GUI Dynamics

Lecture 23
CS211 – Fall 2005

GUI Statics vs. GUI Dynamics

- **Statics:**
  - what’s drawn on the screen
  - **Components**
    - E.g., buttons, labels, lists, sliders
  - **Containers:** components that contain other components
    - E.g., frames, panels, dialog boxes
  - **Layout managers:** control placement and sizing of components

- **Dynamics:**
  - user interactions
  - **Events**
    - E.g., button-press, mouse-click, key-press
  - **Listeners:** an object that responds to an event
  - **Helper classes**
    - E.g., Graphics, Color, Font, FontMetrics, Dimension

Dynamics Overview

- **GUI dynamics:** causing and responding to actions
  - **What actions?**
    - Called events
      - Need to write code that “understands” what to do when an event occurs
    - In Java, you specify what happens by providing an object that “hears” the event
      - In other languages, you specify what happens in response to an event by providing a **function**
  - **What objects do we need?**
    - Events
    - Event listeners

Brief Example Revisited

```java
import java.awt.*;
import java.awt.event.*;
public class Intro extends JFrame {
    private int count;
    private JButton b = new JButton("Push Me!");
    private JLabel label = new JLabel(generateLabel());
    public static void main(String[] args) {
        JFrame f = new Intro();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(200,100);
        f.setVisible(true);
    }
    public Intro() {
        setLayout(new FlowLayout(FlowLayout.LEFT) );
        add(b);
        add(label);
        b.addActionListener( new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                count++;
                label.setText(generateLabel());
            }
        } );
    }
    private String generateLabel() {
        return "Count: " + count;
    }
}
```

Delegation Model

- **Timeline for an event**
  - User (or program) does something to a component
  - Java issues an event object
  - A special type of object (a listener) “hears” the event
    - The listener has a method that “handles” the event
    - The handler can do whatever the programmer programmed

- **What do you need to understand**
  - Events: How a component issues an event
  - Listeners: How to make an object that listens for events
  - Handlers: How to write a method that responds to an events
Events

• An Event is a Java object
  • It is used to indicate to that an action has occurred
  • Examples: mouse clicked, button pushed, menu item selected, key pressed
  • Usually, Events are created by the Java runtime system
    • It’s possible to create your own events, but this is unusual

• Most events are in java.awt.event
  • Some events are in javax.swing.event
  • All events are subclasses of AWTEvent

Kinds of Events

• Each Swing Component can generate one or more kinds of events
  • The possible events depend on the component
    • Example: Clicking a JButton creates an ActionEvent
    • Example: Clicking a JCheckbox creates an ItemEvent
  • The different kinds of events include different information about what has occurred
    • All events have method getSource() which returns the object (e.g., the button or checkbox) on which the Event initially occurred
    • An ItemEvent has a method getStateChange() that returns an integer indicating whether the item (e.g., the checkbox) was selected or deselected

Listeners are Interfaces

• Java provides a way to associate components with their event listeners
  • Example:
    JButton b = new JButton("button text");
    b.addActionListener(an ActionListener object)
  • Note that an ActionListener is an interface
    • Thus any class that implements that interface can be used as an ActionListener

Implementing Listeners

• Which class should be a listener?
  • Java has no restrictions on this, so any class that implements the listener will work
  • Typical choices
    • Top-level container that “contains” whole GUI
      public class MyGUI extends JFrame implements ActionListener
    • Inner classes to create specific listeners for reuse
      private class LabelMaker implements ActionListener
    • Anonymous classes created “on the spot”
      b.addActionListener(new ActionListener() {...});

Listeners and Listener Methods

• When you implement an interface, Java requires that you implement the interface’s methods
  • Thus you are forced to implement all the methods necessary to correctly handle an event
  • Example: ActionListener has one method:
    void actionPerformed(ActionEvent e)
  • Example: MouseInputListener has seven methods:
    void mouseClicked(MouseEvent e)
    void mouseEntered(MouseEvent e)
    void mouseExited(MouseEvent e)
    void mousePressed(MouseEvent e)
    void mouseReleased(MouseEvent e)
    void mouseDragged(MouseEvent e)
    void mouseMoved(MouseEvent e)

Registering Listeners

• How does a component know which listener to use? You must register the listeners
  • This connects listener objects with their source objects
  • Syntax: component.addTypeListener(Listener)

• Example
  b.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      count++;
      label.setText(generateLabel());
    }
  });
Example 1: the Frame is the Listener
import javax.swing.*; import java.awt.*; import java.awt.event.*;
public class ListenerExample1 extends JFrame implements ActionListener {
    private int count;
    private JButton b = new JButton("Push Me!");
    private JLabel label = new JLabel(generateLabel());
    public static void main (String[] args) {
        JFrame f = new ListenerExample1();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(200,100);
        f.setVisible(true);
    }
    public ListenerExample1() {
        setLayout(new FlowLayout(FlowLayout.LEFT) );
        add(b); add(label);
        b.addActionListener(this);
    }
    public void actionPerformed (ActionEvent e) {
        count++;
        label.setText(generateLabel());
    }
    private String generateLabel() {
        return "Count: "+count;
    }
}

Example 2: the Listener is an Inner Class
import javax.swing.*; import java.awt.*; import java.awt.event.*;
public class ListenerExample2 extends JFrame {
    private int count;
    private JButton b = new JButton("Push Me!");
    private JLabel label = new JLabel(generateLabel());
    class Helper implements ActionListener {
        public void actionPerformed (ActionEvent e) {
            count++;
            label.setText(generateLabel());
        }
    }
    public static void main (String[] args) {
        JFrame f = new ListenerExample2();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(200,100); f.setVisible(true);
    }
    public ListenerExample2() {
        setLayout(new FlowLayout(FlowLayout.LEFT) );
        add(b); add(label);
        b.addActionListener(new Helper());
    }
    private String generateLabel() {
        return "Count: "+count;
    }
}

Example 3: the Listener is an Anonymous Class
import javax.swing.*; import java.awt.*; import java.awt.event.*;
public class ListenerExample3 extends JFrame {
    private int count;
    private JButton b = new JButton("Push Me!");
    private JLabel label = new JLabel(generateLabel());
    public static void main (String[] args) {
        JFrame f = new ListenerExample3();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(200,100); f.setVisible(true);
    }
    public ListenerExample3() {
        setLayout(new FlowLayout(FlowLayout.LEFT) );
        add(b); add(label);
        b.addActionListener(new ActionListener() {
            public void actionPerformed (ActionEvent e) {
                count++;
                label.setText(generateLabel());
            }
        });
    }
    private String generateLabel() {
        return "Count: "+count;
    }
}

Adapters
• Some listeners (e.g., MouseInputListener) have lots of
  methods; you don’t always need all of them
  • For instance, I may be interested only in mouse clicks
• For this kind of situation, Java provides adapters
  • An adapter is a predefined class that implements all the
    methods of the corresponding Listener
  • Example: MouseInputAdapter is a class that implements all the
    methods of interface MouseInputListener
  • The adapter methods do nothing
  • To easily create your own listener, you extend the adapter class,
    overriding just the methods that you actually need

Using an Adapter to Count Mouse Entries
import javax.swing.*; import javax.swing.event.*;
import java.awt.*; import java.awt.event.*;
public class AdapterExample extends JFrame {
    private int count; private JButton b = new JButton("Mouse Me!");
    private JLabel label = new JLabel(generateLabel());
    class Helper extends MouseInputAdapter {
        public void mouseEntered (MouseEvent e) {
            count++;
            label.setText(generateLabel());
        }
    }
    public static void main (String[] args) {
        JFrame f = new AdapterExample();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(200,100); f.setVisible(true);
    }
    public AdapterExample() {
        setLayout(new FlowLayout(FlowLayout.LEFT) );
        add(b); add(label);
        b.addMouseListener(new Helper());
    }
    private String generateLabel() {
        return "Count: "+count;
    }
}

Some Notes on Events and Listeners
• A single component can have many listeners
• Multiple components can share the same listener
  • Can use event.getSource() to identify the component to
    which an event belongs
• Take a look at
  http://java.sun.com/docs/books/tutorial/uiswing/events/generalrules.html
  for more information on designing listeners
• You can’t sit down and quickly write a GUI
  • You need to use the API and the Swing Tutorial
    (http://java.sun.com/docs/books/tutorial/uiswing/)