Event Handling

Objectives

- Learn about Java interfaces.
- Be able to use listener interfaces to process events in a GUI application.
Outline

- Java Interfaces
- Event Handling
  - Action events
  - Key events
  - Mouse events

Interfaces

- An interface definition is similar to a class definition except that they can only contain constants and method signatures.
  - Interfaces don’t have fields
  - Methods in interfaces don’t have implementations
- A class can implement an interface by:
  - Declaring that it implements the interface
  - Implementing all the methods defined in the interface
- An interface can be used as a type.
  - A variable of an interface type can be assigned an object of any class that implements the interface.
Defining an Interface

```
public interface Automobile {
    public void signalLeftTurn();
    public void signalRightTurn();
    public void accelerate(double speed);
    public double getSpeed();
    public void turnOnWipers();
    public void turnOffWipers();
    public void turnOnHeadLights();
    ...
}
```

Implementing an Interface

```
public class ToyotaCamry implements Automobile {

    public void signalLeftTurn() {
        // code to activate left turn signal
        // on a toyota camry
    }

    public void accelerate(double speed) {
        // code to accelerate to specified
        // speed on a toyota camry
    }
    ...
}
```
Events and Listeners

- Events are represented in java programs by event objects.
- There are classes for various kinds of events.
- Generally, there is a listener interface for handling each kind of event.
- User interface components create event objects in response to user actions, and send the events to listener objects for processing.

Event Classes

```
java.util
java.awt.event
javax.swing.event
java.awt

EventObject
  ____________________________
   |                           |
   v                           v
ListSelectionEvent           AWTEvent
                             _______________________
                             |                           |
                             v                           v
ComponentEvent               ActionEvent
                             _______________________
                             |                           |
                             v                           v
InputEvent
                             _______________________
                             |                           |
                             v                           v
KeyEvent
                             _______________________
                             |                           |
                             v                           v
MouseEvent
```

Packages

- java.awt
- java.util
- java.awt.event
- javax.swing.event
Events

- ActionEvents are generated when the user:
  - clicks a JButton
  - selects a JMenuItem
  - presses the return/enter key in a JTextField
- KeyEvents are generated when a user presses or releases keys on the keyboard.
- MouseEvents are generated when a user moves the mouse or presses or releases a mouse button.
- A single action by the user may result in several different event objects being created.
- It isn’t necessary to respond to every event.

Listener Interfaces

Diagram showing the inheritance of event listener interfaces::

- ActionListener
-KeyListener
-MouseMotionListener
-MouseListener
-ListSelectionListener
Event Handling

- 3 Steps to set up event handling:
  - Declare that your class implements the necessary interface.
  - Implement the methods in the interface.
  - Register as a listener with the objects that create the events.

Counter Example

- Clicking the Up button increases the count.
- Clicking the Down button decreases the count.
- The count is displayed in a JLabel.
**Step 1**

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

public class Counter extends JFrame implements ActionListener {

    // We need to store these objects in fields so we can
    // send messages to them after they are created.
    private JLabel display = new JLabel("0");
    private JButton up = new JButton("Up");
    private JButton down = new JButton("Down");
    private Integer count = 0;

    Counter objects need to respond to action events
```

**Step 2**

```java
public void actionPerformed(ActionEvent e) {
    Object source = e.getSource(); // Which object generated the event?

    if (source == up) { // Which object generated the event?
        count = count + 1;
    }

    else { // must be down
        count = count - 1;
    }

    // Display the new count
    String s = count.toString();
    display.setText(s);
}

The ActionListener interface has just one method: actionPerformed
```
Step 3

```java
public Counter() {
    // Register with the JButtons
    up.addActionListener(this);
    down.addActionListener(this);

    // Layout the user interface
    this.setDefaultCloseOperation(EXIT_ON_CLOSE);
    this.setLayout(new FlowLayout());
    this.add(display);
    this.add(up);
    this.add(down);
    this.setSize(250, 80);
    this.setVisible(true);
}
```

We need to register with the JButtons so they will send us actionPerformed messages when they’re clicked.

---

The Counter Class main Method

```java
public static void main(String[] args) {
    new Counter();
}
```

The main method just creates a Counter object.
**ActionListener and KeyListener**

```java
public interface ActionListener {
    public void actionPerformed(ActionEvent e);
}

public interface KeyListener {
    public void keyTyped(KeyEvent e);
    public void keyPressed(KeyEvent e);
    public void keyReleased(KeyEvent e);
}
```

**MouseListener and MouseMotionListener**

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

public interface MouseMotionListener {
    public void mouseDragged(MouseEvent e);
    public void mouseMoved(MouseEvent e);
}
```
Alternative Program Structures

1. The Counter object could be a JFrame (as we have seen) by extending the JFrame class.
2. The Counter object could use a JFrame by creating one in its constructor.
3. The Counter could be a new kind of reusable component that we could place in a JFrame or any other container along with other components.

Alternative 2

```java
public class Counter implements ActionListener {

    public Counter() {
        // Register with the JButtons
        up.addActionListener(this);
        down.addActionListener(this);

        // Layout the user interface
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new FlowLayout());
        frame.add(display);
        frame.add(up);
        frame.add(down);
        frame.setSize(250, 80);
        frame.setVisible(true);
    }
}
```
Alternative 3

Extend JPanel instead of JFrame

```java
public class Counter extends JPanel implements ActionListener {

    public Counter() {
        // Register with the JButtons
        up.addActionListener(this);
        down.addActionListener(this);

        // Layout the user interface
        this.setLayout(new FlowLayout());
        this.add(display);
        this.add(up);
        this.add(down);
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new FlowLayout());
        Counter counter = new Counter();
        frame.add(counter);
        frame.setSize(250, 80);
        frame.setVisible(true);
    }
}
```

Eliminate unnecessary code from the constructor.

Write a main method to use our new Counter component.