

CSE 8B—Intro to CS: Java

Winter, 2006

February 21 (Day 13)
Interfaces
Swing Event Handling

Instructor: Neil Rhodes

Review of Swing

Components are visible objects in the GUI

- Example: JButton, JLabel

Containers are areas where the components can be placed

- Containers are in a hierarchy
- Example: Panel, JFrame

2

Events

Two kinds:

- Low-level:
 - Mouse moved
 - Mouse pressed
 - Mouse released
 - Key pressed
 - Key released
- Semantic:
 - Button pressed
 - Checkbox changed
 - Menu item chosen

3

Listeners

When the user interacts with a component:

- Presses a button
- Turns on/off a checkbox

Component sends *event* to *listener*

- Sometimes called *firing the event*

The method in the listener is called an *event handler*

- Different kinds of events cause different methods to execute

In order for this to work, component must know which objects are listening

- Keeps a list of listeners
- A listener can be added to a component

4

Different kinds of listeners

Listen for particular kinds of events

- Have different method that executes

Examples

- **ComponentListener**
 - Listens for changes in component's geometry or visibility
 - Methods: `componentHidden`, `componentMoved`, ...
- **FocusListener**
 - Listens for whether component gained *focus* (ability to receive keyboard input).
 - Methods: `focusGained`, `focusLost`
- **KeyListener**
 - Listens for key presses from component that has the focus
 - Methods: `keyTyped`, `keyPressed`, `keyReleased`, ...
- **MouseListener**
 - Listens for mouse clicks and movements into/out of the component
 - Methods: `mouseEntered`, `mouseExited`, `mouseClicked`

5

Different kinds of listeners

ActionListener

- Listens for user pressing a button (or radio button), or choosing a menu item, or pressing Enter in a text field.
- Methods: `actionPerformed`

ChangeListener

- Listens for changes to the state of a component (usually a slider, color chooser, or spinner).
- Methods: `stateChanged`

ItemListener

- Listens for changes to whether an item is on/off (checkboxes, check box menu items)
- Methods: `itemStateChanged`

ListSelectionListener

- Listens for changes to the selection in a list or table
- Methods: `valueChanged`

6

But what is a listener?

A listener is an object that satisfies the listener *interface*

- It implements the necessary method(s)

Java has syntax for specifying that a class implements particular methods

- Interface

Detour into interfaces
We shall return!

7

Interface

An interface is a property of a class that specifies what methods it must have.

```
public interface Ordered {  
    // does this precede other?  
    public boolean precedes(Object other);  
    // does this follow other?  
    public boolean follows(Object other);  
}
```

```
public class MyClass implements Ordered {  
    public boolean precedes(Object other) {  
        // ... actual implementation here  
    }  
  
    public boolean follows(Object other) {  
        // ... actual implementation here  
    }  
}
```

8

Interfaces vs. inheritance

Java does not support *multiple inheritance*

- A class can have at most one immediate superclass

But, a class can have many interfaces

- The interfaces tell the Java compiler what methods a class must have.
- Classes are usually nouns; interfaces are usually adjectives.

```
- public class MyClass extends Shape implements  
  Colorful, Blinking {...}
```

```
public void handleOrdered(Ordered c) {  
    // ...  
}
```

```
MyClass m;  
handleOrdered(m);      // OK?  
Object o;  
handleOrdered(o);      // OK?
```

9

Additional interface capabilities

Interfaces can be extended

```
public interface MoreOrdered extends Ordered {  
    // returns object preceding this one  
    public Object preceding();  
}
```

Interfaces can define constants

- Must be static final. They are *not* instance variables

```
public interface MonthNumbers {  
    public static final int JANUARY=1, ...  
    DECEMBER=12;  
}
```

10

Common Interfaces

Comparable

- One method:
 - `public int compareTo(Object other) throws ClassCastException;`
- Defines whether the other object is less than, equal to or greater than this object (returns negative, zero, or positive)
- Implements *total ordering* on the class
 - You'll learn about total orderings in CSE 20

Cloneable

- No method!
- Says that it's OK for `Object.clone` to make a field-for-field copy of the object

Serializable

- No method!
- Says that it's OK for this object to be serialized to/from a stream.

11

Comparable

Example implementation

```
class Person implements Comparable {  
    public String firstName, lastName;  
    public int age;  
  
    // compare by lastName, then by firstName, then by age  
    public int compareTo(Object o) throws ClassCastException {  
        if (!(o instanceof Person))  
            throw new ClassCastException("Person object expected");  
        Person p = (Person) o;  
        int d = lastName.compareTo(p.lastName);  
        if (d != 0)  
            return d;  
        else  
            d = firstName.compareTo(p.firstName);  
        if (d != 0)  
            return d;  
        else  
            return age - p.age;  
    }  
}
```

12

Comparable

Example uses

- The Arrays class defines:
 - `public void sort(Object[] arr);`
 - All elements in the array must implement Comparable
- Unfortunately, should actually be defined as:
 - `public void sort(Comparable[] arr);`

13

Comparable

Example uses

```
Object[] arr;  
String[] arr2;  
Sort.doSort(arr); // OK?  
Sort.doSort(arr2); // OK?  
  
import java.util.*;  
public class Sort {  
    public static void doSort(Comparable[] arr) {  
        for (int i = 0; i < arr.length; i++) {  
            int minIndex = i;  
            for (int j = i+1; j < arr.length; j++)  
                if (arr[j].compareTo(arr[minIndex]) < 0)  
                    minIndex = j;  
            swap(arr, i, minIndex);  
        }  
    }  
  
    private static void swap(Comparable[] arr, int i, int j) {  
        Comparable tmp = arr[i];  
        arr[i] = arr[j];  
        arr[j] = tmp;  
    }  
}
```

14

Cloneable

Object defines clone

- `protected Object clone()`
throws `CloneNotSupportedException`
- Does a bit-by-bit copy (shallow copy)
- Will throw an exception if *this* is not Cloneable

So, if you want a class to be cloneable:

- implements Cloneable
- Redefine clone as

```
public MyClass implements Cloneable {  
    OtherClass var;  
    ...  
    public Object clone() throws  
CloneNotSupportedException {  
        MyClass m = (MyClass) super.clone();  
        // clone any instance vars that shouldn't be shared  
        m.var = (OtherClass) m.var.clone();  
        return m;  
    }  
}
```

15

Serializable

Just add implements Serializable to class

16

Back to Listeners

Listeners are interfaces

- public interface **ActionListener** extends `EventListener`
{
 public void actionPerformed(ActionEvent e);
}
- public interface **ChangeListener** extends `EventListener`
{
 public void stateChanged(ChangeEvent e);
}
- etc.

17

Handling an event

Any object can be a listener (as long as it implements the needed interface)

18

Handling a button

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*; // for ActionEvent and ActionListener

public class ButtonTest extends JFrame implements ActionListener {
    public static void main(String[] args) {
        ButtonTest window = new ButtonTest();
        window.setVisible(true);
    }
    ButtonTest() {
        super("ButtonTest");
        setSize(300, 100);

        JButton button = new JButton("Click to close");
        button.addActionListener(this);
        add(button);
    }
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
}
```

19

How one event handler can handle multiple objects

```
public class ButtonTest2 extends JFrame implements ActionListener {
    public static void main(String[] args) {
        ButtonTest2 window = new ButtonTest2();
        window.setVisible(true);
    }

    ButtonTest2() {
        super("ButtonTest2");
        setSize(300, 100);
        setLayout(new FlowLayout());

        JButton button = new JButton("Button 1");
        button.addActionListener(this);
        add(button);
        button = new JButton("Button 2");
        button.addActionListener(this);
        add(button);
    }

    public void actionPerformed(ActionEvent e) {
        System.out.println(e.getActionCommand());
        if (e.getActionCommand().equals("Button 1"))
            System.exit(0);
    }
}
```

20

ActionEvent

Extends AWTEvent which extends EventObject

- Useful method from EventObject:
 - **Object getSource();**
 - Returns the object on which the event occurred.
- From ActionEvent:
 - **String getActionCommand();**
 - Returns the text of the component (button text, for example).
 - Or, returns a string set as the action command (button.setActionCommand("my text"))
 - If two buttons have the same text, use setActionCommand to set unique action commands.

21

Another way of handling multiple objects

```
public class ButtonTest3 extends JFrame implements ActionListener {
    JButton button1;
    JButton button2;
    public static void main(String[] args) {
        ButtonTest3 window = new ButtonTest3();
        window.setVisible(true);
    }
    ButtonTest3() {
        super("ButtonTest3");
        setSize(300, 100);
        setLayout(new FlowLayout());

        button1 = new JButton("Button 1");
        button1.addActionListener(this);
        add(button1);
        button2 = new JButton("Button 2");
        button2.addActionListener(this);
        add(button2);
    }
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == button1)
            System.exit(0);
    }
}
```

22

Using a separate ActionListener

```
import java.awt.event.*; // for ActionEvent and ActionListener
public class ExitOnAction implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
}
```

```
import javax.swing.*;
import java.awt.*;
public class ButtonTest4 extends JFrame {
    public static void main(String[] args) { ... }

    ButtonTest4() {
        super("ButtonTest4");
        setSize(300, 100);
        setLayout(new FlowLayout());

        JButton button1 = new JButton("Button 1");
        button1.addActionListener(new ExitOnAction());
        add(button1);
        JButton button2 = new JButton("Button 2");
        add(button2);
    }
}
```

23

Changing the appearance of a component

Text changes

- For JTextField, JTextArea, JButton, JLabel, can use:
 - **obj.setText("new text");**
- Will automatically redraw

Enabled change

- **obj.setEnabled(false);** //obj is now disabled
- If not enabled, will be dimmed, and clicking will do nothing

Editable change

- Only for JTextField, JTextArea
 - **obj.setEditable(false);** // can't be edited

Location change

- **obj.setLocation(10, 50);** //move to (10, 50)

other changes

- See docs (specific, like JButton, and superclasses, like Component).

24