

## Chapter 7: Events

Programming with Alice and Java  
First Edition

by  
John Lewis  
and  
Peter DePasquale



Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

## Objectives

- Explore the core elements of event processing in Java.
- Learn about the different types of events in Java.
- Explore various components used in a graphical user interface.
- Use listener objects to process Java events.
- See how inner classes can be used effectively to create listeners.

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

7-2

## Event Processing

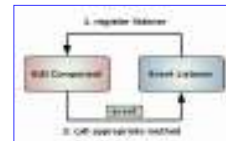
- Event processing is a technique used to create a *graphical user interface* (GUI).
- A GUI *component* is an object that represents a screen element (button, text field, slider, menu).
- *Containers* are components that can hold other components.
- GUI components generate events that represent user actions related to that component.
- A program that is oriented around the GUI, responding to events from the user, is called *event-driven*.

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

7-3

## Event Processing (continued)

- A listener is an object that “waits” for an event to occur and responds when it does occur.
- A relationship between the listener and the component that generates an event should be established.



Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

7-4

## Event Processing (continued)

- Many components and events are predefined by classes in the Java class library.
- A programmer can write her own listener classes to perform any action she desires when events occur.
- To create a Java program that uses GUI, a programmer must:
  - create and set up the necessary components;
  - implement listener classes that define what happens when a particular event occurs;
  - establish the relationship between the listeners and the components that generate the events of interest.

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

7-5

## Buttons



The PushCounter class

```
1 // PushCounter.java: A simple program that demonstrates the use of the PushCounter class.
2 // The PushCounter class is a subclass of JComponent and implements the ActionListener interface.
3 // The PushCounter class has a single method, actionPerformed, which is called when the button is pushed.
4 // The PushCounter class also has a single attribute, count, which is used to keep track of the number of times the button has been pushed.
5
6 import javax.swing.*;
7 import java.awt.*;
8 import java.awt.event.*;
9
10 public class PushCounter {
11     // The count attribute.
12     private int count = 0;
13
14     // The constructor.
15     public PushCounter() {
16         // Create the button.
17         JButton button = new JButton("Push");
18
19         // Add the button to the window.
20         JFrame frame = new JFrame("PushCounter");
21         frame.getContentPane().add(button);
22         frame.setSize(200, 100);
23         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24         frame.setVisible(true);
25     }
26
27     // The actionPerformed method.
28     public void actionPerformed(ActionEvent e) {
29         // Increment the count.
30         count++;
31
32         // Display the count.
33         JOptionPane.showMessageDialog(frame, "Count: " + count);
34     }
35 }
```

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

7-6

## Buttons (continued)

- A **label** can be used to display a line of text or an image.
- A **push** button allows the user to initiate an action with the press of a button.
- A **JButton** is a class that generates this action event.
- Several events are defined in the Java standard library.

```

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

public class Buttons {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Buttons");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 100);

        JLabel label = new JLabel("Click the button");
        JButton button = new JButton("Click Me");

        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                label.setText("Button clicked!");
            }
        });

        frame.add(label);
        frame.add(button);
        frame.setVisible(true);
    }
}
    
```

## Action Events

- To respond to the event a listener object is needed.
- The Listener class represents the action listener.

The **actionPerformed** method is called when the event occurred and **ActionEvent** object is passed to that method

```

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

public class ActionEvents {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Action Events");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 100);

        JButton button = new JButton("Click Me");

        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(frame, "Button clicked!");
            }
        });

        frame.add(button);
        frame.setVisible(true);
    }
}
    
```

## Event Types

Event	What it Indicates	Listener Interface
<b>ActionEvent</b>	A button was pushed, enter was pressed in a text field, or a menu item was selected.	<b>ActionListener</b>
<b>ChangeEvent</b>	An object's state changed in some way.	<b>ChangeListener</b>
<b>ComponentEvent</b>	A component was hidden, moved, resized, or shown.	<b>ComponentListener</b>
<b>ContainerEvent</b>	A component was added or removed from a container.	<b>ContainerListener</b>
<b>FocusEvent</b>	A component gained or lost the keyboard focus.	<b>FocusListener</b>
<b>ItemEvent</b>	The state of selectable item (check box, menu) changed.	<b>ItemListener</b>
<b>KeyEvent</b>	A Keyboard key was pressed.	<b>KeyListener</b>
<b>MouseEvent</b>	The mouse interacted with a component (mouse button push, rollover, drag).	<b>MouseListener</b> <b>MouseMotionListener</b>
<b>WindowEvent</b>	The application window was opened, closed, iconified, maximized, minimized, etc.	<b>WindowFocusListener</b> <b>WindowStateListener</b>

## Another Example



- Controls include a **text box**, **check boxes** and a set of **radio buttons**.
- Instead of using a separate public class for each listener, the public class contains private classes that define listeners.
- These classes are called **inner classes**.
- Only outer classes can make use of the inner class.

```

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

public class AnotherExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Another Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 100);

        JTextField textField = new JTextField(20);
        JRadioButton radio1 = new JRadioButton("Radio 1");
        JRadioButton radio2 = new JRadioButton("Radio 2");
        JButton button = new JButton("Click Me");

        radio1.setSelected(true);
        radio1.setGroup(radio2);

        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(frame, "Button clicked!");
            }
        });

        frame.add(textField);
        frame.add(radio1);
        frame.add(radio2);
        frame.add(button);
        frame.setVisible(true);
    }
}
    
```

## Listener Examples

```

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

public class ListenerExamples {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Listener Examples");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 100);

        JTextField textField = new JTextField(20);
        JCheckBox checkBox = new JCheckBox("Check Box");

        textField.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(frame, "Text field clicked!");
            }
        });

        checkBox.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(frame, "Check box clicked!");
            }
        });

        frame.add(textField);
        frame.add(checkBox);
        frame.setVisible(true);
    }
}
    
```

A text field listener

A check box listener

```

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

public class ListenerExamples {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Listener Examples");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 100);

        JTextField textField = new JTextField(20);
        JCheckBox checkBox = new JCheckBox("Check Box");

        textField.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(frame, "Text field clicked!");
            }
        });

        checkBox.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(frame, "Check box clicked!");
            }
        });

        frame.add(textField);
        frame.add(checkBox);
        frame.setVisible(true);
    }
}
    
```

## Mouse Events

- A mouse event is generated when the mouse interacts with a GUI component.
- Components can generate mouse events that indicate that:

- a mouse button was pressed;
- a mouse button was released;
- a mouse button was clicked;
- the mouse cursor moved over a component;
- the mouse cursor moved off a component;
- the mouse was moved;
- the mouse was dragged.

MouseListener interface

MouseMotionListener interface

## Keyboard Events

- A **keyboard event** (or **key event**) occurs when a keyboard key is pressed.
- Key events allow a program to respond immediately as the user presses keys.
- A listener in Java responds when **any** key is pressed, then decides what to do based on the specific key pressed.



In the **ImageFlicker** program three different images appear in the image frame when user presses 1, 2, or 3. When space bar is pressed one of the images is picked at random.

## Example

### The main method

```
// The main method
// This program displays a panel with three images.
// The user can click on the images to change them.
// The user can also press the space bar to change a random image.
// The user can press the 'q' key to quit the program.

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

public class ImageFlicker {
    public static void main(String[] args) {
        // Create the panel and display it.
        ImageFlickerPanel panel = new ImageFlickerPanel();
        JFrame frame = new JFrame("Image Flicker");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(panel);
        frame.pack();
        frame.setVisible(true);
    }
}
```

Display panel for the program

## Example (continued)

```
// The ImageFlickerPanel class
// This panel displays three images.
// The user can click on the images to change them.
// The user can also press the space bar to change a random image.
// The user can press the 'q' key to quit the program.

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

public class ImageFlickerPanel extends JPanel {
    // The three images to be displayed.
    private Image img1, img2, img3;
    // The index of the current image.
    private int currentImage = 0;

    // Constructor.
    public ImageFlickerPanel() {
        // Set the background color.
        setBackground(Color.WHITE);
        // Set the layout manager.
        setLayout(new BorderLayout());
        // Add the three images.
        add(img1, BorderLayout.CENTER);
        add(img2, BorderLayout.EAST);
        add(img3, BorderLayout.WEST);
    }

    // Method to change the current image.
    public void changeImage() {
        // Increment the current image index.
        currentImage++;
        // If the current image index is 3, reset it to 0.
        if (currentImage == 3) {
            currentImage = 0;
        }
        // Repaint the panel.
        repaint();
    }

    // Method to change a random image.
    public void randomImage() {
        // Generate a random number between 0 and 2.
        int randomIndex = (int) (Math.random() * 3);
        // Change the current image to the random index.
        currentImage = randomIndex;
        // Repaint the panel.
        repaint();
    }

    // Method to quit the program.
    public void quit() {
        System.exit(0);
    }
}
```

The component that generates key events is the one that currently has the **keyboard focus**. This call to the **setFocusable** method sets the keyboard focus to the panel.

## Example (continued)

```
// The ActionListener interface
// This interface defines the methods that must be implemented by any class that implements the ActionListener interface.

import java.awt.event.*;

public interface ActionListener {
    // Method to be called when an action occurs.
    void actionPerformed(ActionEvent e);
}
```

The **KeyListener** interface

## Summary

- A GUI is made up of components, events that represent user actions, and listeners that respond to those events.
- A listener can be created by implementing an appropriate listener interface.
- Listeners are often defined as inner classes because of the intimate relationship between the listener and the GUI components.
- Radio buttons operate as a group, providing a set of mutually exclusive options.
- Java mouse events are separated into two categories with two listener interfaces.
- A listener may have to provide empty method definitions for unused events to satisfy the interface.
- To generate a keyboard event, a component must have the keyboard focus.