Chapter 15. Event-Driven Programming

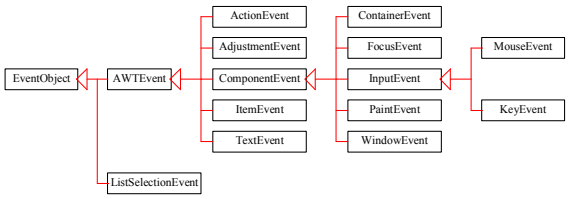
Events

- 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 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

Event Classes



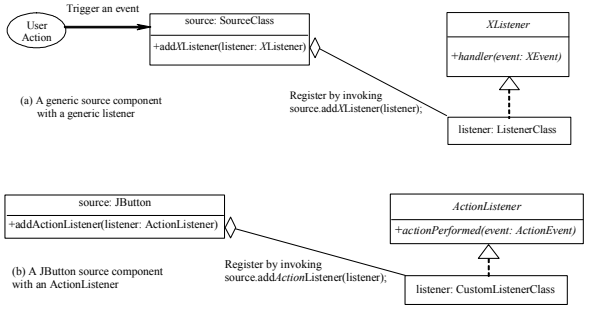
Taste of Event-Driven Programming

- The example displays two buttons in the frame. A message is displayed on the console when a button is clicked.
- HandleEvent.java

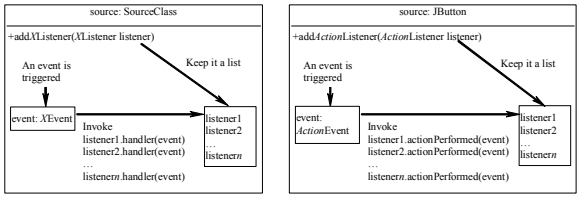
Selected User Actions

User Action	Source Object	Event Type Generated
Click a button	`JButton`	`ActionEvent`
Click a check box	`JCheckBox`	`ItemEvent`, `ActionEvent`
Click a radio button	`JRadioButton`	`ItemEvent`, `ActionEvent`
Press return on a text field	`JTextField`	`ActionEvent`
Select a new item	`JComboBox`	`ItemEvent`, `ActionEvent`
Window opened, closed, etc.	`Window`	`WindowEvent`
Mouse pressed, released, etc.	`Component`	`MouseEvent`
Key released, pressed, etc.	`Component`	`KeyEvent`

The Delegation Model



Internal Function of a Source Component



Selected Event Handlers

Event Class	Listener Interface	Listener Methods (Handlers)
ActionEvent	ActionListener	actionPerformed(ActionEvent)
ItemEvent	ItemListener	itemStateChanged(ItemEvent)
WindowEvent	WindowListener	windowClosing(WindowEvent) windowOpened(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent)
ContainerEvent	ContainerListener	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)
MouseEvent	MouseListener	mousePressed(MouseEvent) mouseReleased(MouseEvent) mouseClicked(MouseEvent) mouseExited(MouseEvent) mouseEntered(MouseEvent)
KeyEvent	KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTypeed(KeyEvent)

Back to the First Example: HandleEvent.java

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

// define the listener class
class OKListenerClass implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    System.out.println("OK button clicked");
  }
}
```


Example: Simple event demo

- A simple event demo program with an OK button
  - SimpleEventDemo.java

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.

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

Inner Classes Example

```
public class Test {
  ...
}

public class A {
  ...
}
```

(a)

```
public class Test {
  ...

  // Inner class
  public class A {
    ...
  }
}
```

(b)

```
// OuterClass.java: inner class demo
public class OuterClass {
  private int data;

  /** A method in the outer class */
  public void m() {
    // Do something
  }

  // An inner class
  class InnerClass {
    /** A method in the inner class */
    public void mi() {
      // Directly reference data and method
      // defined in its outer class
      data++;
      m();
    }
  }
}
```

(c)

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.

```
new SuperClassName/InterfaceName() {
  // Implement or override methods in superclass or interface
  // Other methods if necessary
}
```


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$*n*.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.

javax.swing.Timer	
+Timer(delay: int, listener: ActionListener)	Creates a Timer with a specified delay in milliseconds and an ActionListener.
+addActionListener(listener: ActionListener): void	Adds an ActionListener to the timer.
+start(): void	Starts this timer.
+stop(): void	Stops this timer.
+setDelay(delay: int): void	Sets a new delay value for this timer.

How to Use Timer Class?

- Define an listener class that implements the `ActionListener`

```
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

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

Example: Timer Class

- Count down example – count down every second
  - TimerTester.java
- Shape mover example – move a box to a different position every second
  - TimerTester2.java

Review question

- Which of the following statements are true?

A. You can use the addActionListener method in the Timer class to add a listener.
B. You can specify a delay in the Timer constructor.
C. You must always specify a listener when creating a Timer object.
D. When a timer is created, it is automatically started.

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.math.com/students/wonders/life/life.html
  - 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!