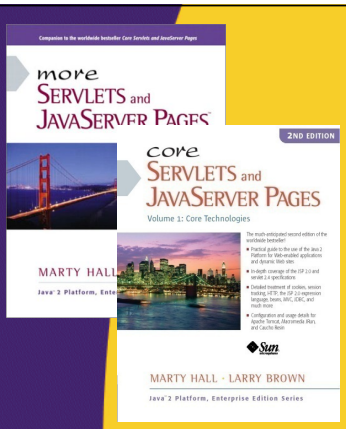
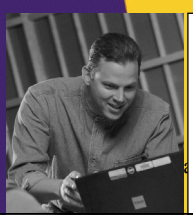
Handling Mouse and Keyboard Events

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/java.html





Topics in This Section

- **General asynchronous event-handling strategy**
- **Event-handling options**
  - 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.add*Xxx*Listener(new MyListenerClass());
    - E.g., addKeyListener, addMouseListener

Using Separate Listener Classes

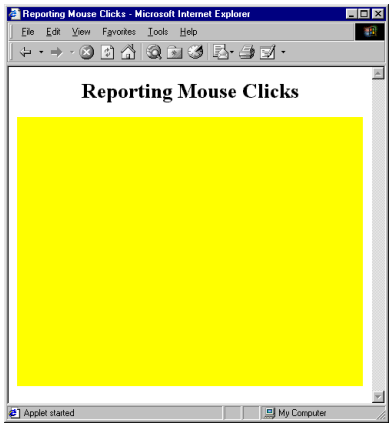
Handling Events with a Separate Listener: Simple Case

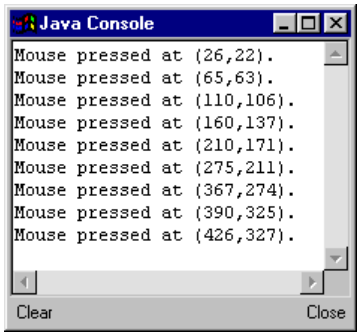
- **Listener does not need to call any methods of the window to which it is attached**

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

public class ClickReporter extends Applet {
  public void init() {
    setBackground(Color.YELLOW);
    addMouseListener(new ClickListener());
  }
}
```

Separate Listener: Simple Case (Continued)

```
import java.awt.event.*;

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





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

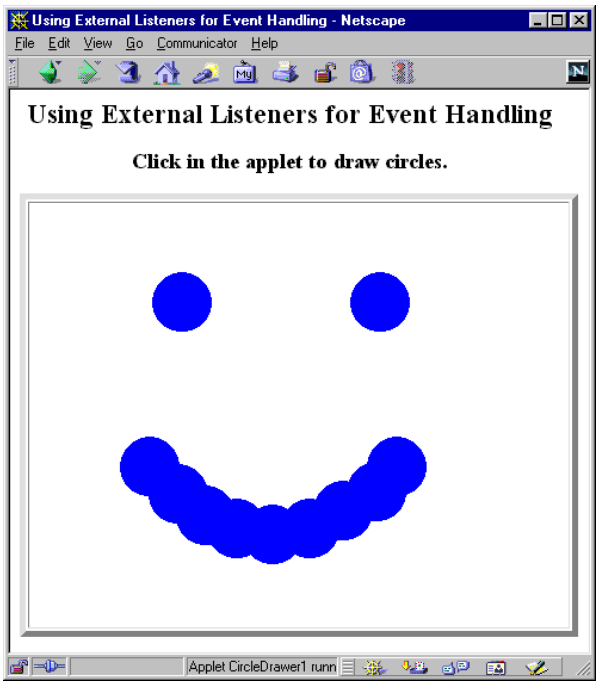
Handling Events with Separate Listener: General Case

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

public class CircleDrawer1 extends Applet {
  public void init() {
    setForeground(Color.BLUE);
    addMouseListener(new CircleListener());
  }
}
```


Separate Listener: General Case (Continued)

```
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)



Implementing a Listener Interface

Review of Interfaces: Syntax

- **Shape interface**

```
public interface Shape {
    public double getArea();   // No body, just specification
}
```

- **Circle class**

```
public class Circle implements Shape {
    public double getArea() {  some real code }
}
```

- **Note**
  - You can implement many interfaces
    - public class MyClass implements Foo, Bar, Baz { … }

Review of Interfaces: Benefits

- **Class can be treated as interface type**
  - public interface Shape {
        public double getArea();
    }
  - public class Circle implements Shape { … }
  - public class Rectangle implements Shape { … }

```
Shape[] shapes =
    { new Circle(…), new Rectangle(…) … };
double sum = 0;
for(Shape s: shapes) {
   sum = sum + s.getArea();  // All Shapes have getArea
}
```

Source Code for MouseListener and MouseAdapter (Simplified)

```
public interface MouseListener {
  public void mouseClicked(MouseEvent e);
  public void mousePressed(MouseEvent e);
  public void mouseReleased(MouseEvent e);
  public void mouseEntered(MouseEvent e);
  public void mouseExited(MouseEvent e);
}
```

```
public abstract class MouseAdapter
       implements MouseListener {
  public void mouseClicked(MouseEvent e) {}
  public void mousePressed(MouseEvent e) {}
  public void mouseReleased(MouseEvent e) {}
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
}
```


Case 2: Implementing a Listener Interface

```
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);
  }
```

When you implement an interface, Eclipse can stub out the methods for you. R-click inside the class, Source, Override/Implement Methods.

Implementing a Listener Interface (Continued)

```
  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);
  }
}
```


Adapters vs. Interfaces: Method Signature Errors

- **What if you goof on the method signature?**
  - public void mousepressed(MouseEvent e)
  - public void mousePressed()
- **Interfaces**
  - Compile time error
- **Adapters**
  - No compile time error, but nothing happens at run time when you press the mouse
- **Solution for adapters (and overriding in Java 5+ in general): @Override annotation**
  - Whenever you *think* you are overriding a method, put "@Override" on the line above the start of the method.
    - If that method is not actually overriding an inherited method, you get a compile-time error.

@Override Example

```
public class CircleDrawer1 extends Applet {
  @Override
  public void init() {
    setForeground(Color.BLUE);
    addMouseListener(new CircleListener());
  }
}
```

```
public class CircleListener extends MouseAdapter {
  private int radius = 25;
  @Override
  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);
  }
}
```


Using Inner Classes (Named & Anonymous)

Review of Inner Classes

- **Class can be defined inside another class**
  - Methods in the inner class can access all methods and instance variables of surrounding class
    - Even private methods and variables
- **Example**

```
public class OuterClass {
  private int count = …;

  public void foo(…) {
     InnerClass inner = new InnerClass();
     inner.bar();
  }

  private class InnerClass {
    public void bar() {
       doSomethingWith(count);
    }
  }
}
```


Case 3: Named Inner Classes

```
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**

```
  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);
    }
  }
}
```


Case 4: Anonymous Inner Classes

```
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);
         }
      });
  }
}
```

Summary of Approaches

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

Event Handler Details and Examples

Standard AWT Event Listeners (Summary)

Listener	Adapter Class (If Any)	Registration Method
ActionListener		addActionListener
AdjustmentListener		addAdjustmentListener
ComponentListener	ComponentAdapter	addComponentListener
ContainerListener	ContainerAdapter	addContainerListener
FocusListener	FocusAdapter	addFocusListener
ItemListener		addItemListener
KeyListener	KeyAdapter	addKeyListener
MouseListener	MouseAdapter	addMouseListener
MouseMotionListener	MouseMotionAdapter	addMouseMotionListener
TextListener		addTextListener
WindowListener	WindowAdapter	addWindowListener

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)

- **ItemListener**
  - 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. Do *not* use this for mousePressed!
      - Applies on release if no movement since press
- **MouseMotionListener**
  - Handles mouse movement
    - mouseMoved(MouseEvent event)
    - mouseDragged(MouseEvent event)
- **MouseInputListener**
  - Combines MouseListener and MouseMotionListener
    - In javax.swing.event package, not java.awt.event
    - You have to call *both* addMouseListener and addMouseMotionListener, so it does not save much

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

Example: Simple Whiteboard

```
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;
  }
```

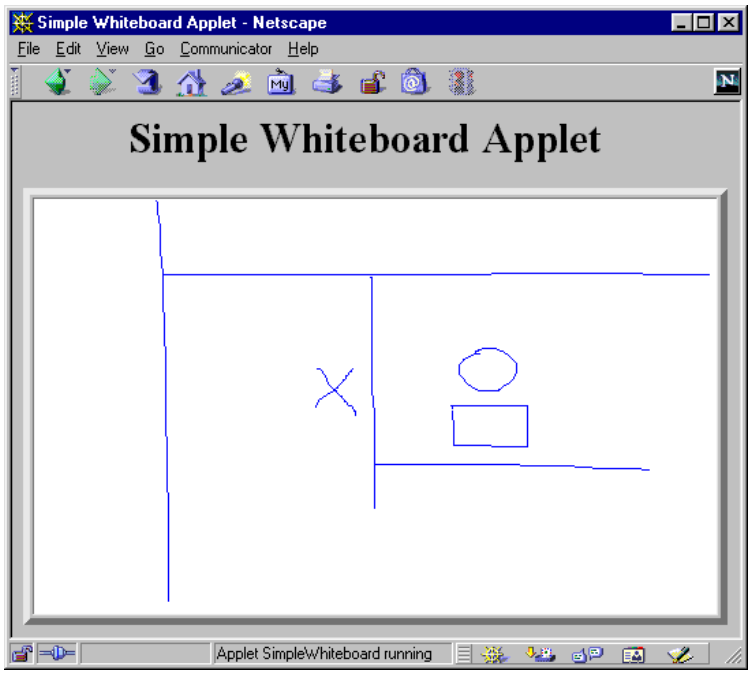

Simple Whiteboard (Continued)

```
  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());
    }
  }
  ...
```

Simple Whiteboard (Continued)

```
  ...
  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)



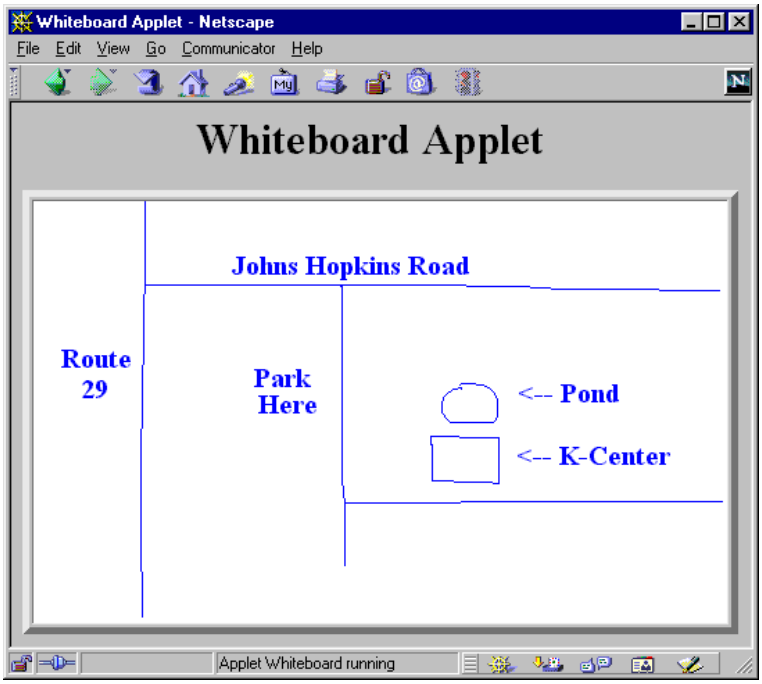
Whiteboard: Adding Keyboard Events

```
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());
  }
```


Whiteboard (Continued)

```
  ...
  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)



Mouse Events: Details

- **MouseListener and MouseMotionListener share event types**
- **Location of clicks**
  - event.getX() and event.getY()
  - You can also use the MouseInfo class for mouse position
- **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.

Combining Listeners: Spelling-Correcting Textfield

- KeyListener corrects spelling during typing
- ActionListener completes word on ENTER
- FocusListener gives subliminal hints

Wrap-Up

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 add*Xxx*Listener
- **Understanding listeners**
  - Methods give specific behavior.
    - Arguments to methods are of type XxxEvent
      - Methods in MouseEvent of particular interest

Preview of Later Topics

- **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

Questions?