Graphical User Interface Programming

Part 2 – Event Handling – 1

Chapter 10, Core Java, Volume I

Contents

Basics of Event Handling

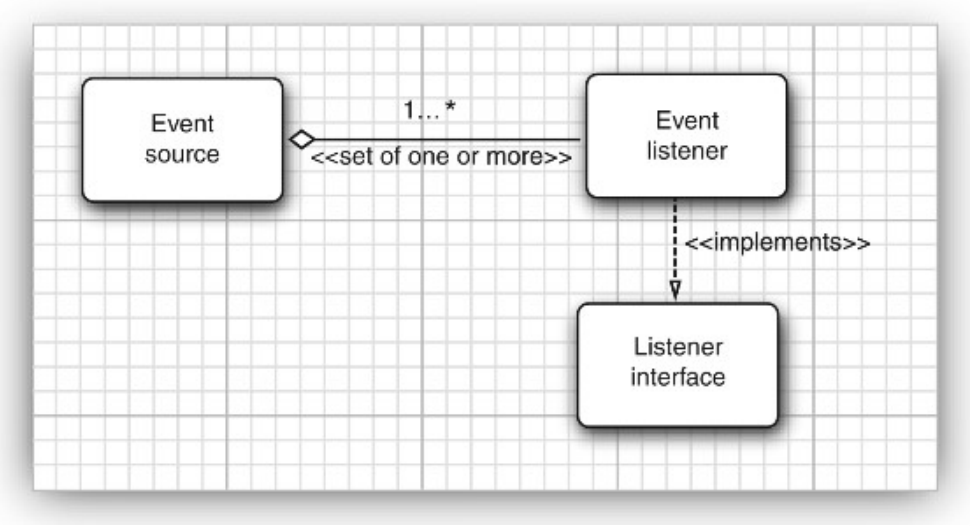
- Event source (button, window, ...) produces events.
- Event object (ActionEvent, WindowEvent, ...) carries information about an event.
- Event listener contains code that reacts to events.
- Listener object implements appropriate listener interface:

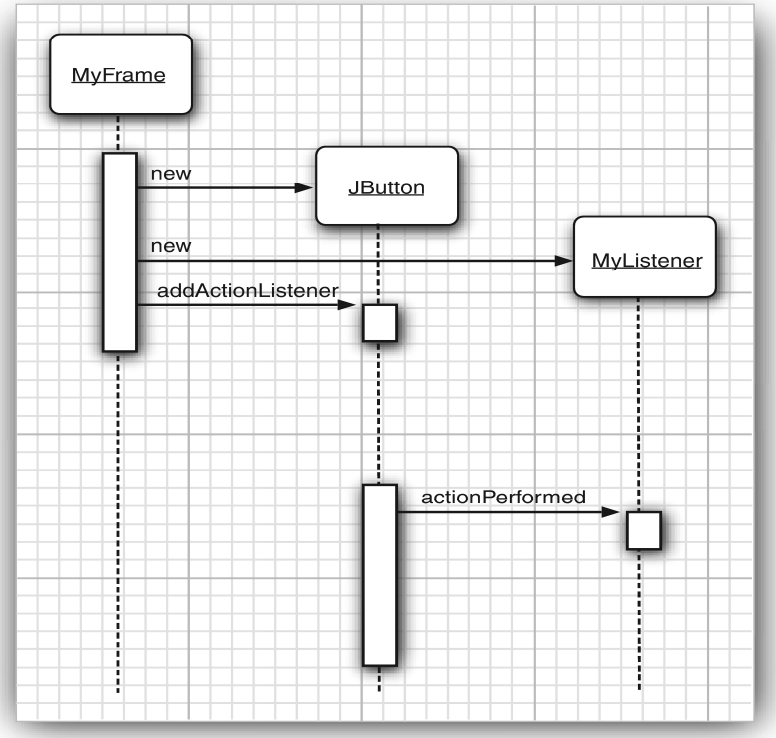
```
class MyListener implements ActionListener
{
   public void actionPerformed(ActionEvent event)
   {
      reaction to button click
   }
}
```

- Add event listener to event source:

```
ActionListener listener = new MyListener();
JButton button = new JButton("OK");
button.addActionListener(listener);
```

Basics of Event Handling





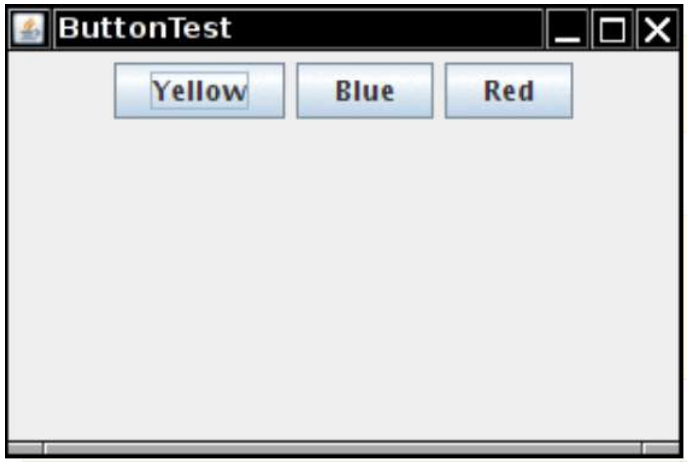
Handling Button Clicks

- Create a JButton object:

```
JButton yellowButton = new JButton("Yellow");
JButton blueButton = new JButton(new ImageIcon("blue-ball.gif"));
```

- Add the buttons to a JPanel, and add the panel to the frame:

```
JPanel buttonPanel = new JPanel();
buttonPanel.add(yellowButton);
buttonPanel.add(blueButton);
frame.add(buttonPanel);
```



- Add a listener to each button:

```
class ColorAction implements ActionListener { . . . }
. . .
yellowButton.addActionListener(new ColorAction(Color.YELLOW));
blueButton.addActionListener(new ColorAction(Color.BLUE));
```

Example: ButtonFrame (Listing 10.5)

```
package button;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ButtonFrame extends JFrame
{
   private JPanel buttonPanel;
   private static final int DEFAULT_WIDTH = 300;
   private static final int DEFAULT_HEIGHT = 200;
 public ButtonFrame()
  {
     setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);

     // create buttons
     JButton yellowButton = new JButton("Yellow");
     JButton blueButton = new JButton("Blue");
     JButton redButton = new JButton("Red");
     buttonPanel = new JPanel();
     // add buttons to panel
     buttonPanel.add(yellowButton);
     buttonPanel.add(blueButton);
     buttonPanel.add(redButton);
     // add panel to frame
     add(buttonPanel);
     // create button actions
     ColorAction yellowAction = new ColorAction(Color.YELLOW);
     ColorAction blueAction = new ColorAction(Color.BLUE);
     ColorAction redAction = new ColorAction(Color.RED);
    // associate actions with buttons
     yellowButton.addActionListener(yellowAction);
     blueButton.addActionListener(blueAction);
     redButton.addActionListener(redAction);
}
```

Example: ButtonFrame (Listing 10.5)

```
  // An action listener that sets the panel's background color.
  // inner class
  private class ColorAction implements ActionListener
  {
    private Color backgroundColor;
    public ColorAction(Color c)
    {
      backgroundColor = c;
    }
    public void actionPerformed(ActionEvent event)
    {
      buttonPanel.setBackground(backgroundColor);
    }
  } // end of ColorAction class
}
```

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

public class ButtonTest
{
  public static void main(String[] args)
  {
    EventQueue.invokeLater(() -> {
      JFrame frame = new ButtonFrame();
      frame.setTitle("ButtonTest");
      frame.setDefaultCloseOperation
                (JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
    });
  }
}
```

Specifying Listeners Concisely

- Use lambda expressions:

```
exitButton.addActionListener(event -> System.exit(0));
```

- If you need actions with parameters, use a helper method:

```
public void makeButton(String name, Color backgroundColor)
{
   JButton button = new JButton(name);
   buttonPanel.add(button);
   button.addActionListener(
      event -> buttonPanel.setBackground(backgroundColor) );
}
```

- Then call:

```
makeButton("yellow", Color.YELLOW);
makeButton("blue", Color.BLUE);
```

- Older code used anonymous inner classes:

```
exitButton.addActionListener( new ActionListener()
   {
      public void actionPerformed(ActionEvent event) { System.exit(0); }
   } );
```

Adapter Classes

- Windows send out events in multiple situations:
  - When the window is opened or closed
  - When a window is minimized or restored
  - When a window becomes inactive or active
  - When the user wants to close the window
- WindowListener interface has a separate method for each of them:

```
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);
}
```

- Suppose you just want to intercept `windowClosing`.

Adapter Classes

- First way: Define a class that implements WindowListener interface.

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

  - Then, install listener:

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

Adapter Classes

- Second way: Extend WindowAdapter instead of implementing WindowListener:

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

- Third way: Use anonymous inner class

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