Unit 2: Event Delegation Model

2

Unit Structure

2.1 LEARNING OBJECTIVE

The objective of this unit is to make the students,

- To learn, understand Event
- To learn, understand Event Source and Event Handlers
- To learn, understand and define different Event and Listeners for various kinds of Events
- To learn, understand adapter classes and their importance

2.2 OUTCOMES

After learning the contents of this chapter, the students will be able to:

- Define different events
- Write event source and event handlers to handle the events
- Adapter classes when they are in need of few methods instead of all methods of handlers

2.3 INTRODUCTION

Java provides the platform to develop interactive GUI application using the AWT and Event classes. This unit discusses various event classes for handling various events like button click, checkbox selection etc. We can define an event as the change in the state of an object when something changes within a graphical user interface. If a user check or uncheck redio button, clicks on a button, or write characters into a text field etc. then an event trigger and creates the relevant event object. This mechanism is a part of Java's Event Delegation Model

2.4 EVENT DELEGATION MODEL

The Event Delegation Model is based on the concept of source and listener. A source triggers an event and sends it to one or more registered listeners. On receiving the event notification, listener processes the event and returns it. The important feature is that the source has a list of registered listeners which will be informed as and when event take place. Only the registered listeners will actually

receive the notification when a specific event is generated. Generally the event Handling is a three step process:

a. Create controls which can generate events (Event Generators).

b. Build objects that can handle events (Event Handlers).

c. Register event handlers with event generators.

2.4.1 EVENT GENERATORS

It is an object that is responsible to generate a particular kind of event. An event is generated when the internal state of an object is changed. A source may trigger more than one kind of event. Every source must register a list of listeners that are interested to receive the notifications when an event is generated. Event source has methods to add or remove listeners.

To register (add) a listener the signature of method is:

public void addNameListener(NameListener eventlistener)

To unregister (remove) a listener the signature of method is:

public void removeNameListener(NameListener eventlistener)

where,

Name is the name of the event and eventlistener is a reference to the event listener.

2.4.2 EVENT LISTENER

An event listener is an object which receives notification when an event is triggered. As already said only registered listeners will receive notifications from the event sources about specific kinds of events. The event listener is responsible to receive these notifications and process them. Technically these listeners are interfaces having various abstract methods for event handling. These interfaces needs to be implemented in the class where the object or source will trigger the event.

For example, consider an action event represented by the class ActionEvent, which is triggered when a user clicks a button or the item of a list. At the user's interaction, an ActionEvent object related to the relevant action is created. This

object will contain both the event source and the specific action taken by the user. This event object is then passed to the related ActionListener object's method:

void actionPerformed(ActionEvent e)

This method will be executed and returns the appropriate response to the user.

2.4.3 REGISTRATION OF LISTENER FOR EVENTS

As we know, we have implemented the interface and set up the methods which will listen for these events and trigger the functionality accordingly. To perform this, we have to use an event listener. To use an event listener the addActionListener() method will be used on the component that will listen for these events - the button.

button.addWindowListener(this);

2.5 TYPES OF EVENTS

There are various types of events that can happen in a Java program. They are,

Event Class / Type	**Generated when**	**Listener Interface**	**Methods to implement**
Action event	Button is pressed	ActionListener	actionPerformed()
Adjustment event	Scroll bar is manipulated	AdjustmentListener	adjustmentValueChanged()
Component event	A control is hidden, moved, resized, or shown	ComponentListener	componentHidden(), componentMoved(), componentResized(), componentShown()
Container event	A control is added or removed from a	ContainerListener	componentAdded(), componentRemoved()

	container		
Focus event	A control gains or loses focus	FocusListener	focusGained(), focusLost()
Item event	An item is selected or deselected	ItemListener	itemStateChanged()
Key event	A key is pressed, released or typed	KeyListener	keyPressed(), keyReleased(), keyTyped()
Mouse event	Mouse is clicked, pressed or released. Mouse pointer enters, leaves a component	MouseListener	mouseClicked(), mouseEntered(), mouseExited(), mousePressed(), mouseReleased()
Mouse event	Mouse is dragged or moved	MouseMotionListener	mouseDragged(), mouseMoved()
Text event	Text value is changed	TextListener	textValueChanged()
Window event	A window is activated, closed, deactivated, deiconfied,	WindowListener	windowActivated(), windowClosed(), windowClosing(), windowDeactivated(),

	opened or quit		windowDeiconified(), windowIconified(), windowOpened()

Table-10: Event classes and their methods

Each interface has their own methods to use to execute some code when certain events occur. For example, the ActionListener interface has a actionPerformed method that can be used to execute some code when a button is clicked. When we implement an interface, we have to define all of it's abstract methods in the program.

Let's check a simple example that will have window events.

```
import java.awt.*;
import java.awt.event.*;
public class winEvents extends Frame implements WindowListener{
}
```

Now we need to implement the methods of the WindowListener interface to specify what happens during window events.

```
//Window event methods
public void windowClosing(WindowEvent we)
{ System.out.println("The frame is closing"); }
public void windowClosed(WindowEvent we)
{ System.out.println("The frame is closed"); }
public void windowDeactivated(WindowEvent we)
{ System.out.println("The frame is deactivated"); }
```

Example:

In the below program, a frame utilizes all the window event methods. See how the program displays different messages as we perform different actions such as minimize and maximize on the frame.

```
import java.awt.*;
import java.awt.event.*;
public class WinEvents extends Frame implements WindowListener
{
  public WinEvents(String str){
    super(str);
   addWindowListener(this);
  }
  public static void main(String[] args){
    Frame fm = new WinEvents("WindowEvent_Example");
    fm.setSize(250, 250);
    fm.setVisible(true);
  }
  public void windowClosing(WindowEvent we){
    System.out.println("The window is closing.....");
    ((Window)we.getSource()).dispose();
  }
  public void windowClosed(WindowEvent we){
    System.out.println("The window has been closed!");
    System.exit(0);
  }
  public void windowActivated(WindowEvent we){
    System.out.println("The window has been activated");
  }
  public void windowDeactivated(WindowEvent we){
    System.out.println("The window has been deactivated");
  }
  public void windowDeiconified(WindowEvent we){
    System.out.println("The window has been restored from a minimized state");
  }
  public void windowIconified(WindowEvent we){
    System.out.println("The window has been minimized");
  }
  public void windowOpened(WindowEvent we){
    System.out.println("The window is now visible");
```

```
  }
}
```

Output: When we perform different operation on window following output will be displayed.

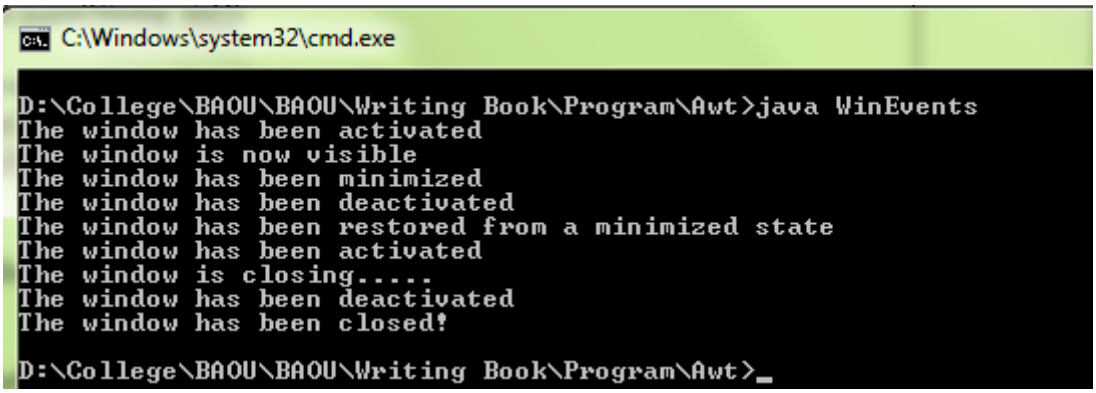

Figure-111: Output of program

After discussing all the window event methods, let us check the key events. The following program depicts the use of KeyListener to handle different key events.

```
import java.awt.BorderLayout;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.Frame;
import java.awt.TextField;
import java.awt.TextArea;
import java.awt.Label;
public class keyListenerTest extends Frame implements KeyListener
{
        TextArea text;
        TextField txtF;
        Label l1;
  keyListenerTest(String str)
  {
          super(str);
          setLayout(null);
          l1=new Label("Enter Key:");
           l1.setBounds(50,50,100,30);
          txtF= new TextField();
```

```
            text = new TextArea();
            txtF.addKeyListener(this);
            txtF.setBounds(160,50,100,30);
             text.setBounds(20,100,300,300);
            add(l1);
            add(txtF);
            add(text);
  }
        public void keyPressed(KeyEvent ke)
        {
                text.append("Key is Pressed\n");
        }
        public void keyReleased(KeyEvent ke)
        {
                text.append("Key is Released\n");
        }
        public void keyTyped(KeyEvent ke)
        {
                text.append("Key is Typed\n");
        }
        public static void main(String args[])
        {
                  Frame frame = new keyListenerTest("KeyListener");
                   frame.setSize(350,1400);
                  frame.setVisible(true);
        }
}
```

Output:

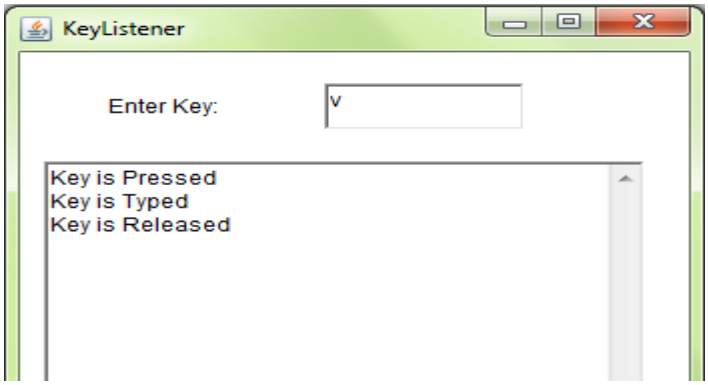

Figure-112: Output of program

➢ **Check Your Progress 1**

1) Define Event. Which Interface is extended by all awt Event Listener?

……

……

2) Write a code to register actionListener for button event.

……

……

3) Differentiate between mouseListener and mouseMotionListener.

……

……

2.6 ADAPTER CLASSES

We have seen that handler class need to implement interface therefore it has to provide implementation of all the methods of that interface. Suppose, an interface has 10 methods, then hander class has to provide implementation of all these 10 methods. Even if the requirement is for one method, class has to provide empty implementation of the remaining 9 methods with null bodies. This becomes a irritating job for a programmer.

Adapter classes are classes that implement all of the methods in their corresponding interfaces with null bodies. If the programmer needs one of the

methods of particular interface then he / she can extend an adapter class and override its methods. They belongs to java.awt.event package.

There is an adapter class for listener interfaces having more than one event handling methods. For example, for WindowListener there is a WindowAdapter class and for MouseMotionListener there is a MouseMotionAdapter class and many more.

Adapter classes provide definitions for all the methods (empty bodies) of their corresponding Listener interface. It means that WindowAdapter class implements WindowListener interface and provide the definition of all methods inside that Listener interface. Consider the following example of WindowAdapter and its corresponding WindowListener interface:

```
public interface WindowListener{
        public void windowOpened ( WindowEvent e )
        public void windowIconified ( WindowEvent e )
        public void windowDeiconified ( WindowEvent e )
        public void windowClosed ( WindowEvent e )
        public void windowActivated ( WindowEvent e )
        public void windowDeactivated ( WindowEvent e )
}
public class WindowAdapter implements WindowListener {
        public void windowOpened ( WindowEvent e ) {}
        public void windowIconified ( WindowEvent e ) {}
        public void windowDeiconified ( WindowEvent e ) {}
        public void windowClosed ( WindowEvent e ) {}
        public void windowActivated ( WindowEvent e ) {}
        public void windowDeactivated ( WindowEvent e ) {}
}
```

Now in the below class WinEvents, if we extend the above handler class then due to inheritance, all the methods of the adapter class will be available inside handler class as adapter classes has already provided implementation with empty bodies. So, we only need to override and provide implementation of method of our interest.

```
public class WinEvents extends WindowAdapter{...}
```

Example: Following program demonstrates the use of WindowAdapter class.

```
import java.awt.*;
import java.awt.event.*;
public class adapterTest extends Frame
{
	Label lblTest;
	adapterTest(String str)
	{
		super(str);
		setLayout(new FlowLayout(FlowLayout.LEFT));
		lblTest = new Label();
		add(lblTest);
		addMouseListener(new MyAdapter(lblTest));
	}
	public static void main(String str[])
	{
		Frame at=new adapterTest("AdapterClass");
		at.setSize(250,250);
		at.setVisible(true);
	}
}
class MyAdapter extends MouseAdapter
{
	Label lblTest;
	MyAdapter(Label lbl)
	{
		lblTest = lbl;
	}
	public void mouseClicked(MouseEvent me)
	{
		lblTest.setText("Mouse is Clicked");
	}
}
```

Output:

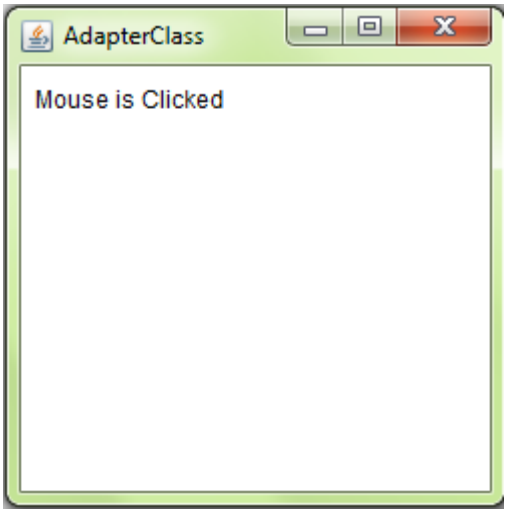

Figure-113: Output of program

➢ Check Your Progress 2

1) What is the use of WindowListener?

……

……

2) What is the use of the Window class?

……

……

3) Why do we use adapter class? List all adapter classes of java AWT.

……

……

The following example handles action event along with item event when a user clicks a button, check a checkbox or changes a value from choice.

Example:

```
import java.awt.*;
import java.awt.event.*;
public class AwtControl extends Frame
implements ActionListener, ItemListener
{
        Button button;
        Checkbox mca;
        Choice city;
        TextField TxtF,TxtF1,TxtF2;
        AwtControl(String str)
        {
           super(str);
                setLayout(new FlowLayout());
                button = new Button("BAOU");
                mca = new Checkbox("MCA");
                city = new Choice();
                TxtF = new TextField(50);
                TxtF1 = new TextField(50);
                TxtF2 = new TextField(50);
                button.addActionListener(this);
                mca.addItemListener(this);
                city.addItemListener(this);
                city.addItem("Ahmedabad");
                city.addItem("Sadra");
                city.addItem("Randheja");

                add(button);
                add(mca);
                add(city);
                add(TxtF); add(TxtF1);add(TxtF2);

        }
        public void actionPerformed(ActionEvent e)
        {
                String action = e.getActionCommand();
```

```
                if(action.equals("BAOU"))
                {
                        TxtF.setText("BAOU is in Ahmedabad");
                }
        }
        public void itemStateChanged(ItemEvent e)
        {
                if (e.getSource() == mca)
                {
                        TxtF1.setText("MCA at Gujarat Vidyapith: " + mca.getState() +
".");
                }
                else if (e.getSource() == city)
                {
                        TxtF2.setText(city.getSelectedItem() + " is selected.");
                }
        }
        public static void main(String arg[])
        {
                Frame frame = new AwtControl("AWT Event");
             frame.setSize(1400,200);
                  frame.setVisible(true);
        }
}
```

Output:

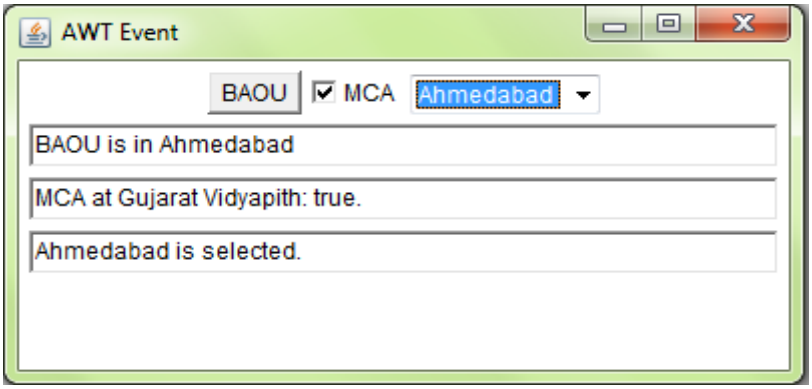

Figure-114: Output of program

2.7 LET US SUM UP

Throughout the unit, we saw that AWT provides various event classes and listener interfaces to fire and handle events triggered through user interaction. We have seen that how ActionEvent class helps to handle the event triggered by button like controls. Same way, in the unit we have discussed various other Event classes and their respective listener for event handling. At last, we have discussed Adapter class to relieve the programmer from writing all the listener methods.

2.8 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

- **Check Your Progress 1**

1. Event is an act which indicates the changes in the status of an Object. The java.util.EventListener interface is extended by all the AWT event listeners.

2.

```
public class handler implements ActionListener
{
        handler()
        {
                Button jb;
                jb=new Button("Click");
                // Registering Event listener with object.
                jb.addActionListener(this);
        }
        public void actionPerformed(ActionEvent ae)
         {
           System.out.println("Button Clicked");
         }
}
```

3. MouseListener handles event when mouse is Released, Clicked, Exited, Entered or pressed while MouseMotionListener handles event when mouse is Dragged or Moved.

➢ **Check Your Progress 2**

1. WindowListener interface is implemented to handle the tasks like Opening a window, Closing a window, Iconifying a window, Deiconifying a window, Activating a window, Deactivating a window and Maximizing the window.
2. The window class can be used to create a plain, open window without a border or menu. Sometimes, the window class can also be used to display introduction or welcome screens.
3. The main benefit of adapter class is that we can override any one or two methods we wants instead of all methods of an interface. But with the listener, we must have to override all the abstract methods. For example, to minimize the window, all the 7 abstract methods of WindowListener should be overridden atleast with empty bodies. But if we use WindowAdapter class then we need to implement method windowIconified().
   Different adapter classes are WindowAdapter, MouseAdapter, MouseMotionAdapter and KeyAdapter.

2.9 FURTHER READING

7) Java: The Complete Reference by Schildt Herbert. Ninth Edition
8) Let Us Java by Yashavant Kanetkar. 3rd Edition
9) Core Java Volume I — Fundamentals by Cay S. HorstMann, Gary Cornell, 9th Edition
10) https://www.javatpoint.com/

2.10 ASSIGNMENTS

6) Discuss Event delegation model with diagram in detail.
7) Explain different methods of KeyListener with its signature.
8) Write a program to implement a single key event with the help of adapter classes.