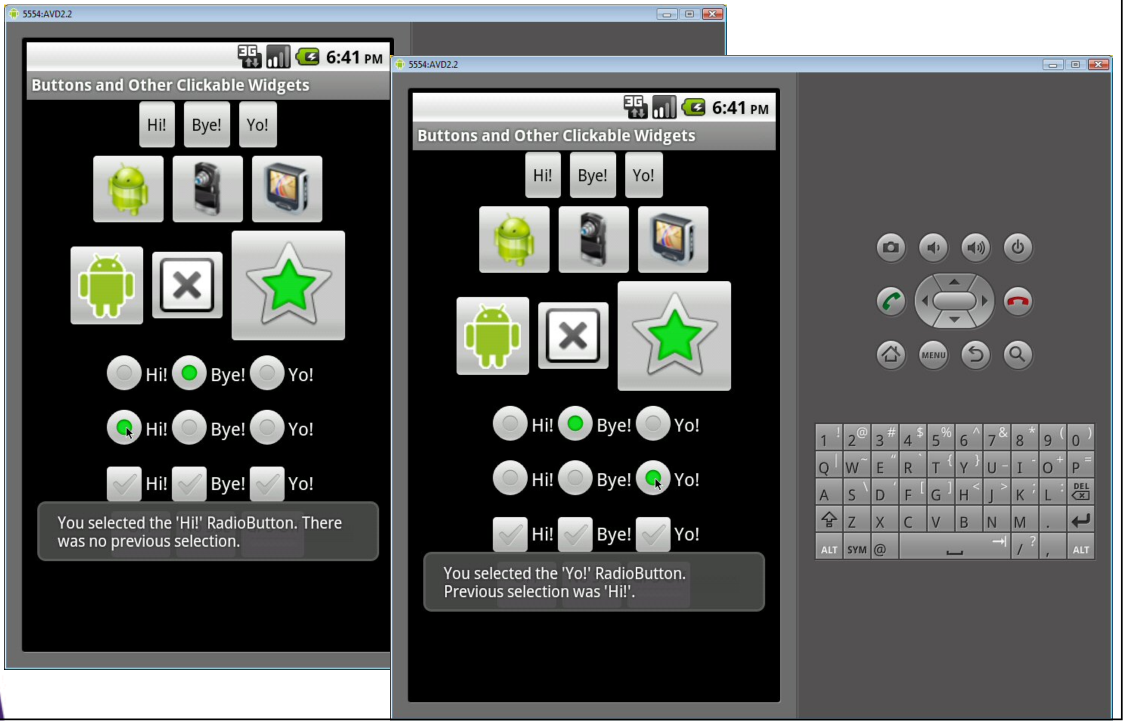
Top of the inner class

## Java (Relevant Parts, Continued)

```
        @Override
        public void onCheckedChanged(RadioGroup group, int checkedId) {
            RadioButton newChecked =
                    (RadioButton)findViewById(checkedId);
            String message;
            if (mLastChecked == null) {  // No previous selection
                message = String.format(mNewSelectionMessageTemplate,
                                        newChecked.getText());
            } else {
                message = String.format(mChangedSelectionMessageTemplate,
                                        newChecked.getText(),
                                        mLastChecked.getText());
            }
            mLastChecked = newChecked;
            showToast(message);
        }
    }
}
```

Bottom of the inner class. Keeps track of current and previous selections.

## Results (Emulator)



You selected the 'Hi!' RadioButton. There was no previous selection.

You selected the 'Yo!' RadioButton. Previous selection was 'Hi!'.

# CheckBox

## CheckBox

- **Idea**
  - A button with two states (checked and unchecked)
    - Has visual indicator to show whether it is checked
    - In Java, use isChecked() to determine state. Use setChecked to programmatically change the state.
  - Same text in both states (unlike ToggleButton)
- **Main Listener types**
  - View.OnClickListener
  - No Listener at all
    - Take no action when CheckBox is clicked, but instead query the CheckBox later to find if it is checked or not
- **Key XML attributes**
  - android:text, android:onClick
    - Same as in previous examples

## XML: Layout File Entry (Part of res/layout/buttons.xml)

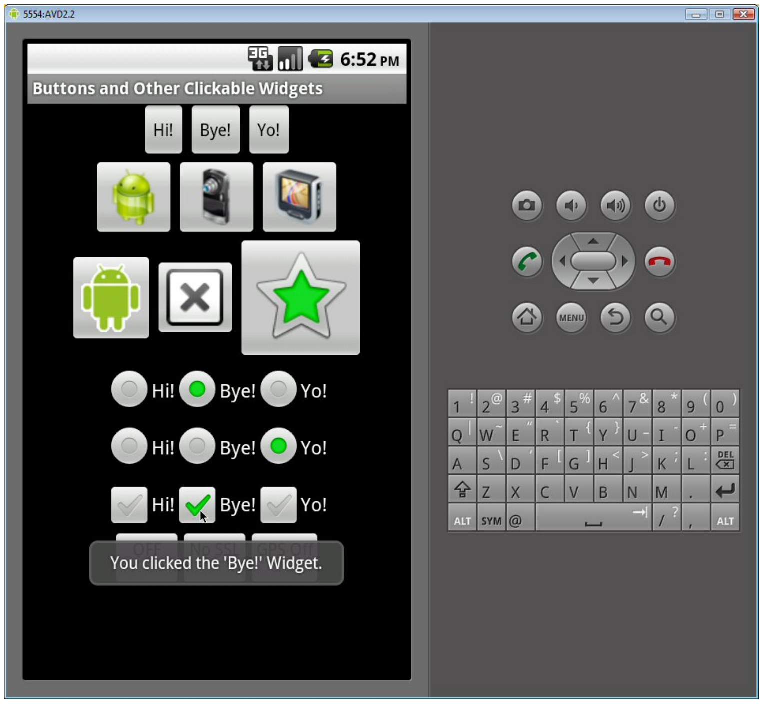
```
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_horizontal">
    <CheckBox
        android:text="@string/hi_label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="showButtonText"/>
    <CheckBox
        android:text="@string/bye_label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="showButtonText"/>
    <CheckBox
        android:text="@string/yo_label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="showButtonText"/>
</LinearLayout>
```

Note that the class name is CheckBox, not Checkbox (as in AWT).

## Strings File and Java Code

- **Nothing new for this example**
  - Strings file
    - Already showed button labels and button_message_template
  - Java code
    - Already showed makeToast and showButtonText

## Results (Emulator)


5554:AVD2.2
6:52 PM
Buttons and Other Clickable Widgets
Hi!
Bye!
Yo!
Hi!
Bye!
Yo!
Hi!
Bye!
Yo!
Hi!
Bye!
Yo!
You clicked the 'Bye!' Widget.

## ToggleButton

## ToggleButton

- **Idea**
  - A button with two states (checked and unchecked)
    - Has visual indicator to show whether it is checked
    - In Java, use isChecked() to determine state. Use setChecked to programmatically change the state.
  - Has different text for each state (unlike CheckBox)
- **Main Listener types**
  - View.OnClickListener
  - No Listener at all
    - Take no action when ToggleButton is clicked, but instead query the ToggleButton later to find if it is checked or not
- **Key XML attributes**
  - android:textOn, android:textOff
    - The text for the two states. If you omit this, then the text is automatically ON and OFF (in caps)
  - android:onClick
    - Same as in previous examples

## XML: Layout File Entry (Part of res/layout/buttons.xml)

```
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_horizontal">
    <ToggleButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="showToggleButtonInfo"/>
    <ToggleButton
        android:textOn="@string/ssl_toggle_on"
        android:textOff="@string/ssl_toggle_off"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="showToggleButtonInfo"/>
    <ToggleButton
        android:textOn="@string/gps_toggle_on"
        android:textOff="@string/gps_toggle_off"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="showToggleButtonInfo"/>
</LinearLayout>
```

No textOn or textOff attributes, so the defaults of ON and OFF will be used.

## XML: Strings File Entries (Part of res/values/strings.xml)

```
<string name="ssl_toggle_on">Use SSL</string>
<string name="ssl_toggle_off">No SSL</string>
<string name="gps_toggle_on">GPS On</string>
<string name="gps_toggle_off">GPS Off</string>
<string name="toggle_button_message_template"
        formatted="false">
    You turned the ToggleButton %s.
    Label is now \'%s\'.
</string>
```

The event handler method will use String.format, this template, the state of the ToggleButton (on or off), and the text to produce a message that will be shown in a Toast when a ToggleButton is clicked.

## Java (Relevant Parts)

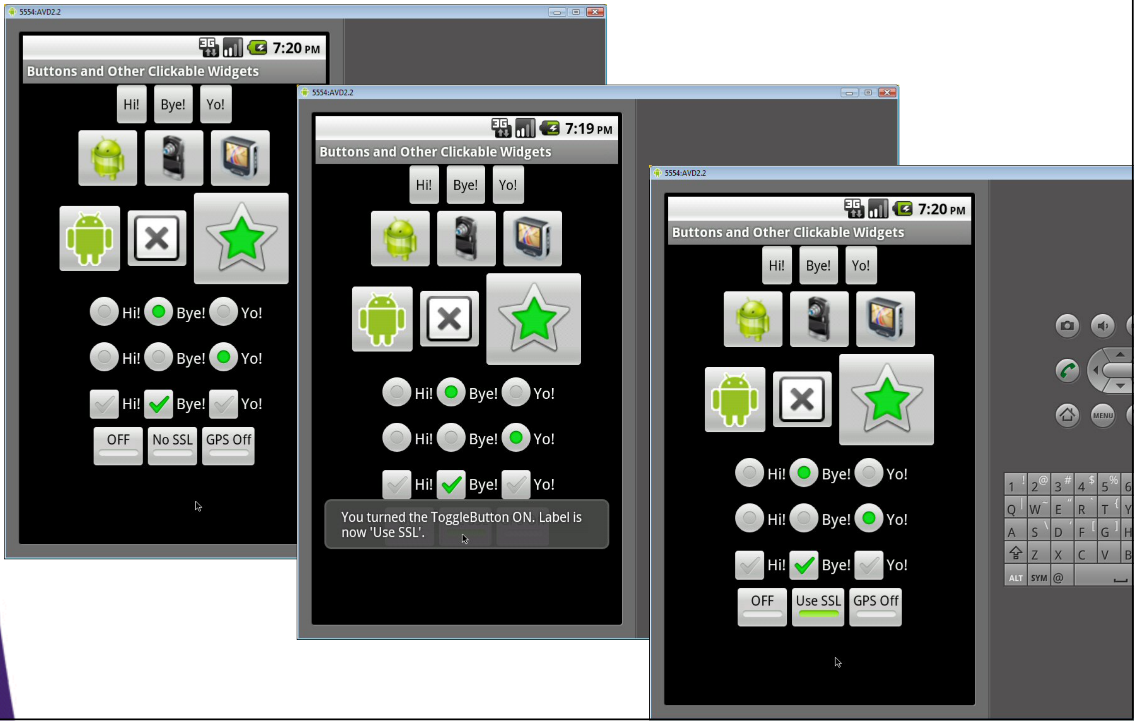
```
public class ButtonActivity extends Activity {
    private String mToggleButtonMessageTemplate;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
        mToggleButtonMessageTemplate =
                getString(R.string.toggle_button_message_template);
    }
```

## Java (Relevant Parts, Continued)

This is the method specified for the ToggleButtons via the android:onClick attribute in the layout file.

```
public void showToggleButtonInfo(View clickedToggleButton) {
    ToggleButton toggleButton =
            (ToggleButton)clickedToggleButton;
    String status;
    if (toggleButton.isChecked()) {
        status = "ON";
    } else {
        status = "OFF";
    }
    CharSequence label = toggleButton.getText();
    String message =
            String.format(mToggleButtonMessageTemplate,
                          status, label);
    showToast(message);
}
```

## Results (Emulator)



You turned the ToggleButton ON. Label is now 'Use SSL'.

## Wrap-Up

## Summary

- **Click handling is consistent among buttons**
  - Button, ImageButton, RadioButton, CheckBox, ToggleButton
    - Can specify event handler method with android:onClick
    - Or can set programmatically as in events lecture
- **ImageButton**
  - Can have single image or set of three.
    - Specify with android:src
    - Images and image XML files go in res/drawable folder
- **RadioGroup**
  - Surrounds RadioButtons. Can have its own Listener if you need to track previous selection.
- **ToggleButton**
  - Similar behavior to CheckBox. But has android:textOn and android:textOff instead of a fixed label.

## Questions?