Chapter 15. Event-Driven Programming

Event Programming

- Event programming
  - the flow of the program is determined by user actions (mouse clicks, key presses) or messages from other programs.

- Components
  - Events: user actions or other events
  - Event sources: graphical user interface (GUI) components or other sources that generate the events
  - Event listener (handler): reactions on events

- Basic steps
  - Define event handler
  - Register event handler with event sources

Events

- An event can be defined as a type of signal to the program that something has happened.

- The event is generated by external user actions such as mouse movements, mouse clicks, and keystrokes, or by the operating system or program activities, such as a timer.

Event Classes

- AWTEvent
- EventObject
- AdjustmentEvent
- ComponentEvent
- TextEvent
- ItemEvent
- ActionEvent
- InputEvent
- WindowEvent
- MouseEvent
- KeyEvent
- ContainerEvent
- FocusEvent
- PaintEvent
- ListSelectionEvent

Selected User Actions

<table>
<thead>
<tr>
<th>User Action</th>
<th>Source Object</th>
<th>Event Type Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click a button</td>
<td>JButton</td>
<td>ActionEvent</td>
</tr>
<tr>
<td>Click a check box</td>
<td>JCheckBox</td>
<td>ItemEvent, ActionEvent</td>
</tr>
<tr>
<td>Click a radio button</td>
<td>JRadioButton</td>
<td>ItemEvent, ActionEvent</td>
</tr>
<tr>
<td>Press return on a text field</td>
<td>JTextField</td>
<td>ActionEvent</td>
</tr>
<tr>
<td>Select a new item</td>
<td>JComboBox</td>
<td>ItemEvent, ActionEvent</td>
</tr>
<tr>
<td>Window opened, closed, etc.</td>
<td>Window</td>
<td>WindowEvent</td>
</tr>
<tr>
<td>Mouse pressed, released, etc.</td>
<td>Component</td>
<td>MouseEvent</td>
</tr>
<tr>
<td>Key released, pressed, etc.</td>
<td>Component</td>
<td>KeyEvent</td>
</tr>
</tbody>
</table>
The Delegation Model

- Trigger an event
- Listener: Listens
  - Trigger an event
  - Handle the event

(b) A generic source component with a generic listener
(source: SourceClass)
- A generic source component
- With a generic listener
- Source: JButton
- addActionListener
  - Listener: ActionListener

ActionEvent
ToListener

Internal Function of a Source Component

- Source: SourceClass
- add X Listener(listener)
- Source: JButton
- addActionListener
- listener: CustomListenerClass

Back to the First Example:
HandleEvent.java

1. create event source, event listener, and register listener to
   the source
   ```java
   JButton jbt = new JButton("OK");
   ActionListener listener = new OKListener();
   jbt.addActionListener(listener);
   ```

Example: Simple event demo

- A simple event demo program with an OK button
  ```java
  // define the listener class
class OKListenerClass implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      System.out.println("OK button clicked");
    }
  }
  ```

Review questions

- Which of the following statements are true?
  A. Each event class has a corresponding listener interface.
  B. The listener object's class must implement the corresponding event-listener interface.
  C. A source may have multiple listeners.
  D. The listener object must be registered by the source object.

Selected Event Handlers

<table>
<thead>
<tr>
<th>Event Class</th>
<th>Listener Interface</th>
<th>Listener Methods (Handlers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionEvent</td>
<td>ActionListener</td>
<td>actionPerformed(ActionEvent)</td>
</tr>
<tr>
<td>WindowEvent</td>
<td>WindowListener</td>
<td>windowClosing(WindowEvent)</td>
</tr>
<tr>
<td>ContainerEvent</td>
<td>ContainerListener</td>
<td>componentAdded(ContainerEvent)</td>
</tr>
<tr>
<td>MouseEvent</td>
<td>MouseListener</td>
<td>mousePressed(MouseEvent)</td>
</tr>
<tr>
<td>KeyEvent</td>
<td>KeyListener</td>
<td>keyPressed(KeyEvent)</td>
</tr>
</tbody>
</table>


Listener class as Inner Classes

- A listener class is designed specifically to create a listener object for a GUI component (e.g., a button).
- It is appropriate to define the listener class inside the frame class as an inner class.

Inner Classes

- An inner class, or nested class, is a class defined within the scope of another class.
- Defined inside a class but outside its methods.
- Defined inside a method.
- Inner classes can make programs simple and concise.
- Compiler turns an inner class into a regular class file OuterClassName$InnerClassName.class.
- An inner class can reference the data and methods defined in the outer class in which it nests.

Example: Defining Listener Class as an Inner Class

- A simple event demo program with an OK button.
- SimpleEventDemo.java
- The event demo program using inner class:
  - SimpleEventDemoInnerClass.java

Anonymous Inner Classes

- Inner class listeners can be shortened using anonymous inner classes.
- An anonymous inner class is an inner class without a name.
- It combines declaring an inner class and creating an instance of the class in one step.

Anonymous Inner Classes

- An anonymous inner class must always extend a superclass or implement an interface.
- An anonymous inner class must implement all the abstract methods in the superclass or in the interface.
- An anonymous inner class always uses the no-arg constructor from its superclass to create an instance.
- If an anonymous inner class implements an interface, the constructor is Object.
- An anonymous inner class is compiled into a class named OuterClassName$InnerClassName.class.
- For example, if the outer class Test has two anonymous inner classes, these two classes are compiled into Test$1.class and Test$2.class.
Example: Defining Listener Class as an Anonymous Inner Class

- A simple event demo program with an OK button
  - SimpleEventDemo.java
- The same event demo program using inner class
  - SimpleEventDemoInnerClass.java
- The event demo program using anonymous inner class
  - SimpleEventDemoAnonymousInnerClass.java

The Timer Class

- Some non-GUI components can fire events. The Timer class is a source component that fires an ActionEvent at a predefined rate.
- The Timer class can be used to control animations.

How to Use Timer Class?

- Define an listener class that implements the ActionListener

```java
class MyListener implements ActionListener {
    void actionPerformed(ActionEvent event) {
      // This action will be executed at each timer event
    }
}
```

- Add listener to timer and start the timer

```java
MyListener listener = new MyListener();
Timer t = new Timer(interval, listener);
t.start();
```

Game of Life (for extra credit or just for fun!)

- A mathematical game invented by mathematician John Conway in 1970
- Game rules
  - A dead cell with exactly three live neighbors becomes a live cell (birth).
  - A live cell with two or three live neighbors stays alive (survival).
  - In all other cases, a cell dies or remains dead (overcrowding or loneliness).
- Resources and demos
  - http://www.ibiblio.org/lifepatterns/
- Implementation
  - Use timer class for animation
  - Use two dimensional arrays for cell updates
  - Use Graphics for fancy graphical implementation
- Enjoy life!