Delegation Event Model in Java

Introduction

1. The Event model is based on two things, i.e., Event Source and Event Listeners.

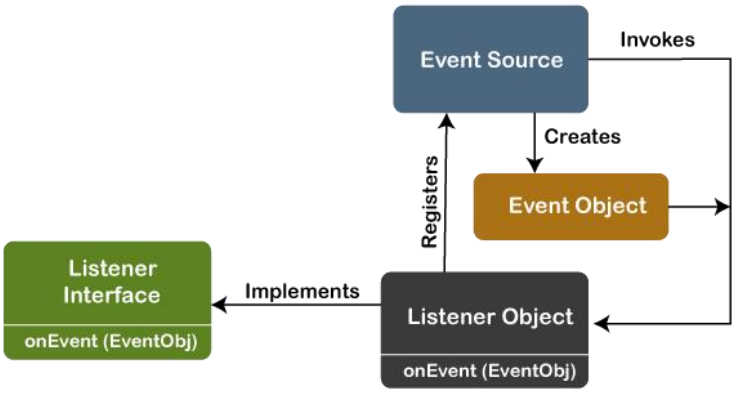
- Event Source means any object which creates the message or event.
- Event Listeners are the object which receives the message or event.

2. The Delegation Event model is defined to handle events in GUI programming languages.
3. The GUI stands for Graphical User Interface, where a user graphically/visually interacts with the system.
4. The GUI programming is inherently event-driven; whenever a user initiates an activity such as a mouse activity, clicks, scrolling, etc., each is known as an event that is mapped to a code to respond to functionality to the user. This is known as event handling.
5. the Delegation Event Model in Java, the Delegation event model is based upon Event Source, Event Listeners, and Event Objects.

- Event Source is the class used to broadcast the events.
- Event Listeners are the classes that receive notifications of events.
- Event Object is the class object which describes the event.

Event Processing in Java

1. Java support event processing since Java 1.0. It provides support for AWT Abstract Window Toolkit), which is an API used to develop the Desktop application.
2. In Java 1.0, the AWT was based on inheritance. To catch and process GUI events for a program, it should hold subclass GUI components and override action() or handleEvent() methods. The below image demonstrates the event processing.



3. But, the modern approach for event processing is based on the Delegation Model.
4. It defines a standard and compatible mechanism to generate and process events.
5. In this model, a source generates an event and forwards it to one or more listeners. The listener waits until it receives an event.
6. Once it receives the event, it is processed by the listener and returns it.

7. The UI elements are able to delegate the processing of an event to a separate function.

Note:
The key advantage of the Delegation Event Model is that the application logic is completely separated from the interface logic.

8. In this model, the listener must be connected with a source to receive the event notifications.
9. Thus, the events will only be received by the listeners who wish to receive them.
10. So, this approach is more convenient than the inheritance-based event model (in Java 1.0).
11. In the older model, an event was propagated up the containment until a component was handled.
12. This needed components to receive events that were not processed, and it took lots of time. The Delegation Event model overcame this issue.
13. Basically, an Event Model is based on the following three components:

- Events
- Events Sources
- Events Listeners

Events

1. The Events are the objects that define state change in a source.
2. An event can be generated as a reaction of a user while interacting with GUI elements.
3. Some of the event generation activities are moving the mouse pointer, clicking on a button, pressing the keyboard key, selecting an item from the list, and so on.
4. The Events may also occur that may be not related to user interaction, such as a timer expires, counter exceeded, system failures, or a task is completed, etc. We can define events for any of the applied actions.

Event Sources

1. A source is an object that causes and generates an event.
2. It generates an event when the internal state of the object is changed.
3. The sources are allowed to generate several different types of events.
4. A source must register a listener to receive notifications for a specific event.
5. Each event contains its registration method.

"**public void** addTypeListener (TypeListener e1)"

6. The Type is the name of the event, and e1 is a reference to the event listener.
7. For example, for a keyboard event listener, the method will be called as **addKeyListener()**.
8. For the mouse event listener, the method will be called as **addMouseMotionListener()**.

9. When an event is triggered using the respected source, all the events will be notified to registered listeners and receive the event object. This process is known as event multi-casting.
10. When the specified event occurs, it will be notified to the registered listener. This process is known as **uni casting** events.
11. A source should contain a method that unregistered a specific type of event from the listener if not needed. Below is an example of the method that will remove the event from the listener.

Event Listeners

1. An event listener is an object that is invoked when an event triggers.
2. The listeners require two things; first, it must be registered with a source; however, it can be registered with several resources to receive notification about the events.
3. Second, it must implement the methods to receive and process the received notifications.
4. The methods that deal with the events are defined in a set of interfaces.
5. These interfaces can be found in the java.awt.event package.
6. For example, the MouseMotionListener interface provides two methods when the mouse is dragged and moved. Any object can receive and process these events if it implements the MouseMotionListener interface.

Types of Events

The events are categories into the following two categories:

The Foreground Events:

1. The foreground events are those events that require direct interaction of the user.
2. These types of events are generated as a result of user interaction with the GUI component.
3. For example, clicking on a button, mouse movement, pressing a keyboard key, selecting an option from the list, etc.

The Background Events :

1. The Background events are those events that result from the interaction of the end-user.
2. For example, an Operating system interrupts system failure (Hardware or Software).

To handle these events, we need an event handling mechanism that provides control over the events and responses.

The Delegation Model

The Delegation Model is available in Java since Java 1.1. it provides a new delegation-based event model using AWT to resolve the event problems. It provides a convenient mechanism to support complex Java programs.

Design Goals

The design goals of the event delegation model are as following:
1. It is easy to learn and implement.
2. It supports a clean separation between application and GUI code.
3. It provides robust event handling program code which is less error-prone (strong compile-time checking)
4. It is Flexible, can enable different types of application models for event flow and propagation.
5. It enables run-time discovery of both the component-generated events as well as observable events.
6. It provides support for the backward binary compatibility with the previous model.

Java Program to Implement the Event Delegation Model

The below is a Java program to handle events implementing the event delegation model:

TestApp.java:

```
import java.awt.*;
import java.awt.event.*;
public class TestApp {
   public void search() {
      System.out.println("Searching...");
   }
   public void sort() {
      System.out.println("Sorting....");
   }
   public void delete(){
      System.out.println("Deleting....");
   }
   static public void main(String args[]) {
      TestApp app = new TestApp();
      GUI gui = new GUI(app);
   }
}
class Command implements ActionListener  {
   static final int SEARCH = 0;
   static final int SORT = 1;
```

```
    static final int DELETE =2;
    int id;
    TestApp app;
    public Command(int id, TestApp app) {
        this.id = id;
        this.app = app;
    }
    public void actionPerformed(ActionEvent e) {
        switch(id) {
            case SEARCH:
                app.search();
                break;
            case SORT:
                app.sort();
                break;
            case DELETE:
                app.delete();
        }
    }
}
class GUI {
    public GUI(TestApp app) {
        Frame f = new Frame();
        f.setLayout(new FlowLayout());
        Command searchCmd = new Command(Command.SEARCH, app);
        Command sortCmd = new Command(Command.SORT, app);
        Command deleteCmd = new Command(Command.DELETE, app);
        Button b;
        f.add(b = new Button("Search"));
        b.addActionListener(searchCmd);
        f.add(b = new Button("Sort"));
        b.addActionListener(sortCmd);
        f.add(b = new Button("Delete"));
        b.addActionListener(deleteCmd);
        List l;
        f.add(l = new List());
        l.add("Coding Ninjas");
        l.add("CodeStudio");
        l.addActionListener(sortCmd);
        f.pack();
        f.show();
    }
}
```

Event Classes in Java

Event Class	Listener Interface	Description
ActionEvent	ActionListener	An event that indicates that a component-defined action occurred like a button click or selecting an item from the menu-item list.
AdjustmentEvent	AdjustmentListener	The adjustment event is emitted by an Adjustable object like Scrollbar.
ComponentEvent	ComponentListener	An event that indicates that a component moved, the size changed or changed its visibility.
ContainerEvent	ContainerListener	When a component is added to a container (or) removed from it, then this event is generated by a container object.
FocusEvent	FocusListener	These are focus-related events, which include focus, focusin, focusout, and blur.
ItemEvent	ItemListener	An event that indicates whether an item was selected or not.
KeyEvent	KeyListener	An event that occurs due to a sequence of keypresses on the keyboard.
MouseEvent	MouseListener & MouseMotionListener	The events that occur due to the user interaction with the mouse (Pointing Device).
MouseWheelEvent	MouseWheelListener	An event that specifies that the

Event Class	Listener Interface	Description
		mouse wheel was rotated in a component.
TextEvent	TextListener	An event that occurs when an object's text changes.
WindowEvent	WindowListener	An event which indicates whether a window has changed its status or not.

Different interfaces consists of different methods which are specified below.

Listener Interface	Methods
ActionListener	actionPerformed()
AdjustmentListener	adjustmentValueChanged()
ComponentListener	componentResized() componentShown() componentMoved() componentHidden()
ContainerListener	componentAdded() componentRemoved()
FocusListener	focusGained() focusLost()
ItemListener	itemStateChanged()
KeyListener	keyTyped() keyPressed() keyReleased()
MouseListener	mousePressed() mouseClicked()

Listener Interface	Methods
	mouseEntered() mouseExited() mouseReleased()
MouseMotionListener	mouseMoved() mouseDragged()
MouseWheelListener	mouseWheelMoved()
TextListener	textChanged()
WindowListener	windowActivated() windowDeactivated() windowOpened() windowClosed() windowClosing() windowIconified() windowDeiconified()

Flow of Event Handling

1. User Interaction with a component is required to generate an event.
2. The object of the respective event class is created automatically after event generation, and it holds all information of the event source.
3. The newly created object is passed to the methods of the registered listener.
The method executes and returns the result.

Code-Approaches

The three approaches for performing event handling are by placing the event handling code in one of the below-specified places.
1. Within Class
2. Other Class
3. Anonymous Class

Java Adapter Classes

Java adapter classes provide the default implementation of listener interfaces. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it saves code.

Pros of using Adapter classes:

1. It assists the unrelated classes to work combinedly.
2. It provides ways to use classes in different ways.
3. It increases the transparency of classes.
4. It provides a way to include related patterns in the class.
5. It provides a pluggable kit for developing an application.
6. It increases the reusability of the class.

Note: The adapter classes are found in java.awt.event, java.awt.dnd and javax.swing.event packages.

java.awt.event Adapter classes

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

java.awt.dnd Adapter classes

Adapter class	Listener interface
DragSourceAdapter	DragSourceListener
DragTargetAdapter	DragTargetListener

javax.swing.event Adapter classes

Adapter class	Listener interface
MouseInputAdapter	MouseInputListener

InternalFrameAdapter	InternalFrameListener

Java WindowAdapter Example

In the following example, we are implementing the WindowAdapter class of AWT and one its methods windowClosing() to close the frame window.

AdapterExample.java

```
// importing the necessary libraries
import java.awt.*;
import java.awt.event.*;
  public class AdapterExample {
// object of Frame
    Frame f;
// class constructor
    AdapterExample() {
// creating a frame with the title
        f = new Frame ("Window Adapter");
// adding the WindowListener to the frame
// overriding the windowClosing() method
        f.addWindowListener (new WindowAdapter() {
            public void windowClosing (WindowEvent e) {
                f.dispose();
            }
        });
         // setting the size, layout and
        f.setSize (400, 400);
        f.setLayout (null);
        f.setVisible (true);
    }
  // main method
public static void main(String[] args) {
    new AdapterExample();    }}
```