Android Programming:
Widget Event Handling

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

For live Android training, please see courses at http://courses.coreservlets.com/.

Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this Android tutorial. Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
  - Android development, JSF 2, servlets/JSP, Ajax, jQuery, Java 6 programming, custom mix of topics
  - Ajax courses can concentrate on 1 library (jQuery, Prototype/Scriptaculous, Ext-JS, Dojo, etc.) or survey several
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Spring, Hibernate/JPA, EJB3, GWT, RESTful and SOAP-based Web Services

Contact hall@coreservlets.com for details

Topics in This Section

- **Using a separate Listener class**
- **Using a named inner class**
- **Using an anonymous inner class**
- **Using the main Activity**
  - And having it implement the Listener interface
- **Using the main Activity**
  - And specifying the method in the layout file (main.xml)
- **Copying and renaming Eclipse Android projects**

5

Using a Separate Listener Class

Idea

- **Goal**
  - Change color of a TextView when Button or RadioButton is pressed. Different colors depending on which pressed.
- **Approach**
  - Use an external class that implements View.OnClickListener
    - Import android.view.View.OnClickListener, then say "implements OnClickListener"
- **Advantages**
  - You can pass arguments to change behavior
  - Separate classes generally promote loose coupling
    - So, if event handler can be applied to different controls, it can be change independently from rest of app.
      - But, in most real situations, behavior is tightly coupled to app anyhow.
- **Disadvantages**
  - If you want to call code in main Activity, you need reference
  - Even then, that code in main Activity must be public

7
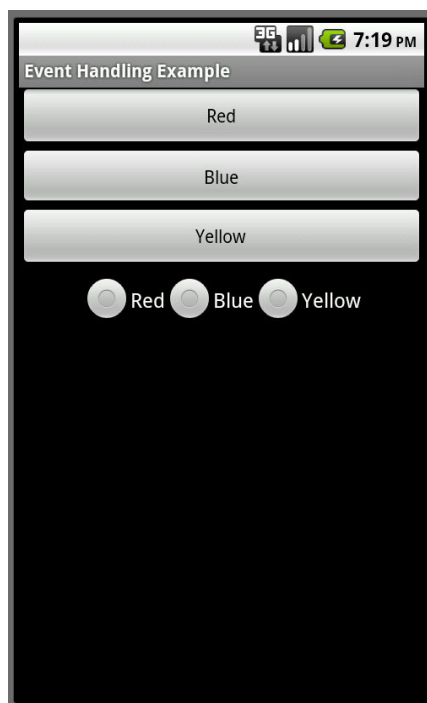
Summary of Layout

8

res/layout/main.xml

```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">
    <Button
        android:id="@+id/button1"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:text="@string/red_prompt"/>
    <Button
        android:id="@+id/button2"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:text="@string/blue_prompt"/>
    <Button
        android:id="@+id/button3"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:text="@string/yellow_prompt"/>
```

Overall layout is a vertical stack of graphical items.

This part defines the 3 buttons shown on the previous slide.

Each button is given an id so that it can be found in Java via findViewById, then assigned an event handler via setOnClickListener.

The text (Button label) is taken from strings.xml instead of entered directly here, because the same label will also be used for RadioButtons.

res/layout/main.xml (Continued)

```xml
    <RadioGroup
        android:gravity="center_horizontal"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:orientation="horizontal">
        <RadioButton
            android:id="@+id/radio_button1"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:text="@string/red_prompt"/>
        <RadioButton
            android:id="@+id/radio_button2"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:text="@string/blue_prompt"/>
        <RadioButton
            android:id="@+id/radio_button3"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:text="@string/yellow_prompt"/>
    </RadioGroup>
```

A horizontal RadioGroup gives the same layout as a horizontal LinearLayout, except that it contains only RadioButtons. A RadioGroup also means that only one of the RadioButtons inside can be selected at any given time.

res/layout/main.xml (Continued)

```
    <TextView
        android:id="@+id/color_region"
        android:layout_height="match_parent"
        android:layout_width="match_parent"/>
</LinearLayout>
```

This defines the blank region at the bottom that will change colors when the Buttons or RadioButtons are clicked. I used a TextView because I might later want to put some text inside.

res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Event Handling Example</string>
    <string name="red_prompt">Red</string>
    <string name="blue_prompt">Blue</string>
    <string name="yellow_prompt">Yellow</string>
</resources>
```

main.xml refers to these names with @string/red_prompt, @string/blue_prompt, and @string/yellow_prompt.

Each string is used as label for one Button and one RadioButton.

Main Activity Class

```
public class Events1Example extends Activity {
    private View mColorRegion;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mColorRegion = findViewById(R.id.color_region);
        Button b1 = (Button)findViewById(R.id.button1);
        Button b2 = (Button)findViewById(R.id.button2);
        Button b3 = (Button)findViewById(R.id.button3);
        RadioButton r1 =
                (RadioButton)findViewById(R.id.radio_button1);
        RadioButton r2 =
                (RadioButton)findViewById(R.id.radio_button2);
        RadioButton r3 =
                (RadioButton)findViewById(R.id.radio_button3);
```

This part just looks up the controls that were defined in main.xml, and assigns them to variables. Note the Android coding convention that non-public instance variables (data members) start "m".

13

Main Activity Class (Continued)

```
        b1.setOnClickListener(new ColorSetter(Color.RED, this));
        b2.setOnClickListener(new ColorSetter(Color.BLUE, this));
        b3.setOnClickListener(new ColorSetter(Color.YELLOW, this));
        r1.setOnClickListener(new ColorSetter(Color.RED, this));
        r2.setOnClickListener(new ColorSetter(Color.BLUE, this));
        r3.setOnClickListener(new ColorSetter(Color.YELLOW, this));
    }


    public void setRegionColor(int color) {
        mColorRegion.setBackgroundColor(color);
    }
}
```

Assigns a separate class as the event handler for each of the Buttons and RadioButtons.

Good news: you can pass arguments to the event handler (the colors) so that the same event handler class can have different behaviors for different controls.

Bad news: you have to pass a reference to the main Activity ("this" above) so that the event handler can call back to code in the Activity.

Since this method will be called by method in separate event handler class, it must be public.

14

Event Handler Class

```
public class ColorSetter implements OnClickListener {
    private int regionColor;
    private Events1Example mainActivity;

    public ColorSetter(int regionColor,
                       Events1Example mainActivity) {
        this.regionColor = regionColor;
        this.mainActivity = mainActivity;
    }

    @Override
    public void onClick(View v) {
        mainActivity.setRegionColor(regionColor);
    }
}
```

Event handler must store a reference to the main Activity so that it can call
back to it. Another option in this particular case would be to pass the TextView
to the event handler, but passing the main Activity is a more general solution.
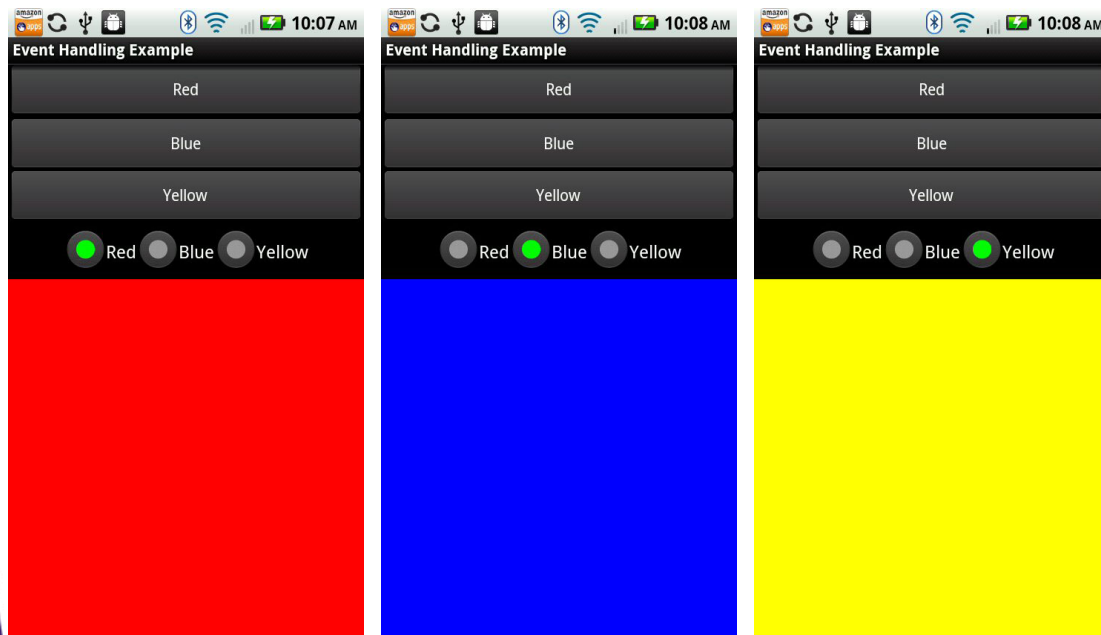
15

Results on Emulator
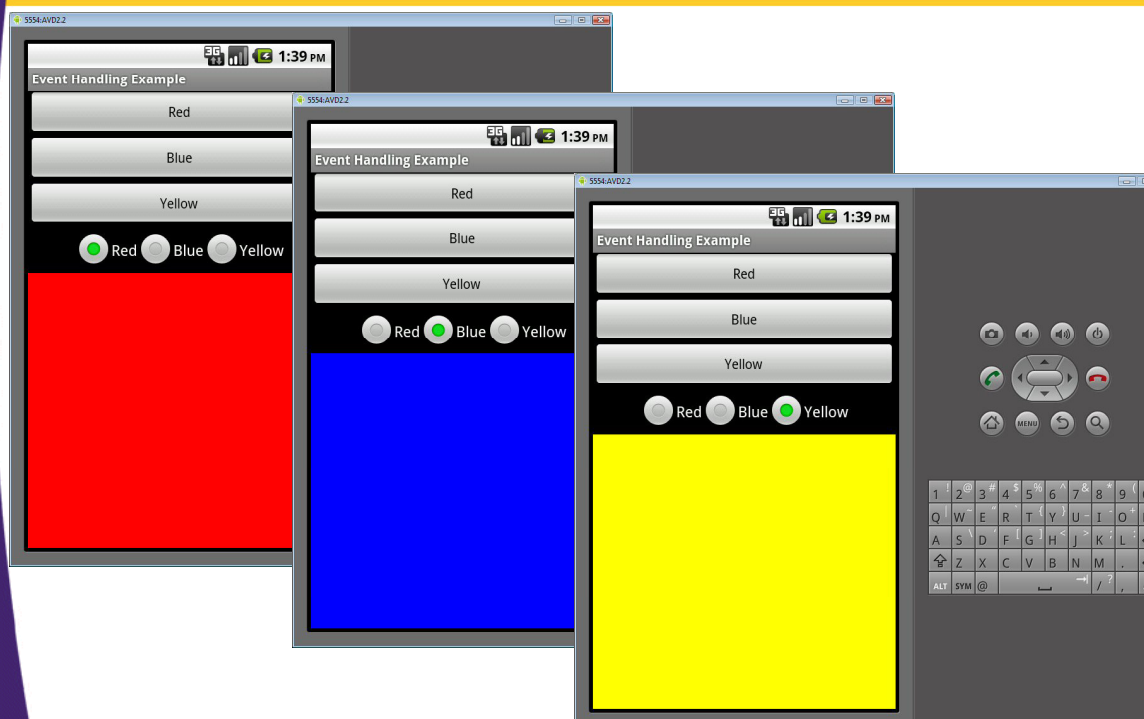
16

Results on Physical Phone

© 2011 Marty Hall

Using a Named Inner Class for Event Handling

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, JSF 2.0, Java 6, Ajax, jQuery, GWT, Spring, Hibernate, RESTful Web Services, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Idea

- **Goal**
  - Change color of a TextView when Button or RadioButton is pressed. Different colors depending on which pressed.
    - Same as previous example
- **Approach**
  - Use an inner class that implements View.OnClickListener
- **Advantages**
  - You can pass arguments to change behavior
  - Event handler methods can access private data of Activity. No reference is needed to call to Activity.
- **Disadvantages**
  - Since Listener class is in same file as Activity, it is more tightly coupled, and cannot be changed independently

XML Files:
Same as Previous Example

- **res/layout/main.xml**
  - Defines vertical LinearLayout that contains 3 Buttons, a horizontal RadioGroup (with 3 RadioButtons), and a TextView.
  - The Buttons, RadioButtons, and TextView have ids so that they can be referred to in the Java code
- **res/values/strings.xml**
  - Defines the app name and the labels of the Buttons and RadioButtons

Main Activity Class

```
public class Events2Example extends Activity {
    private View mColorRegion;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mColorRegion = findViewById(R.id.color_region);
        Button b1 = (Button)findViewById(R.id.button1);
        Button b2 = (Button)findViewById(R.id.button2);
        Button b3 = (Button)findViewById(R.id.button3);
        RadioButton r1 =
                (RadioButton)findViewById(R.id.radio_button1);
        RadioButton r2 =
                (RadioButton)findViewById(R.id.radio_button2);
        RadioButton r3 =
                (RadioButton)findViewById(R.id.radio_button3);
```

Except for the class name, this top part of the Activity is *exactly* the same as the previous example.

Main Activity Class (Continued)

```
        b1.setOnClickListener(new ColorSetter(Color.RED));
        b2.setOnClickListener(new ColorSetter(Color.BLUE));
        b3.setOnClickListener(new ColorSetter(Color.YELLOW));
        r1.setOnClickListener(new ColorSetter(Color.RED));
        r2.setOnClickListener(new ColorSetter(Color.BLUE));
        r3.setOnClickListener(new ColorSetter(Color.YELLOW));
    }

    private void setRegionColor(int color) {
        mColorRegion.setBackgroundColor(color);
    }
```

Assigns an inner class as the event handler for each of the Buttons and RadioButtons.

As with the previous example, you can pass arguments to the event handler (the colors) so that the same event handler class can have different behaviors for different controls.

However, since the event handler is in the same class, you do not have to supply a reference to the main Activity class.

Since this method will only be called by method in inner event handler class, it is allowed to be private.

Note no closing brace. This class is not finished yet (continued on next slide)

Event Handler Class
# (Part of Main Activity Class)

```java
private class ColorSetter implements OnClickListener {
    private int regionColor;

    public ColorSetter(int regionColor) {
        this.regionColor = regionColor;
    }

    @Override
    public void onClick(View v) {
        setRegionColor(regionColor);
    }
}
}
```

Event handler can directly call methods in the main Activity, even if the method is private.
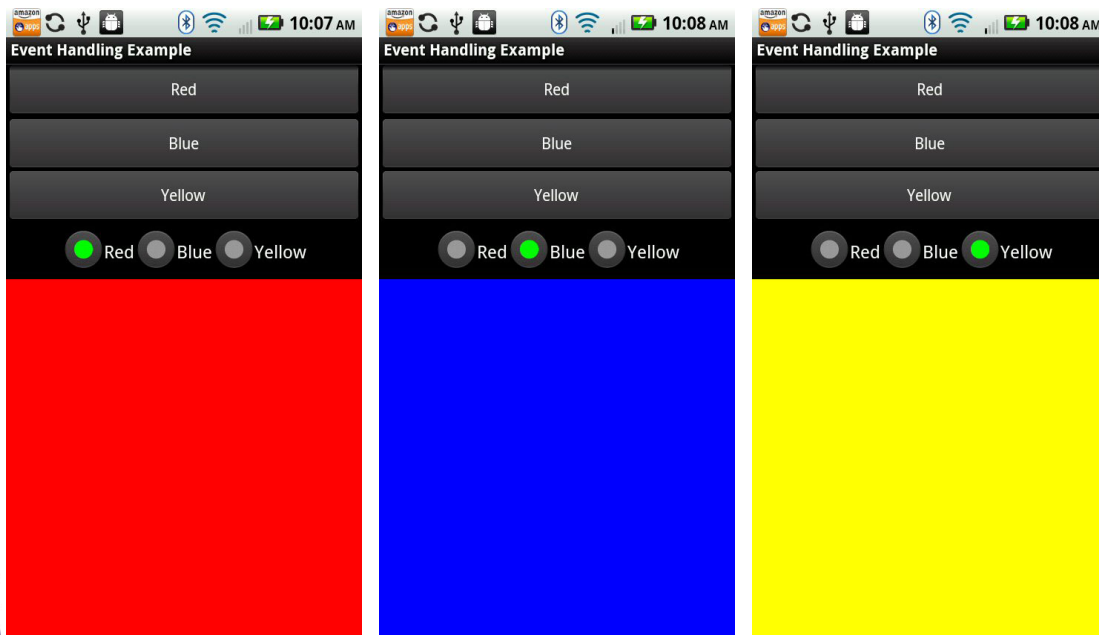
Closes off the main Activity class.

Results on Emulator
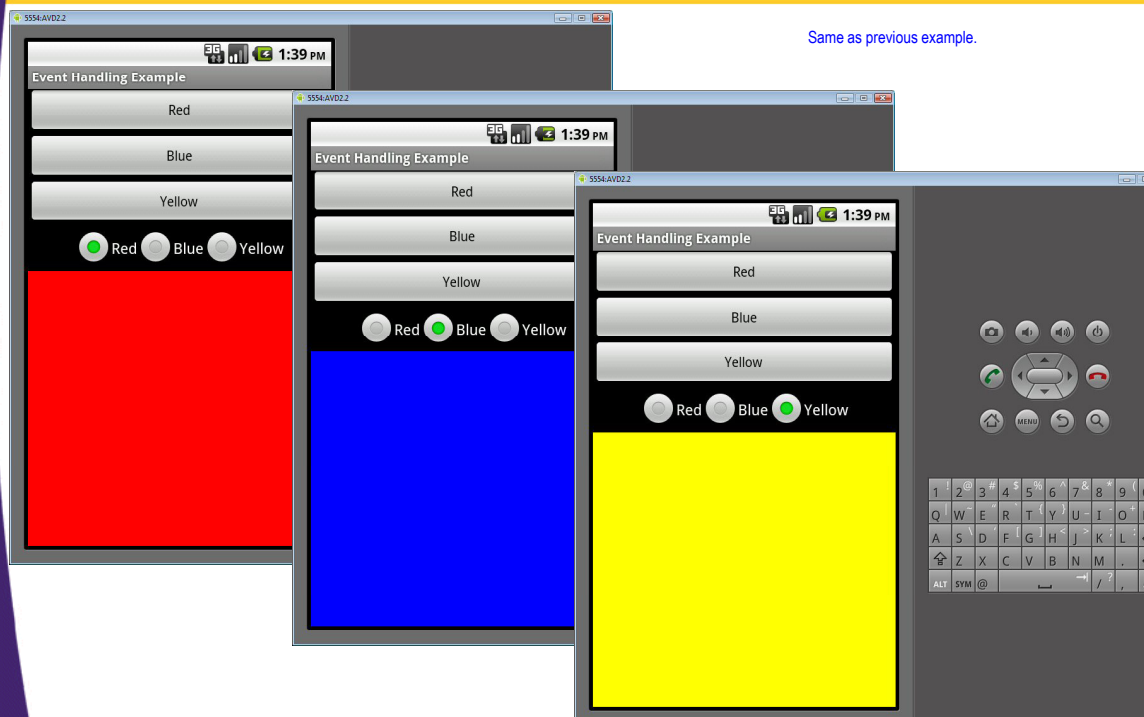
Same as previous example.

Results on Physical Phone

25

© 2011 Marty Hall

Using an Anonymous Inner Class for Event Handling

Idea

- **Goal**
  - Randomly change color of TextView when Button is pressed.
- **Approach**
  - Use an anonymous inner class that implements the Listener
- **Advantages**
  - Assuming that each class is applied to a single control only, same advantages as named inner classes, but shorter.
    - This approach is widely used in Swing, SWT, AWT, and GWT.
- **Disadvantages**
  - If you applied the handler to more than one control, you would have to cut and paste the code for the handler.
    - This approach should be applied for a single control only
  - If the code for the handler is long, it makes the code harder to read by putting it inline.
    - This approach is usually used only when handler code is short

res/layout/main.xml

```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">
    <Button
        android:id="@+id/color_button"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:text="@string/button_prompt"/>
    <TextView
        android:id="@+id/color_region"
        android:layout_height="match_parent"
        android:layout_width="match_parent"/>
</LinearLayout>
```

res/values/strings.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Event Handling Example</string>
    <string name="button_prompt">Random Color</string>
</resources>
```

Main Activity Class Attempt 1: Named Inner Class

```java
public class Events3Example extends Activity {
    private View mColorRegion;
    private int[] mColorChoices =
            { Color.BLACK, Color.BLUE, ...};

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mColorRegion = findViewById(R.id.color_region);
        Button colorButton =
                (Button)findViewById(R.id.color_button);
        colorButton.setOnClickListener(new ColorRandomizer());
    }

    private void setRegionColor(int color) {
        mColorRegion.setBackgroundColor(color);
    }
```

There is nothing wrong with this approach. However, this event handler class is only used on this line of code. Furthermore, the code for ColorRandomizer (next page) is relatively short. So, you can make it a bit more concise with an anonymous inner class.

Main Activity Class Attempt 1: Named Inner Class (Continued)

```java
    private class ColorRandomizer
            implements OnClickListener {
        @Override
        public void onClick(View v) {
            Random generator = new Random();
            int index = generator.nextInt(mColorChoices.length);
            setRegionColor(mColorChoices[index]);
        }
    }
}
```

Main Activity Class Refactored: Anonymous Inner Class

```java
public class Events3Example extends Activity {
    private View mColorRegion;
    private int[] mColorChoices =
            { Color.BLACK, Color.BLUE, ...};


    private void setRegionColor(int color) {
        mColorRegion.setBackgroundColor(color);
    }
```

Main Activity Class Refactored: Anonymous Inner Class (Cont.)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    mColorRegion = findViewById(R.id.color_region);
    Button colorButton =
            (Button)findViewById(R.id.color_button);
    colorButton.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            Random generator = new Random();
            int index = generator.nextInt(mColorChoices.length);
            setRegionColor(mColorChoices[index]);
        }
    });
}
```

This defines the class and instantiates it all in one fell swoop. If you have never seen anonymous inner classes before, the confusion is probably not worth the code savings over a named inner class. However, once you are used to it, it is more concise and arguably easier to understand because the behavior is shown where it is used. This approach is very commonly used by Swing, SWT, AWT, and GWT programmers. This is also very analogous to anonymous functions (closures) that are widely used in functional programming languages.
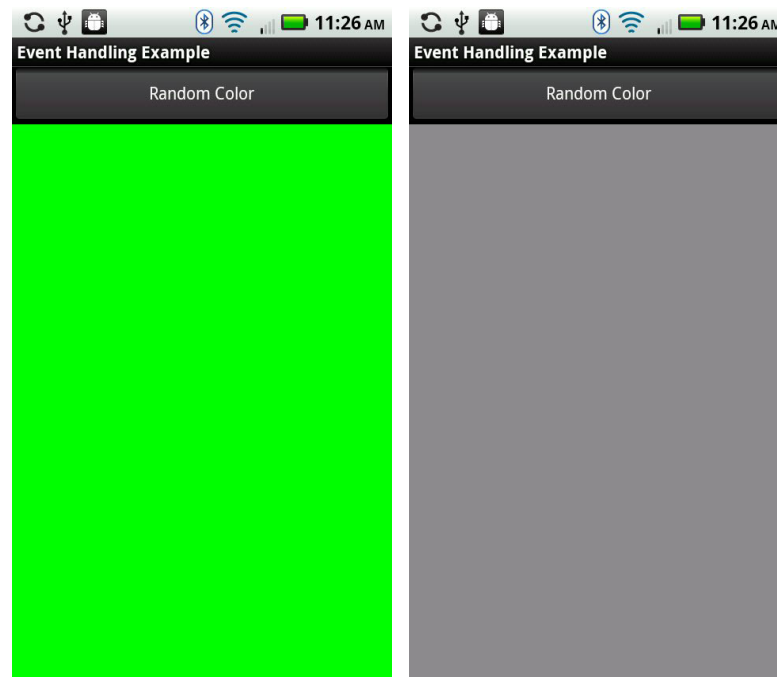
33

Results on Emulator
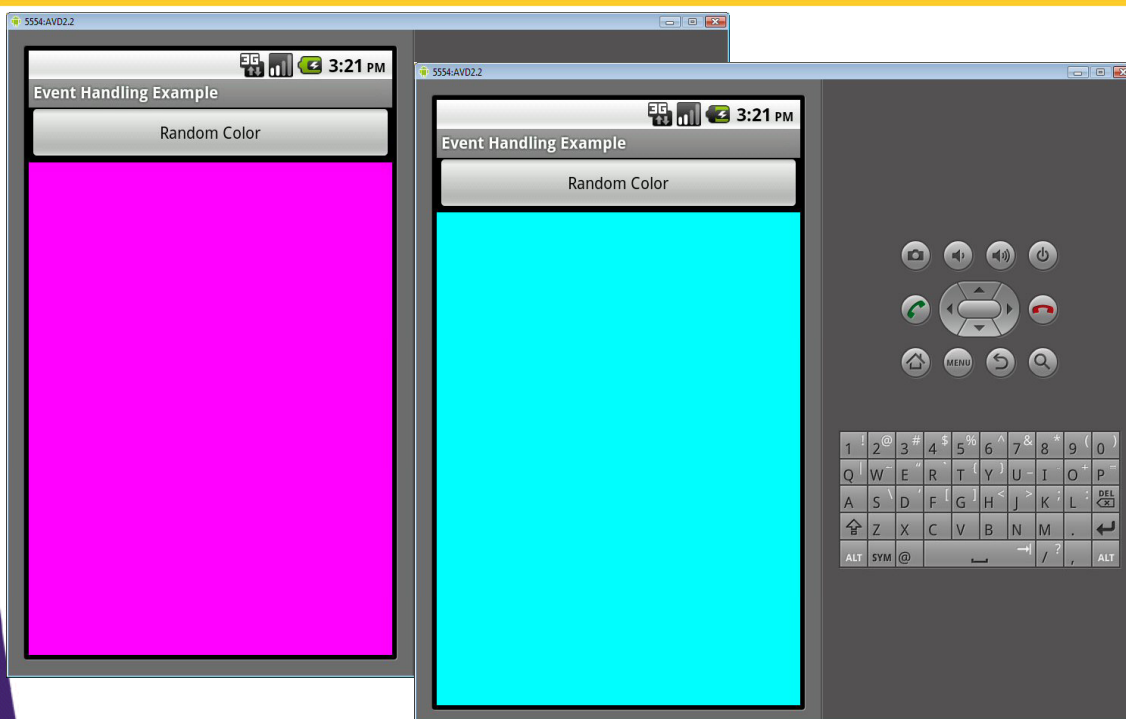
34

Results on Physical Phone

Handling Events by Having Main Activity Implement Listener Interface

Idea

- **Goal**
  - Randomly change color of TextView when Button is pressed.
    - Same as previous example
- **Approach**
  - Have the main Activity implement the Listener interface. Put the handler method in the main Activity. Call setOnClickListener(this).
- **Advantages**
  - Assuming that the app has only a single control of that Listener type, this is the shortest and simplest of the approaches.
- **Disadvantages**
  - Scales poorly to multiple controls unless they have *completely* identical behavior.
    - If you assigned "this" as the handler for more than one control of the same Listener type, the onClick (or whatever) method would have to have cumbersome if statements to see which control was clicked
    - This approach should be applied when your app has only a single control of that Listener type
  - You cannot pass arguments to the Listener.
    - So, again, works poorly for multiple controls

XML Files: Same as Previous Example

- **res/layout/main.xml**
  - Defines vertical LinearLayout that contains a Button and a TextView.
  - The Button and TextView have ids so that they can be referred to in the Java code
- **res/values/strings.xml**
  - Defines the app name and the label of the Button

Main Activity Class

```
public class Events5Example extends Activity
                            implements OnClickListener {
    private View mColorRegion;
    private int[] mColorChoices =
            { Color.BLACK, Color.BLUE, ... };

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mColorRegion = findViewById(R.id.color_region);
        Button colorButton =
                (Button)findViewById(R.id.color_button);
        colorButton.setOnClickListener(this);
    }
```
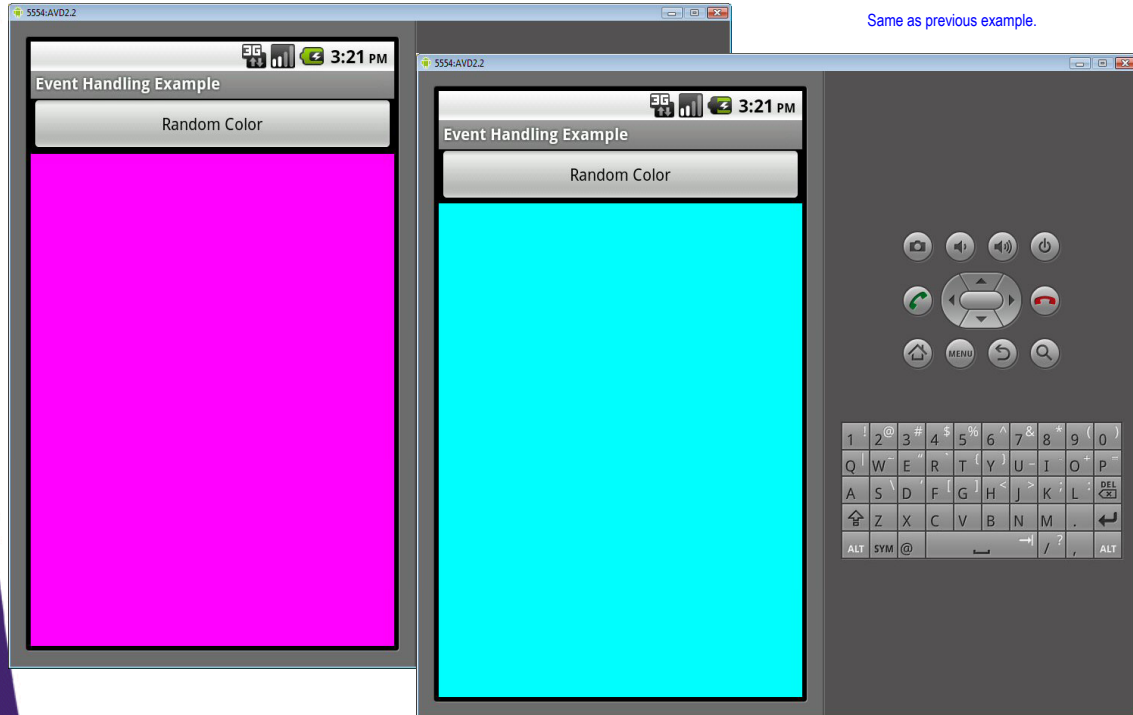
Main Activity Class (Continued)

```
    private void setRegionColor(int color) {
        mColorRegion.setBackgroundColor(color);
    }

    @Override
    public void onClick(View v) {
        Random generator = new Random();
        int index = generator.nextInt(mColorChoices.length);
        setRegionColor(mColorChoices[index]);
    }
}
```
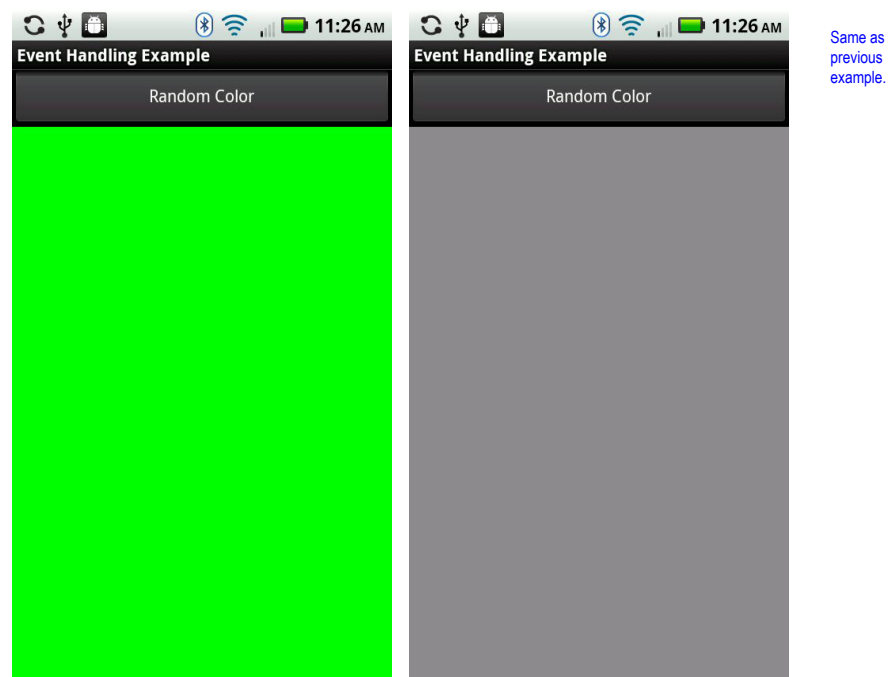
Results on Emulator

Same as previous example.

41

Results on Physical Phone

Same as previous example.

42

Handling Events by Specifying the Event Handler Method in main.xml

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, JSF 2.0, Java 6, Ajax, jQuery, GWT, Spring, Hibernate, RESTful Web Services, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Idea

- **Goal**
  - Randomly change color of TextView when Button is pressed.
    - Same as previous example
- **Approach**
  - Put the handler method in the main Activity. Do not implement a Listener interface or call setOnClickListener. Have the layout file (main.xml) specify the handler method via the android:onClick attribute.
- **Advantages**
  - Assuming that the app has only a single control of that Listener type, mostly the same advantages (short/simple code) as the previous approach where the Activity implemented the interface.
  - More consistent with the "do layout in XML" strategy
  - You can supply different method names for different controls, so not nearly as limited as interface approach.
- **Disadvantages**
  - You cannot pass arguments to Listener.
  - Less clear to the Java developer which method is the handler for which control

44

res/layout/main.xml

```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">
    <Button
        android:id="@+id/color_button"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:text="@string/button_prompt"
        android:onClick="randomizeColor"/>
    <TextView
        android:id="@+id/color_region"
        android:layout_height="match_parent"
        android:layout_width="match_parent"/>
</LinearLayout>
```

This is the name of the event handler method in the main class. This method must have a void return type and take a View as an argument. However, the method name is arbitrary, and the main class need not implement any particular interface.

res/values/strings.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Event Handling Example</string>
    <string name="button_prompt">Random Color</string>
</resources>
```

Unchanged from the previous two examples

Main Activity Class

```
public class Events6Example extends Activity {
    private View mColorRegion;
    private int[] mColorChoices =
            { Color.BLACK, Color.BLUE, ... };

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mColorRegion = findViewById(R.id.color_region);
        // No need to look up the button or assign event handler
    }
```

Main Activity Class (Continued)

```
    private void setRegionColor(int color) {
        mColorRegion.setBackgroundColor(color);
    }

    public void randomizeColor(View v) {
        Random generator = new Random();
        int index = generator.nextInt(mColorChoices.length);
        setRegionColor(mColorChoices[index]);
    }
}
```

Matches method name given for
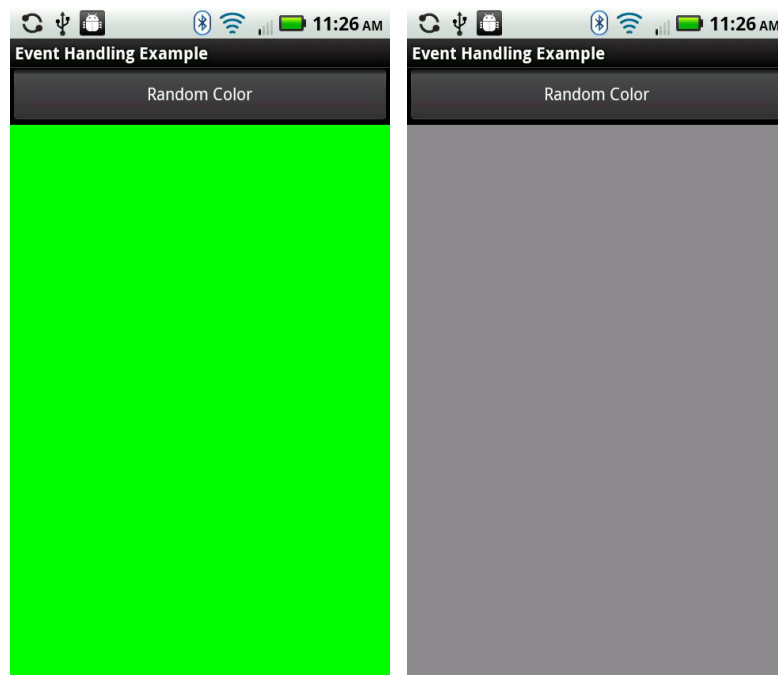android:onClick in main.xml

Results on Emulator

Same as previous example.

49

Results on Physical Phone
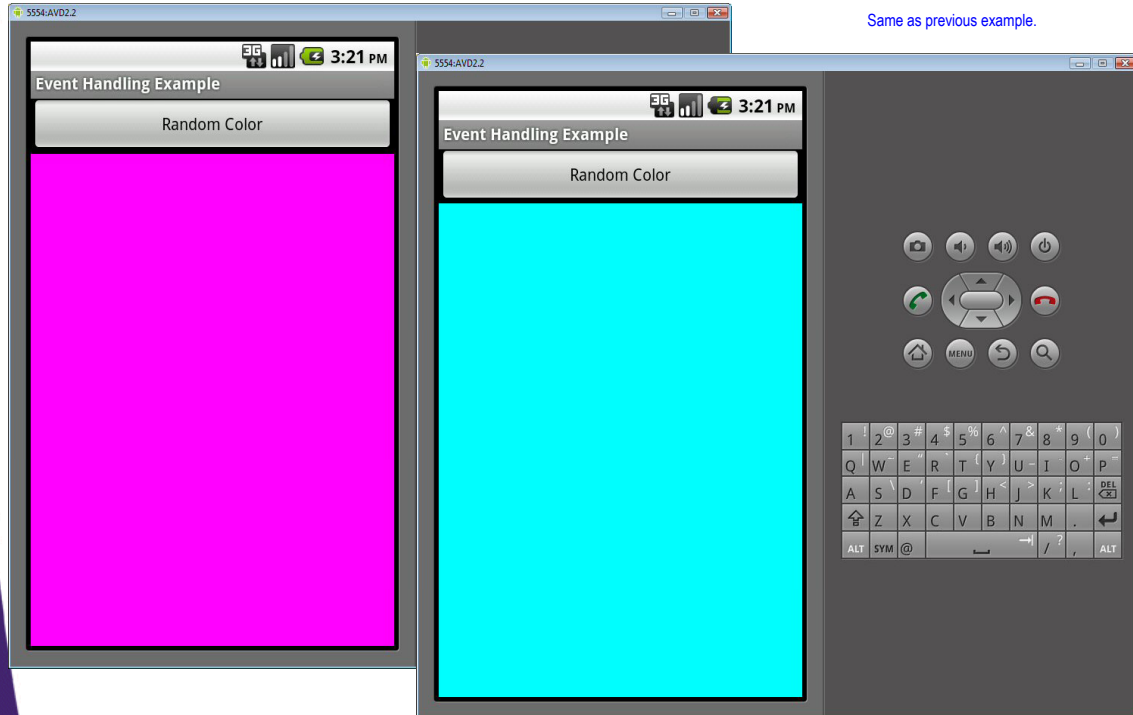
Same as previous example.

50

Aside: Copying Android Projects in Eclipse

Steps to Copy Projects

- **Issues**
  - The projects in this tutorial are very similar to each other.
    - So, you want to copy/rename previous project instead of making new project and copying many separate pieces
    - But, package names on devices must be unique
    - Renaming package requires care. Poor Eclipse support.
- **Steps (order of steps 2 and 3 matter!)**
  1. R-click old project. R-click and choose Paste. New name.
  2. R-click new project, Android Tools → Rename Application Package. New name. *Unselect* the Java classes, and leave selection for manifest only. OK when asked to update launch configuration.
  3. R-click src/projectName in new project. Refactor → Rename. OK when warned package exists.
  4. Optional: R-click main Activity. Refactor → Rename.

Wrap-Up

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, JSF 2.0, Java 6, Ajax, jQuery, GWT, Spring, Hibernate, RESTful Web Services, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

© 2011 Marty Hall

Approaches: Single or Multiple Controls

- **Situation**
  - Same handler code may be applied to multiple controls
- **Options**
  - Use a separate event handler class
    - Pros: can pass args to handler to customize behavior, easier to change independently of main app
    - Cons: if handler will call code in main Activity, must pass "this" and must make methods public
  - Use a named inner class
    - Pros: can pass args to handler to customize behavior, no need to pass "this" reference, methods can be private
      - This is my overall favorite and most widely used approach for Widget event handling
    - Cons: handler tightly coupled to main Activity

54

Approaches: Single Control

- **Situation**
  - Handler code will be applied only to a single control
- **Options**
  - Use an anonymous inner class
    - Pros: same as named inner class, but more concise
    - Cons: confusing to newbies or if handler code is long
  - Put handler method in Activity, implement interface, call setOnClickListener(this)
    - Pros: simple code
    - Cons: can't pass arguments to handler class
  - Put handler method in Activity, no interface, specify method with android:onClick in main.xml
    - Pros: one method per control, but can specify different methods for each control. More XML-oriented. Less Java code.
    - Cons: more confusing to Java developer (arguably)

55

Questions?