

# Object Oriented Programming with Java

COMS M0103  
2004

## Lecture 13 -- Event Handling

Tim Kovacs  
kovacs@cs.bris.ac.uk

Frederik Vercauteren  
frederik@cs.bris.ac.uk

## Overview of Lecture

1. The event model of how GUIs respond to the user
2. An example of a panel and 3 buttons that change its colour
3. How to simplify the example using a helper class and an anonymous inner class
4. A bad alternative to inner classes
5. A more complex event handling example of how to exit an application when a window closes
6. How adapter classes can simplify implementing listener interfaces
7. A list of AWT listener interfaces and adapter classes

2

## 1 The Event Model

- GUIs respond to *events* like clicking on buttons and moving sliders
- *Event sources* (like buttons) generate *event objects*
- *Event listeners* (also called *event handlers*) receive event objects

### Event listeners

- implement the methods defined by *listener interfaces*
- *register* with event sources to get events from them
  - this technique is called a *callback*

### Different event sources generate different types of events

- buttons generate `ActionEvent` objects, windows `WindowEvents`
- different types of events require different listener interfaces

3

## 1 Event Handling

3 steps to use an event listener:

- 1) Implement `xxxListener` interface
- 2) Create an object of this class: the event listener
- 3) Register listener with source using `addxxxListener()`

```
...  
ActionListener listener = ...; // create listener  
JButton button = new JButton("Ok"); // create source  
button.addActionListener(listener); // register listener  
...
```

4

## 2 JButton Example

This example:

- adds `JButtons` to a panel
- gives them event handlers
- uses inner and anonymous classes

Each button changes the panel background to a certain colour

5

## 2 Working with JButton

Button constructors accept a string, `ImageIcon` or both:

```
JButton quitButton = new JButton("Quit");  
JButton imageButton = new  
    JButton(new ImageIcon("star.gif"));
```

If we extend `JPanel` we add components like this:

```
class ButtonPanel extends JPanel {  
    public ButtonPanel() {  
        add(quitButton);  
        add(imageButton);  
    }  
}
```

Note: we add components directly to `ButtonPanel`, not to a content pane as we do with `JFrame`.

6

## 2 ActionListener Interface

JButton clicks are handled by the ActionListener interface and its only method: actionPerformed

Step 1 of using an event listener:

```
class ColorAction implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        // set JPanel background colour
    }
    ...
}
```

Steps 2 and 3 occur in ButtonPanel constructor:

```
public ButtonPanel() {
    ...
    ColorAction yellowAction = new
        ColorAction(Color.yellow); // step 2
    yellowButton.addActionListener(yellowAction); //step 3
    ...
}
```

7

## 2 More on ActionListener

ActionListener interface also handles:

- selection from list boxes
- menu item selection
- enter/return key in text field
- Timer components going off

8

## 2 ColorAction: An Inner Class

ColorAction handles JButton clicks and sets ButtonPanel's background colour.

**But** - how does ColorAction know which ButtonPanel object to change? (There's only 1, but could have been more.)

Solutions:

- 1) Store ButtonPanel object in ColorAction object, or:
- 2) Make ColorAction an inner class of ButtonPanel.
  - This allows us to call setBackground method of 'outer' class (ButtonPanel)
  - setBackground is actually inherited by ButtonPanel from JPanel

9

## 2 ButtonTest.java (1/4)

```
/**
 * @author Cay Horstmann. Core Java vol.1 p. 282
 */

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

public class ButtonTest {
    public static void main(String[] args) {
        ButtonFrame frame = new ButtonFrame();

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.show();
    }
}
```

10

## 2 ButtonTest.java (2/4)

```
/**
 * A frame with a button panel
 */
class ButtonFrame extends JFrame {
    public ButtonFrame() {
        setTitle("ButtonTest");
        setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);

        // add panel to frame
        ButtonPanel panel = new ButtonPanel();
        Container contentPane = getContentPane();
        contentPane.add(panel);
    }

    public static final int DEFAULT_WIDTH = 300;
    public static final int DEFAULT_HEIGHT = 200;
}
```

11

## 2 ButtonTest.java (3/4)

```
// A panel with three buttons.
class ButtonPanel extends JPanel {
    public ButtonPanel() {
        // create buttons
        JButton yellowButton = new JButton("Yellow");
        JButton blueButton = new JButton("Blue");
        JButton redButton = new JButton("Red");

        // add buttons to panel
        add(yellowButton);
        add(blueButton);
        add(redButton);

        // create button actions
        ColorAction yellowAction = new ColorAction(Color.yellow);
        ColorAction blueAction = new ColorAction(Color.blue);
        ColorAction redAction = new ColorAction(Color.red);
    }
}
```

12

## 2 ButtonTest.java (4/4)

```
// Associate actions with buttons
yellowButton.addActionListener(yellowAction);
blueButton.addActionListener(blueAction);
redButton.addActionListener(redAction);
}

//Action listener that sets panel color - note inner class.
private class ColorAction implements ActionListener {
    public ColorAction(Color c) {
        backgroundColor = c;
    }

    public void actionPerformed(ActionEvent event) {
        setBackground(backgroundColor); // ButtonPanel method
    }
    private Color backgroundColor;
}
}
```

13

## 3 Simplification: Helper Method

If we add a helper method:

```
void makeButton(String name, Color backgroundColor) {
    JButton button = new JButton(name);
    add(button);
    ColorAction action = new ColorAction(backgroundColor);
    button.addActionListener(action);
}
```

The constructor simplifies to:

```
public ButtonPanel() {
    makeButton("yellow", Color.yellow);
    makeButton("blue", Color.blue);
    makeButton("red", Color.red);
}
```

14

## 3 Simplification: Anonymous Class

Since ColorAction is only used once we can make it anonymous and make it an inner class:

```
void makeButton(String name, final Color
backgroundColor) {
    JButton button = new JButton(name);
    add(button);
    button.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent event) {
            setBackground(backgroundColor);
        }
    });
}
```

15

## 4 Components as Event Listeners

- Some people do not like making inner classes to listen to events
- Bad alternative: component affected by event is listener. Eg ButtonPanel listens for button clicks

```
Class ButtonPanel extends JPanel implements
ActionListener {
    . . .
    public void actionPerformed(ActionEvent event)
```

ButtonPanel makes itself the listener for all buttons:

```
yellowButton.addActionListener(this);
blueButton.addActionListener(this);
redButton.addActionListener(this);
```

16

## 4 Components as Event Listeners

- Now ButtonPanel's actionPerformed is called for all 3 buttons
- It must find out which button was clicked:

```
Object source = event.getSource();
If (source == yellowButton) . . .
else if (source == blueButton) . . .
else if (source == redButton) . . .
```

This is no simpler than using inner classes  
(especially when the panel has a lot of components)

17

## 5 Closing Windows

From Java 1.3 the easy way to close windows is:

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

But sometimes you want to capture the window closing event e.g. to ask whether to save changes

Window closing causes JFrame to generate a WindowEvent. Catch it with:

```
WindowListener listener = . . . ; // make listener
Frame.addWindowListener(listener); // register it
```

18

## 5 Closing Windows

The WindowListener interface:

```
public interface WindowListener {
    void windowOpened(WindowEvent e);
    void windowClosing(WindowEvent e);
    void windowClosed(WindowEvent e);
    void windowIconified(WindowEvent e);
    void windowDeiconified(WindowEvent e);
    void windowActivated(WindowEvent e);
    void windowDeactivated(WindowEvent e);
}
```

Any class implementing WindowListener must implement all of it, even though we only want windowClosing

19

## 5 Closing Windows

```
class Terminator implements WindowListener
{
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
    void windowOpened(WindowEvent e) {}
    void windowClosing(WindowEvent e) {}
    void windowClosed(WindowEvent e) {}
    void windowIconified(WindowEvent e) {}
    void windowDeiconified(WindowEvent e) {}
    void windowActivated(WindowEvent e) {}
    void windowDeactivated(WindowEvent e) {}
}
```

20

## 6 Adapter Classes

Adapter classes implement an interface with empty methods

Extending the adapter for an interface means you don't have to write lots of empty methods:

E.g. WindowAdapter is an empty WindowListener

```
class Terminator extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
}
```

21

## 6 Compressing the Code

We register a Terminator object to listen for WindowEvents:

```
WindowListener listener = new Terminator( );
frame.addWindowListener(listener);
```

To save space we can make the Terminator object anonymous:

```
frame.addWindowListener(new Terminator( ));
```

To save more space we can make Terminator an anonymous inner class:

```
frame.addWindowListener(new WindowAdapter( ) {
    public void windowClosing(WindowEvent e) {
        System.exit(0); . . .
    }
});
```

22

## 6 Notes on Adapter Classes

1. We usually can't use adapters if components are listeners because Java does not have multiple inheritance.  
E.g. myJFrame cannot extend both JFrame and an adapter

2. Spell carefully when using adapter classes:  
windowclosing is not the same as windowClosing

If you type windowclosing, you get a new method instead of overriding windowClosing

windowClosing events will still trigger the empty windowClosing method and nothing will happen

23

## 7 AWT Event Listeners and Adapters

- ActionListener
- AdjustmentListener
- ComponentListener
- ContainerListener
- FocusListener
- ItemListener
- KeyListener
- MouseListener
- MouseMotionListener
- MouseWheelListener
- TextListener
- WindowListener
- WindowFocusListener
- WindowStateListener

### AWT Adapter Classes

- ComponentAdapter
- ContainerAdapter
- FocusAdapter
- KeyAdapter
- MouseAdapter
- MouseMotionAdapter
- WindowAdapter

See also javax.swing.event

24

### Further Event-Handling Example

See MouseTest.java for a longer example.

- Handles mouse click events.
- Draws boxes on a JPanel.
- From Core Java vol. 1, example 8-3.

25

### Summary

Event sources

- generate event objects
- send them to the event listeners

Event listeners

- register with event sources to get event objects
- implement listener interfaces
- sometimes extend adapter classes so they do not need to implement the whole interface

26

### Test Yourself

- Anonymous Class
- Adapter Class
- Callback
- Event Listener/Event Handler
- Event Object
- Event Source
- Inner Class
- Listener Interface

27