

COURSE: ADVANCED JAVA PROGRAMMING (22517)



Unit 3 Event Handling

By

Mr. Vijay M Bande
Lecturer in Computer Engineering
Government Polytechnic Khamgaon

COURSE OUTCOMES(CO's)

- a) Develop Programs using GUI Framework (AWT and Swings)
- b) **Handle events of AWT and Swings Components.**
- c) **Develop programs to handle events in Java Programming**
- d) Develop Java programs using networking basics.
- e) Develop programs using Database.
- f) Develop programs using Servlets.



UNIT OUTCOMES(UO's)

Unit	Unit Outcomes (UOs) (in cognitive domain)	Topics and Sub-topics
Unit– III Event Handling	<p>3a. Use delegation event model to develop event driven program for the given problem.</p> <p>3b. Use relevant AWT/ swing component(s) to handle the given event.</p> <p>3c. Use Adapter classes in Java program to solve the given problem.</p> <p>3d. Use inner classes in java program to solve the given problem.</p>	<p>3.1 The delegation Event Model:Event sources, Event listeners</p> <p>3.2 Event classes: The Action Eventclass, the Item Event class, the Key Event class, theMouse Event class, the Text Event class, the Window Event class.</p> <p>3.3 Adapter classes.</p> <p>3.4 Inner classes.</p> <p>3.5 Event listener interfaces:ActionListener Interface, ItemListener Interface, KeyListener Interface, MouseListenerInterface, MouseMotion Interface, TextListener Interface, WindowsListener Interface.</p>



EVENT HANDLING

- Any program that uses **GUI (graphical user interface)** such as Java application written for windows, is event driven.
- Event describes the change of state of any object.
- Events are generated as result of user interaction with the graphical user interface components.

EVENT HANDLING

- Changing the state of an object is known as an **event**.
- **For example:** clicking on a button, Entering a character in Textbox, moving the mouse, selecting an item from list, scrolling the page, etc.
- The **java.awt.event** package provides many event classes and Listener interfaces for event handling.

DELEGATION EVENT MODEL

- The modern approach to handling events is based on the delegation event model.
- The delegation event model **provides a standard mechanism for a source to generate an event and send it to a set of listeners.**

DELEGATION EVENT MODEL

- Its concept is quite simple: **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 received, the listener processes the event and then returns.
- The advantage of this design 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” the processing of an event to a separate piece of code.

DELEGATION EVENT MODEL

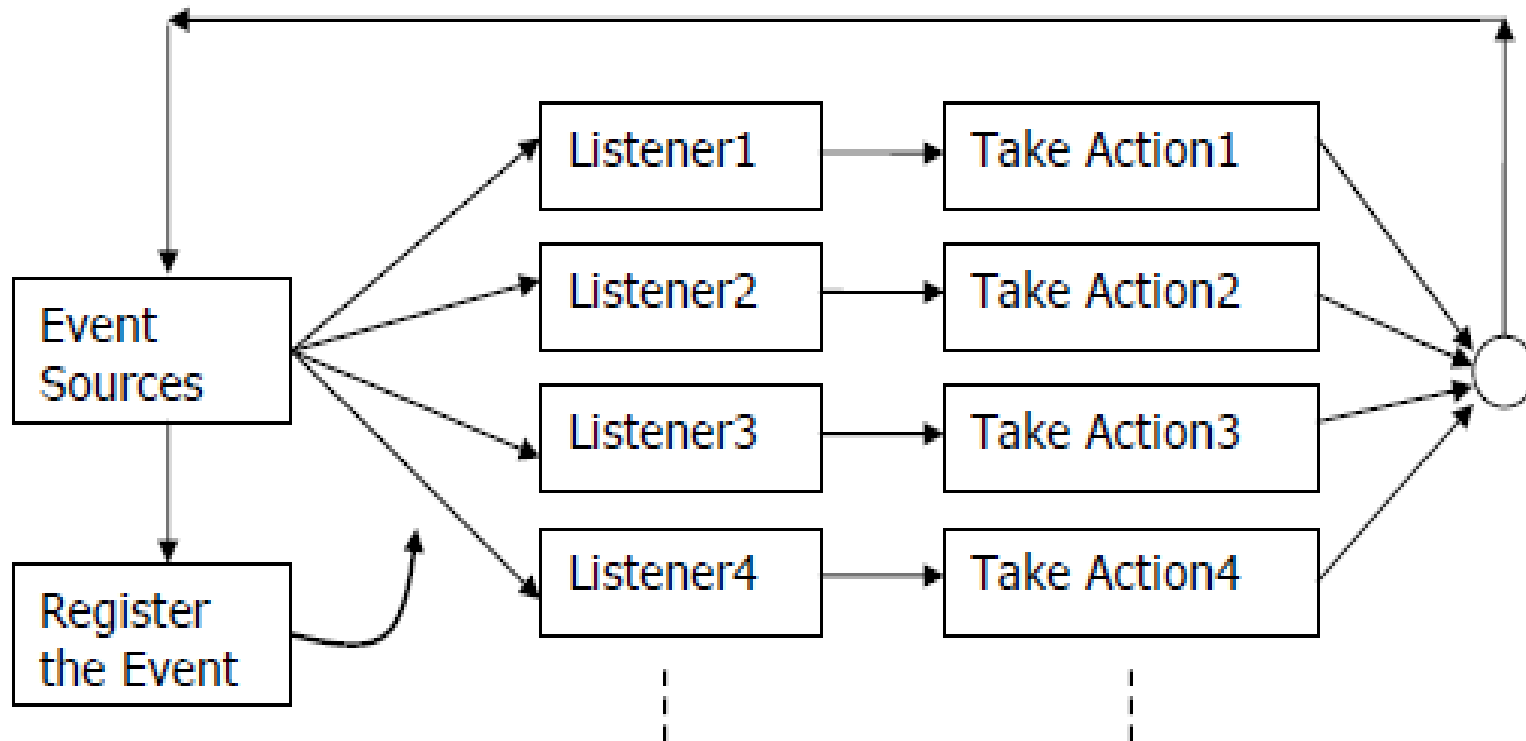


Fig. Delegation Event Model



DELEGATION EVENT MODEL

- In the delegation event model, **listener must register with a source in order to receive an event notification.**
- Notification are sent only to listeners that want to receive them.
- There are mainly three parts in delegation event model.
 - **Events.**
 - **Event sources.**
 - **Event Listeners.**

COMPONENTS OF EVENT HANDLING

○ Events

- An event is a change of state of an object.
- It can be generated as a consequence of a **person interacting with the elements** in a graphical user interface.
- Some of the activities that cause events to be generated are **pressing a button, entering a character via the keyboard, selecting an item in a list, and clicking the mouse.**

EVENT

- Events may also occur that are not directly caused by interactions with a user interface.
- For example, an event may be generated **when a timer expires, a counter exceeds a value, software or hardware failure occurs, or an operation is completed.**
- We are free to define events that are appropriate for an application.

EVENT

Event Class	Description
ActionEvent	Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.
AdjustmentEvent	Generated when a scroll bar is manipulated.
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible.
ContainerEvent	Generated when a component is added to or removed from a container.
FocusEvent	Generated when a component gains or loses keyboard focus.
InputEvent	Abstract superclass for all component input event classes.
ItemEvent	Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.
KeyEvent	Generated when input is received from the keyboard.
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
MouseWheelEvent	Generated when the mouse wheel is moved.
TextEvent	Generated when the value of a text area or text field is changed.
WindowEvent	Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Table 24-1 Commonly Used Event Classes in `java.awt.event`

EVENT SOURCES

- A source is **an object that generates an event**. This 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 in order for the listeners to receive notifications about a specific type of event.
- Each type of event has its own registration method.

EVENT SOURCES

- Here is the general form to register listeners:
 - **public void addTypeListener(TypeListener el)**
For example: **b.addActionListener(this);**
- 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()**.
- The method that registers a **mouse motion listener** is called **addMouseMotionListener()**.
- When an event occurs, all registered listeners are notified and receive a copy of the event object.

EVENT SOURCES

- The general form of unregister listener method is this:
 - `public void removeTypeListener(TypeListener el)`
- Here, **type** is an object that is notified when an event listener. For example, to remove a keyboard listener, you would call `removeKeyListener()`

Sources generating Events

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.
Menu Item	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.
Text Component	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.



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 methods that receive and process events are defined in a set of interfaces found in **java.awt.event**.

EVENT LISTENERS

- 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.

IMPORTANT EVENT CLASSES AND INTERFACE



EVENT CLASSES

- The classes that represent events are at the core of Java's event handling mechanism.
- At the root of the Java event class hierarchy is **EventObject**, which is in **java.util**.
- It is the superclass for all events. **EventObject contains two methods: getSource() and toString().**
- The getSource() method returns the source of the event. Its general form is shown here:
 - Object getSource()
- As expected, toString() returns the string equivalent of the event.



EVENT CLASSES

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

Event Classes	Description	Listener Interface
ActionEvent	generated when button is pressed, menu-item is selected, list-item is double clicked	ActionListener
MouseEvent	generated when mouse is dragged, moved, clicked, pressed or released and also when it enters or exit a component	MouseListener
KeyEvent	generated when input is received from keyboard	KeyListener
ItemEvent	generated when check-box or list item is clicked	ItemListener
TextEvent	generated when value of textarea or textfield is changed	TextListener
MouseWheelEvent	generated when mouse wheel is moved	MouseWheelListener



Event Classes	Description	Listener Interface
WindowEvent	generated when window is activated, deactivated, deiconified, iconified, opened or closed	WindowListener
ComponentEvent	generated when component is hidden, moved, resized or set visible	ComponentEventListener
ContainerEvent	generated when component is added or removed from container	ContainerListener



○ **Steps to handle events:**

- Implement appropriate interface in the class.
- Register the component with the listener.



REGISTRATION METHODS

- 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){}
```



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:
 1. **public static final int ALT_MASK** The alt modifier.
 - An indicator that the alt key was held down during the event.
 2. **public static final int SHIFT_MASK** The shift modifier.
 - An indicator that the shift key was held down during the event.
 3. **public static final int CTRL_MASK** The control modifier.
 - An indicator that control key was held down during the event.
 4. **public static final int META_MASK** The meta modifier.
 - An indicator that the meta key was held down during the event.



ACTIONEVENT CLASS (CONSTRUCTORS)

- In addition, there is an integer constant, **ACTION_PERFORMED**, which can be used to identify action events.
- **ActionEvent** has these **three** constructors:
 1. **ActionEvent(Object src, int type, String cmd)**
 2. **ActionEvent(Object src, int type, String cmd, int modifiers)**
 3. **ActionEvent(Object src, int type, String cmd, long when, int modifiers)**

Here,

- src is a reference to the object that generated this event.
 - The type of the event is specified by type, and
 - its command string is cmd.
 - The argument modifiers indicates which modifier keys (alt, ctrl, meta, and/or shift) were pressed when the event was generated.
 - The when parameter specifies when the event occurred.
- 

ACTIONEVENT CLASS (METHODS)

1. **String** **getActionCommand()**

-You can obtain the command name for the invoking **ActionEvent** object.

2. **int** **getModifiers()**

-returns a value that indicates which modifier keys (alt, ctrl, meta, and/or shift) were pressed when the event was generated.

3. **long** **getWhen()**

-returns the time at which the event took place



ITEMEVENT CLASS

- An **ItemEvent** is generated when a check box or a list item is clicked or when a checkable menu item is selected or deselected.
- There are **two** types of item events, which are identified by the following integer constants:

DESELECTED	The user deselected an item.
SELECTED	The user selected an item.

In addition, **ItemEvent** defines one integer constant, **ITEM_STATE_CHANGED**, that signifies a change of state



ITEMEVENT CLASS (CONSTRUCTORS AND METHODS)

○ **ItemEvent**(ItemSelectable src, int type, Object entry, int state)

Here, src is a reference to the component that generated this event.

The type of the event is specified by type.

The specific item that generated the item event is passed in entry.

The current state of that item is in state.

○ **Methods:**

- **Object getItem()**

-used to obtain a reference to the item that changed.

- **ItemSelectable getItemSelectable()**

-used to obtain a reference to the ItemSelectable object that generated an event.

- **int getStateChange()**

-returns the state change (that is, SELECTED or DESELECTED) for the event.



KEYEVENT CLASS

- A **KeyEvent** is generated when keyboard input occurs.
- There are **three** types of key events, which are identified by these integer constants:
 1. **KEY_PRESSED**,
 2. **KEY_RELEASED**, and
 3. **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 keypresses result in characters.
For example, pressing shift does not generate a character



KEYEVENT CLASS

- 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.
- Here are some others:

VK_ALT	VK_DOWN	VK_LEFT	VK_RIGHT
VK_CANCEL	VK_ENTER	VK_PAGE_DOWN	VK_SHIFT
VK_CONTROL	VK_ESCAPE	VK_PAGE_UP	VK_UP

The **VK constants specify virtual key codes** and are independent of any modifiers, such as control, shift, or alt.



KEYEVENT CLASS (CONSTRUCTORS)

- **KeyEvent** is a subclass of **InputEvent**.

KeyEvent(Component src, int type, long when, int modifiers, int code, char ch)

Here, src is a reference to the component that generated the event.

The type of the event is specified by type.

The system time at which the key was pressed is passed in when.

The modifiers argument indicates which modifiers were pressed when this key event occurred.

The virtual key code, such as VK_UP, VK_A, and so forth, is passed in code.

The character equivalent (if one exists) is passed in ch. **If no valid character exists, then ch contains CHAR_UNDEFINED.**

For KEY_TYPED events, code will contain VK_UNDEFINED.



KEYEVENT CLASS (METHODS)

- **char getKeyChar()**
-returns the character that was entered
- **int getKeyCode()**
-returns the key code

NOTE:

1. **If no valid character is available**, then **getKeyChar()** returns **CHAR_UNDEFINED**.
2. **When a KEY_TYPED event occurs**, **getKeyCode()** returns **VK_UNDEFINED**.



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_PRESSED	The mouse was pressed.
MOUSE_RELEASED	The mouse was released.
MOUSE_WHEEL	The mouse wheel was moved.



MOUSEEVENT CLASS (CONSTRUCTORS)

- **MouseEvent** is a subclass of **InputEvent**.

- Here is one of its constructors:

MouseEvent(Component src, int type, long when, int modifiers, int x, int y, int clicks, boolean triggersPopup)

Here, src is a reference to the component that generated the event.

The type of the event is specified by type.

The system time at which the mouse event occurred is passed in when.

The modifiers argument indicates which modifiers were pressed when a mouse event occurred.

The coordinates of the mouse are passed in x and y.

The click count is passed in clicks.

The triggersPopup flag indicates if this event causes a pop-up menu to appear on this platform.

MOUSEEVENT CLASS (METHODS)

1. **int getX() int getY()**

-These return the X and Y coordinates of the mouse within the component when the event occurred.

2. **Point getPoint()**

-It returns a Point object that contains the X,Y coordinates in its integer members: x and y.

3. **void translatePoint(int x, int y)**

-changes the location of the event

4. **int getClickCount()**

-obtains the number of mouse clicks for this event.



MOUSEEVENT CLASS (METHODS)

5. boolean isPopupTrigger()

-tests if this event causes a pop-up menu to appear on this platform.

6. int getButton()

-It returns a value that represents the button that caused the event.

For most cases, the return value will be one of these constants defined by **MouseEvent**:

NOBUTTON	BUTTON1	BUTTON2	BUTTON3
----------	---------	---------	---------

NOTE: The **NOBUTTON** value indicates that no button was pressed or released.



TEXT EVENT CLASS

- 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 one constructor for this class is shown here:
TextEvent(Object src, int type)



WINDOWEVENT CLASS

- There are ten types of window events. The **WindowEvent** class defines integer constants that can be used to identify them.
- The constants and their meanings are shown here:

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 gained input focus.
WINDOW_ICONIFIED	The window was iconified.
WINDOW_LOST_FOCUS	The window lost input focus.
WINDOW_OPENED	The window was opened.
WINDOW_STATE_CHANGED	The state of the window changed.



WINDOWEVENT CLASS (CONSTRUCTORS)

- **WindowEvent** is a subclass of **ComponentEvent**.
- It defines several constructors
 1. **WindowEvent**(Window src, int type)
 2. **WindowEvent**(Window src, int type, Window other)
 3. **WindowEvent**(Window src, int type, int fromState, int toState)
 4. **WindowEvent**(Window src, int type, Window other, int fromState, int toState)



WINDOWEVENT CLASS (METHODS)

- **Window getWindow()**

- It returns the Window object that generated the event.

- **WindowEvent** also defines methods that **return the opposite window** (when a focus or activation event has occurred), **the previous window state**, and **the current window state**.

- **These methods are shown here:**

1. Window getOppositeWindow()
2. int getOldState()
3. int getNewState()



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; generates item events when an item is selected or deselected.
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.

EVENT LISTENER INTERFACES

- listeners are created by implementing one or more of the interfaces defined by the **java.awt.event** package.
- When an event occurs, the event source invokes the appropriate method defined by the listener and provides an event object as its argument.



EVENT LISTENER INTERFACES

Interface	Description
ActionListener	Defines one method to receive action events.
AdjustmentListener	Defines one method to receive adjustment events.
ComponentListener	Defines four methods to recognize when a component is hidden, moved, resized, or shown.
ContainerListener	Defines two methods to recognize when a component is added to or removed from a container.
FocusListener	Defines two methods to recognize when a component gains or loses keyboard focus.
ItemListener	Defines one method to recognize when the state of an item changes.
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed.
MouseListener	Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.
MouseMotionListener	Defines two methods to recognize when the mouse is dragged or moved.
MouseWheelListener	Defines one method to recognize when the mouse wheel is moved.
TextListener	Defines one method to recognize when a text value changes.
WindowFocusListener	Defines two methods to recognize when a window gains or loses input focus.
WindowListener	Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

THE ACTIONLISTENER INTERFACE

- This interface defines the **actionPerformed()** method that is invoked when an action event occurs.
- Its general form is shown here:

void actionPerformed(ActionEvent ae**)**



THE ITEMListener INTERFACE

- This interface defines the **itemStateChanged()** method that is invoked when the state of an item changes.
- Its general form is shown here: void
itemStateChanged(ItemEvent ie)



THE KEYLISTENER INTERFACE

○ This interface defines **three** methods.

1. **void keyPressed(KeyEvent ke)**
2. **void keyReleased(KeyEvent ke)**
3. **void keyTyped(KeyEvent ke)**

- 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.
- For example, if a user presses and releases the a key, three events are generated in sequence: **key pressed, typed, and released.**
- If a user presses and releases the home key, two key events are generated in sequence: **key pressed and released**



THE MOUSELISTENER INTERFACE

- This interface defines **five** methods.
- If the mouse is pressed and released at the same point, **mouseClicked()** is invoked.
- When the mouse enters a component, the **mouseEntered()** method is called. When it leaves, **mouseExited()** is called.
- The **mousePressed()** and **mouseReleased()** methods are invoked when the mouse is pressed and released, respectively.
- The general forms of these methods are shown here:
 1. **void mouseClicked(MouseEvent me)**
 2. **void mouseEntered(MouseEvent me)**
 3. **void mouseExited(MouseEvent me)**
 4. **void mousePressed(MouseEvent me)**
 5. **void mouseReleased(MouseEvent me)**



THE MOUSEMOTIONLISTENER INTERFACE

- This interface defines **two** methods.
- The **mouseDragged()** method is called multiple times as the mouse is dragged.
- The **mouseMoved()** method is called multiple times as the mouse is moved.
- Their general forms are shown here:
 1. **void mouseDragged(MouseEvent me)**
 2. **void mouseMoved(MouseEvent me)**



THE TEXTLISTENER INTERFACE

- This interface defines the **textValueChanged()** method that is invoked when a change occurs in a text area or text field.
- Its general form is shown here:

void textValueChanged(TextEvent te)



THE WINDOWLISTENER INTERFACE

- This interface defines **seven** methods.
 1. **void windowActivated(WindowEvent we)**
 2. **void windowClosed(WindowEvent we)**
 3. **void windowClosing(WindowEvent we)**
 4. **void windowDeactivated(WindowEvent we)**
 5. **void windowDeiconified(WindowEvent we)**
 6. **void windowIconified(WindowEvent we)**
 7. **void windowOpened(WindowEvent we)**



THE WINDOWLISTENER INTERFACE

- The **windowActivated()** and **windowDeactivated()** methods are invoked when a window is activated or deactivated, respectively.
- If a window is iconified, the **windowIconified()** method is called.
- When a window is deiconified, the **windowDeiconified()** method is called.
- When a window is opened or closed, the **windowOpened()** or **windowClosed()** methods are called, respectively.
- The **windowClosing()** method is called when a window is being closed.

