Event Handling

Delegation Event Model

- The **delegation event model**, it defines standard and consistent mechanism to generate and process events.
- Here, **source** generates an event and sends it to one or more **listeners.** Listener processes the event and then returns.
- Listener must register with a source in order to receive an event notification. Notifications are sent only to listeners that want to receive them.
- Advantage of this design is, the application that processes events is cleanly separated from the user interface logic that generates these events.

Event handling

- Event Handling is the mechanism that controls the event and decides what should happen if an event occurs.
- **Steps**:
  a) The User clicks the button and the event is generated.
  b) Now the object of concerned event class is created automatically and information about the source and the event get populated with in same object.
  c) Event object is forwarded to the method of registered listener class.
  d) The method is now executed and then returns.

Event

- An **event** is object that describes the state change in a source. Event can be generated when user interacts with elements in a GUI.

Types of Event

- **Foreground Events** - Those events which require the direct interaction of user. They are generated as consequences of a person interacting with the graphical components in Graphical User Interface.
- For example: clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page etc.
- **Background Events** - Those events that don't require the direct interaction of end user are known as background events.
- For example: Operating system interrupts, hardware or software failure, timer expiry, an operation completion etc.

Event Source

- A **source** is an object that generates an event. It occurs when internal state of that object changes in some way. Sources may generate more than one type of event.
- Source must register listeners in order for the listeners to receive notifications

  ```
  public void addTypeListener(TypeListener el)
  ```

  type is name of the event and el is reference to event listener
- Example: The method that registers a keyboard event listener is called **addKeyListener()**

- When event occurs all registered listeners are notified and receive a copy of the event object. This is known as multicasting the event.
- Some sources may allow only one listener to register. It is called as unicasting the event. General form is

```
public void addTypeListener(TypeListener el) throws java.util.TooManyListenersException
```

- Source also provides a method that allows listener to unregister a specific type of event. General form is

```
public void removeTypeListener(TypeListener el)
```

- Example: To remove keyboard listener, you use **removeKeyListener()**

<u>Event Listeners</u>

- A **Listener** is an object that is notified when event occurs. It has two major requirements
- First, it must have registered with one or more sources to receive notifications about specific type of events.
- Second, it must implement methods to receive and process these notifications.
- Methods that receives and process events are defined in set of interfaces found in awt.

  The following table shows some of the classes & respective listener interfaces belonging to java.awt.event package.

Event Classes	**Listener Interfaces**
ActionEvent	ActionListener
AdjustmentEvent	AdjustmentListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener
InputEvent	InputListener
ItemEvent	ItemListener
KeyEvent	KeyListener
MouseEvent	MouseListener
MouseWheelEvent	MouseWheelListener
TextEvent	TextListener
WindowEvent	WindowListener

<u>Event Classes</u>

- An **Event class** provides a consistent, easy-to-use means of encapsulating events.
- **EventObject** is super class of all events. It is at the root of java event class hierarchy, which is in java.util package.
- It has one constructor

```
EventObject(Object src)
```

  Here src is the object that generates this event.
- It has two methods

  getSource() - returns the source of the event.

```
Object getSource()
```

  toString() - returns the string equivalent of the event.

1. **ActionEvent**
    - Generated when button is pressed, list item is double-clicked or item is selected.
    - The ActionEvent class defines 4 integer constants ALT_MASK, CTRL_MASK, META_MASK and SHIFT_MASK
    - The getActionCommand() method return command string associated with action

2. **AdjustmentEvent -**
    - Generated when scroll bar is manipulated.
    - BLOCK_DECREMENT, BLOCK_INCREMENT, TRACK, UNIT_DECREMENT, UNIT_INCREMENT
    - The getAdjustable() method returns object that generated the event
      Adjustable getAdjustable()

3. **ComponentEvent -**
    - Generated when component is hidden, moved, resized or becomes visible.
    - COMPONENT_HIDDEN, COMPONENT_MOVED, COMPONENT_RESIZED, COMPONENT_SHOWN
    - The getComponent() returns the component that generated event
      Component getComponent()

4. **ContainerEvent**
    - Generated when component is added or removed from container.
    - The getContainer() returns container that generated the event.
      Container getContainer()
    - The getChild returns reference to component that was added or removed
      Component getChild()
5. **FocusEvent -**
    - Generated when component gains or losses keyboard focus.
    - It has 2 types identified by integer constants FOCUS_GAINED, FOCUS_LOST.
    - The getOppositeComponent() returns the other component.

6. **InputEvent -**
    - The abstract class InputEvent is subclass of ComponentEvent and is the superclass for component input events.
    - It has 8 types identified by integer constants ALT_MASK, CTRL_MASK, BUTTON1_MASK, BUTTON2_MASK etc.
    - The methods such as isAltDown(), isControlDown() are used to test which modifiers were pressed.

7. **ItemEvent -**
    - generated when a checkbox or list item is clicked, item is selected or deselected.
    - It has one integer constant ITEM_STATE_CHANGED & it signifies change of state.
    - The getItem() is used to obtain reference to the item that generated an event.
    - The getItemSelected() used to obtain ref to the object that generated an event.

8. **KeyEvent -**
    - generated when input is received from the keyboard
    - There are 3types of key events: KEY_PRESSED, KEY_RELEASED, KEY_TYPED.
    - There are many integer constants, eg: VK_0 through VK_9, VK_A through VK_Z etc define the ASCII equivalents of the numbers and letters.
    - The VK constants specify virtual key & are independent of any modifiers such as control, shift or alt.
    - KeyEvent is subclass of InputEvent.
    - The getKeyChar() method returns the character that was entered & getKeyCode() returns the key code associated with a key.
    - For no values entered, method returns CHAR_UNDFINED or VK_UNDEFINED

9. **MouseEvent -**
    - generated when there is mouse manipulation.
    - There are 8 types of mouse events & integer constants are used to identify them.
    - MOUSE_CLICKED, MOUSE_DRAGGED, MOUSE_ENTERED, MOUSE_EXITED, MOUSE_MOVED, MOUSE_PRESSED, MOUSE_RELEASED, MOUSE_WHEEl.
    - The int getX() & int getY() are commonly used methods used to obtain the X & Y co-oridantes of the mouse.
    - The int getClickCount() method obtains the number of mouse clicks performed.
    - The isPopupTrigger() method tests if event causes a pop-up menu to appear on this platform.

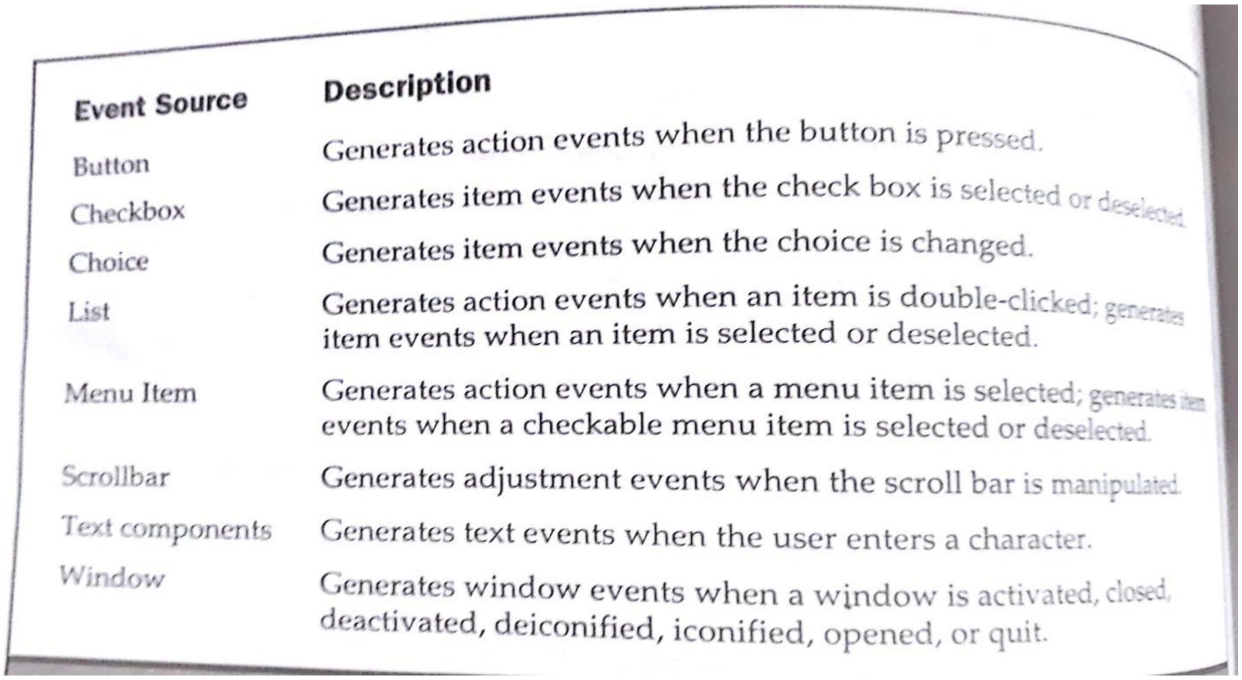
10. **MouseWheelEvent -**
    - generated when mouse scrolling is performed
    - This class encapsulates a mouse wheel event. It is subclass of MouseEvent.
    - It defines 2 integer constants WHEEL_BLOCK_SCROLL, WHEEL_UNIT_SCROLL
    - The getScrollType() method is used to identify the type of scroll performed.

11. **TextEvent -**
    - generated by text fields and text areas when characters are entered by user.
    - It has integer constant called TEXT_VALUE_CHANGED.
    - TextEvent(Pbjectsrc, int type) src is reference to the object that generated the event, type specifies type of event.
    - TextEvent object does not include characters currently in the text component that generated the event, hence no methods are available here.

12. **WindowEvent -**
    - generated when window is activated/deactivated, gain/lost focus etc.
    - There are 10 types of Windows event, defined using integer constants.
    - The getWindow() method returns the opposite window(when focus event has occurred), the previous window state, and current window state.
    - The int getOldState() and int getNewState() method returns the old & new state of the WindowEvent.

<u>Sources of Events</u>

- A **source** is an object that generates an event.
- It occurs when internal state of that object changes in some way.
- Sources may generate more than one type of event.
- Following table lists some of the user interfaces that can generate the events.
- In addition to GUI, other components such as an applet can also generate events.



Event Source	Description
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 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.

<u>Event Listener Interface</u>

- An **EventListener** represents the interfaces responsible to handle events.
- when an event occurs, source invokes the appropriate method defined by the listener and provides an event object as its argument.
- Its compulsory to implement all the methods under a interface.
- The most commonly used EventListeners interfaces along with the methods under it are listed below for reference:

1. **ActionListener**
   void actionPerformed(ActionEvent ae)

2. **AdjustmentListener**
   void adjustmentValueChanged(AdjustmentEvent ae)

3. **ComponentListener**
   void componentResized(CompoenentEvent ce)
   void compoenentMoved(CompoenentEvent ce)
   void componentShown(CompoenentEvent ce)
   void componentHidden(CompoenentEvent ce)

4. **ContainerListener**
   void componentAdded(ContainerEvent ce)
   void componentRemoved(ContainerEvent ce)

5. **FocusListener**
   void focusGained(FocusEvent fe)
   void focusLost(FocusEvent fe)
6. **ItemListener**
   void itemStateChanged(ItemEvent ie)

7. **KeyListener**
   void keyPressed(KeyEvent ke)
   void keyReleased(KeyEvent ke)
   void keyTyped(KeyEvent ke)

8. **MouseListener**
   void mouseClicked(MouseEvent me)
   void mouseEntered(MouseEvent me)
   void mouseExited(MouseEvent me)
   void mousePressed(MouseEvent me)
   void mouseReleased(MouseEvent me)

9. **MouseMotionListener**
   void mouseDragged(MouseEvent me)
   void mouseMoved(MouseEvent me)

10. **MouseWheelListener**
    void mouseWheelMoved(MouseWheelEvent mwe)

11. **TextListener**
    void textChanged(TextEvent te)

12. **WindowFocusListener**
    void windowGainedFocus(WindowEvent we)
    void windowLostFocus(WindowEvent we)

13. **WindowListener**
    void windowActivated(WindowEvent we)
    void windowClosed(WindowEvent we)
    void windowClosing(WindowEvent we)
    void windowDeactivated(WindowEvent we)
    void windowDeconified(WindowEvent we)
    void windowIconified(WindowEvent we)
    void windowOpened(WindowEvent we)

Handling Windows Events

1. Handling Mouse Events

- To handle mouse events, we must implement **MouseListener** and **MouseMotionListener** interfaces.
- The interfaces contain methods that receive and process the various types of mouse events.
- Inside init() method, the applet registers itself as a listener to mouse events by using **addMouseListener**() and **addMouseMotionListener**().
- **Example**

```
import  java.awt.*;
import  java.awt.event.*;
import  java.applet.*;
/*<applet code="MouseEvents" width=300 height=400></applet>*/
public class MouseEvents extends Applet implements MouseListener, MouseMotionListener
{
        String msg=" ";
        public void init()
        {
                addMouseListener(this);
                addMouseMotionListener(this);
        }
        public void paint(Graphics g)
        {
                g.drawString(msg,20,20);
        }
        public void mousePressed(MouseEvent me)
        {
                msg="mouse pressed";
                repaint();
        }
        public void mouseClicked(MouseEvent me)
        {
                msg="mouse clicked";
                repaint();
        }
        public void mouseEntered(MouseEvent me)
        {
                msg="mouse entered";
                repaint();
        }
        public void mouseExited(MouseEvent me)
        {
                msg="mouse exited";
                repaint();
        }
        public void mouseReleased(MouseEvent me)
        {
                msg="mouse released";
                repaint();
        }
        public void mouseMoved(MouseEvent me)
        {
                msg="mouse moved";
                repaint();
        }
        public void mouseDragged(MouseEvent me)
        {
                msg="mouse Dragged";
                repaint();
        }
}
```

2. Handling Keyboard Events

- To handle keyboard events, we must implement **KeyListener** interface.
- The **KeyEvents** class extends Applet class & implements the interface
- **Example - Lab Assignment Program - 2**

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*<applet code="KeyEvents" width=300 height=400></applet>*/
public class KeyEvents extends Applet implements KeyListener
{
        String msg="KeyEvents--->";
        public void init()
        {
                addKeyListener(this);
        }
        public void paint(Graphics g)
        {
                g.drawString(msg,20,20);
        }
        public void keyPressed(KeyEvent k)
        {
                showStatus("KeyDown");
                int key=k.getKeyCode();
                switch(key)
                {
                        case  KeyEvent.VK_UP:
                                showStatus("Move to Up");
                                break;
                        case  KeyEvent.VK_DOWN:
                                showStatus("Move to Down");
                                break;
                        case  KeyEvent.VK_LEFT:
                                showStatus("Move to Left");
                        break;
                case  KeyEvent.VK_RIGHT:
                        showStatus("Move to Right");
                        break;
            }
            repaint();
        }
        public void keyReleased(KeyEvent k)
        {
            showStatus("Key Up");
        }
        public void keyTyped(KeyEvent k)
        {
            msg+=k.getKeyChar();
            repaint();
        }
    }
```

Adapter classes

- Adapter classes are useful when you want to receive and process only some of the events that are handled.
- They are used in order to reduce the burden which is imposed by interfaces(which makes compulsion of implementing all the methods).
- **Example - Lab Assignment Program - 1**

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*<applet code = "AdapterDemo" width=300 height=400></applet>*/
public class AdapterDemo extends Applet
{
    public void init()
    {
        addMouseListener(new MyMouseAdapter(this));
        addMouseMotionListener(new MyMouseMotionAdapter(this));
    }
}
class MyMouseAdapter extends MouseAdapter
{
    AdapterDemo ad;
    public MyMouseAdapter(AdapterDemo ma)
    {
        ad = ma;
    }
    public void mousePressed(MouseEvent me)
    {
        ad.showStatus("Mouse Pressed");
    }
    public void mouseReleased(MouseEvent me)
```

```
		{
			ad.showStatus("Mouse Released");
		}
	}
	class MyMouseMotionAdapter extends MouseMotionAdapter
	{
		AdapterDemo ad;
		public MyMouseMotionAdapter(AdapterDemo ma)
		{
			ad = ma;
		}
		public void mouseMoved(MouseEvent me)
		{
			ad.showStatus("Mouse Moved");
		}
	}
```

Inner classes

- A class defined within another class is called inner class.
- In InnerClassDemo, since MyMouseAdapter is inside its scope it has access to all of variables and methods within its scope.
- Advantage - It no longer needs to do this by storing reference to the applet.
- **Example**

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/* <applet code="InnerClassDemo" width=300 height=400> </applet>*/
public class InnerClassDemo extends Applet
{
    public void init()
    {
        addMouseListener(new MyMouseAdapter());
    }
    class MyMouseAdapter extends MouseAdapter
    {
        public void mousePressed(MouseEvent me)
        {
            showStatus("Mouse Pressed");
        }
    }
}
```