Unit 4

Event Handling:-

1. **Events**
2. **Event sources**
3. **Event classes**
4. **Event Listeners**
5. **Delegation event model**
6. **Handling mouse events**
7. **Handling keyboard events**
8. **Adapter classes**

Event Handling:-

1. Events

What is an Event?

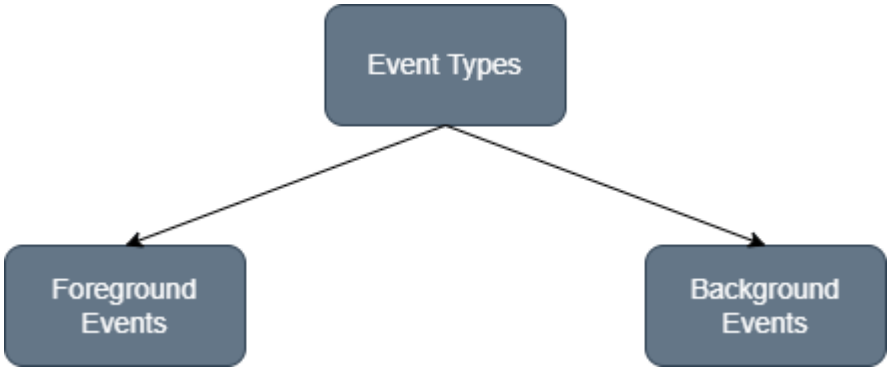
Change in the state of an object is known as event i.e. event describes the change in state of source. Events are generated as result of user interaction with the graphical user interface components.

For example, clicking on a button, moving the mouse, entering a character through keyboard,selecting an item from list, scrolling the page are the activities that causes an event to happen.

Java provides a package **java.awt.event** that contains several event classes.

We can classify the events in the following two categories:

1. Foreground Events
2. Background Events



Foreground Events

Foreground events are those events that require user interaction to generate. In order to generate these foreground events, the user interacts with components in GUI. When a user clicks on a button, moves the cursor, and scrolls the scrollbar, an event will be fired.

Background Events

Background events don't require any user interaction. These events automatically generate in the background. OS failure, OS interrupts, operation completion, etc., are examples of background events.

2.Event sources

A source is an object that generates an event. Generally sources are components. Sources may generate more than one type of event. A source must register listeners in order for the listeners to receive notifications about a specific type of event. Each type of event has its own registration method. Here is the general form:

Syntax:-

public void addTypeListener (TypeListener el)

Here, 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( ).**

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)

Sources of Events:

Event Source	Description
Button	Generates action events when the button is pressed.
Check box	Generates item events when the check box is selected or deselected.
Choice	Generates item events when the choice is changed.
List	Generates action events when an item is double-clicked;
Menu item	Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected.
Scroll bar	Generates adjustment events when the scroll bar is manipulated.
Text components	Generates text events when the user enters a character.
Window	Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Registration Methods:

For registering the component with the Listener, many classes provide the registration methods. For example:

 Button

o public void addActionListener(ActionListener a){}

 MenuItem

o public void addActionListener(ActionListener a){}

 TextField

o public void addActionListener(ActionListener a){}

o public void addTextListener(TextListener a){}

 TextArea

o public void addTextListener(TextListener a){}

 Checkbox

o public void addItemListener(ItemListener a){}

 Choice

o public void addItemListener(ItemListener a){}

 List

o public void addActionListener(ActionListener a){}

o public void addItemListener(ItemListener a){}

 Mouse

o public void addMouseListener(MouseListener a){}

4.Event Listeners

A listener is an object that is notified when an event occurs. It has two major requirements.

1.It must have been registered with one or more sources to receive notifications about specific types of events.

2. It must implement methods to receive and process these notifications.The methods that receive and process events are defined in a set of interfaces found in java.awt.event package.

An event listener registers with an event source to receive notifications about the events of a particular type. Various event listener interfaces defined in the java.awt.event

package are given below:

Interface	Description
ActionListener	Defines the actionPerformed() method to receive and process action events. *void actionPerformed(ActionEvent ae)*
MouseListener	Defines five methods to receive mouse events, such as when a mouse is clicked, pressed, released, enters, or exits a component *void mouseClicked(MouseEvent me)* *void mouseEntered(MouseEvent me)* *void mouseExited(MouseEvent me)* *void mousePressed(MouseEvent me)* *void mouseReleased(MouseEvent me)*
MouseMotionListener	Defines two methods to receive events, such as when a mouse is dragged or moved. *void mouseDragged(MouseEvent me)* *void mouseMoved(MouseEvent me)*
AdjustmentListner	Defines the adjustmentValueChanged() method to receive and process the adjustment events. *void adjustmentValueChanged(AdjustmentEvent ae)*
TextListener	Defines the textValueChanged() method to receive and process an event when the text value changes. *void textValueChanged(TextEvent te)*
WindowListener	Defines seven window methods to receive events. *void windowActivated(WindowEvent we)* *void windowClosed(WindowEvent we)* *void windowClosing(WindowEvent we)* *void windowDeactivated(WindowEvent we)* *void windowDeiconified(WindowEvent we)* *void windowIconified(WindowEvent we)* *void windowOpened(WindowEvent we)*
ItemListener	Defines the itemStateChanged() method when an item has been *void itemStateChanged(ItemEvent ie)*
WindowFocusListener	This interface defines two methods: **windowGainedFocus( )** and **windowLostFocus( )**. These are called when a window gains or loses input focus. Their general forms are shown here: *void windowGainedFocus(WindowEvent we)* *void windowLostFocus(WindowEvent we)*
ComponentListener	This interface defines four methods that are invoked when a component is resized, moved, shown, or hidden. Their general forms are shown here: *void componentResized(ComponentEvent ce)* *void componentMoved(ComponentEvent ce)* *void componentShown(ComponentEvent ce)* *void componentHidden(ComponentEvent ce)*

ContainerListener	This interface contains two methods. When a component is added to a container, **componentAdded( )** is invoked. When a component is removed from a container, **componentRemoved( )** is invoked. Their general forms are shown here: *void componentAdded(ContainerEvent ce)* *void componentRemoved(ContainerEvent ce)*
FocusListener	This interface defines two methods. When a component obtains keyboard focus, **focusGained( )** is invoked. When a component loses keyboard focus, **focusLost( )** is called. Their general forms are shown here: *void focusGained(FocusEvent fe)* *void focusLost(FocusEvent fe)*
KeyListener	This interface defines three methods. *void keyPressed(KeyEvent ke)* *void keyReleased(KeyEvent ke)* *void keyTyped(KeyEvent ke)*

3.Event classes

Event Classes and Listener Interfaces:

The java.awt.event package provides many event classes and Listener interfaces for event handling. At the root of the Java event class hierarchy is **EventObject**, which is in **java.util**. It is the super class for all events. It's one constructor is shown here:

EventObject(Object *src*) - Here, *src* is the object that generates this event.

EventObject contains two methods:

Object getSource( ) - Object on which event initially occurred.
String toString( ) - toString( ) returns the string equivalent of the event.

The class **AWTEvent**, defined within the **java.awt** package, is a subclass of **EventObject**. It is the superclass (either directly or indirectly) of all AWT-based events used by the delegation event model. Its **getID( )** method can be used to determine the type of the event. The signature of this method is shown here:

int getID( )

The package **java.awt.event** defines many types of events that are generated by various user interface elements

Event Class	Description	Listener Interface
ActionEvent	Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.	ActionListener
AdjustmentEvent	Generated when a scroll bar is manipulated.	AdjustmentListener
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible.	ComponentListener
ContainerEvent	Generated when a component is added to or removed from a container.	ContainerListener
FocusEvent	Generated when a component gains or losses keyboard focus.	FocusListener
InputEvent	Abstract super class for all component input event classes.	
ItemEvent	Generated when a check box or list item is clicked	ItemListener
KeyEvent	Generated when input is received from the keyboard.	KeyListener
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.	MouseListener and MouseMotionListener
TextEvent	Generated when the value of a text area or text field is changed.	TextListener
WindowEvent	Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.	WindowListener

Useful Methods of Component class:

Method	Description
public void add(Component c)	inserts a component.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.

The ActionEvent Class:

An ActionEvent is generated when a button is pressed, a list item is double-clicked, or a menu item is selected.

The ActionEvent class defines four integer constants that can be used to identify any modifiers associated with an action event: ALT_MASK, CTRL_MASK, META_MASK (Ex. Escape), and SHIFT_MASK.

ActionEvent has these three constructors:

- ActionEvent(Object src, int type, String cmd)
- ActionEvent(Object src, int type, String cmd, int modifiers)
- ActionEvent(Object src, int type, String cmd, long when, int modifiers)

You can obtain the command name for the invoking ActionEvent object by using the getActionCommand( ) method, shown here:

String getActionCommand( ) -Returns the command string associated with this action

The AdjustmentEvent Class:

An **AdjustmentEvent** is generated by a scroll bar. There are five types of adjustment events.

BLOCK_DECREMENT	The user clicked inside the scroll bar to decrease its value.
BLOCK_INCREMENT	The user clicked inside the scroll bar to increase its value.
TRACK	The slider was dragged.
UNIT_DECREMENT	The button at the end of the scroll bar was clicked to decrease its value.
UNIT_INCREMENT	The button at the end of the scroll bar was clicked to increase its value.

The ComponentEvent Class:

A **ComponentEvent** is generated when the size, position, or visibility of a component is changed. There are four types of component events. The **ComponentEvent** class defines integer constants that can be used to identify them:

COMPONENT_HIDDEN	The component was hidden.
COMPONENT_MOVED	The component was moved.
COMPONENT_RESIZED	The component was resized.
COMPONENT_SHOWN	The component became visible.

ComponentEvent is the superclass either directly or indirectly of **ContainerEvent**, **FocusEvent**, **KeyEvent**, **MouseEvent**, and **WindowEvent**, among others.

The **getComponent( )** method returns the component that generated the event. It is shown here:

Component getComponent( )

The ContainerEvent Class:

A **ContainerEvent** is generated when a component is added to or removed from a container. There are two types of container events. The **ContainerEvent** class defines constants that can be used to identify them: **COMPONENT_ADDED** and **COMPONENT_REMOVED**.

The FocusEvent Class:

A **FocusEvent** is generated when a component gains or loses input focus. These events are identified by the integer constants **FOCUS_GAINED** and **FOCUS_LOST**.

The InputEvent Class:

The abstract class **InputEvent** is a subclass of **ComponentEvent** and is the super class for component input events. Its subclasses are **KeyEvent** and **MouseEvent**.

InputEvent defines several integer constants that represent any modifiers, such as the control key being pressed, that might be associated with the event. Originally, the InputEvent class defined the following eight values to represent the modifiers:

ALT_MASK	ALT_GRAPH_MASK	BUTTON2_MASK	BUTTON3_MASK
BUTTON1_MASK	CTRL_MASK	META_MASK	SHIFT_MASK

However, because of possible conflicts between the modifiers used by keyboard events and mouse events, and other issues, the following extended modifier values were added:

ALT_DOWN_MASK	ALT_GRAPH_DOWN_MASK	BUTTON1_DOWN_MASK
BUTTON2_DOWN_MASK	BUTTON3_DOWN_MASK	CTRL_DOWN_MASK
META_DOWN_MASK	SHIFT_DOWN_MASK	

The KeyEvent Class

A KeyEvent is generated when keyboard input occurs. There are three types of key events, which are identified by these integer constants: **KEY_PRESSED, KEY_RELEASED,** and **KEY_TYPED.**

The first two events are generated when any key is pressed or released. The last event occurs only when a character is generated. Remember, not all key presses result in characters. For example, pressing shift does not generate a character.

There are many other integer constants that are defined by KeyEvent. For example, VK_0 through VK_9 and VK_A through VK_Z define the ASCII equivalents of the numbers and letters.

The MouseEvent Class:

There are eight types of mouse events. The **MouseEvent** class defines the following integer constants that can be used to identify them:

MOUSE_CLICKED	The user clicked the mouse
MOUSE_DRAGGED	The user dragged the mouse
MOUSE_ENTERED	The mouse entered a component
MOUSE_EXITED	The mouse exited from a component.
MOUSE_MOVED	The mouse moved
MOUSE_RELEASED	The mouse was released.
MOUSE_WHEEL	The mouse wheel was moved.

Two commonly used methods in this class are **getX( )** and **getY( )**. These return the X and Y coordinates of the mouse within the component when the event occurred. Their forms are shown here:

int getX( )
int getY( )

The TextEvent Class:

Instances of this class describe text events. These are generated by text fields and text areas when characters are entered by a user or program. TextEvent defines the integer constant **TEXT_VALUE_CHANGED.**

The WindowEvent Class:

The **WindowEvent** class defines integer constants that can be used to identify different types of events:

WINDOW_ACTIVATED	The window was activated.
WINDOW_CLOSED	The window has been closed.
WINDOW_CLOSING	The user requested that the window be closed.
WINDOW_DEACTIVATED	The window was deactivated.
WINDOW_DEICONIFIED	The window was deiconified.
WINDOW_GAINED_FOCUS	The window was iconified.
WINDOW_ICONIFIED	The window gained input focus.
WINDOW_LOST_FOCUS	The window lost input focus.
WINDOW_OPENED	The window was opened.

What is Event Handling?

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism have the code which is known as event handler that is executed when an event occurs. Java Uses the Delegation Event Model to handle the events.

Steps to perform Event Handling Following steps are required to perform event handling:

1. Register the component with the Listener
2. Implement the concerned interface

5.Delegation event model

The Delegation Event model is defined to handle events in GUI programming languages. The GUI stands for Graphical User Interface, where a user graphically/visually interacts with the system.

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.

Event Processing in Java

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. 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.

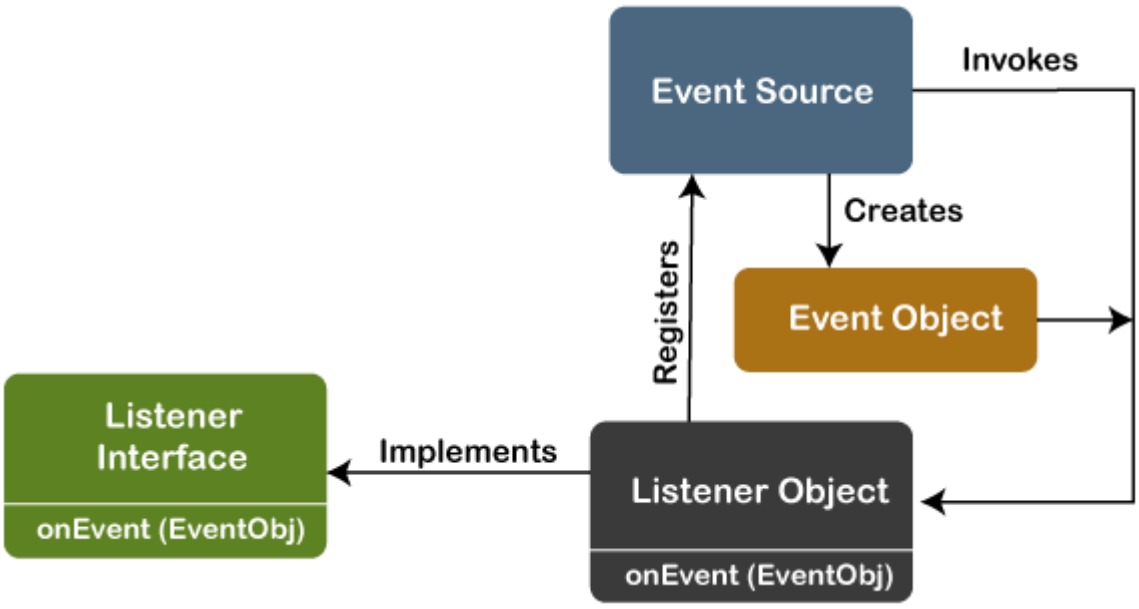


Fig.Delegation Event Model

But, the modern approach for event processing is based on the Delegation Model. It defines a standard and compatible mechanism to generate and process events.

In this model, a source generates an event and forwards it to one or more listeners. The listener waits until it receives an event. Once it receives the event, it is processed by the listener and returns it. The UI elements are able to delegate the processing of an event to a separate function.

In this model, the listener must be connected with a source to receive the event notifications. Thus, the events will only be received by the listeners who wish to receive them. So, this approach is more convenient than the inheritance-based event model (in Java 1.0).

In the older model, an event was propagated up the containment until a component was handled. This needed components to receive events that were not processed, and it took lots of time. The Delegation Event model overcame this issue.

Basically, an Event Model is based on the following three components:

- Events
- Events Sources
- Events Listeners

Events

The Events are the objects that define state change in a source. An event can be generated as a reaction of a user while interacting with GUI elements. 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. We can also consider many other user operations as events.

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

A source is an object that causes and generates an event. It generates an event when the internal state of the object is changed. The sources are allowed to generate several different types of events.

A source must register a listener to receive notifications for a specific event. Each event contains its registration method. Below is an example:

1. **public void** addTypeListener (TypeListener e1)

From the above syntax, the Type is the name of the event, and e1 is a reference to the event listener. For example, for a keyboard event listener, the method will be called as **addKeyListener()**. For the mouse event listener, the method will be called as **addMouseMotionListener()**. 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 multicasting. In few cases, the event notification will only be sent to listeners that register to receive them.

Some listeners allow only one listener to register. Below is an example:

1. **public void** addTypeListener(TypeListener e2) **throws** java.util.TooManyListenersException

From the above syntax, the Type is the name of the event, and e2 is the event listener's reference. When the specified event occurs, it will be notified to the registered listener. This process is known as **unicasting** events.

A source should contain a method that unregisters 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.

```
1. public void removeTypeListener(TypeListener e2?)
```

From the above syntax, the Type is an event name, and e2 is the reference of the listener. For example, to remove the keyboard listener, the **removeKeyListener()** method will be called.

The source provides the methods to add or remove listeners that generate the events. For example, the Component class contains the methods to operate on the different types of events, such as adding or removing them from the listener.

Event Listeners

An event listener is an object that is invoked when an event triggers. 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. Second, it must implement the methods to receive and process the received notifications.

The methods that deal with the events are defined in a set of interfaces. These interfaces can be found in the java.awt.event package.

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.

6.Handling mouse events

The Java MouseListener is notified whenever you change the state of mouse. It is notified against MouseEvent. The MouseListener interface is found in java.awt.event package. It has five methods to handle mouse events.

Methods of MouseListener interface

The signature of 5 methods found in MouseListener interface are given below:

```
1. public abstract void mouseClicked(MouseEvent e);
2. public abstract void mouseEntered(MouseEvent e);
3. public abstract void mouseExited(MouseEvent e);
4. public abstract void mousePressed(MouseEvent e);
5. public abstract void mouseReleased(MouseEvent e);
```

Java MouseListener Example

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class MouseListenerExample extends Frame implements MouseListener{
4.     Label l;
```

```
5.    MouseListenerExample(){
6.        addMouseListener(this);
7.
8.        l=new Label();
9.        l.setBounds(20,50,100,20);
10.       add(l);
11.       setSize(300,300);
12.       setLayout(null);
13.       setVisible(true);
14.   }
15.   public void mouseClicked(MouseEvent e) {
16.       l.setText("Mouse Clicked");
17.   }
18.   public void mouseEntered(MouseEvent e) {
19.       l.setText("Mouse Entered");
20.   }
21.   public void mouseExited(MouseEvent e) {
22.       l.setText("Mouse Exited");
23.   }
24.   public void mousePressed(MouseEvent e) {
25.       l.setText("Mouse Pressed");
26.   }
27.   public void mouseReleased(MouseEvent e) {
28.       l.setText("Mouse Released");
29.   }
30.public static void main(String[] args) {
31.   new MouseListenerExample();
32.}
33.}
```

Output:

7.Handling keyboard events

KeyboardEvent objects describe a user interaction with the keyboard; each event describes a single interaction between the user and a key (or combination of a key with modifier keys) on the keyboard. It is notified against KeyEvent. The KeyListener interface is found in java.awt.event package. It has three methods to handle mouse events.

Methods of KeyListener interface

The signature of 3 methods found in keyListener interface are given below:

void keyPressed(KeyEvent ke)

void keyReleased(KeyEvent ke)

void keyTyped(KeyEvent ke)

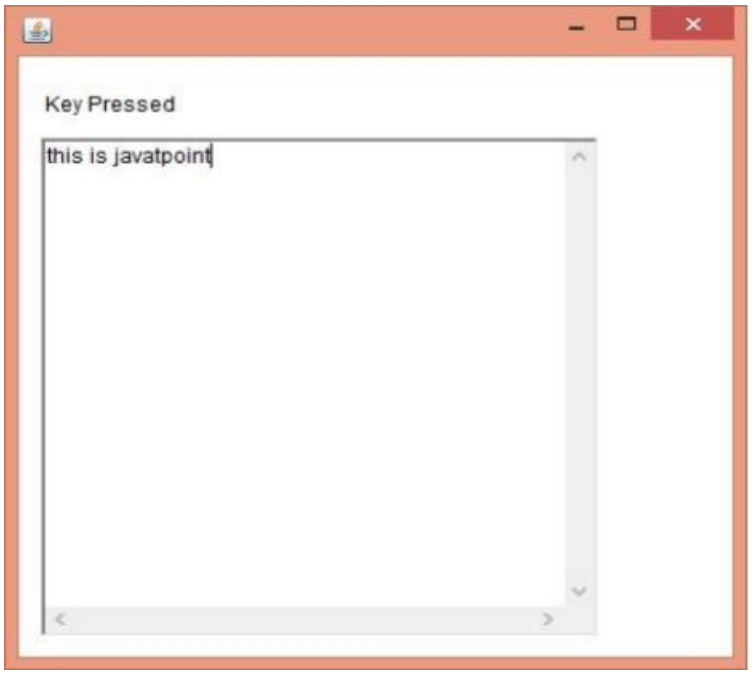
Example program:-

```
import java.awt.*;
    import java.awt.event.*;
    public class KeyListenerExample extends Frame implements KeyListener{
        Label l;
        TextArea area;
        KeyListenerExample(){
            l=new Label();
            l.setBounds(20,50,100,20);
            area=new TextArea();
            area.setBounds(20,80,300, 300);
            area.addKeyListener(this);
            add(l);
           add(area);
            setSize(400,400);
            setLayout(null);
            setVisible(true);
        }
        public void keyPressed(KeyEvent e) {
            l.setText("Key Pressed");
        }
```

```
    public void keyReleased(KeyEvent e) {
        l.setText("Key Released");
    }
    public void keyTyped(KeyEvent e) {
        l.setText("Key Typed");
    }

    public static void main(String[] args) {
        new KeyListenerExample();
    }
}
```

OUTPUT:



8.Adapter classes

Java provides a special feature, called an *adapter class,* that can simplify the creation of event handlers in certain situations. An adapter class provides an empty implementation of all methods in an event listener interface. Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface.

For example,

MouseListener	MouseAdapter
void mouseClicked(MouseEvent me)	*void mouseClicked(MouseEvent me){ }*
void mouseEntered(MouseEvent me)	*void mouseEntered(MouseEvent me) { }*
void mouseExited(MouseEvent me)	*void mouseExited(MouseEvent me) { }*
void mousePressed(MouseEvent me)	*void mousePressed(MouseEvent me) { }*
void mouseReleased(MouseEvent me)	*void mouseReleased(MouseEvent me) { }*

Table: Commonly used Listener Interfaces implemented by Adapter Classes

Adapter Class	Listener Interface
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener

The adapter classes are found in **java.awt.event package.**

Example Program:-

WindowAdapter Example

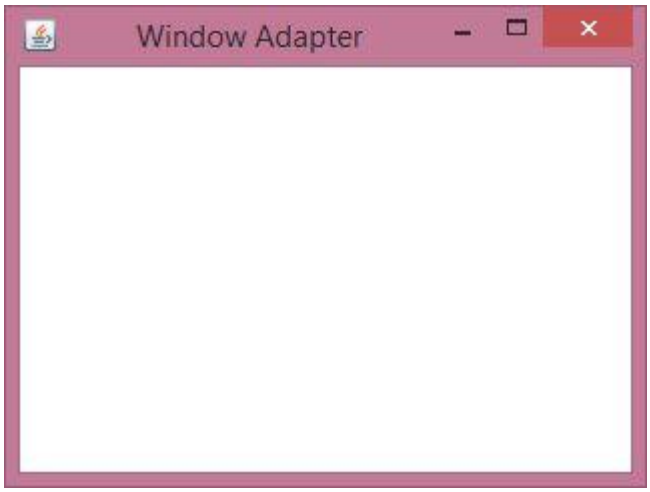
In the following example, we are implementing the WindowAdapter class of AWT and one its methods windowClosing() to close the frame window.

AdapterExample.java

```
1. // importing the necessary libraries
2. import java.awt.*;
3. import java.awt.event.*;
4.
5. public class AdapterExample {
6. // object of Frame
7.     Frame f;
8. // class constructor
9.     AdapterExample() {
10.// creating a frame with the title
```

```
11.       f = new Frame ("Window Adapter");
12.// adding the WindowListener to the frame
13.// overriding the windowClosing() method
14.//using annonymous class here
15.       f.addWindowListener (new WindowAdapter() {
16.          public void windowClosing (WindowEvent e) {
17.             f.dispose();
18.          }
19.       });
20.        // setting the size, layout and
21.       f.setSize (400, 400);
22.       f.setLayout (null);
23.       f.setVisible (true);
24.    }
25.
26.// main method
27.public static void main(String[] args) {
28.    new AdapterExample();
29.}
30.}
```

Output:



Q.Distinguish Event Listeners from Event Adapters.

Event Listeners:-

When we implement a listener interface in any class then we must have to implement all the methods declared in that interface because all the methods in an interface are abstract and must

be override in class which implement it. For example consider the following program which demonstrates handling of key events by implementing listener interface.

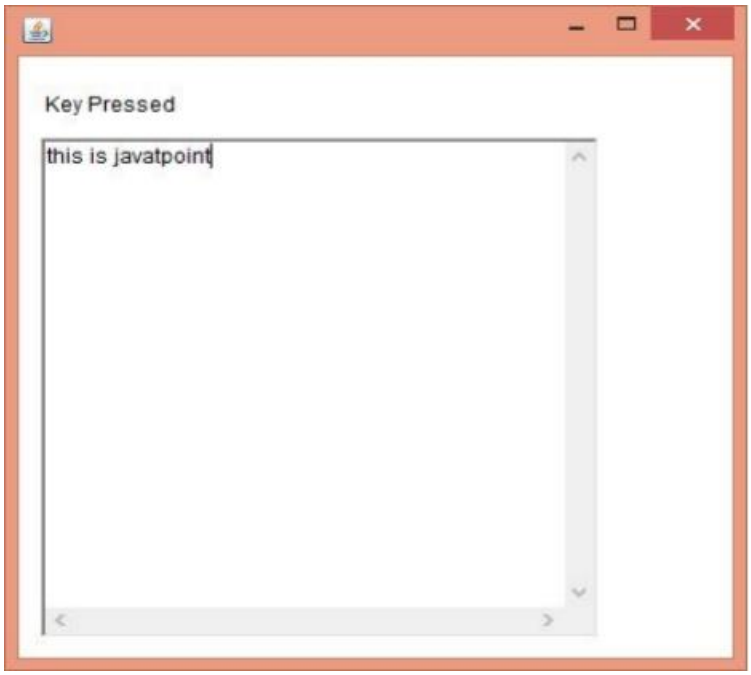
KeyListenerExample.java

```
import java.awt.*;
  import java.awt.event.*;
  public class KeyListenerExample extends Frame implements KeyListener{
    Label l;
    TextArea area;
    KeyListenerExample(){
      l=new Label();
      l.setBounds(20,50,100,20);
      area=new TextArea();
      area.setBounds(20,80,300, 300);
      area.addKeyListener(this);
      add(l);
     add(area);
      setSize(400,400);
      setLayout(null);
      setVisible(true);
    }
    public void keyPressed(KeyEvent e) {
      l.setText("Key Pressed");
    }
    public void keyReleased(KeyEvent e) {
      l.setText("Key Released");
    }
    public void keyTyped(KeyEvent e) {
      l.setText("Key Typed");
    }

    public static void main(String[] args) {
      new KeyListenerExample();
```

```
    }
}
```

OUTPUT:



Our above example for handling key events implements KeyListener interface so the class KeyEvent has to implement all the three methods listed below.

1. public void keyTyped(KeyEvent e)
2. public void keyPressed(KeyEvent e)
3. public void keyReleased(KeyEvent e)

This can be inconvenient because if we want to use only one or two methods in your program then? It is not suitable solution to implement all the methods every time even we don't need them. Adapter class makes it easy to deal with this situation. An adapter class provides empty implementations of all methods defined by that interface.

Adapter classs are very useful if you want to override only some of the methods defined by that interface. Here the names of Listener interface and corresponding interface are given which are in java.awt.event package.

Adapter Class	Listener Interface
ComponentAdapter	ComponentListner
ContainerAdapter	ContainerListner
FocusAdapter	FocusListner
KeyAdapter	KeyListner
MouseAdapter	MouseListner
MouseMotionAdapter	MouseMotionListner
WindowAdapter	WindowListner

Now consider a situation in which we want to perform any action only when key Typed then we should have to override keyTyped() method. In this case if we use the listener interface then we must have to implements all the above three methods, the adapter class KeyAdapter will minimize programmer's work. Following example demonstrate the use of Adapter class in place of Listener interface.

KeyListenerExample.java

```
import java.awt.*;
   import java.awt.event.*;
   public class KeyListenerExample extends KeyAdapter{
      Frame f;
      Label l;
      TextArea area;
      KeyListenerExample(){
         f=new Frame("Adapter class");
         l=new Label();
         l.setBounds(20,50,100,20);
         area=new TextArea();
         area.setBounds(20,80,300, 300);
         area.addKeyListener(this);
```

```
        f.add(l);
f.add(area);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }

    public void keyTyped(KeyEvent e) {
        l.setText("Key Typed");
    }

    public static void main(String[] args) {
        new KeyListenerExample();
    }
}
```