ADVANCED JAVA

BCA-V SEM

NOTES
Prepared By

Prof. Prabhu Kichadi, M.Tech
KLES SSMS BCA College Athani

UNIT – I

Event Handling: Event, Event Source, Event Classes, Event Listener interface, Examples, Handling Windows Events, Adapter Classes, Inner classes.

Event Handling

Event: Event is the change in the state of an object/source when user does some action on it. Event is associated with time and task. Whenever an event occurs task/code associated with an event must be executed.

Events are generated when user interacts with the elements in a Graphical User Interface **(GUI).**

Eg:

1. Clicking on a Button,
2. Chooses a menu item,
3. Presses Enter in a text field,
4. Selecting A radio button,
5. Selecting OR deselecting a checkbox.

Button state changes from unclicked to clicked.
Checkbox changes state from unchecked to checked.

Event Handling/EVENT DELEGATION MODEL:

"Event Handling is the mechanism that controls the event and decides what should happen if an event occurs." This mechanism has the code which is known as **event handler** that is executed when an event occurs.

Java uses **EVENT DELEGATION MODEL** to handle event.

There mainly two participants we need to understand regrading event handling using **event delegation model**:

1. Event Source:

2. Event Listener/Handler:

Event Sources: These are the objects of components on which event occurs. Source is responsible for providing information of the occurred event to its handler.

Eg: Button, JButton, MenuItem, JMenuItem, Scrollbar, List, Component, Window, etc.

Event Handler/Listener: Event listeners are interfaces which is responsible to handle events. It is also known as event handler. Listener is responsible for generating response to an event.

Eg:

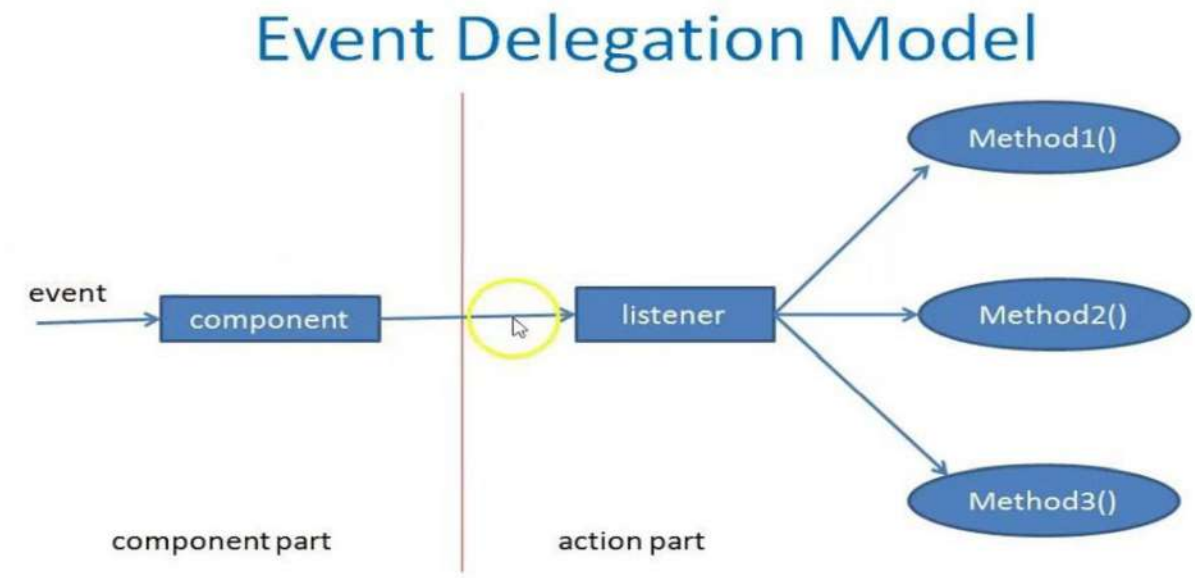
ActionListener ----> Handles Button, MenuItem, List related events (Action Events)
MouseListener ----> Handles events related to mouse movements.
KeyListener --- -> Handles events related to Text Components like Key Presses, Released, Entered etc.

If we want to write handler code, we must create an **event handler class** that must implements **event listeners**<<i>>, and provide definitions for abstract methods of listeners to corresponding events.

Ex: EHCButton is an EvenetListener class for clicking a button event.

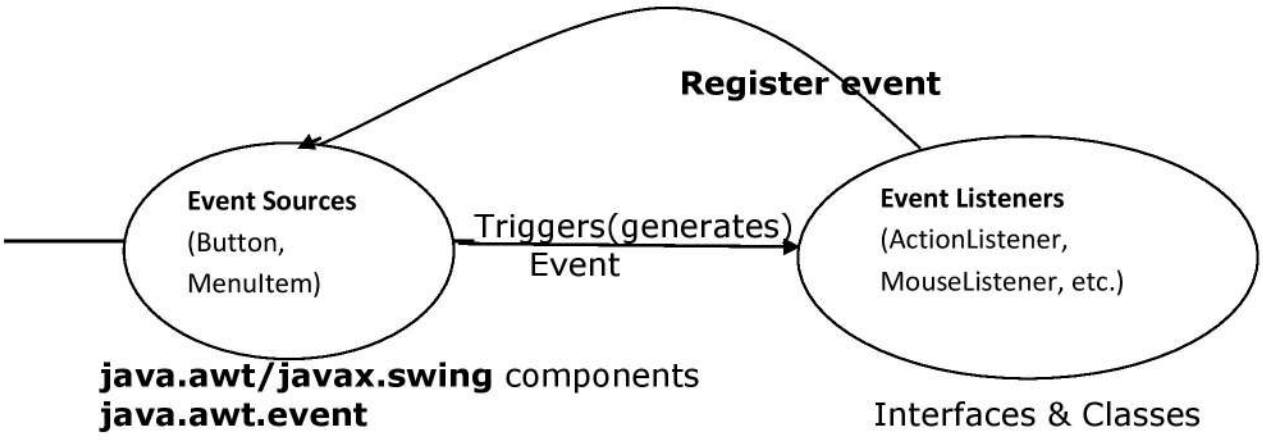
```
class EHCButton implements ActionListener
{
        public void actionPerfomed(ActionEvent e)
        {
                //write the code after event occurs,
                //write response code after an event occurs
        }
}
```

Steps used in Event Delegation Model for event handing:
1. Event Listeners must **register** to the sources which generates an event.
2. User Click on a component like Button (Source) and the event is generated.
3. Now the object concerned event class is created automatically and information about the source and the event get populated with in same object.
4. Event object is forwarded to the method of registered listener class.
5. The method is now getting executed & returns.



Event Delegation Model:

- Events in java are handled through ***Event Delegation Model,*** it defines a consistent mechanism to generate & process events.
- Its concept is simple ***Source*** generates an event and sends its to one or more ***listeners***.
- Then l***istene***r waits until it receives an event. Once the event is received, the Listener processes the event and returns.
- Advantage of this approach is ***application logic is separated*** from user interface logic.
- In the event delegation model, listeners must ***register*** with a source in order to receive an event notification.



Event Sources:

The source is an object on which event occurs. Source is responsible for providing information of the occurred event to its handler. Java provide as with classes for source object. Event source is an **java.awt** Component There are many GUI components from **java.awt** package which are the sources for event generation.

Some of AWT components are **event sources** and descriptions:

Event Sources	Descriptions
Button	Generates action events when the button is pressed
Checkbox	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; generates item events when an item is selected or deselected
MenuItem	Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected
Scrollbar	Generates adjustment events when the scroll bar is manipulated
Window	Generates window events when a window is activated, closed, deactivated, DE iconified, iconified, opened, or quit
Textfield	Generates text events when the user enters a character.
TextArea	Generates text events when the user enters a character.

Event Listener

- It is an object which is notified when an event occurs.
- Listeners are **Java Interfaces** and since we are covering advance topics you must have the knowledge of interfaces that when we implement an interface by a class, we must implement all interface's methods because all the methods in an interface are abstract. Implementing an interface without adding its required abstract methods result in a syntax error.
- A listener has two major requirements,
- It must have been **registered** with one or more sources to receive notifications about specific types of events.
- It must implement methods to receive and process these notifications.
- Some common Java interfaces and the events they listen for and handle

Listeners	Descriptions
ActionListener	Listens for and handles button clicks
KeyListener	Listens for and handles **key events**
MouseListener	Listens for and handles **mouse events**
MouseMotionListener	Listens for and handles **mouse drag and move events**
TextListener	Listens for and handles **text changing events**
ItemListener	Listens for and handles **Checkbox and List events**
AdjustmentListener	Listens for and handles **scrolling events – Scrollbar**
Runnable	Listens for and handles **Threads** and **Run**
WindowListener	Listens for and handles **Window events**

Event Classes

- Classes that represent events.
- **EventObject** is the superclass for all events which means it is the root of the Java event class hierarchy.
- **EventObject** contains two methods. **getSource()** and **toString().**
- getSource() method returns the source and toString() method returns the string equivalent of the event .
- **AWTEvent** is a subclass of **EventObject** class and it is superclass for all AWT events .
- Package **java.awt.event** is used for defining interfaces and classed that is used for event handling in AWT and SWING.
- Some common event classes in the package **java.awt.event** are given below

Event Classes and Associated Listener Interfaces

Event Classes	Event Listener Interfaces	Event Sources
ActionEvent	ActionListener	Button, MenuItem, List
MouseEvent	MouseListener and MouseMotionListener	Mouse Movements
MouseWheelEvent	MouseWheelListener	Mouse Wheel Movements
KeyEvent	KeyListener	Text Components
ItemEvent	ItemListener	Checkbox, List
TextEvent	TextListener	Text Components
AdjustmentEvent	AdjustmentListener	Component, Scrollbar
WindowEvent	WindowListener	Window
ComponentEvent	ComponentListener	Component
ContainerEvent	ContainerListener	Component
FocusEvent	FocusListener	Component

Event Registration methods:

Every event listener must be registered to the event sources to make active response from handler to the corresponding events by the sources. Every event source has methods to register events to the listeners, Format of Method to register an event listener is

addTypeListener();

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

- **Button**
  - public void addActionListener(ActionListener a){ }
- **MenuItem**
  - public void addActionListener(ActionListener a){ }
- **TextField**
  - public void addActionListener(ActionListener a){ }
  - public void addTextListener(TextListener a){ }
- **TextArea**
  - public void addTextListener(TextListener a){ }
- **Checkbox**
  - public void addItemListener(ItemListener a){ }
- **Choice**
  - public void addItemListener(ItemListener a){ }
- **List**
  - public void addActionListener(ActionListener a){ }
  - public void addItemListener(ItemListener a){ }

Event Listener interfaces and their event handling methods:

The event delegation model contains two main components. First are the event sources and second are the listeners. Most of the listener interfaces are available in the ***java.awt.event*** package. In Java, there are several event listeners interfaces which are listed below:

1. ActionListener :This interface deals with the action events. Following is the event handling method available in the ActionListener interface:

void actionPerformed(ActionEvent ae)

2. AdjustmentListener: This interface deals with the adjustment event generated by the scroll bar. Following is the event handling method available in the AdjustmentListener interface:

void adjustmentValueChanged(AdjustmentEvent ae)

3. ComponentListener:

This interface deals with the component events. Following are the event handling methods available in the ComponentListener interface:

void componentResized(ComponentEvent ce)
void componentMoved(ComponentEvent ce)
void componentShown(ComponentEvent ce)
void componentHidden(ComponentEvent ce)

4. ContainerListener: This interface deals with the events that can be generated on containers. Following are the event handling methods available in the ContainerListener interface:

void componentAdded(ContainerEvent ce)
void componentRemoved(ContainerEvent ce)

5. FocusListener: This interface deals with focus events that can be generated on different components or containers. Following are the event handling methods available in the FocusListener interface:

void focusGained(FocusEvent fe)
void focusLost(FocusEvent fe)

6. ItemListener: This interface deals with the item event. Following is the event handling method available in the ItemListener interface:
void itemStateChanged(ItemEvent ie)

7. KeyListener: This interface deals with the key events. Following are the event handling methods available in the KeyListener interface:
void keyPressed(KeyEvent ke)
void keyReleased(KeyEvent ke)
void keyTyped(KeyEvent ke)

8. MouseListener: This interface deals with five of the mouse events. Following are the event handling methods available in the MouseListener interface:

void mouseClicked(MouseEvent me)
void mousePressed(MouseEvent me)
void mouseReleased(MouseEvent me)
void mouseEntered(MouseEvent me)
void mouseExited(MouseEvent me)

9. MouseMotionListener: This interface deals with two of the mouse events. Following are the event handling methods available in the MouseMotionListener interface:

void mouseMoved(MouseEvent me)
void mouseDragged(MouseEvent me)

10. MouseWheelListener: This interface deals with the mouse wheel event. Following is the event handling method available in the MouseWheelListener interface:

void mouseWheelMoved(MouseWheelEvent mwe)

11. TextListener: This interface deals with the text events. Following is the event handling method available in the TextListener interface:

void textValueChanged(TextEvent te)

12. WindowFocusListener: This interface deals with the window focus events. Following are the event handling methods available in the WindowFocusListener interface:

void windowGainedFocus(WindowEvent we)
void windowLostFocus(WindowEvent we)

13.WindowListener: This interface deals with seven of the window events. Following are the event handling methods available in the WindowListener interface:

void windowActivated(WindowEvent we)
void windowDeactivated(WindowEvent we)
void windowIconified(WindowEvent we)
void windowDeiconified(WindowEvent we)
void windowOpened(WindowEvent we)
void windowClosed(WindowEvent we)
void windowClosing(WindowEvent we)

General Steps for GUI Creation & Event Handling Demonstrations:

1. Create a Frame (Container)
2. Set the Properties size, visibility etc.
3. Create Component elements Like Button, MenuItem, Checkbox, etc.
4. Set properties for components also.
5. Register component to Listeners. - addTypeListener()
6. Add components to Frame or Container. - add() method
7. Implement Listeners & override abstract methods (Handler code)
8. Execute the code.

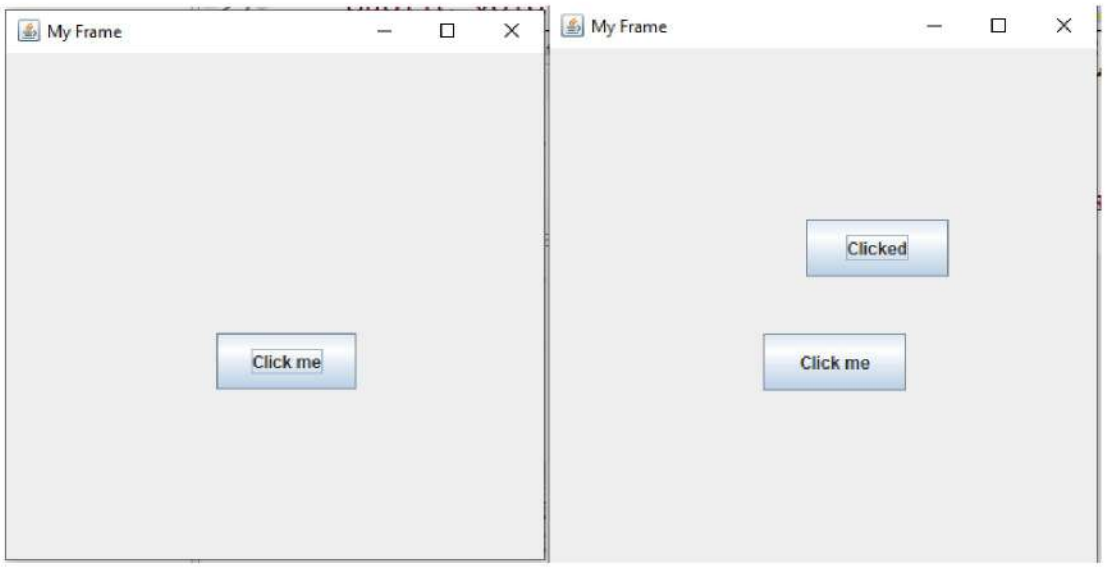
Examples:

1. Button Click Event & ActionListener: On Button Click Event

```
package com.kle.bca;
import java.awt.event.*;
import javax.swing.*;

public class MyFrame implements ActionListener
{
	JFrame fr;
	JButton btn;
	JButton bt;

	public MyFrame()
	{
		//Prepare GUI
		fr = new JFrame("My Frame");
		fr.setBounds(200, 200, 400, 400);
		fr.setLayout(null);
```

```
            fr.setVisible(true);
            fr.setDefaultCloseOperation(fr.EXIT_ON_CLOSE);

            btn = new JButton("Click me");
            btn.setBounds(150,200,100,40);
            fr.add(btn);


            //Register listener to source
            btn.addActionListener(this);

      }

      public void actionPerformed(ActionEvent ae)
      {

            bt = new JButton("Clicked");
            bt.setBounds(180,120,100,40);
            fr.add(bt);
      }

      public static void main(String[] args)
      {
            MyFrame f1 = new MyFrame();
      }


}
```

Output:

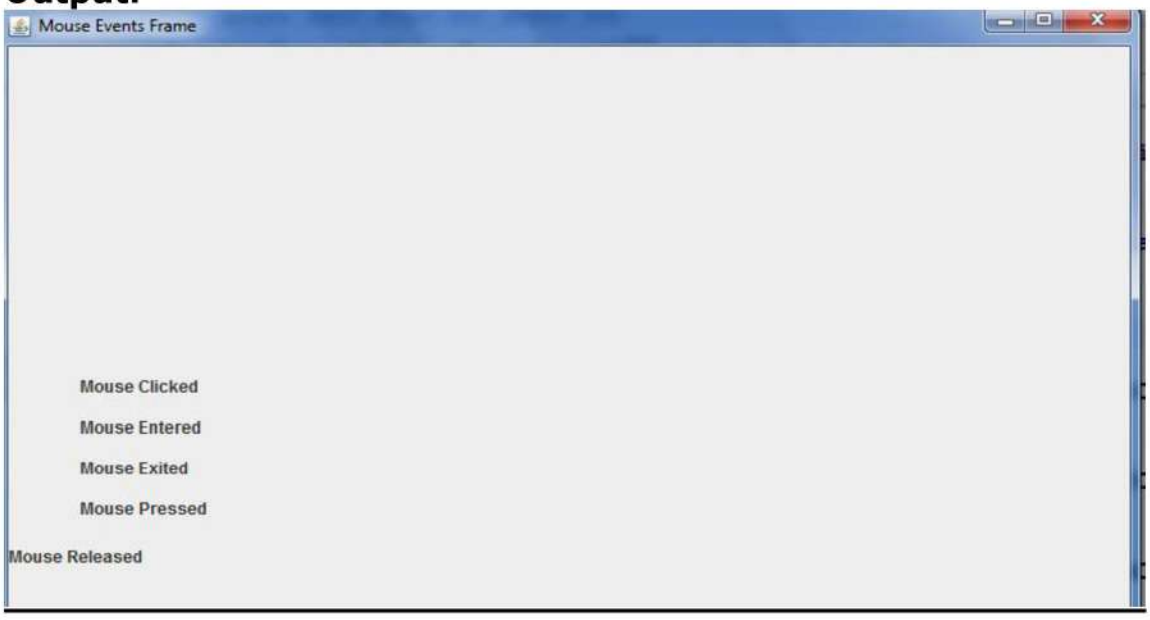


2. MouseEvent & MouseListener Example (Lab Program)

```
package com.kle.bca;

import  java.awt.event.MouseEvent;
import  java.awt.event.MouseListener;
import  javax.swing.JFrame;
import  javax.swing.JLabel;

public  class  MouseEvents  extends  JFrame implements  MouseListener
{

      JLabel  label1, label2, label3, label4, label5;
      public  MouseEvents( )
      {
            label1 = new JLabel( );
            label1.setBounds(50, 50, 400, 400);

            label2 = new JLabel( );
            label2.setBounds(50, 80, 400, 400);

            label3 = new JLabel( );
            label3.setBounds(50, 110, 400, 400);

            label4 = new JLabel( );
            label4.setBounds(50, 140, 400, 400);

            label5 = new JLabel( );
            label5.setBounds(50, 170, 400, 400);


            setVisible(true);
            setBounds(25, 25, 800, 800);
            add(label1);
            add(label2);
            add(label3);
            add(label4);
            add(label5);
            addMouseListener(this);

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

```
	public  void  mouseClicked(MouseEvent arg0)
	{

		label1.setText("Mouse Clicked");

	}
	public  void  mouseEntered(MouseEvent arg0)
	{

		label2.setText("Mouse Entered");
	}
	public  void  mouseExited(MouseEvent arg0)
	{
		label3.setText("Mouse Exited");
	}
	public  void  mousePressed(MouseEvent arg0)
	{

		label4.setText("Mouse Pressed");
	}
	public  void mouseReleased(MouseEvent arg0)
	{

		label5.setText("Mouse Released");
	}
}
```

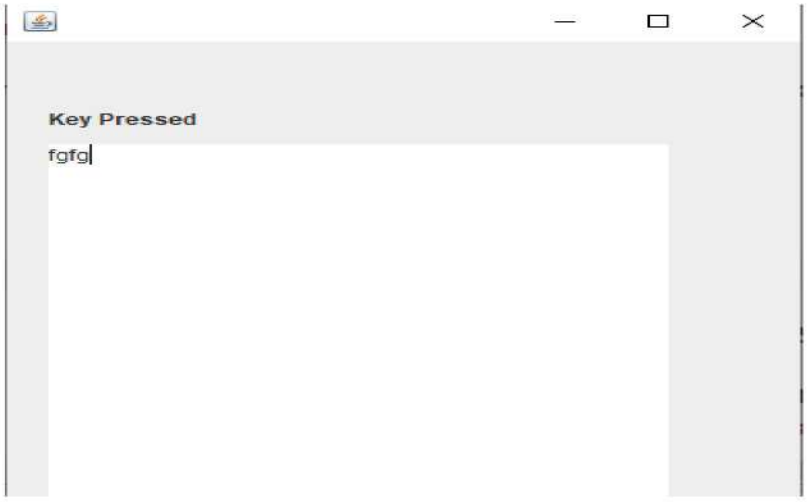
Output:



3. KeyEvent & KeyListener Example(Lab Program)

```
package com.kle.bca;

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

public class KeyListenerDemo extends JFrame implements KeyListener
{
         JLabel l;
         JTextArea txa;

        KeyListenerDemo()
        {
                l=new JLabel();
                l.setBounds(20,50,100,20);
               txa=new JTextArea();
               txa.setBounds(20,80,300, 300);
               txa.addKeyListener(this);
                       add(l);
                add(txa);
                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)
        {
             KeyListenerDemo d = new KeyListenerDemo();
        }
}
```

Output:


Key Pressed
fgfg

Handling Windows Events:

Window is a top-level container with no-border, no-menu bar It could be used to implement a pop-up menu. The default layout for a window is BorderLayout.

This type of events occurs when user opened, closed, activated, deactivated, iconified, or de iconified window.

- The AWT WindowEvent is the class that represents the change in state of a window.
- The Java ***WindowListener*** is interface, it is notified whenever you change the state of window. It is notified against **WindowEvent.**
- This low-level event is generated by a Window object when it is opened, closed, activated, deactivated, iconified, or de iconified, or when focus is transferred into or out of the Window.

The methods of WindowListener are given below,

1. **public abstract void** windowActivated(WindowEvent e);
2. **public abstract void** windowClosed(WindowEvent e);
3. **public abstract void** windowClosing(WindowEvent e);
4. **public abstract void** windowDeactivated(WindowEvent e);
5. **public abstract void** windowDeiconified(WindowEvent e);
6. **public abstract void** windowIconified(WindowEvent e);
7. **public abstract void** windowOpened(WindowEvent e);

Example on WindowEvent & WindowListener:

```
package com.kle.bca;

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

public class WindowListenerDemo extends JFrame implements
WindowListener
{
	public WindowListenerDemo()
	{
		addWindowListener(this);

		 setSize(400,400);
		setLayout(null);
		 setVisible(true);

	}
```

```
public static void main(String[] args)
{
	WindowListenerDemo w = new WindowListenerDemo();
}


public void windowActivated(WindowEvent e)
{

	 System.out.println("activated");
}


public void windowClosed(WindowEvent e)
{

	 System.out.println("window closed");
}


public void windowClosing(WindowEvent e)
{
	 System.out.println("closing");
	 dispose();

}


public void windowDeactivated(WindowEvent e)
{
	System.out.println("deactivated");
}


public void windowDeiconified(WindowEvent e)
{
	System.out.println("deiconified");
}


public void windowIconified(WindowEvent e)
{
	System.out.println("iconified");
}


public void windowOpened(WindowEvent e)
```

```
        {
                System.out.println("opened");
        }
}
```

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.
- If we are inheriting Adapter class then it is not required to implement listeners.

The Adapter classes with their corresponding listener interfaces are given below.

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

Mouse Events: Mouse Clicked : through Adapter Class-> MouseAdapter class

Example:

```
package com.kle.bca;

import java.awt.Color;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.event.*;

public class MouseAdapterDemo extends MouseAdapter
{
	Frame f;
	public MouseAdapterDemo()
	{
		f=new Frame("Mouse Adapter");
```

```
            f.addMouseListener(this);

            f.setSize(300,300);
            f.setLayout(null);
            f.setVisible(true);
        }

        public void mouseClicked(MouseEvent e)
        {
            System.out.println("Mouse Clicked");
            /*Graphics g=f.getGraphics();
              g.setColor(Color.red);
                  g.fillOval(e.getX(),e.getY(),30,30); */
        }

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

Java Inner Classes:

- Java inner class or nested class is a class which is declared inside the class or interface.
- We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.
- Additionally, it can access all the members of outer class including private data members and methods.

Syntax:

```
class Outer
{

      class Inner
      {

      }

}
```

Advantage of java inner classes

1) Nested classes represent a special type of relationship that is it can access all the members (data members and methods) of outer class including private.

2) Nested classes are used to develop more readable and maintainable code because it logically group classes and interfaces in one place only.

3) Code Optimization: It requires less code to write.

Types of inner classes:

1. **Member inner class**
2. **Anonymous inner class**
3. **Local inner class**

Member Inner Class	A class created within class and outside method.
Anonymous Inner Class	A class created for implementing interface or extending class. Its name is decided by the java compiler.
Local Inner Class	A class created within method.

Question Bank