Managing Input Events in Swing

Week 5 Workshop

8.02.2008

Lyn Bartram

SCHOOL OF INTERACTIVE ARTS + TECHNOLOGY

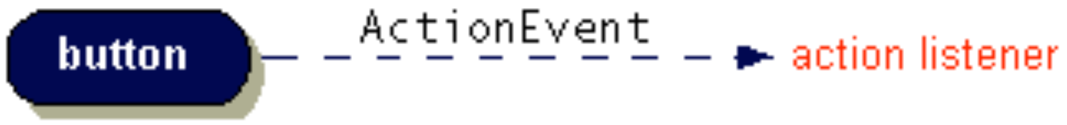
Today

- Introduction
- Java’s event delegation model – event sources and event listeners
- Event classes
- Examples
  - Window events
  - Adding simple buttons

  - a mouse tracker

Events Handling

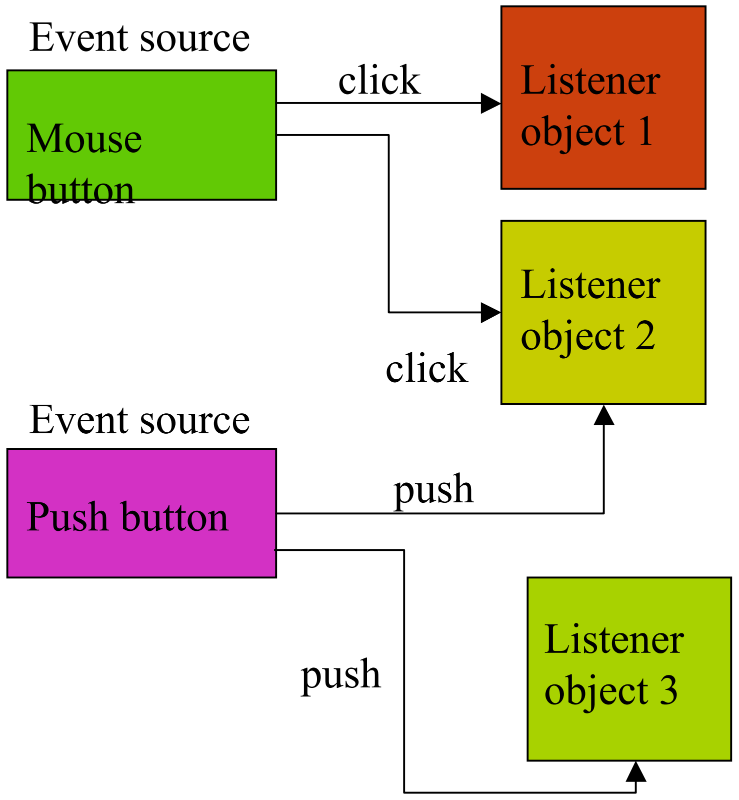
- Every time a user types a character or pushes a mouse button, an ***event*** occurs.
- Any object can be notified of an event by registering as an ***event listener*** on the appropriate ***event source***.
- Multiple listeners can register to be notified of events of a particular type from a particular source.



SCHOOL OF INTERACTIVE ARTS + TECHNOLOGY

Java’s event delegation model – event sources and event listeners

- Java allows objects to be designated *event listeners* which can listen for *specific* types of events (for example a mouse button click)
  - Event listeners are *registered* with the particular *event sources* whose events they handle
  - One object can be a listener for several sources

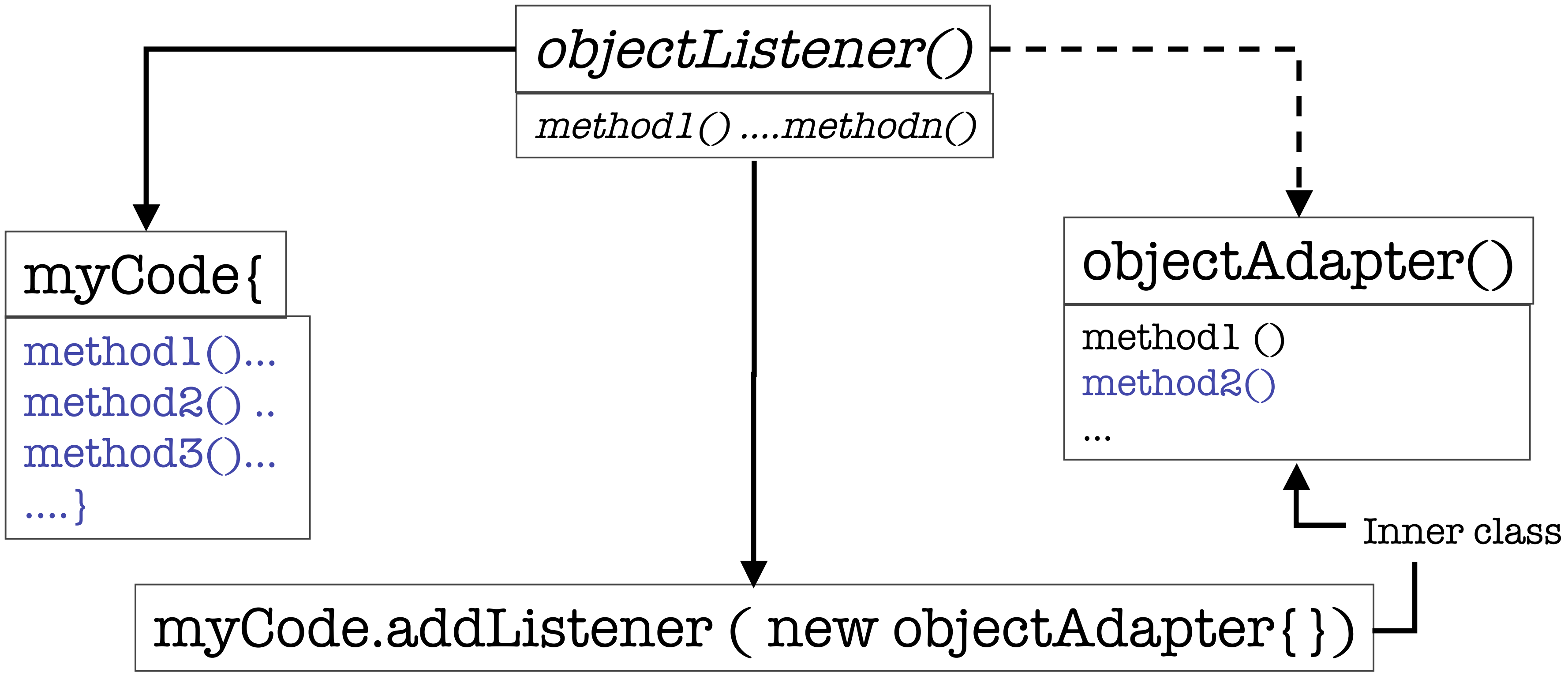


Implementing an Event Handler

- Implement a listener interface or extend a class that implements a listener interface.
- Register an instance of the event handler class as a listener upon one or more components.
- Implement the methods in the listener interface to handle the event.

- In terms of Java objects and methods, event handling works as follows
  - An event source registers all listener objects
  - The event source sends out *event objects* to all registered listener objects
  - Each listener object uses information encapsulated in the event object to call the appropriate listener method
  - Listener objects implement the appropriate listener *interface*
    - Not a UI!
    - Have to implement all the functions in the interface, OR
    - Use an Adapter object (with an inner class)

Adding a listener



Types of Event Listeners

Act that results in event	Listener type
User clicks a button, presses Return while typing in a text field, or chooses a menu item	ActionListener
User closes a frame (main window)	WindowListener
User presses a mouse button while the cursor is over a component	MouseListener
User moves the mouse over a component	MouseMotionListener
Component becomes visible	ComponentListener
Component gets the keyboard focus	FocusListener
Table or list selection changes	ListSelectionListener

Event classes

- Event classes are arranged in an inheritance tree with the base class being *EventObject*
- Event classes are in the package *java.awt.event*
- Event objects encapsulate information about the event such as the event source
- Each event class has a corresponding event listener class

Example 1: Simple window events

- Create a simple Frame

```
public static void main(String[ ]  args)  {
 {
  FrameExample1 app = new FrameExample1();
  app.setSize(400,300);
  app.setVisible(true);
  app.addWindowListener(         //   Start of an inner class
   new WindowAdapter( )
    {
      public void windowClosing( WindowEvent e )
      {
       System.exit(0);     // Stop the program upon window closing
      }
    }
   ); //   Note the   );   <- Note especially; this is correct!  Because of the inner class
 }
```

SCHOOL OF INTERACTIVE ARTS + TECHNOLOGY

Example 2: Adding Buttons

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class SomeButtons extends JFrame implements ActionListener
 {
          private JButton jbt1, jbt2, jbt3, jbt4;
          public static void main(String[ ]  args)
  {
          ButtonFrame frame = new ButtonFrame ( )
          frame.pack( );
          frame.setTitle("Some Buttons");
          frame.setVisible(true);
   }  // End of main method
```

Adding buttons

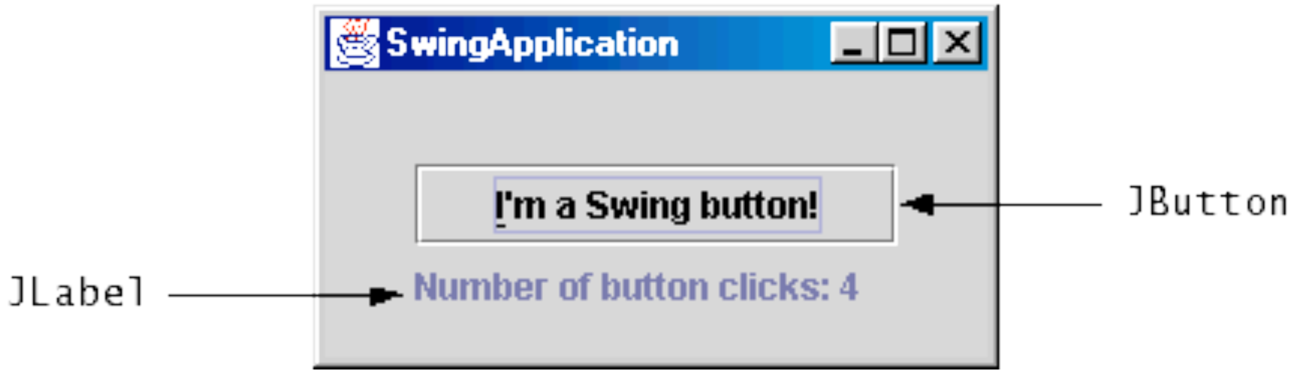
```
public ButtonFrame ( ) {
   JPanel p1 = new JPanel(); // Create panel p1,  add 2 buttons
   p1.setLayout (new FlowLayout( ) );
   p1.add(jbt1 = new JButton("Button 1"));
   p1.add(jbt2 = new JButton("Button 2"));

   JPanel p2 = new JPanel( ); // Create panel p2;  add 2 more buttons
   p2.setLayout(new FlowLayout());
   p2.add(jbt3 = new JButton("Button 3"));
   p2.add(jbt4 = new JButton("Button 4"));
```

Class/object can itself be a Listener

```
// Place panels p1 and p2 into the frame of class ButtonFrame
   getContentPane().setLayout(new FlowLayout());
   getContentPane().add(p1);
   getContentPane().add(p2);

   jbt1.addActionListener(this);   // Register listeners for the 4 buttons
   jbt2.addActionListener(this);
   jbt3.addActionListener(this);
   jbt4.addActionListener(this);

public void actionPerformed(ActionEvent  e)
  {     System.out.println(e.getActionCommand() + " was clicked");  }

} // End of class SomeButtons
```

SCHOOL OF INTERACTIVE ARTS + TECHNOLOGY

Example 2b: simple button listener



```
button.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    numClicks++;
    label.setText(labelPrefix + numClicks);
  }});
```

- The following example shows a simple user interface to select the background colour
  - Start up a simple program with a JFrame
  - Class *ButtonPanel* is the panel containing the push buttons and the event handling (key parts emboldened)

```
class ButtonPanel extends JPanel   implements ActionListener
{
    public ButtonPanel()
    {
        // Create buttons and add listeners
    }

    public void actionPerformed(ActionEvent evt)
    {
        // Handle button press events
    }

    private JButton yellowButton;
    private JButton blueButton;
    private JButton redButton;
}
```

```
public ButtonPanel()
{
      yellowButton = new JButton("Yellow");
      blueButton = new JButton("Blue");
      redButton = new JButton("Red");

      add(yellowButton);
      add(blueButton);
      add(redButton);

      yellowButton.addActionListener(this);
      blueButton.addActionListener(this);
      redButton.addActionListener(this);
   }
public void actionPerformed(ActionEvent evt)
{
      Object source = evt.getSource();
      Color color = getBackground();
      if (source == yellowButton) color = Color.yellow;
      else if (source == blueButton) color = Color.blue;
    else if (source == redButton) color = Color.red;
      setBackground(color);
      repaint();
}
```

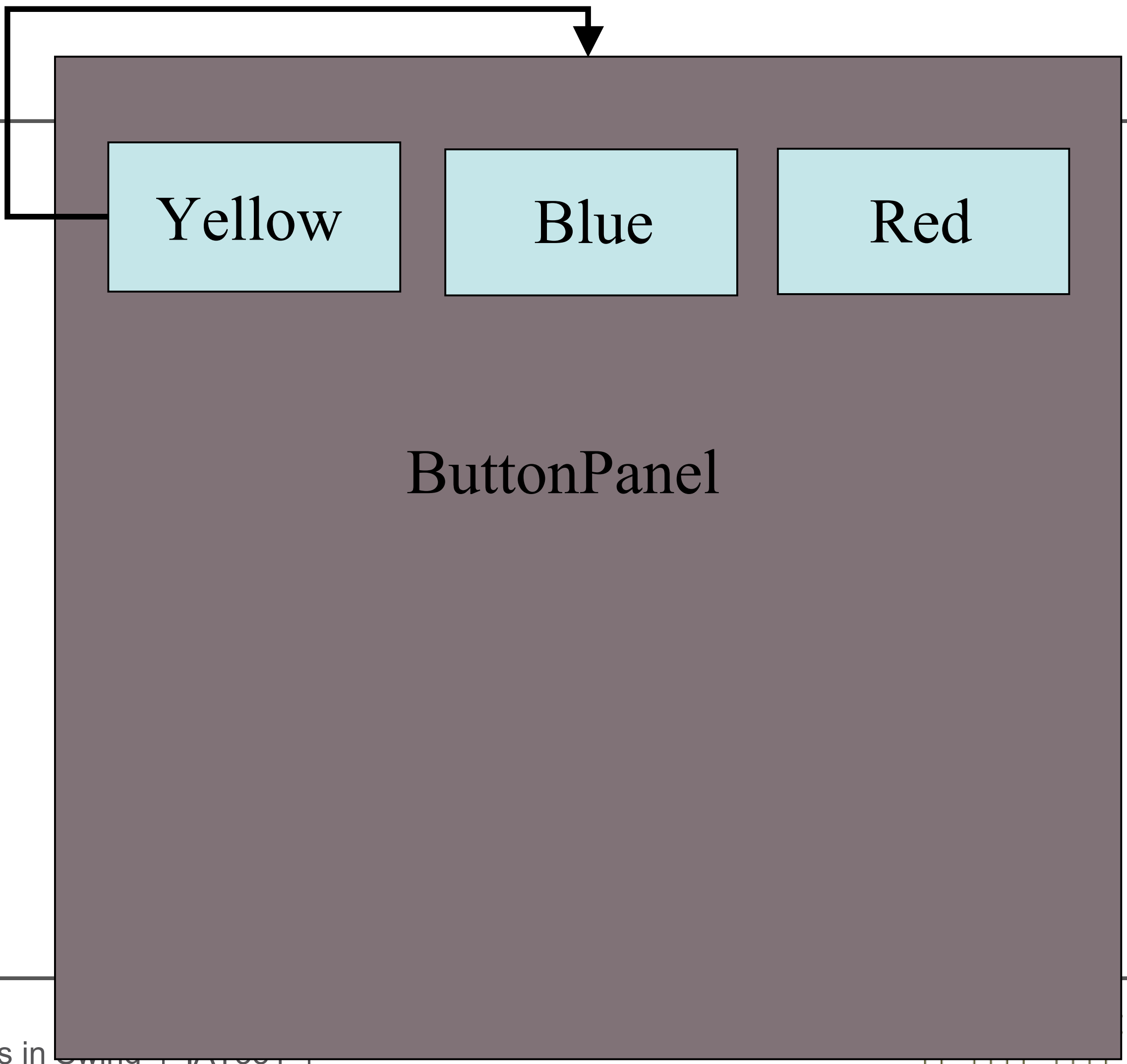
Check it out

- It should look like this ….

http://www.eee.bham.ac.uk/spannm/Java%20Stuff/ButtonTestApplet/ButtonTestApplet.html

- `class ButtonPanel extends JPanel   implements ActionListener`
  - The panel object implements the *ActionListener* interface and an implementation of the method *ActionPerformed(),* which is the event handling method which must be provided
- `yellowButton.addActionListener(this);`
  - The *JButton* object *yellowButton* registers the *ButtonPanel* object as a listener for button presses

yellowButton.addActionListener(this)



- `ButtonPanel.actionPerformed(ActionEvent evt)` is called automatically when one of the buttons is pressed
  - *evt* is an *ActionEvent* object which can be used to determine which of the buttons was pressed

- `Object source = evt.getSource();`
  - This returns the object which was the source of the event
  - *Object* is the super class so an object of any class can be assigned to it

- We have already seen two examples of events and corresponding listeners
  - *ActionEvent* with listener *ActionListener* generated by (amongst other things) a button press
  - *WindowEvent* with listener *WindowListener* generated when a user tries to close a window
- Events are also generated by keyboard presses and mouse drags and clicks which are handled by appropriate listeners
- Some events (such as a *PaintEvent*) are generated automatically when a window is moved/resized so that it is repainted

Example 3 – a mouse tracker

- A mouse tracker program keeps track of the motion of the mouse and mouse clicks
- Uses event listeners
  - *MouseListener*
    - Listens for mouse button clicks
  - *MouseMotionListener*
    - Listens for mouse moves and drags
- We need to implement the following methods in the listener interfaces

Tracking mouse events

- MouseListener interface
  - Methods :
    - mousePressed
    - mouseReleased
    - mouseEntered
    - mouseExited
    - mouseClicked
- MouseMotionListener
  - Methods :
    - mouseDragged
    - mouseMoved

http://www.eee.bham.ac.uk/spannm/Java%20Stuff/MouseTrackerApplet/MouseTrackerApplet.html

- sample applet
- The implementation of the event handlers is straighforward
  - Uses `event.getX()` and `event.getY()` to determine the mouse position
  - `mouseEntered()` puts up a dialog box (see later) so that the user can select when ready to track

```
public class MouseTrackerApplet extends JApplet implements MouseListener, MouseMotionListener
{
      public MouseTrackerApplet()
      {
        getContentPane().add(new Jlabel(), BorderLayout.SOUTH);
        addMouseListener(this);
        addMouseMotionListener(this);
      }

      public void mouseClicked(MouseEvent event) {..}
      public void mousePressed(MouseEvent event) {..}
      public void mouseReleased(MouseEvent event) {..}
      public void mouseEntered(MouseEvent event) {..}
      public void mouseExited(MouseEvent event) {..}
      public void mouseDragged(MouseEvent event) {..}
      public void mouseMoved(MouseEvent event) {..}
      .
      .
}
```

```
public void mouseClicked(MouseEvent event)
{
     statusBar.setText("Clicked at [" + event.getX() + ", " +
                              event.getY() + "]");
     // could be newx=event.getX(); and then redraw x in paintComponent()
}
public void mouseEntered(MouseEvent event)
{
     if (!entered)
   {
     JOptionPane.showMessageDialog(null,"Mouse in window");
     entered=true;
     }
}
```

Handling Mouse Events   Painter.java

- The next example uses the `MouseDragged` event handler to create a simple drawing program.
- The user can draw pictures with the mouse by dragging the mouse on the background of the window.

- Since the method mousemoved is not used in the Painter.java program, the `MouseMotionListener` is defined as a subclass of `MouseMotionAdapter`.
- Since `MouseMotionAdapter` defines `mouseMoved` and `mouseDragged`, we can override the `mouseDragged` method to provide the functionality for the drawing program.

Handling Mouse Events   Painter.java

```
//  Painter.java
// Using class MouseMotionAdapter.
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

public class Painter extends JFrame {
  private int   xValue = -10, yValue = -10;
```

Handling Mouse Events   Painter.java

```
public Painter( )  {     super( "A simple paint program" );
    getContentPane().add( new Label( "Drag the mouse to draw" ),
     BorderLayout.SOUTH );
    addMouseMotionListener(      // Register mouse motion listener
      new MouseMotionAdapter( ) {
        public void mouseDragged( MouseEvent e )
        {     //  An anonymous inner class that extends class MouseMotionListener
          xValue = e.getX( );
          yValue = e.getY( );
          repaint( );              // Initiate drawing of the next oval on the background
        }
      }
    );   // end of inner class
```

The anonymous inner class inherits a default implementation of both mouseMoved() and mouseDragged()

Handling Mouse Events   Painter.java

```
setSize( 300, 150 );    // Set the window size
    show( );                 // Display the window
  }
  public void paint( Graphics g )  // Use Graphics class
  {
    g.fillOval( xValue, yValue, 4, 4 );    // Draw an oval
  }
```

Handling Mouse Events   Painter.java

```
public static void main( String[ ]  args )
  {
    Painter app = new Painter( );   // Create a new instance of Painter class
    app.addWindowListener(       // Register a window  listener  (start of  inner
     class)
      new WindowAdapter( ) {
        public void windowClosing( WindowEvent  e )
        {   The program stops when the user clicks the  [X]  in upper-right corner
          System.exit( 0 );    // Halt program on  window closing
        }
      }
    );    // end of inner class
  }
}
```

Building GUI’s

- Swing has a large number of classes for GUI components
  - Text input
    - JTextField
  - Labels
    - JLabel
  - Buttons
    - JButton
  - Check boxes (for choosing options)
    - JCheckBox

Swing Input components (just a sample)

- Radio buttons (for choosing 1 from several options)
  - JRadioButton
- Lists
  - JList
- Drop down boxes (combo boxes)
  - JComboBox
- Scroll bars
  - JScrollBar
- Menus (a bit more involved)
  - *JMenuBar*, *JMenu*, *JMenuItem*
- Diaog boxes (quite a bit more involved!)
  - *JOptionPane*
- File chooser dialog box (very useful!)
  - *JFileChooser*