CS205 Object Oriented Programming in Java

Module 4 - **Advanced features of Java**

(Part 6)

Prepared by

Renetha J.B.

AP

Dept.of CSE,

Lourdes Matha College of Science and Technology

Topics

- ☑ **Event handling:**
  - ❑ **Event Handling Mechanisms**
  - ❑ **Delegation Event Model**

Event Handling

- There are several types of events,
  - Events can be generated by
    - the mouse (click, move, drag mouse etc.)
    - the keyboard (type, press, release etc.)
    - different GUI controls, such as a
      - push button
      - scroll bar
      - check box
- **Event Handling** is the mechanism that **controls the event** and **decides what should happen if an event occurs**.

Event Handling(contd.)

- Events are supported by a number of packages, including

  java.util

  java.awt

  java.awt.event

- Event handling is an integral part in the **creation** of **applets** and other types of **GUI-based programs**.
- Applets are **event-driven programs** that use a graphical user interface(GUI) to interact with the user.
- Any program that uses a graphical user interface is event driven.
  - Thus, we cannot write these types of programs without a solid command of event handling.

Event Handling Mechanisms

- The **two ways** in which events are handled changed significantly between the (Two event handling mechanisms)
  - **original version of Java (1.0) event handling** and
  - **modern versions of Java (beginning with version 1.1) event handling**
- The 1.0 method of event handling is still supported, but it is not recommended for new programs.
  - Many of the methods that support the old 1.0 event model have been deprecated.
- The **modern approach** is the way that **events should be handled by all new programs**

The Delegation Event Model

- The **modern approach** to handling events is based on the ***delegation event model***
  - It defines standard and consistent mechanisms to **generate and process events.**
- Concept of ***delegation event model*** **:**
  - A **source** **generates an event** and **sends** it to one or more **listeners**.
  - In this scheme, the listener simply **waits until it receives an event**.
  - Once an event is received, the listener processes the event and then returns.

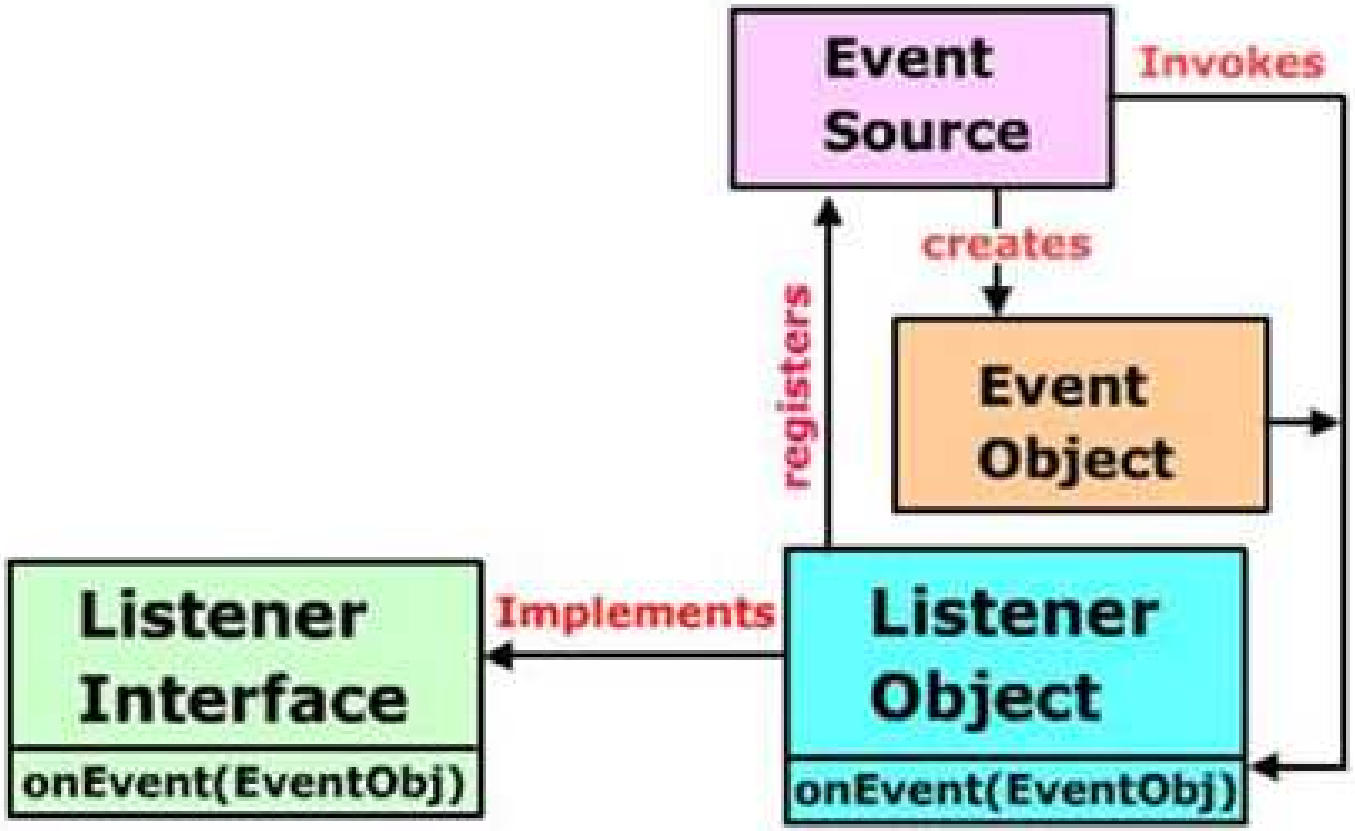
The Delegation Event Model(contd.)

- The **advantage** of **Delegation Event Model** is that the **application logic** that **processes events** is cleanly separated from the **user interface logic** that **generates those events.**
- A user interface element is able **to "delegate"(entrust) the processing of an event to a separate piece of code.**
- In the delegation event model, **listeners must register with a source in order to receive an event notification**.
  - **Benefit:** Notifications are sent only to listeners that want to receive them.

The Delegation Event Model(contd.)

- In previous models, an event was *propagated up the containment hierarchy* **until it was handled** by a component.
  - This required components to receive events that they did not process, and it wasted valuable time.
- **The delegation event model eliminates this overhead**

The Delegation Event Model(contd.)



The Delegation Event Model -Event

- In the delegation model, an **event** is an **object** that describes a **state change in a source.**
- Events can be caused with or without user interaction.

The Delegation Event Model -Event

- Some events are caused by interactions with a user interface such as:
  - pressing a button,
  - entering a character via the keyboard,
  - selecting an item in a list,
  - clicking the mouse.

The Delegation Event Model –Event (contd.)

- Events may also occur that are not directly caused by interactions with a user interface.
  - Example: an **event** may be generated
    - when a timer expires,
    - a counter exceeds a value,
    - a software or hardware failure occurs,
    - an operation is completed

The Delegation Event Model – Event Sources

- A **event source** is an **object** that **generates an event.**
  - Event occurs when the internal state of that object changes in some way.
- Sources may generate more than one type of event.
- A **source** must **register listeners,** then only listeners can receive notifications about a specific type of event.
- Each type of event has its own registration method.

The Delegation Event Model – Event Sources (contd.)

General form of listener registration is:

public void ***add****TypeListener*(*TypeListener* el)

- Type is the name of the event, and el is a reference to the event listener.
- For example, the method that **registers a keyboard event listener** is called **addKeyListener( ).**
- The method that **registers a mouse motion listener** is called **addMouseMotionListener( ).**

The Delegation Event Model – Event Sources (contd.)

- When an event occurs, **all registered listeners are notified** and receive a copy of the event object. This is known as ***multicasting*** the event.
  - In all cases, notifications are sent only to listeners that register to receive them.
- Some sources may allow **only one listener to register.** The general form of such a method is this:

```
public void addTypeListener(TypeListener el) throws
java.util.TooManyListenersException
```

  - When such an event occurs, that single registered listener is notified. This is known as **unicasting** *the event.*

The Delegation Event Model – Event Sources (contd.)

- A source must also provide a method that allows a **listener to unregister** an interest in a specific type of event. The general form of such a method is this:

```
public void removeTypeListener(TypeListener el)
```

  - Here, *Type is the name of the event, and el is a reference to the event listener.*

  – *For example,* to remove a keyboard listener, call **removeKeyListener( ).**

- The **methods that add or remove listeners** are provided by the **source** that generates events.

  – For example, the **Component class** provides methods to add and remove keyboard and mouse event listeners.

The Delegation Event Model – Event Listeners

- A **listener** is an object that is **notified when an event occurs.**
- It has two major requirements.
  - First, it must have been **registered with one or more sources** to receive notifications about specific types of events.
  - Second, it **must implement methods** to receive and process these notifications.

The Delegation Event Model – Event Listeners (contd.)

- The methods that receive and process events are defined in a set of interfaces found in **java.awt.event.**
  - For example, the **MouseMotionListener interface** defines two methods to **receive notifications when the mouse is dragged or moved.**
    - Any object may receive and process one or both of these events if it provides an implementation of this interface.

Reference

- Herbert Schildt, Java: **The Complete Reference, 8/e, Tata McGraw Hill, 2011**.