Handling Mouse and Keyboard Events
Agenda

- General event-handling strategy
- Handling events with separate listeners
- Handling events by implementing interfaces
- Handling events with named inner classes
- Handling events with anonymous inner classes
- The standard AWT listener types
- Subtleties with mouse events
- Examples
General Strategy

• **Determine what type of listener is of interest**
  – 11 standard AWT listener types, described on later slide.
    • ActionListener, AdjustmentListener, ComponentListener, ContainerListener, FocusListener, ItemListener, KeyListener, MouseListener, MouseMotionListener, TextListener, WindowListener

• **Define a class of that type**
  – Implement interface (KeyListener, MouseListener, etc.)
  – Extend class (KeyAdapter, MouseAdapter, etc.)

• **Register an object of your listener class with the window**
  – `w.addXxxListener(new MyListenerClass());`
    • E.g., addKeyListener, addMouseListener
Handling Events with a Separate Listener: Simple Case

- Listener does not need to call any methods of the window to which it is attached

```java
import java.applet.Applet;
import java.awt.*;

public class ClickReporter extends Applet {
    public void init() {
        setBackground(Color.yellow);
        addMouseListener(new ClickListener());
    }
}
```
import java.awt.event.*;

public class ClickListener extends MouseAdapter {
    public void mousePressed(MouseEvent event) {
        System.out.println("Mouse pressed at (" +
                          event.getX() + "," +
                          event.getY() + ").");
    }

    }

Separate Listener: Simple Case (Continued)
Generalizing Simple Case

• What if ClickListener wants to draw a circle wherever mouse is clicked?
• Why can’t it just call getGraphics to get a Graphics object with which to draw?
• General solution:
  – Call event.getSource to obtain a reference to window or GUI component from which event originated
  – Cast result to type of interest
  – Call methods on that reference
import java.applet.Applet;
import java.awt.*;

public class CircleDrawer1 extends Applet {
    public void init() {
        setForeground(Color.blue);
        addMouseListener(new CircleListener());
    }
}
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class CircleListener extends MouseAdapter {
    private int radius = 25;

    public void mousePressed(MouseEvent event) {
        Applet app = (Applet)event.getSource();
        Graphics g = app.getGraphics();
        g.fillOval(event.getX()-radius,
                    event.getY()-radius,
                    2*radius,
                    2*radius);
    }
}
Separate Listener: General Case (Results)

Click in the applet to draw circles.
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class CircleDrawer2 extends Applet
    implements MouseListener {
    private int radius = 25;

    public void init() {
        setForeground(Color.blue);
        addMouseListener(this);
    }
}
public void mouseEntered(MouseEvent event) {}
public void mouseExited(MouseEvent event) {}
public void mouseReleased(MouseEvent event) {}
public void mouseClicked(MouseEvent event) {}

public void mousePressed(MouseEvent event) {
    Graphics g = getGraphics();
g.fillOval(event.getX()-radius,
    event.getY()-radius,
    2*radius,
    2*radius);
}
}
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class CircleDrawer3 extends Applet {
    public void init() {
        setForeground(Color.blue);
        addMouseListener(new CircleListener());
    }
}
Named Inner Classes (Continued)

- Note: still part of class from previous slide

```java
private class CircleListener
    extends MouseAdapter {
    private int radius = 25;

    public void mousePressed(MouseEvent event) {
        Graphics g = getGraphics();
        g.fillOval(event.getX()-radius,
                    event.getY()-radius,
                    2*radius,
                    2*radius);
    }
}
```
public class CircleDrawer4 extends Applet {
    public void init() {
        setForeground(Color.blue);
        addMouseListener
            (new MouseAdapter() {
                private int radius = 25;

                public void mousePressed(MouseEvent event) {
                    Graphics g = getGraphics();
                    g.fillOval(event.getX()-radius, 
                                event.getY()-radius, 
                                2*radius, 
                                2*radius);
                }
            });
    }
}
Event Handling Strategies: Pros and Cons

• **Separate Listener**
  – Advantages
    • Can extend adapter and thus ignore unused methods
    • Separate class easier to manage
  – Disadvantage
    • Need extra step to call methods in main window

• **Main window that implements interface**
  – Advantage
    • No extra steps needed to call methods in main window
  – Disadvantage
    • Must implement methods you might not care about
Event Handling Strategies: Pros and Cons (Continued)

- **Named inner class**
  - Advantages
    - Can extend adapter and thus ignore unused methods
    - No extra steps needed to call methods in main window
  - Disadvantage
    - A bit harder to understand

- **Anonymous inner class**
  - Advantages
    - Same as named inner classes
    - Even shorter
  - Disadvantage
    - Much harder to understand
### Standard AWT Event Listeners (Summary)

<table>
<thead>
<tr>
<th>Listener</th>
<th>Adapter Class (If Any)</th>
<th>Registration Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionListener</td>
<td></td>
<td>addActionListener</td>
</tr>
<tr>
<td>AdjustmentListener</td>
<td></td>
<td>addAdjustmentListener</td>
</tr>
<tr>
<td>ComponentListener</td>
<td>ComponentAdapter</td>
<td>addComponentListener</td>
</tr>
<tr>
<td>ContainerListener</td>
<td>ContainerAdapter</td>
<td>addMouseListener</td>
</tr>
<tr>
<td>FocusListener</td>
<td>FocusAdapter</td>
<td>addMouseListener</td>
</tr>
<tr>
<td>ItemListener</td>
<td></td>
<td>addMouseListener</td>
</tr>
<tr>
<td>KeyListener</td>
<td>KeyAdapter</td>
<td>addMouseListener</td>
</tr>
<tr>
<td>MouseListener</td>
<td>MouseAdapter</td>
<td>addMouseListener</td>
</tr>
<tr>
<td>MouseMotionListener</td>
<td>MouseMotionAdapter</td>
<td>addMouseListener</td>
</tr>
<tr>
<td>TextListener</td>
<td></td>
<td>addMouseListener</td>
</tr>
<tr>
<td>WindowListener</td>
<td>WindowAdapter</td>
<td>addMouseListener</td>
</tr>
</tbody>
</table>
Standard AWT Event Listeners (Details)

• **ActionListener**
  – Handles buttons and a few other actions
    • `actionPerformed(ActionEvent event)`

• **AdjustmentListener**
  – Applies to scrolling
    • `adjustmentValueChanged(AdjustmentEvent event)`

• **ComponentListener**
  – Handles moving/resizing/hiding GUI objects
    • `componentResized(ComponentEvent event)`
    • `componentMoved (ComponentEvent event)`
    • `componentShown(ComponentEvent event)`
    • `componentHidden(ComponentEvent event)`
Standard AWT Event Listeners (Details Continued)

• **ContainerListener**
  – Triggered when window adds/removes GUI controls
    • `componentAdded(ContainerEvent event)`
    • `componentRemoved(ContainerEvent event)`

• **FocusListener**
  – Detects when controls get/lose keyboard focus
    • `focusGained(FocusEvent event)`
    • `focusLost(FocusEvent event)`
Standard AWT Event Listeners (Details Continued)

- **ItemListerner**
  - Handles selections in lists, checkboxes, etc.
    - `itemStateChanged(ItemEvent event)`

- **KeyListener**
  - Detects keyboard events
    - `keyPressed(KeyEvent event) -- any key pressed down`
    - `keyReleased(KeyEvent event) -- any key released`
    - `keyTyped(KeyEvent event) -- key for printable char released`
Standard AWT Event Listeners (Details Continued)

- **MouseListener**
  - Applies to basic mouse events
    - `mouseEntered(MouseEvent event)`
    - `mouseExited(MouseEvent event)`
    - `mousePressed(MouseEvent event)`
    - `mouseReleased(MouseEvent event)`
    - `mouseClicked(MouseEvent event)` — Release without drag
  - Applies on release if no movement since press

- **MouseMotionListener**
  - Handles mouse movement
    - `mouseMoved(MouseEvent event)`
    - `mouseDragged(MouseEvent event)`
Standard AWT Event Listeners (Details Continued)

- **TextListener**
  - Applies to textfields and text areas
    - `textValueChanged(TextEvent event)`

- **WindowListener**
  - Handles high-level window events
    - `windowOpened, windowClosing, windowClosed, windowIconified, windowDeiconified, windowActivated, windowDeactivated`
      - `windowClosing` particularly useful
Mouse Events: Details

- **MouseListener and MouseMotionListener** share event types
- **Location of clicks**
  - event.getX() and event.getY()
- **Double clicks**
  - Determined by OS, not by programmer
  - Call event.getClickCount()
- **Distinguishing mouse buttons**
  - Call event.getModifiers() and compare to 
    MouseEvent.Button2_MASK for a middle click and 
    MouseEvent.Button3_MASK for right click.
  - Can also trap Shift-click, Alt-click, etc.
Simple Example: Spelling-Correcting Textfield

- KeyListener corrects spelling during typing
- ActionListener completes word on ENTER
- FocusListener gives subliminal hints
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class SimpleWhiteboard extends Applet {
    protected int lastX=0, lastY=0;

    public void init() {
        setBackground(Color.white);
        setForeground(Color.blue);
        addMouseListener(new PositionRecorder());
        addMouseMotionListener(new LineDrawer());
    }

    protected void record(int x, int y) {
        lastX = x; lastY = y;
    }
}
private class PositionRecorder extends MouseAdapter {
    public void mouseEntered(MouseEvent event) {
        requestFocus(); // Plan ahead for typing
        record(event.getX(), event.getY());
    }

    public void mousePressed(MouseEvent event) {
        record(event.getX(), event.getY());
    }
}

...
... 
private class LineDrawer extends MouseMotionAdapter {
    public void mouseDragged(MouseEvent event) {
        int x = event.getX();
        int y = event.getY();
        Graphics g = getGraphics();
        g.drawLine(lastX, lastY, x, y);
        record(x, y);
    }
}
}
Simple Whiteboard (Results)
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class Whiteboard extends SimpleWhiteboard {
    protected FontMetrics fm;

    public void init() {
        super.init();
        Font font = new Font("Serif", Font.BOLD, 20);
        setFont(font);
        fm = getFontMetrics(font);
        addKeyListener(new CharDrawer());
    }
}
private class CharDrawer extends KeyAdapter {
    // When user types a printable character, draw it and shift position rightwards.

    public void keyTyped(KeyEvent event) {
        String s = String.valueOf(event.getKeyChar());
        getGraphics().drawString(s, lastX, lastY);
        record(lastX + fm.stringWidth(s), lastY);
    }
}
Whiteboard (Results)
Summary

• **General strategy**
  – Determine what type of listener is of interest
    • Check table of standard types
  – Define a class of that type
    • Extend adapter separately, implement interface, extend adapter in named inner class, extend adapter in anonymous inner class
  – Register an object of your listener class with the window
    • Call addXxxListener

• **Understanding listeners**
  – Methods give specific behavior.
    • Arguments to methods are of type XxxEvent
      – Methods in MouseEvent of particular interest
Questions?
• Whiteboard had freehand drawing only
  – Need GUI controls to allow selection of other drawing methods
• Whiteboard had only “temporary” drawing
  – Covering and reexposing window clears drawing
  – After cover multithreading, we’ll see solutions to this problem
    • Most general is double buffering
• Whiteboard was “unshared”
  – Need network programming capabilities so that two different whiteboards can communicate with each other