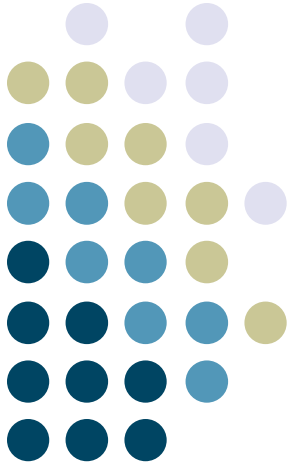
Java Swing GUI Programming 2

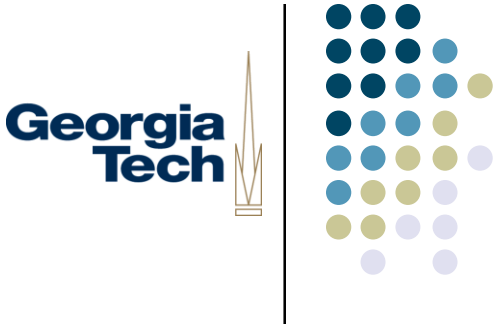


John Stasko

CS 6452

Prototyping Interactive Systems

Learning Objectives



- Inner classes
- Event model of Swing
  - Handler objects for UI components
  - Callback methods in handler objects
  - Action listeners
- Push buttons, check boxes, radio buttons
- Option pane dialog boxes
- Text fields
- Font controls

Precursor

```
public class MyClass
{
   private int count;

   private void doSomething(int a)
   {
      // code here
   }

}
```

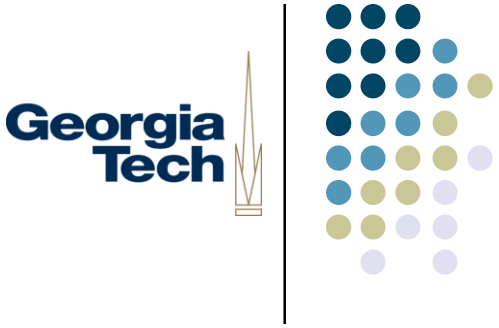
```
public class MyClass
{
   private int count;

   private void doSomething(int a)
   {
      myClass m = new MyClass();
   }

   private class myClass
   {
      private int total;

      public void meth1(String name)
      {
         // code here
      }
   }
}
```

Can have class inside too
It can access private data and methods of outer class
Outer class can access its public methods
Outside classes cannot access it if it is private

Event-driven Programming

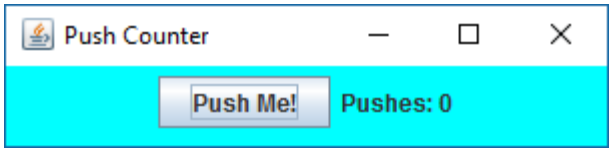
- Events – Actions (mouse button, key press) "generated" by component
- Listener – Object waits for event and responds appropriately

Typical GUI Program



- 1. Set up necessary component
- 2. Implement listener classes that define what to do when an event occurs
- 3. Establish relationships between listeners and components that generate events

Example



Each click causes the counter to go up by 1

PushCounter program

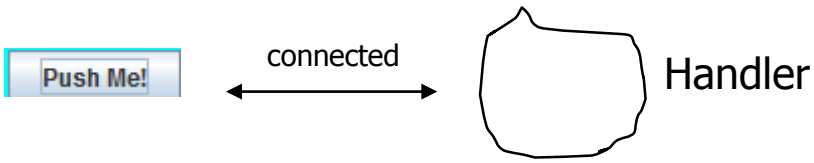
Panel Class

```
public class PushCounterPanel extends JPanel
{
   private int count;
   private JButton push;
   private JLabel label;

   public PushCounterPanel ()
   {
      count = 0;

      push = new JButton ("Push Me!");
      push.addActionListener (new ButtonListener());
      label = new JLabel ("Pushes: " + count);

      add(push);
      add(label);

      setPreferredSize (new Dimension(300, 40));
      setBackground (Color.cyan);
   }

   private class ButtonListener implements ActionListener
   {
      public void actionPerformed (ActionEvent event)
      {
         count++;
         label.setText("Pushes: " + count);
      }
   }
}
```

Concepts

For each interactive component such as a button, we must assign an object to "handle" or "manage" it

When the user does something in the UI to that interactive component, its handler object is notified (a special method in it is called)



Concepts

`JButton` – Component that allows you to push a button
Generates action event – Need method to handle it

```
push.addActionListener(ButtonListener)
```

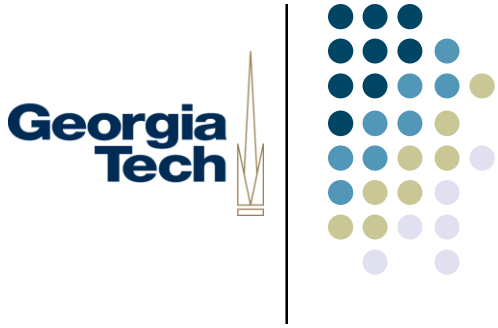
Call to connect a handler object with this button

Object that handles events

We use a private inner class, `ButtonListener`

`actionPerformed(ActionEvent)` called automatically by Java when button is pressed ("callback" method)

Concepts



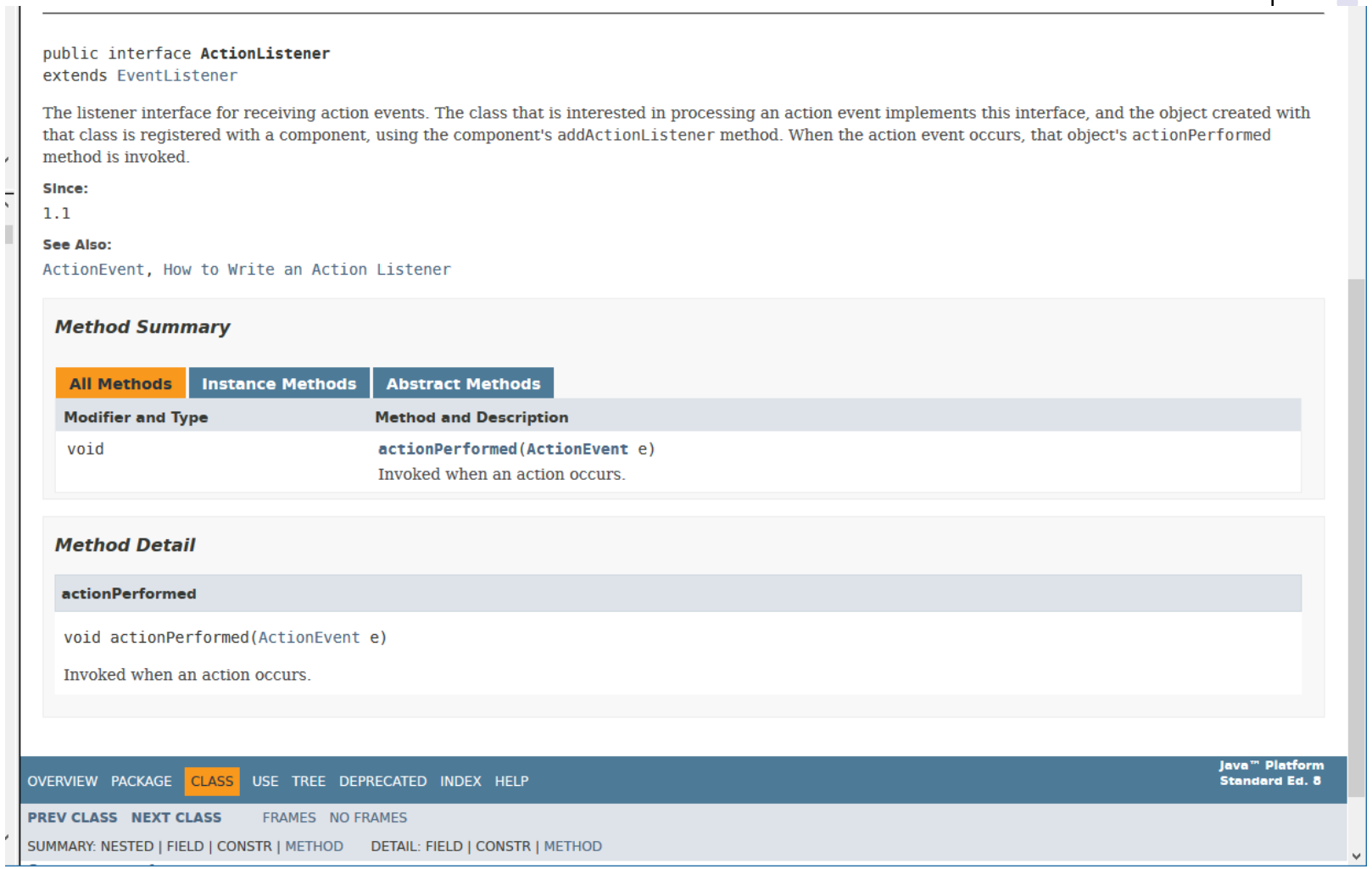
Can create an "inner class", one that lives inside of another

Can create instances of it

It has access to all of the class's instance data just as methods do

Here, it implements the ActionListener interface
What does that mean? It must provide an `actionPerformed(ActionEvent)` method

ActionListener Interface



public interface **ActionListener**
extends EventListener

The listener interface for receiving action events. The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's addActionListener method. When the action event occurs, that object's actionPerformed method is invoked.

Since:

1.1

See Also:

ActionEvent, How to Write an Action Listener

Method Summary

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method and Description
void	**actionPerformed(ActionEvent e)** Invoked when an action occurs.

Method Detail

actionPerformed

void actionPerformed(ActionEvent e)

Invoked when an action occurs.

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP — Java™ Platform Standard Ed. 8

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Concepts

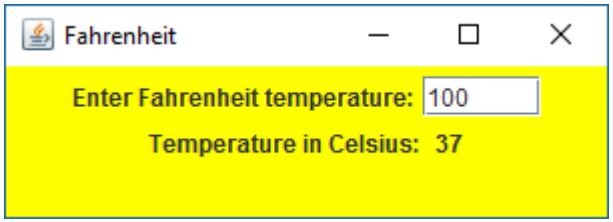
How do we do the counting?
Each time button is pressed, the handler method is called
We keep an int counter (instance variable in class) and increment it in there

Still need to update the UI
Change the string of the label component to have the new counter value

Next Component

- `JTextField` – Interactive text entry field
  - Constructor takes integer that is # of characters in the field
  - `actionPerformed(ActionEvent)` is called when "Return" is hit in field

Example



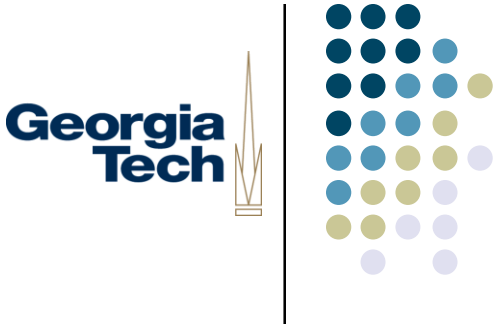
Enter Fahrenheit temperature and have it converted

Fahrenheit program

Panel Class

```
public class FahrenheitPanel extends JPanel
{
   private JLabel inputLabel, outputLabel, resultLabel;
   private JTextField fahrenheit;

   public FahrenheitPanel()
   {
      inputLabel = new JLabel ("Enter Fahrenheit temperature:");
      outputLabel = new JLabel ("Temperature in Celsius: ");
      resultLabel = new JLabel ("---");

      fahrenheit = new JTextField (5);
      fahrenheit.addActionListener (new TempListener());

      add (inputLabel);
      add (fahrenheit);
      add (outputLabel);
      add (resultLabel);

      setPreferredSize (new Dimension(300, 75));
      setBackground (Color.yellow);
   }

   private class TempListener implements ActionListener
   {
      public void actionPerformed (ActionEvent event)
      {
         int fahrenheitTemp, celsiusTemp;

         String text = fahrenheit.getText();

         fahrenheitTemp = Integer.parseInt (text);
         celsiusTemp = (fahrenheitTemp-32) * 5/9;

         resultLabel.setText (Integer.toString (celsiusTemp));
      }
   }
}
```

Concepts



Three labels and a text field in the panel

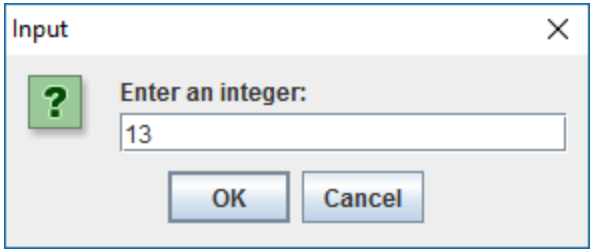
When new value entered, handler method called
It must pull the entered value out of the text label
then change the converted temperature and display that

Entered new value is a String that must be converted to int
Converted int then must be made into a String for label

`Integer.parseInt(String)` & `Integer.toString(int)`

New Component

Swing contains class `JOptionPane` that helps to create dialog boxes



3 kinds

Message – Just displays message

Input – Prompts user to enter one string

Confirm – Yes/no answer

Methods

Usually just a String

```
static String showInputDialog(Object msg)
static int showConfirmDialog(Component parent, Object msg)
static int showMessageDialog(Component parent, Object msg)
```

If `parent` is null, dialog is centered on screen

EvenOdd program

Code

```
import javax.swing.JOptionPane;

public class EvenOdd
{
   public static void main (String[] args)
   {
      String numStr, result;
      int num, again;

      do
      {
         numStr = JOptionPane.showInputDialog ("Enter an integer: ");
         num = Integer.parseInt(numStr);

         result = "That number is " + ((num%2 == 0) ? "even" : "odd");

         JOptionPane.showMessageDialog (null, result);
         again = JOptionPane.showConfirmDialog (null, "Do Another?");
      }
      while (again == JOptionPane.YES_OPTION);
   }
}
```

Just one file this time, all in main()

New Component

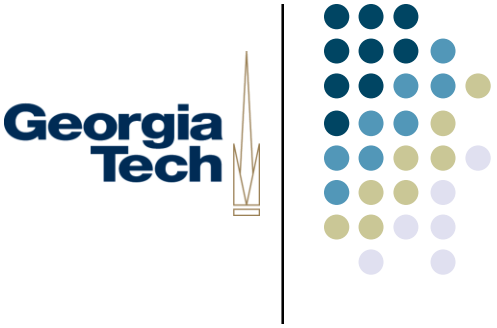
`JCheckBox`

Works similarly to `JButton` we saw before
- Need listener (`ItemListener`)
- Event is generated when checked/unchecked (`itemStateChanged`)

To check state of checkbox, call `isSelected()`, returns boolean

StyleOptions program

Panel Code



```
public class StyleOptionsPanel extends JPanel
{
   private JLabel saying;
   private JCheckBox bold, italic;

   public StyleOptionsPanel()
   {
      saying = new JLabel ("Say it with style!");
      saying.setFont (new Font ("Helvetica", Font.PLAIN, 36));

      bold = new JCheckBox ("Bold");
      bold.setBackground (Color.cyan);
      italic = new JCheckBox ("Italic");
      italic.setBackground (Color.cyan);

      StyleListener listener = new StyleListener();
      bold.addItemListener (listener);
      italic.addItemListener (listener);

      add (saying);
      add (bold);
      add (italic);

      setBackground (Color.cyan);
      setPreferredSize (new Dimension(300, 100));
   }
// continues...
```

```
// continued ...
   private class StyleListener implements ItemListener
   {
      public void itemStateChanged (ItemEvent event)
      {
         int style = Font.PLAIN;

         if (bold.isSelected())
            style = Font.BOLD;

         if (italic.isSelected())
            style += Font.ITALIC;

         saying.setFont (new Font ("Helvetica", style, 36));
      }
   }
}
```

Items of Note

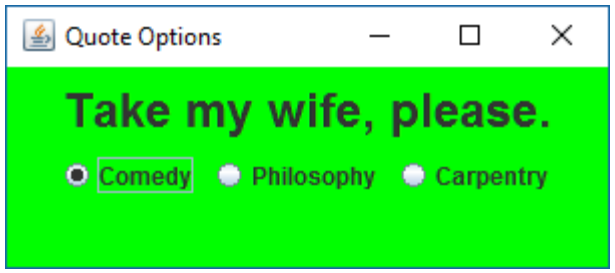
- Use of checkboxes
- New font
- Using one listener for two objects
- Note how listener accesses two checkbox objects which are instance data in class
- Doesn't matter which box generates the event
- Note PLAIN, BOLD, ITALIC constants in Font class

New Component

`JRadioButton` Push one in, others pop out

Works differently: Create `ButtonGroup()` object then add each radio button to it. (It handles the push behavior.)

Uses `ActionListener` to catch `actionPerformed` event



QuoteOptions program

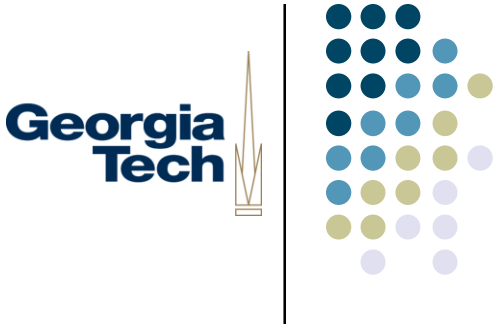
Panel Code

```
public class QuoteOptionsPanel extends JPanel
{
   private JLabel quote;
   private JRadioButton comedy, philosophy, carpentry;
   private String comedyQuote, philosophyQuote, carpentryQuote;

   public QuoteOptionsPanel()
   {
      comedyQuote = "Take my wife, please.";
      philosophyQuote = "I think, therefore I am.";
      carpentryQuote = "Measure twice. Cut once.";

      quote = new JLabel (comedyQuote);
      quote.setFont (new Font ("Helvetica", Font.BOLD, 24));

      comedy = new JRadioButton ("Comedy", true);
      comedy.setBackground (Color.green);
      philosophy = new JRadioButton ("Philosophy");
      philosophy.setBackground (Color.green);
      carpentry = new JRadioButton ("Carpentry");
      carpentry.setBackground (Color.green);

      ButtonGroup group = new ButtonGroup();
      group.add (comedy);
      group.add (philosophy);
      group.add (carpentry);

      QuoteListener listener = new QuoteListener();
      comedy.addActionListener (listener);
      philosophy.addActionListener (listener);
      carpentry.addActionListener (listener);

      add (quote);
      add (comedy);
      add (philosophy);
      add (carpentry);

      setBackground (Color.green);
      setPreferredSize (new Dimension(300, 100));
   }
// continues ...
```

```
// continued ...
  private class QuoteListener implements ActionListener
  {
     public void actionPerformed (ActionEvent event)
     {
        Object source = event.getSource();

        if (source == comedy)
           quote.setText (comedyQuote);
        else
           if (source == philosophy)
              quote.setText (philosophyQuote);
           else
              quote.setText (carpentryQuote);
     }
  }
}
```

We get the source button in the `actionPerformed` method to determine which one was picked

Learning Objectives



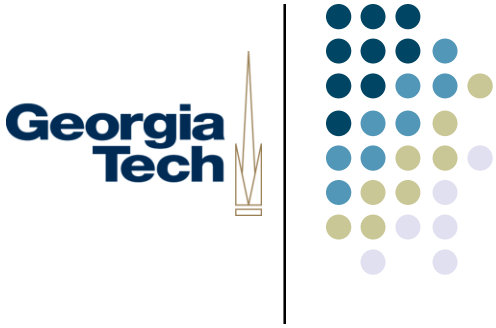
- Inner classes
- Event model of Swing
  - Handler objects for UI components
  - Callback methods in handler objects
  - Action listeners
- Push buttons, check boxes, radio buttons
- Option pane dialog boxes
- Text fields
- Font controls

Reading

- Barkhuus, et al
  "Picking Pockets on the Lawn..."

- Do reading review for Tuesday

HW 7



- Pick a Color
- Use principles from all the examples we've seen
- Requires both paintComponent() edits and doing event handling
- Due Thursday @ 1:30pm

Next Time

- Communication in a GUI program
- Positioning components in a window