Java Programming

EVENT HANDLING

Windows based Java Programs

- Console based Programming
  - Every thing is predetermined
  - The program code determines the sequence of events
- Window based programming
  - The operation is driven by what you do with the GUI
  - Selecting menu items, buttons, or keyboard causes particular actions within a program
  - The specific program that is executed next is not known
Event driven Programming

- The signals that a program receives from the OS as a result of your actions are called events
- A window based program is called event driven program
- Heart of the GUI programming
- Unlike the old rigid old sequential programs,
  - it puts user in charge, user control the sequence of program
  - Application waits for the user action
  - This approach is called event driven programming

The event-handling process

- Suppose a user clicks a button in the GUI
- Button is the source of the event
- When a button is clicked, it will create a new object that represents and identifies this event...in this case the type is ActionEvent
- The object will create information about event and its source
- This object is passed to a method that handles the event
- The event object corresponding to the button object will be passed to a listener object that has previously registered an interest in this kind of event.
- A listener is called Target of an event
Delegation event model

- The way in which events are handled in Java, using listener objects, is called delegation event model.
- We can make objects of any class listener objects by making the class implement a listener interface.
- In case of the button, the `ActionListener` interface needs to be implemented to receive events from button.
- `actionPerformed()` is `ActionListener` is called when the event occurs and the event object is passed as an argument.
- We register listener with the source by calling a particular method of the source. In this case `addActionListener()` method.

Java Events

- Events are objects
  - Objects that represent user initiated actions
  - Examples:
    - button clicked → `ActionEvent`
    - mouse dragged → `MouseEvent`
    - Enter key pressed → `KeyEvent`
  - AWT and Swing Components generate *(fire)* Events
  - `EventObject`; root event class for all event objects
  - `AWTEvent`; root event class for all AWT events
  - `Package java.awt.event`
    - Provides interfaces and classes for dealing with different types of events fired by AWT components.
Events Tell Us

- Who fired the event?
- What type is it?
- Was it consumed?

AWT Events

- Low-level Events - always same meaning
  - **All Components** - ComponentEvent, FocusEvent, KeyEvent, MouseEvent, PaintEvent
  - **Containers** - ContainerEvent
  - **Windows** - WindowEvent

- Semantic Events - event meaning is different depending on who fires it
  - ActionEvent, ItemEvent, TextEvent
Semantic AWT Events

- Not fired by all Components

- Example 1:
  - AdjustmentEvent indicates an item was selected or deselected
    - Fired by a ScrollBar
    - Not fired by a Button

- Example 2:
  - ActionEvent indicates a component-defined action has occurred
  - Fired by a ComboBox, Button, CheckBox and many more

---

Event Handling

- Programmer choice to decide how to handle the generated event
  - Ignore the Event
  - Have the Event handled by the component where the event was generated (Self Contained Event handling)
  - Delegate event handling to some other object called Listeners (Event Delegation)
Self Contained Event handling

- Self contained component is the one that handles the events that it generates
  - None of the AWT & swing components are self contained
- If you want a component to handle its own events, you need to create a subclass. The subclass must do two things
  - Enable receipt of events by calling enableEvents() with necessary event mask setting
  - Provide a processActionEvent() method, which will be called when the component is activated

Event Delegation

- Some time component on which event was generated is not best suited to handle its own event
- The process of assigning an object to handle a component's events is called delegation.
- The event handling objects are called Listeners
JAVA Event Types

Key Methods of Event

- **Object getSource()**
  - In ObjectEvent, return the component in which event took place.

- **int getID()**
  - In AwtEvent, return int that describes the nature of the event e.g. on MouseEvent it will give MOUSE_PRESSED, MOUSE_DRAGGED.
Event Listeners
- Interfaces to support dispatching of events
- Each Event class has a corresponding Listener interface
- Multiple listeners for the same event type
- Each interface will have one or more method corresponding to types of events

Example of Listener
```java
public interface ActionListener {
    public void actionPerformed(ActionEvent e);
}
```
```java
public interface ItemListener {
    public void itemStateChanged(ItemEvent e);
}
```
```java
public interface ComponentListener {
    public void componentHidden(ComponentEvent e);
    public void componentMoved(ComponentEvent e);
    public void componentResized(ComponentEvent e);
    public void componentShown(ComponentEvent e);
}
```
Registering Listeners

- Listeners register themselves with component
  - public void addXXXListener(XXXListener)
  - addActionListener, addItemListener, etc.
- Multiple listeners can be registered for the same event on a component
  - One event can trigger numerous responses
  - Events are broadcast to all listeners

Multiple Listeners

```
Component
  addEventListener
  eventHappened
  Event

EventListener
  addEventListener
  eventHappened
  Event
```

Wiring a Listener

- Define a class to implement the Listener Interface

```java
public class Applet extends Applet implements ActionListener
public class MyClass implements ActionListener {
    ...
    public void actionPerformed(ActionEvent e) {
        // here's where I do stuff when the action happens
    ...
```

- Add the implementation of the Interface

```java
public void actionPerformed(ActionEvent e) {
    // here's where I do stuff when the action happens
```

- Add class as a Listener to Component

```java
Button ok = new Button("OK")
ok.addActionListener(this);
```

Example

```java
public class Beeper extends JPanel implements ActionListener {
    JButton button;

    public Beeper() {
        super(new BorderLayout());
        button = new JButton("Click Me");
        button.setPreferredSize(new Dimension(200, 80));
        add(button, BorderLayout.CENTER);
        button.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        Toolkit.getDefaultToolkit().beep();
    }

    private static void createAndShowGUI() {
        //Create and set up the window.
        JFrame frame = new JFrame("Beeper");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //Create and set up the content pane.
        JComponent newContentPane = new Beeper();
        newContentPane.setOpaque(true);
        frame.setContentPane(newContentPane);
        //Display the window.
        frame.pack();
        frame.setVisible(true);
    }
```
Another Example

```java
public class RadioButtonDemo extends JPanel implements ActionListener {
    static String birdString = "Bird";
    static String catString = "Cat";
    static String dogString = "Dog";
    static String rabbitString = "Rabbit";
    static String pigString = "Pig";
    JLabel picture;
    public RadioButtonDemo() {
        super(new BorderLayout());
        //Create the radio buttons.
        JRadioButton birdButton = new JRadioButton(birdString);
        birdButton.setMnemonic(KeyEvent.VK_B);
        birdButton.setActionCommand(birdString);
        birdButton.setSelected(true);
        JRadioButton catButton = new JRadioButton(catString);
        catButton.setMnemonic(KeyEvent.VK_C);
        catButton.setActionCommand(catString);
        JRadioButton dogButton = new JRadioButton(dogString);
        dogButton.setMnemonic(KeyEvent.VK_D);
        dogButton.setActionCommand(dogString);
        JRadioButton rabbitButton = new JRadioButton(rabbitString);
        rabbitButton.setMnemonic(KeyEvent.VK_R);
        rabbitButton.setActionCommand(rabbitString);
        JRadioButton pigButton = new JRadioButton(pigString);
        pigButton.setMnemonic(KeyEvent.VK_P);
        pigButton.setActionCommand(pigString);
        //Group the radio buttons.
        ButtonGroup group = new ButtonGroup();
        group.add(birdButton);
        group.add(catButton);
        group.add(dogButton);
        group.add(rabbitButton);
        group.add(pigButton);
        //Register a listener for the radio buttons.
        birdButton.addActionListener(this);
        catButton.addActionListener(this);
        dogButton.addActionListener(this);
        rabbitButton.addActionListener(this);
        pigButton.addActionListener(this);
        //Set up the picture label.
        picture = new JLabel(createImageIcon("images/" + birdString + ".gif"));
        //The preferred size is hard-coded to be the width of
        //the widest image and the height of the tallest image.
        //A real program would compute this.
        picture.setPreferredSize(new Dimension(177, 122));
        //Put the radio buttons in a column in a panel.
        JPanel radioPanel = new JPanel(new GridLayout(0, 1));
        radioPanel.add(birdButton);
        radioPanel.add(catButton);
        radioPanel.add(dogButton);
        radioPanel.add(rabbitButton);
        radioPanel.add(pigButton);
        add(radioPanel, BorderLayout.LINE_START);
        add(picture, BorderLayout.CENTER);
        setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));
    }
    /** Listens to the radio buttons. */
    public void actionPerformed(ActionEvent e) {
        picture.setIcon(createImageIcon("images\" + e.getActionCommand() + ".gif"));
    }
}
```
Another Example

```java
/** Returns an ImageIcon, or null if the path was invalid. */
protected static ImageIcon createImageIcon(String path) {
    java.net.URL imgURL = RadioButtonDemo.class.getResource(path);
    if (imgURL != null) {
        return new ImageIcon(imgURL);
    } else {
        System.err.println("Couldn't find file: "+ path);
        return null;
    }
}

/**
* Create the GUI and show it. For thread safety,
* this method should be invoked from the
* event-dispatching thread.
*/
private static void createAndShowGUI() {
    //Create and set up the window.
    JFrame frame = new JFrame("RadioButtonDemo");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Create and set up the content pane.
    JComponent newContentPane = new RadioButtonDemo();
    newContentPane.setOpaque(true); //content panes must be opaque
    frame.setContentPane(newContentPane);

    //Display the window.
    frame.pack();
    frame.setVisible(true);
}
```

Java Event Types, Listeners & Listener Methods

<table>
<thead>
<tr>
<th>Event Class</th>
<th>Listener Interface</th>
<th>Listener Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionEvent</td>
<td>ActionListener</td>
<td>actionPerformed()</td>
</tr>
<tr>
<td>AdjustmentEvent</td>
<td>AdjustmentListener</td>
<td>adjustmentValueChanged()</td>
</tr>
<tr>
<td>ComponentEvent</td>
<td>ComponentListener</td>
<td>componentHidden()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>componentMoved()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>componentResized()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>componentShown()</td>
</tr>
<tr>
<td>ContainerEvent</td>
<td>ContainerListener</td>
<td>componentAdded()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>componentRemoved()</td>
</tr>
<tr>
<td>FocusEvent</td>
<td>FocusListener</td>
<td>focusGained()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>focusLost()</td>
</tr>
</tbody>
</table>
### Java Event Types, Listeners & Listener Methods

<table>
<thead>
<tr>
<th>Event Class</th>
<th>Listener Interface</th>
<th>Listener Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemEvent</td>
<td>ItemListener</td>
<td>itemStateChanged()</td>
</tr>
<tr>
<td>KeyEvent</td>
<td>KeyListener</td>
<td>keyPressed()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>keyReleased()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>keyTyped()</td>
</tr>
<tr>
<td>MouseEvent</td>
<td>MouseListener</td>
<td>mouseClicked()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mouseEntered()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mouseExited()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mousePressed()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mouseReleased()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mouseDragged()</td>
</tr>
<tr>
<td></td>
<td>MouseMotionListener</td>
<td>mouseMoved()</td>
</tr>
<tr>
<td>TextEvent</td>
<td>TextListener</td>
<td>textValueChanged()</td>
</tr>
<tr>
<td>WindowEvent</td>
<td>WindowListener</td>
<td>windowActivated()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>windowClosed()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>windowClosing()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>windowDeactivated()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>windowDeiconified()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>windowIconified()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>windowOpened()</td>
</tr>
</tbody>
</table>