




# Unit VIII

## Event Handling

Tutor: Pema Galey

# Learning Outcomes

In this session, you will learn about:

- Event Handling
  - Event Sources
  - Event Listeners
  - Event Handlers
- Adaptor Classes

# Introduction to Event Handling

- An object that describes a change of state in a source component is called an **event**.
- In Java, events are supported by the classes and interfaces defined in the `java.awt.event` package.

- Identifying the sources of events:
  - An event source is an object that generates an event.
  - Event sources are the AWT GUI components, such as buttons, choice lists, and scroll bars.
  - An event source can generate various types of events depending upon the change of state of the event source.

# Event Handling Process

- Event Handling process.

# Event Handling

- The following table lists the various **event sources** and the types of events they generate.

| ***Event Source*** | ***Description*** |
|---|---|
| *Checkbox* | *Creates an item event when a check box is selected or deselected.* |
| *Button* | *Creates an action event when a button is pressed.* |
| *List* | *Creates an action event when an item is double-clicked; creates an item event when an item is selected or deselected.* |
| *Scrollbar* | *Creates an adjustment event when a scroll bar is scrolled.* |
| *Text components* | *Creates a text event when a text character is entered in the text component.* |
| *Window* | *Creates a window event when a window is activated, deactivated, opened, closed, or quit.* |

# Event Handling

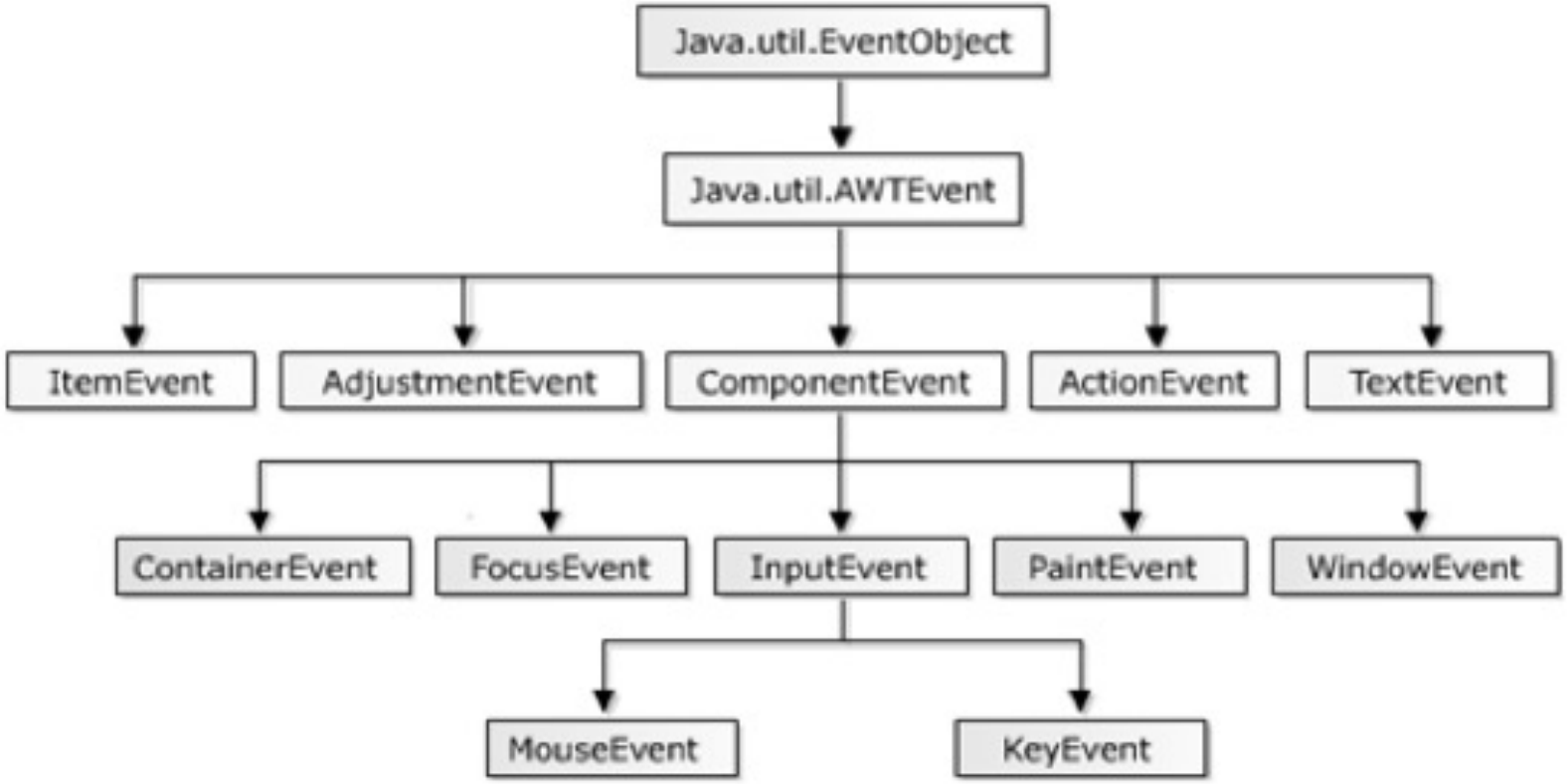
- Event listeners and event handlers:
  - An **event listener**:
    - Listens for a specific event and is notified when that specific event occurs.
    - Registers with one or more event sources to receive notifications about specific types of events and processes the events.
  - An **event-handler**:
    - Is called by the event listener whenever a specific event occurs.

# Event Handling

- The **delegation event model**:
  - Is based on the concept that source generates the event and notifies one or more event listeners.
  - Allows you to specify the objects that are to be notified when a specific event occurs.

# Event Classes

- At the root of the Java event class hierarchy is **EventObject**, which is in **java.util**. It is the superclass for all events.



# Event Classes

- The `ActionEvent` class:
  - **ActionEvent** is generated by an AWT component, when a component-specific action is performed.
  - The following list describes the various constructors of the ActionEvent class:
    - `public ActionEvent(Object source, int id, String command)`
    - `public ActionEvent(Object source, int id, String command,int modifiers)`
  - The various methods included in the **ActionEvent** class are:
    - `String getActionCommand()`
    - `int getModifiers()`

# Event Classes

The **MouseEvent** class:

- It extends the java.awt.event.InputEvent class.
- It is used for the following events:
  - **Mouse events**:

  1) Pressing a mouse button & 2) Releasing a mouse button

  3) Clicking a mouse button & 4) Entering of mouse in a component area

  5) Exiting of mouse from a component area

  - **Mouse motion events**:

  1) Moving a mouse & 2)Dragging a mouse
- It defines some integer constants that can be used to identify several types of mouse events.

# Event Classes

- The following table lists some integer constants defined in the **`MouseEvent`** class.

| ***Constant*** | ***Description*** |
|---|---|
| `MOUSE_CLICKED` | *Identifies the event of mouse clicking.* |
| `MOUSE_DRAGGED` | *Identifies the event of dragging of mouse.* |
| `MOUSE_MOVED` | *Identifies the event of mouse moving.* |
| `MOUSE_PRESSED` | *Identifies the event of mouse pressing.* |
| `MOUSE_RELEASED` | *Identifies the event of mouse releasing.* |
| `MOUSE_ENTERED` | *Identifies the event of mouse entering an AWT component.* |
| `MOUSE_EXITED` | *Identifies the event of mouse exiting an AWT component.* |

# Event Classes

The syntax of one of the constructors of the **MouseEvent** class is:

```
public MouseEvent(Component source, int eventType,long when, int
modifiers, int x, int y, int clickCount, boolean triggersPopup)
```

| ***Parameter*** | ***Description*** |
|---|---|
| `source` | *A component that generates an event* |
| `eventType` | *An integer that specifies an event ID* |
| `when` | *A long int that gives the time of occurrence of an event* |
| `modifiers` | *The modifiers keys, such as CTRL, ALT, and SHIFT are pressed when an event is triggered* |
| `X` | *The horizontal X coordinates for mouse position* |
| `Y` | *The vertical Y coordinates for mouse position* |
| `clickCount` | *The number of mouse clicks associated with a mouse event* |
| `triggersPopup` | *A boolean value that is true if an event causes a popup menu to appear* |

# Event Classes

The following table lists some of the commonly used methods in the `MouseEvent` class.

| Method | Description |
|---|---|
| `public int getX()` | Returns the horizontal x coordinate of the mouse position relative to a source component. |
| `public int getY()` | Returns the vertical y coordinate of the mouse position relative to a source component. |
| `public point getPoint()` | Returns the `Point` object. The `Point` object contains the x and y coordinates of the mouse position relative to a source component. |
| `public void translatePoint(int x, int y)` | Translates the coordinates of a mouse event to a new position by adding x and y offsets. |
| `public int getClickCount()` | Returns the number of mouse clicks associated with an event. |

# Event Listener Interfaces

An **Event Listener**:

- Registers with an event source to receive notifications about the events of a particular type.
- Is created by implementing one or more event listener interfaces defined by the `java.awt.event` package.
- Interface defines one or more methods to handle events.
- Handles different types of the events generated by the AWT components are defined in the `java.awt.event` package.

# Event Listener Interfaces

Some of the event listener interfaces from **java.awt.event** package.

| *Interface* | *Description* |
|---|---|
| `ActionListener` | *Defines the* `actionPerformed()` *method to receive and process action events.* |
| `MouseListener` | *Defines five methods to receive mouse events, such as when a mouse is clicked, pressed, released, enters, or exits a component.* |
| `MouseMotionListener` | *Defines two methods to receive events, such as when a mouse is dragged or moved.* |
| `AdjustmentListner` | *Defines the* `adjustmentValueChanged()` *method to receive and process the adjustment events.* |
| `TextListener` | *Defines the* `textValueChanged()` *method to receive and process an event when the text value changes.* |
| `WindowListener` | *Defines seven window methods to receive events. These methods are:* `windowActivated()`, `windowClosed()`, `windowClosing()`, `windowDeactivated()`, `windowDeiconified()`, `windowIconified()`, *and* `windowOpened()`. |
| `ItemListener` | *Defines the* `itemStateChanged()` *method when an item has been selected or deselected by the user.* |

# Event Listener Interfaces

### The ActionListener Interface

- This interface defines the **actionPerformed( )** method that is invoked when an action event occurs.

```
void actionPerformed(ActionEvent ae)
```

### The AdjustmentListener Interface

- This interface defines the **adjustmentValueChanged( )** method that is invoked when an adjustment event occurs.

```
void adjustmentValueChanged(AdjustmentEvent ae)
```

# Example

```
import java.awt.*;
import java.awt.event.*;

public class ButtonDemo extends Frame implements ActionListener{
        Button submit;
        public ButtonDemo(){
                submit=new Button(”Submit");
                submit.addActionListener(this);
        }
        public void actionPerformed(ActionEvent ae){
                String str=ae.getActionCommand();

                if(str.equals(”Submit")){
                        //…
                }
        }
         public static void main(String arg[]){}
}
```

# Event Listener Interfaces

## The ItemListener Interface:

- This interface defines the **itemStateChanged( )** method that is invoked when the state of an item changes.

```
void itemStateChanged(ItemEvent ie)
```

## The KeyListener Interface

- This interface defines three methods. The `keyPressed( )` and `keyReleased( )` methods are invoked when a key is pressed and released, respectively. The `keyTyped( )` method is invoked when a character has been entered.

# Event Listener Interfaces

**The MouseListener Interface:** is implemented for receiving various mouse events, such as when you press, click, release, enter, and exit.

| *Method* | *Description* |
|---|---|
| `void mouseClicked(MouseEvent me)` | *Performs an action when the mouse button clicks a component.* |
| `void mousePressed(MouseEvent me)` | *Performs an action when mouse button presses a component.* |
| `void mouseReleased(MouseEvent me)` | *Performs an action when the mouse button releases a component.* |
| `void mouseEntered(MouseEvent me)` | *Performs an action when a mouse enters a component area.* |
| `void mouseExited(MouseEvent me)` | *Performs an action when a mouse exits a component area.* |

# Event Listener Interfaces

Using the **MouseMotionListener** interface:

- A mouse motion event is generated when the mouse is moved or dragged.
- A class that processes any mouse motion event implements the MouseMotionListener interface.
- The following syntax is used to define the MouseMotionListener interface:
  - `public interface MouseMotionListener extends EventListener`
- The following list represents the various methods of the MouseMotionListener interface:
  - `public void mouseDragged(MouseEvent me)`
  - `public void mouseMoved(MouseEvent me)`

# Adapter Classes

# Adapter Classes

- An adapter class provides an empty implementation of the event-handling methods in an event listener interface.
- The adapter classes are useful as they allow you to receive and process only some of the events that are handled by a particular event listener interface.

# Adapter Classes

| Adapter Class | Description |
|---|---|
| `KeyAdapter` | Provides empty implementation of the methods of the `KeyListener` interface, such as `keyPressed()`, `keyReleased()`, and `keyTyped()` methods for receiving keyboard events. |
| `MouseAdapter` | Provides empty implementation of the methods of the `MouseListener` interface, such as `mouseClicked()`, `mousePressed()`, `mouseExited()`, `mouseEntered()`, and `mouseReleased()` methods for receiving mouse events. |
| `MouseMotionAdapter` | Provides empty implementation of the methods of the `MouseMotionListener` interface, such as `mouseDragged()` and `mouseMoved()` methods for receiving mouse motion events. |
| `WindowAdapter` | Provides empty implementation of the methods of the `WindowListener` and `WindowFocusListener` interfaces, such as `windowActivated()`, `windowClosed()`, `windowClosing`, `windowOpened()`, and `windowGainedFocus()` methods for receiving window events. |
| `FocusAdapter` | Provides empty implementation of the methods of the `FocusListener` interface, such as `focusGained()` and `focusLost()` methods for receiving keyboard focus events. |

# Adapter Classes

**Using the MouseAdapter class:**

- The `MouseAdapter` class provides empty implementation of mouse event-handling methods.
- A class that acts as a listener for mouse events extends the `MouseAdapter` class and overrides the required methods.

**Using the MouseMotionAdapter class:**

- The `MouseMotionAdpater` class provides empty implementation of the mouse motion event-handling methods.
- A class that acts as a listener for mouse motion events extends the `MouseAdapter` class and overrides the required method.

# Example:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class ListenerClass extends MouseAdapter{
        public void mouseClicked(MouseEvent me){
                AdapterTesterDemo at;
                at=(AdapterTesterDemo)me.getSource();
                at.l.setText("Mouse clicked at: "+me.getX()+", "+me.getY());
        }
}

class AdapterTesterDemo extends JFrame implements ActionListener{

        //codes

}
```

# Lab Work

Design a registration form by including following information:

- Name → TextField
- Student_ID → TextField
- Date of Birth (DD/MM/YYYY) → Dropdown List
- Programme → Dropdown List
- Semester → Dropdown List
- Contact Number → TextField (Check number format i.e. 8 digits only)
- Address → TextArea
- **Submit & Reset** Button → Button

*While user click on **Submit** button, display all the entered information in new page. And if user click on Reset button, clear all the entered information*

# Lab Work

- Complete the menu task of **MenuDemo** with certain actions as follows:
  - Save
  - Open
  - Exit/Close
  - Many more….

- Draw a **rectangle** shape using MouseMotionListener interface or MouseMotionAdapter class.

# Thank you!