1.00 Lecture 18

Swing Event Model

Reading for next time: Text 9.5, 18.3

GUI Event Model: Paradigm Shift

- Operating system (Windows, JVM) runs the show:
  - Monitors keystroke, mouse, other I/O events from sources
  - Dispatches event messages to programs that need to know
  - Each program decides what to do when the event occurs
- This is the reverse of console-oriented programming, where the program runs the show, and asks the operating system (OS) to get input when it wants it
- Event sources: menus, buttons, scrollbars, etc.
  - Have methods allowing event listeners to register with them
  - When event occurs, source sends message (an event object) to all registered listener objects
    - EventObject is the superclass
      - ActionEvent, MouseEvent, etc. are subclasses that we use
- Event listeners: objects in your program that respond to events
Exercise 1: Button Events

• Download ButtonFrame, ButtonTest, ButtonPanel
• We will build an application
  – User presses button
  – Application shows number of button presses
• Demo

![Button Events Example](image1)

© Oracle. All rights reserved. This content is excluded from our Creative Commons license. For more information, see [http://ocw.mit.edu/fairuse](http://ocw.mit.edu/fairuse).

Exercise 1 cont.

• Preliminaries:
  – Complete ButtonTest as shown on next page:
    • main():
      – Create new ButtonFrame object (inherits from JFrame, ButtonFrame class to be written next)
      – Sets default close operation
      – Sets frame visible
  – Complete ButtonFrame:
    • Set title
    • Set size
    • Get contentPane
    • Create ButtonPanel object (ButtonPanel written next)
    • Add the ButtonPanel object to the contentPane
  – Use last lecture’s notes as a guide
**Exercise 1: ButtonTest/Frame**

```java
import javax.swing.*;
public class ButtonTest {
    public static void main(String[] args) {
        // Create new frame (what class?)
        // Set default close option
        // Show frame (set visible)
    }
} // main has 3 lines of SwingTest main (last lecture)
```

```java
import java.awt.*;
import javax.swing.*;
public class ButtonFrame extends JFrame {
    public ButtonFrame() {
        // Call superclass constructor with "Button Example"
        // Set size of frame (200 by 200)
        // Get content pane
        // Create new panel (ButtonPanel, to be written next)
        // Add panel to content pane in BorderLayout.CENTER
    }
} // ButtonFrame has rest of SwingTest main(last lecture)
```

**SwingTest from last lecture**

```java
import java.awt.*;
import javax.swing.*;

public class SwingTest {
    public static void main(String args[]) {
        JFrame frame = new JFrame("Welcome to 1.00");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(500,400);
        Container contentPane = frame.getContentPane(); // No J
        // We never draw on a JFrame. Instead we update
        // components that are added to its contentPane
        JPanel panel = new JPanel();
        panel.setBackground(Color.yellow);
        // Add panel to the contentPane of the JFrame
        contentPane.add(panel, BorderLayout.CENTER);
        frame.setVisible(true);
    }
}
```
Exercise 2: ButtonPanel, p.1

- Complete ButtonPanel constructor on next slide:
  - Create JButton, JLabel, Font objects
  - Add the button and label objects to the ButtonPanel
  - Tell the JButton object to send an ActionEvent to the object (ButtonPanel) that is the ActionListener. If panel is the listener, and button is the JButton object
    
    ```java
    button.addActionListener(panel);
    ```
  - In this example, since we are in the ButtonPanel constructor, and we need to refer to ButtonPanel itself, we use the keyword this as the argument
  - this is a reference to the current object; Java provides it automatically for every object
    - It’s a hidden first argument in every method call
  - Last, read the actionPerformed method to understand what it does
Exercise 2: Button, p.2

```java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class ButtonPanel extends JPanel implements ActionListener {
    private int i = 0;
    private JLabel countLabel;
    private JButton countButton;

    public ButtonPanel() {
        // Create new JButton with prompt: "Show count"
        // Create new JLabel (for output) with text: "Count= 0"
        // Create new Font font1: "Monospaced", Font.PLAIN, size 24
        countLabel.setFont(font1);
        countButton.setFont(font1);
        // Add your button to ButtonPanel (use add() method)
        // Add your label to ButtonPanel (use add() method)
        // Make the ButtonPanel object be the action listener
        // (We're in the ButtonPanel constructor, so use this)
    }
    public void actionPerformed(ActionEvent e) {
        i++;
        countLabel.setText("Count= " + i);
    }
}
```

Exercise 3: Clock

- You’ll complete a clock:

- ClockFrame: written for you
  - Main: creates new ClockFrame
  - Constructor:
    - Gets contentPane, creates ClockPanel, adds to contentPane
- ClockPanel: complete two blocks of code
  - Constructor: Creates 2 buttons, 2 labels, adds to panel
  - Overrides paintComponent(), has actionPerformed()
import java.awt.*;
import javax.swing.*;

public class ClockFrame extends JFrame{
    public ClockFrame() {
        super("Clock Test");   // Or setTitle(…)
        setSize(300, 200);
        ClockPanel clock = new ClockPanel();
        Container contentPane = getContentPane();
        contentPane.add(clock, BorderLayout.CENTER);
    }

    public static void main(String[] args) {
        ClockFrame frame = new ClockFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}

import java.awt.event.*;

public class ClockPanel extends JPanel implements ActionListener {
    private JButton tickButton, resetButton;
    private JLabel hourLabel, minuteLabel;
    private int minutes = 720;   // 12 noon

    public ClockPanel(){
        JPanel bottomPanel = new JPanel();
        tickButton = new JButton("Tick");
        resetButton = new JButton("Reset");
        hourLabel = new JLabel("12:");
        minuteLabel = new JLabel("00");
        bottomPanel.add(tickButton);
        bottomPanel.add(resetButton);
        bottomPanel.add(hourLabel);
        bottomPanel.add(minuteLabel);
        setLayout(new BorderLayout());  // Next lecture
        add(bottomPanel, BorderLayout.SOUTH);
        // Who will listen to the button events? Your code here
    }
}
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D) g;

    Shape e = new Ellipse2D.Double(100, 0, 100, 100);
    g2.draw(e);

    double hourAngle = 2*Math.PI*(minutes- 3*60)/(12*60);
    double minuteAngle = 2*Math.PI * (minutes - 15) / 60;

    Line2D.Double hour = new Line2D.Double(150, 50,
            150 + (int) (30 * Math.cos(hourAngle)),
            50 + (int) (30 * Math.sin(hourAngle)));
    g2.draw(hour);

    Line2D.Double m = new Line2D.Double(150, 50,
            150 + (int) (45 * Math.cos(minuteAngle)),
            50 + (int) (45 * Math.sin(minuteAngle)));
    g2.draw(m);
}

public void setLabels() {
    // Doesn't handle midnight
    int hours = minutes/60;
    int min = minutes - hours*60;
    hourLabel.setText(hours+ ":");
    if (min < 10)    // Minutes should be two digits
        minuteLabel.setText("0" + min);
    else
        minuteLabel.setText("" + min);
}

public void actionPerformed(ActionEvent e) {
    if (e.getSource().equals(tickButton))
        // Complete this code: update clock and labels
    else
        // Reset button
        // Complete this code: update clock and labels
}
Clock Exercise Methods

- **Methods**
  - **paintComponent(Graphics g):**
    - This method draws the clock and the hours and minutes hands based on minutes
  - **setLabels():**
    - This method sets the hour and minute labels to the correct values based on minutes. It is a helper method you can call when writing `actionPerformed()`
  - **actionPerformed():**
    - If the event is from the tick button,
      - increment minutes by one and repaint the clock
      - repaint() will call the paintComponent() method which will redraw the clock with the clock hands adjusted to the new minutes value
    - We never call paintComponent() directly; always use repaint(). JVM manages the calls to paintComponent() –repaint() is a request to call paintComponent(). JVM must repaint when other apps obscure, etc.
    - Update the labels
    - If the event is from the reset button
      - Reset minutes to 720 (noon), repaint the clock and update the labels

Clock Exercise 3

- **ClockFrame should compile and run after you’ve placed it in Eclipse**
  - It won’t, alas, do anything
- **To make it do something:**
  - Hook up the listener to the buttons in the ClockPanel constructor
    - The ClockPanel object is the listener that updates the display, so use the ‘this’ keyword
  - Complete the actionPerformed() method in class ClockPanel
- **If you have time:**
  - Move the clock figure away from the top of the frame
  - Make the clock circle, hour and minute hands be different colors and thicknesses
  - Handle midnight correctly