Today’s lecture

- **GUIs:** Listening to Events
- Also covers:
  - Stepwise Refinement
  - Inner classes & anonymous classes
- Demos on website:
  - Download the demo zip file
  - Demos of sliders, scroll bars, combobox listener, adapters, etc.

---

**Listening to events:** mouse click, mouse movement into or out of a window, a keystroke, etc.

- An **event** is a mouse click, a mouse movement into or out of a window, a keystroke, etc.
- To be able to “listen to” a kind of event, you have to:
  1. Have some class C implement an interface IN that is connected with the event.
  2. In class C, override methods required by interface IN; these methods are generally called when the event happens.
  3. Register an object of class C as a **listener** for the event. That object’s methods will be called when event happens.

We show you how to do this for clicks on buttons, clicks on components, and keystrokes.

---

**Example: JButton**

Instance: associated with a “button” on the GUI, which can be clicked to perform an action

- jb1 = new JButton(); // jb1 has no text on it
- jb2 = new JButton("first"); // jb2 has label "first" on it
- jb2.isEnabled(); // true iff a click on button can be detected
- jb2.setEnabled(); // set enabled property
- jb2.addActionListener(object); // object's w/ method called when jb2 clicked

At least 100 more methods; these are most important

JButton is in package javax.swing
Listening to a JButton

1. Implement interface ActionListener:
   ```java
   public class C extends JFrame implements ActionListener
   {
   ...
   }
   ```

2. In class C override actionPerformed, which is to be called when button is clicked:
   ```java
   /* Process click of button */
   public void actionPerformed(ActionEvent e) {
   ...
   }
   ```

3. Add an instance of class C an “action listener” for button:
   ```java
   button.addActionListener(this);
   ```

Example: A JPanel that is painted

- The JFrame content pane has a JPanel in its CENTER and a “reset” button in its SOUTH.
- The JPanel has a horizontal box b, which contains two vertical Boxes.
- Each vertical Box contains two instances of class Square.
- Click a Square that has no pink circle, and a pink circle is drawn.
- Click a Square that has a pink circle, and the pink circle disappears.
- Click the rest button and all pink circles disappear.

Aside: The “Graphics” class

An object of abstract class Graphics has methods to draw on a component (e.g. on a JPanel, or canvas).

**Major methods:**

- `drawString` string, x0, y0
- `drawLine` x1, y1, x2, y2
- `drawArc` x, y, width, height
- `getFont()`

Given a Graphics object, you use it to draw on a component.

```java
Graphics g = getGraphics();
g.drawString("Hello", 10, 10);
g.setLine(10, 10, 30, 30);
g.setFont(Font f);
```

Class Square

```java
public class Square extends JPanel implements ActionListener
{
    public void paint(Graphics g) {
        if ((x+y) % 2 == 0) g.setColor(Color.green);
        else g.setColor(Color.red);
        g.fillRect(x, y, width, height);
    }
    public void actionPerformed(ActionEvent e) {
        if (hasDisk) {
            hasDisk = !hasDisk;
            repaint();
        }
    }
}
```
Listening to mouse events
(click/press/release/enter/leave a component)

In package java.awt.event

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

Having to write all of these in a class that implements MouseListener, even though you don’t want to use all of them, can be a pain. So, an adapter class is provided that implements them, albeit with empty methods.

A class that listens to a mouseclick in a Square

This class has several methods (that do nothing) that process mouse events:
- mouse click in a Square
- mouse press in a Square
- mouse release in a Square
- mouse enters component in Square
- mouse leaves component in Square
- mouse dragged beginning in Square

public class MouseDemo2 extends JFrame

implements ActionListener {

    public void actionPerformed(ActionEvent e) {
        // Complement "has pink disk" property
        comp.setHasPinkDisk(!comp.getHasPinkDisk());
        if (comp instanceof Square) {
            ((Square)comp).complementDisk();
        }
    }

    public MouseDemo2()

    extends MouseDemo2()

    extends JFrame()

    extends MouseInputAdapter {

/** Constructor: ... */

public MouseDemo2() {

    mouseEntered() {
        // When mouse enters the component
        e = new MouseEvent(this, MouseEvent.MOUSE_MOVED, 0, 0,
        int x, int y, int button, int modifiers, int pressCount);
        MouseInputEvent me = new MouseInputEvent((Component)e.getSource(),
        int button, int x, int y, int focusChange, long when, long time);
        addMouseListener();
        addMouseMotionListener();
        addMouseWheelListener();
        addMouseDragListener();
        addMouseWheelForcedListener();
}

public void mouseClicked(MouseEvent e) {

    MouseClickEvent me = new MouseClickEvent(
    int x, int y, int button, int pressCount, int which);
    MouseInputEvent me = new MouseEvent((Component)e.getSource(),
    int button, int x, int y, int focusChange, long when, long time);
public class ButtonDemo3 extends JFrame implements ActionListener {

private JButton wB, eB;  // Have a different listener for each button.

public ButtonDemo3() {
    // Add buttons to content pane,
    //   enable one, disable the other
    eB.addActionListener(this);  // eB
    eB.addActionListener(new BeListener());  // eB
    wB.addActionListener(this);  // wB
}

public void actionPerformed(ActionEvent e) {
    boolean b = eB.isEnabled();
    eB.setEnabled(!b);
    wB.setEnabled(b);
}

private class BeListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        boolean b = eB.isEnabled();
        eB.setEnabled(!b);
    }  
}

// A listener for eB
private class BeListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        boolean b = eB.isEnabled();
        eB.setEnabled(!b);
    }
}

/**
 * Solution to problem: Make BeListener an inner class.
 */
public class ButtonDemo3 extends JFrame implements ActionListener {

private JButton wB, eB;  // Can reference all the fields and methods

public ButtonDemo3() {
    /*
     * Inside-out rule says that methods in here
     * Can reference all the fields and methods
     */
    eB.addActionListener(new BeListener());
}

public void actionPerformed(ActionEvent e) {
    boolean b = eB.isEnabled();
    eB.setEnabled(!b);
    wB.setEnabled(b);
}

private class BeListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        boolean b = eB.isEnabled();
        eB.setEnabled(!b);
    }
}

/**
 * Inside-out rule then gives access to wB, eB
 */

/**
 * Problem: can't give a function as a parameter
 */
public class BeListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        boolean b = eB.isEnabled();
        eB.setEnabled(!b);
    }
}

/**
 * Make BeListener an inner class.
 */
public void m() {  
    eB.addActionListener(new C());
}

public class C implements IN {
    public void aP(ActionEvent e) {
        body
    }
}

/**
 * Java says: provide class C that wraps method; give eB
 * an object of class C
 */
public void m() {  
    eB.addActionListener(new C());  
}

/**
 * Have a class for which only one object is created?
 * Use an anonymous class.
 */
public class ButtonDemo3 extends JFrame implements ActionListener {

private JButton wB, eB;  // Can only create one BeListener

public ButtonDemo3() {
    eB.addActionListener(new BeListener());
    wB.addActionListener(new BeListener());
}

public void actionPerformed(ActionEvent e) {
    boolean b = eB.isEnabled();
    eB.setEnabled(!b);
    wB.setEnabled(b);
}

private class BeListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        boolean b = eB.isEnabled();
        eB.setEnabled(!b);
    }
}

/**
 * Using an anonymous class to replace “new BeListener()”
 */
public class ButtonDemo3 extends JFrame implements ActionListener {

    public ButtonDemo3() {
        /*
         * Inside-out rule says that methods in here
         * Can reference all the fields and methods
         */
        eB.addActionListener(new BeListener());
        wB.addActionListener(new BeListener());
    }

    public void actionPerformed(ActionEvent e) {
        /*
         * Inside-out rule says that methods in here
         * Can reference all the fields and methods
         */
        boolean b = eB.isEnabled();
        eB.setEnabled(!b);
        wB.setEnabled(b);
    }
}

private class BeListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        boolean b = eB.isEnabled();
        eB.setEnabled(!b);
    }
}

public class C implements IN {
    public void aP(ActionEvent e) {
        body
    }
}

public class C implements IN {
    public void aP(ActionEvent e) {
        body
    }
}

/**
 * Use sparingly, and only when the anonymous class has 1 or 2 methods in it,
 * because the syntax is ugly, complex, hard to understand.
 */
public class ButtonDemo3 extends JFrame implements ActionListener {

private JButton wB, eB;  // Can only create one BeListener

public ButtonDemo3() {
    eB.addActionListener(new BeListener());
    wB.addActionListener(new BeListener());
}

public void actionPerformed(ActionEvent e) {
    boolean b = eB.isEnabled();
    eB.setEnabled(!b);
    wB.setEnabled(b);
}

private class BeListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        boolean b = eB.isEnabled();
        eB.setEnabled(!b);
    }
}

public class ButtonDemo3 extends JFrame implements ActionListener {

private JButton wB, eB;  // Can only create one BeListener

public ButtonDemo3() {
    eB.addActionListener(new BeListener());
    wB.addActionListener(new BeListener());
}

public void actionPerformed(ActionEvent e) {
    boolean b = eB.isEnabled();
    eB.setEnabled(!b);
    wB.setEnabled(b);
}

private class BeListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        boolean b = eB.isEnabled();
        eB.setEnabled(!b);
    }
}

public class C implements IN {
    public void aP(ActionEvent e) {
        body
    }
}

public class C implements IN {
    public void aP(ActionEvent e) {
        body
    }
}
Using an anonymous class to replace “new BeListener()”

**BEFORE:**
```java
eB.addActionListener(new BeListener(){
    public void actionPerformed(ActionEvent e) { …}
});
```

**AFTER:**
```java
eB.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e) { …}
});
```