Chapter 18

EVENT HANDLING

In this chapter, you will learn about event handling. In a GUI environment, events such as mouse click, a key press, press of a button, etc. are to be captured and appropriate actions are to be carried out. Java uses a delegation event model to handle events and they are discussed in this chapter.

18.1 Delegation Event Model

In a Graphical User Interface(GUI) environment, actions are initiated by the press of a button, click of a button, a key press, etc. Therefore, appropriate mechanisms are needed to capture such events and to react to the events by executing a piece of code. Java provides such mechanisms. Events in Java are handled by delegation event model. In this model, there is a source, which generates events. There is a listener, which can listen to the happenings of an event and initiate an action (fig. 18.1).

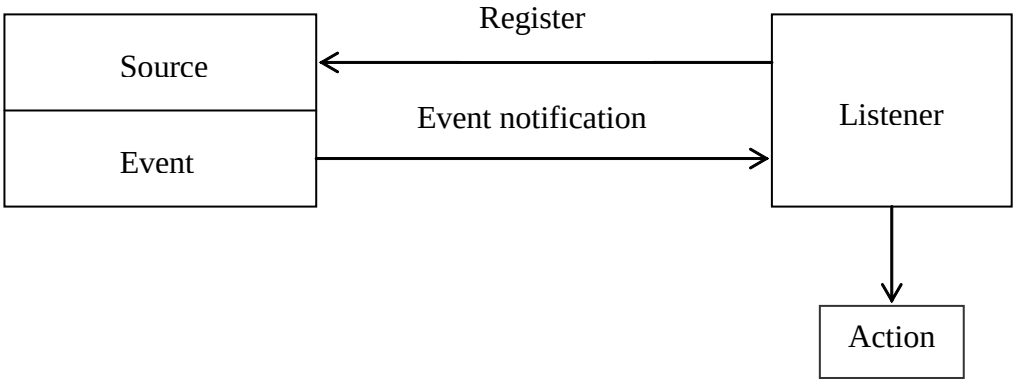

Fig.18.1 Source and Listener of Events

A listener has to register with a source. Any number of listeners can register with a source, except in a few cases. A listener can register with many event sources. When an event takes place, it is notified to the listeners, which are registered with the source. The listener then initiates an action.

18.2 Events

An event is an object that describes the change of state of a source. For example, a mouse click is an event from the source mouse. The superclass of all events is **java.util.EventObject**. The superclass of all AWT events is **java.util.AWTEvent** and is a subclass of **EventObject**. The class hierarchy is given in fig.18.2.

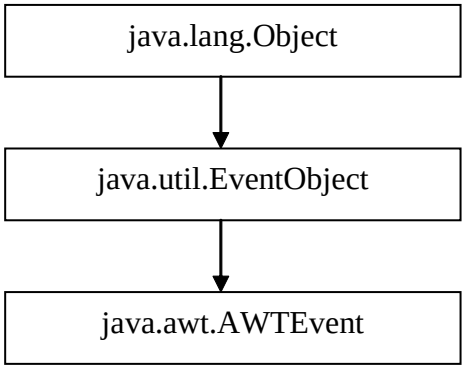

Fig.18.2 Class Hierarchy of AWT Event Class

AWTEvent class is an abstract class and contains many subclasses, which are concrete and are packaged in **java.awt.event**. One important method defined in **AWTEvent** class is **getID(),** which returns an **int** representing the type of event.

An event is an object that describes the change of state of a source.

Some of the classes defined in **java.awt.event** package are given in table 18.1. All of them are subclasses of **AWTEvent** class.

Table 18.1 Some of the Event Classes Defined in java.awt.event

	Name of Class	Event Dealt
1.	ActionEvent	This class deals with high-level event. The event occurs when the component-specific action takes place
2.	AdjustmentEvent	This class deals with events generated by the adjustable objects like scrollbar
3.	ComponentEvent	This class deals with the lower-level event. The event occurs when the component is moved, resized or visibility is changed
4.	ItemEvent	This class deals with the events generated when a check box or list item is clicked
5.	KeyEvent	This class deals with the events generated by key strokes
6.	MouseEvent	This class deals with events generated by mouse clicks and movements
7.	TextEvent	This class deals with events generated by the change of object's text
8.	WindowEvent	This class deals with events generated by the change of window status

The event class hierarchy is given in fig. 18.3.

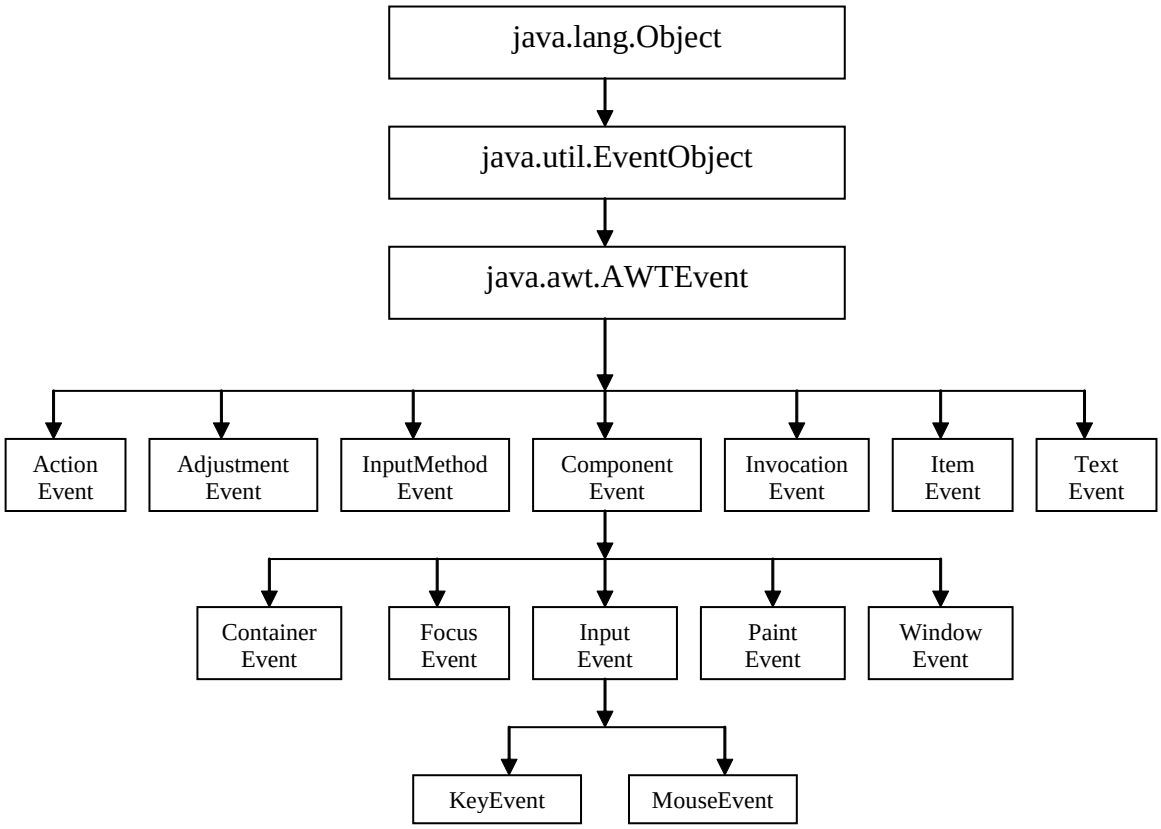

Fig.18.3 Event Class Hierarchy

18.2.1 The ActionEvent Class

An **ActionEvent** is generated when a button is pressed or a menu item is selected. This is a high-level or semantic event. This is in contrast to a low-level event like mouse click. This class has the following int type constants.

ALT_MASK - The alt modifier
An indicator that the alt key was held down during the event

CTRL_MASK - The control modifier
An indicator that the control key was held down during the event

META_MASK - The meta modifier
An indicator that the meta key was held down during the event

SHIFT_MASK - The shift modifier
An indicator that the shift key was held down during the event

Constructors

The constructors for **ActionEvent** class are :

ActionEvent (Object src, int id, String cmd)
ActionEvent (Object src, int id, String cmd, int modifier)

where src is the source object that generated this event, id is an int that identifies the event, cmd is the command associated with the event and modifier indicates which modifier key was pressed when the event was generated.

Methods

Some of the methods defined in **ActionEvent** class are :

String getActionCommand()
Returns the command name for the invoking ActionEvent object

int getModifier()
Returns an int value that indicates which modifier key was pressed when the event was generated

String paramString()
Returns a string identifying the event

18.2.2 The AdjustmentEvent Class

The adjustment event is generated by a scroll bar. Five types of adjustment events are defined for the adjustment of a scroll bar. Each type is identified by an integer constant. The constants are:

BLOCK_DECREMENT	-	The mouse is clicked inside the scroll bar to decrease its value
BLOCK_INCREMENT	-	The mouse is clicked inside the scroll bar to increase its value
TRACK	-	The slider is dragged
UNIT_DECREMENT	-	The button at the end of the scroll bar is clicked to decrease its value
UNIT_INCREMENT	-	The button at the end of the scroll bar is clicked to increase its value

The constants representing the different locations of a scroll bar are shown in fig. 18.4.

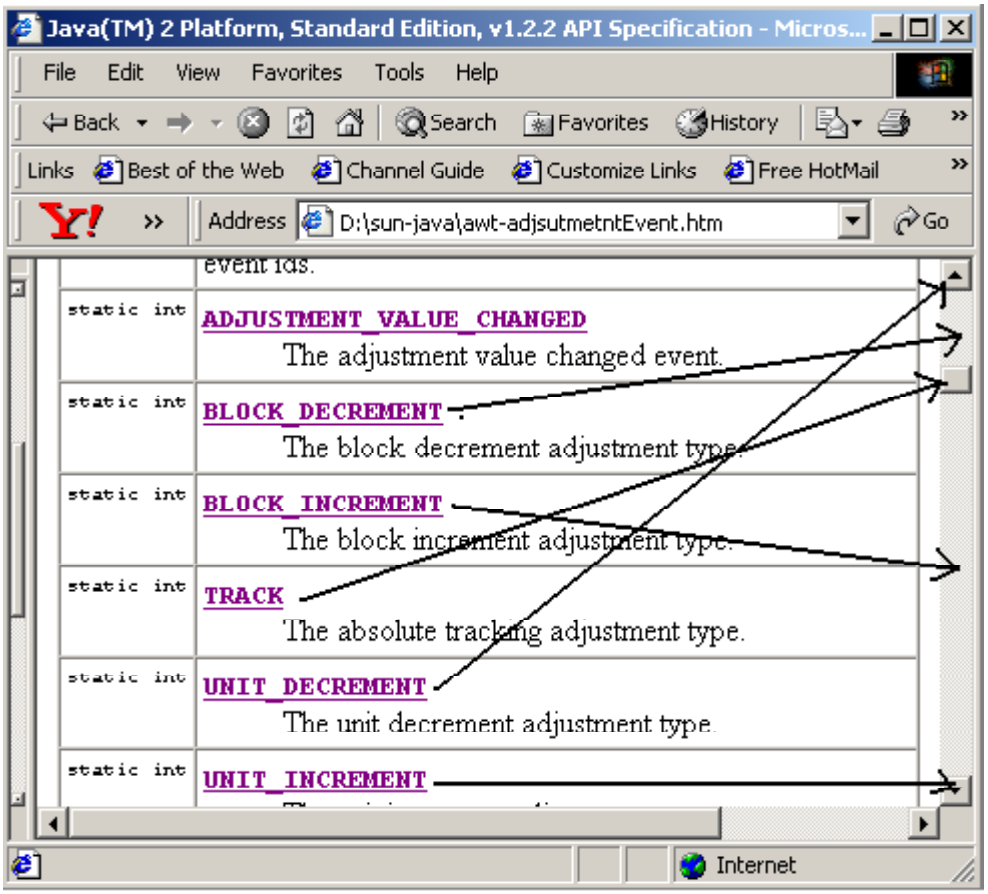

Fig.18.4 Constants Representing Different Locations on a Scroll Bar

Other integer constant associated with the event is:

ADJUSTMENT_VALUE_CHANGED, which represents the adjustment value changed event.

Constructor

The constructor for this class is:

AdjustmentEvent(Adjustable src, int id, int type, int value)
where src is the adjustable object where the event originated,
id is the event type,
type is the adjustment type,
value is the current value of the adjustment.

Methods

The methods defined in this class are :

Adjustable getAdjustable()
Returns the adjustable object where this event originated

int getAdjustableType()
Returns the type of adjustment which caused the value changed event

int getValue()
Returns the current value in the adjustment event

String paramString()
Returns a string representing the state of this event

18.2.3 The ComponentEvent Class

A component is an object having a graphical representation that can be displayed on the screen and that can interact with the user. Buttons, checkboxes and scroll bars are examples for component. Events generated by these components are called component events. An event is generated when a component is moved, changed in size or changed in visibility. It is a low-level event.

The event class has the following integer constants, each representing an event:

COMPONENT_MOVED	-	This event indicates that the component position has changed
COMPONENT_RESIZED	-	This event indicates that the component size has changed
COMPONENT_SHOWN	-	This event indicates that the component was made visible
COMPONENT_HIDDEN	-	This event indicates that the component was made invisible

Constructor

This class has the following constructor:

ComponentEvent(Component src, int id)
where src is the component object that generated event and
id indicates the type of event.

Methods

The methods defined in this class are :

Component getComponent()
Returns the originator of the event

String paramString()
Returns the string identifying the event

18.2.4 The ItemEvent Class

A semantic event indicates that an item, like check box or choice is selected or deselected. This is a high-level event. The following integer constants are defined in this class representing an event:

DESELECTED	This state change value indicates that an item is deselected
ITEM_STATE_CHANGED	This event indicates that an item's state has changed
SELECTED	This state change value indicates that an item is selected

Constructor

The constructor for this **ItemEvent** class is :

ItemEvent(ItemSelectable src, int id, Object item, int stateChange)
where, src is the ItemSelectable object that originated the event, id indicates the type of event, item is an object that is affected by the event, stateChange indicates whether the item was selected or deselected.

Methods

Methods defined in this class are :

ItemSelectable getItemSelectable()
Returns the ItemSelectable object that originated the event

Object getItem()
Returns the item Object that was affected by the event

int getStateChange()
Returns an integer that indicates whether the item was selected or deselected

String paramString()
Returns a string identifying the event

18.2.5 The KeyEvent Class

The key event is generated when a key is pressed, typed or released. The key typed event is generated only when a character is generated. For example, pressing an **Alt** key does not produce a character. Java uses Virtual Key code to represent which key on the keyboard has been pressed rather than which character was generated. There are several integer constants defined in this class. Some of them are given below:

KEY_TYPED	This event is generated when a character is entered.
KEY_PRESSED	This event is generated when a key is pushed down.
KEY_RELEASED	This event is generated when a key is released.
VK_0 to VK_9	Represents the keys ASCII 0 to ASCII 9
VK_A to VK_Z	Represents the keys ASCII A to ASCII Z

Other Virtual Key (VK) constants which are self explained are :

VK_ENTER	**VK_BACK_SPACE**	**VK_TAB**
VK_CANCEL	**VK_CLEAR**	**VK_SHIFT**
VK_CONTROL	**VK_CAPS_LOCK**	**VK_ESCAPE**
VK_SPACE	**VK_PAGE_UP**	**VK_PAGE_DOWN**
VK_LEFT	**VK_UP**	**VK_RIGHT**
VK_DOWN	**VK_COMMA**	**VK_MINUS**
VK_DELETE	**VK_F1 to VK_F24**	**VK_AMPERSAND**
VK_LESS	**VK_CUT**	**VK_COPY**

Constructors

The constructors defined in this class are:

KeyEvent (Component src, int id, long when, int modifier, int keycode, char keyChar)

KeyEvent (Component src, int id, long when, int modifier, int keyCode)
where, src is the Component that originated the event,
id is an integer identifying the event type,
when is a long integer which specifies the time at which the event occured,
modifier is the modifier key down during the event,
keyCode is the integer code for an actual key,
keyChar is the Unicode character generated by this event.

Methods

Methods defined in this class are :

int getKeyCode()
Returns the integer code for an actual key on the keyboard

void setKeyCode(int keyCode)
Sets the keyCode value to represent a physical key

void setKeyChar(char keyChar)
Sets the keychar value to represent a logical character

char getKeyChar()
Returns the Unicode character defined for this key event; if no valid Unicode character is defined for this event, the character generated is CHAR_UNDEFINED.

String getKeyText(int keyCode)
Returns a String describing the keyCode such as “HOME”, “F1” or “E”

String getKeyModifierText(int modifiers)
Returns a String describing the modifier keys such as “Shift” or “shift + ctrl” that were held down during the event

boolean isActionKey()
Returns true if the key is an action key

String paramString()
Returns a parameter string identifying this event

18.2.6 The MouseEvent Class

A mouse event is generated by mouse action in a component. The following events are generated in a component by a mouse action. Two types of events, mouse event and mouse motion event, are generated by a mouse. Mouse events are generated when a mouse button is pressed, released, clicked, mouse enters a component or mouse exits a component. Mouse motion events are generated when the mouse is moved or dragged. The following integer constants are defined for the mouse events:

MOUSE_CLICKED This represents the mouse clicked event. This MouseEvent occurs when a mouse button is pressed and released.

MOUSE_ENTERED This represents the mouse entered event. This MouseEvent occurs when a mouse cursor enteres a component’s area.

MOUSE_EXITED This represents the mouse exited event. This MouseEvent occurs when a mouse cursor exits a component’s area.

MOUSE_PRESSED This represents mouse pressed event. This MouseEvent occurs when a mouse button is pushed down.

MOUSE_RELEASED This represents mouse released event. This MouseEvent occurs when a mouse button is released.

MOUSE_DRAGGED	This represents a mouse dragged event. This MouseMotionEvent occurs when a mouse is dragged
MOUSE_MOVED	This represents a mouse moved event. This MouseMotionEvent is generated when the mouse is moved

Constructor

The constructor for this class is:

MouseEvent(Component src, int id, long when, int modifiers,
int x, int y, int clickCount, boolean pupTrig)
where src is the Component that originated the event,
id identifies the event,
when gives the time the event occurred,
modifiers is the modifier key down during the event,
x is the x co-ordinate of the mouse location,
y is the y co-ordinate of the mouse location,
clickCount is the number of mouse clicks associated with the event,
pupTrig is true, if this event is a trigger for a popup menu.

Methods

Methods defined in this class are:

int getX()
Returns an integer representing the x position of the event relative to the component

int getY()
Returns an integer representing the y position of the event relative to the component

void translatePoint(int x, int y)
Translates the event's co-ordinates to a new position by adding x and y to the x and y of the current position

int clickCount()
Returns the number of clicks associated with this event

boolean isPopupTrigger()
Returns true, if this event is the popup–menu trigger for this platform

String paramString()
Returns a string identifying this event

18.2.7 The TextEvent Class

A text event is generated when the text of an object is changed. This class has an integer constant **TEXT_VALUE_CHANGED,** which indicates that the object's text is changed.

Constructor

This class has the following constructor:

TextEvent(Object src, int id)

where src is the text component object that originated the event and id identifies the type of event.

Method

The method defined in this class is:

String paramString()

Returns a string identifying this text event

18.2.8 The WindowEvent Class

A window event indicates the change in the status of the window. This event is generated when it is opened, closed, about to close, activated, deactivated, iconified or deiconified. This class has integer constants that represent different window events. Some of them are given below:

WINDOW_ACTIVATED	This represents a window activated-event. This event occurs when the window becomes the user's active window.
WINDOW_CLOSED	This represents a window-closed event. This event occurs after the window has been closed.
WINDOW_CLOSING	This represents a window is closing event. This event occurs when the user attempts to close the window from the windows system menu.
WINDOW_DEICONIFIED	This represents a window-deiconified event. This event occurs when the window has been changed from a minimized state to a normal state.
WINDOW_ICONIFIED	This represents a window iconified event. This event occurs when the window has been changed from a normal state to a minimized state.
WINDOW_OPENED	This represents a window-opened event. This event occurs when the window is made visible.

Constructor

The constructor for this class is:

WindowEvent(Window src, int id)

where, src is the Window object that originated the event and id indicates the type of event.

Method

The method defined in this class is:

Window getWindow()

Returns the Window object that originated the event

18.3 Event Listeners

As mentioned earlier, when events occur they are notified to listeners, which are registered with the source. An event listener has to initiate some action when the happening of an event is notified to it. What action is to be taken depends on the problem handled by the programmer. Therefore, the methods designed to handle events are to be open. Several interfaces are designed for listeners with event as a type signature. The listener interfaces are defined in **java.awt.event** package. Some of the listener interfaces are given in table 18.2.

Table 18.2 Some Listener Interfaces and Methods Defined in Them

Name of Interface	Interface Methods
1. ActionListener	void actionPerformed(ActionEvent ae)
2. AdjustmentListener	void adjustmentValueChanged (AdjustmentEvent ae)
3. ComponentListener	void componentHidden(ComponentEvent ce) void componentMoved(ComponentEvent ce) void componentResized(ComponentEvent ce) void componentShown(ComponentEvent ce)
4. ItemListener	void itemStateChanged(ItemEvent ie)
5. KeyListener	void keyPressed(KeyEvent ke) void keyReleased(KeyEvent ke) void keyTyped(KeyEvent ke)

6.	MouseListener	void mouseClicked(MouseEvent me) void mouseEntered(MouseEvent me) void mouseExited(MouseEvent me) void mousePressed(MouseEvent me) void mouseReleased(MouseEvent me)
7.	MouseMotionListener	void mouseDragged(MouseEvent me) void mouseMoved(MouseEvent me)
8.	TextListener	void textValueChanged(TextEvent te)
9.	WindowListener	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)

18.4 Registering Listeners with Source

Listeners have to register with the event source to get event notification. **add** methods are used for registering. All types of listeners have the same syntax for the **add** method. The general form of **add** method is :

public void addTypeListener(TypeListener el)

where type indicates the type of event and el is the event listener. For example, for mouse motion event, **addMouseMotionListener()** is used. The add methods defined in various classes are given in table 18.3.

Table 18.3 Some of the add/remove Methods Defined in Component Class

	Method	Class
1.	addComponentListener(ComponentListener cl)	Component
2.	addKeyListener(KeyListener kl)	Component
3.	addMouseMotionListener(MouseMotionListener ml)	Component
4.	addMouseListener(MouseListener ml)	Component
5.	removeKeyListener(KeyListener kl)	Component
6.	removeMouseMotionListener(MouseMotionListener ml)	Component
7.	removeMouseListener(MouseListener ml)	Component
8.	addWindowListener(WindowListener wl)	Window
9.	removeWindowListener(WindowListener wl)	Window

10.	addActionListener(ActionListener al)	Button, List, MenuItem, TextField
11.	removeActionListener(ActionListener al)	TextField
12.	addAdjustmentListener(AdjustmentListener al)	Scrollbar
13.	addItemListener(ItemListener il)	Check box CheckboxMenuItem Choice, List

Listeners listen to the occurrence of an event and take appropriate action. Listeners have to register with an event source. add methods are used for registering with the source.

18.5 Example Programs

Having seen what an event is, how it is generated and how to listen to such events, we will see how to write programs to handle the events.

18.5.1 Mouse Event Handling

To listen to an event, all the methods in the corresponding listener interface are to be implemented. There are five **MouseEvent** and two **MouseMotionEvent**. To handle them, **MouseEventListener** and **MouseMotionListener** interfaces are to be implemented. Further, to register the listener with the event source, **addMouseEventListener** and **addMouseMotionListener** methods are to be used. The following program 18.1 illustrates the use of mouse handling methods:

Program 18.1

```
// This program illustrates the use of MouseEvent
// and MouseMotionEvent methods.
/*
<applet code = Mouse1 width = 200 height =100 >
</applet>
*/
import java.applet.Applet;
import java.awt.Color;
import java.awt.Graphics;
```

```
import java.awt.event.*;
public class Mouse1
    extends Applet
    implements MouseListener, MouseMotionListener
   {
   String txt = " Nothing";
   int x = 10, y = 30;
   public void init()
      {
      addMouseListener(this);
      addMouseMotionListener(this);
      }
    public void mouseClicked(MouseEvent me)
      {
      txt = "Mouse Clicked";
      repaint();
      }
    public void mouseEntered(MouseEvent me)
      {
      txt = "Mouse Entered";
      repaint();
      }
    public void mouseExited(MouseEvent me)
      {
      txt = "Mouse Exited";
      repaint();
      }
    public void mousePressed(MouseEvent me)
      {
      txt = "Mouse Pressed";
      setForeground(Color.cyan);
      repaint();
      }
    public void mouseReleased(MouseEvent me)
      {
      txt = "Mouse Released";
      setForeground(Color.magenta);
      repaint();
      }
    public void mouseDragged(MouseEvent me)
      {
      txt = "Mouse Dragged";
      setForeground(Color.red);
      repaint();
      }
    public void mouseMoved(MouseEvent me)
      {
      txt = "Mouse Moved";
      setForeground(Color.green);
      repaint();
```

```
  }
public void paint(Graphics gp)
    {
    gp.drawString(txt, 20, 40);
    showStatus("Mouse event handling");
    }
}
```

The above program gives the following outputs for various mouse events:

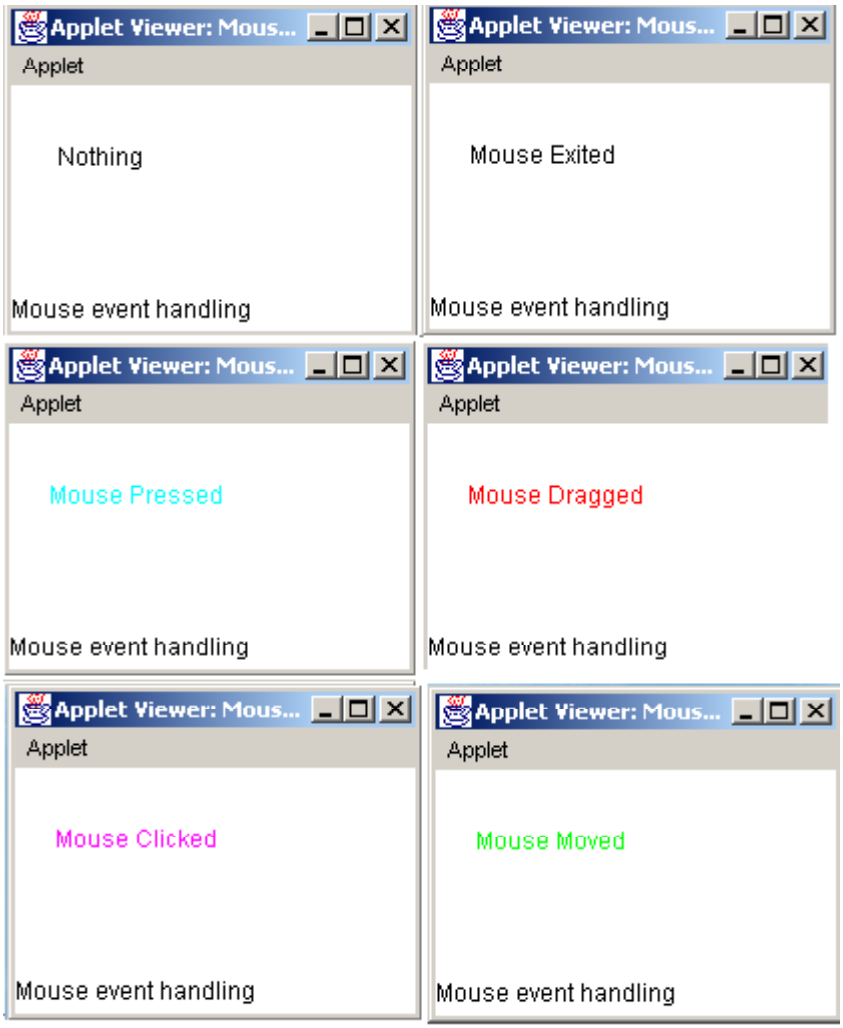

Fig.18.5 Output Screens for Program 18.1 (Six Screens)

18.5.2 Key Event Handling

The following program 18.2 illustrates the use of **KeyEventListener** and **KeyEvent** methods:

Program 18.2

```
// This program illustrates the use of KeyEvent  methods.
/*
<applet code = Key width = 200 height =100 >
</applet>
*/
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
public class Key
    extends Applet
    implements KeyListener
    {
    String txt = " ";
    String txt1 = " ";
    String txt2 = " ";
    String txt3 = " ";
    int kcode;
    char ch;
    int drawnnumber;
    public void init()
        {
        addKeyListener(this);
        requestFocus();
        }
    public void keyTyped(KeyEvent ke)
        {
        ch = ke.getKeyChar();
        txt1 += ch;
        if (txt1.length() > 25)
            txt1 = " ";
        txt = "Key Typed ";
        repaint();
        }
    public void keyPressed(KeyEvent ke)
        {
        kcode = ke.getKeyCode();
        if (kcode == ke.VK_F1)
            txt2 = "You have typed F1 key";
        if (kcode == ke.VK_SHIFT)
            txt3 = "You have typed Shift  key";
        txt = "Key Pressed";
        repaint();
        }
    public void keyReleased(KeyEvent ke)
        {
        txt = "Key released";
        repaint();
```

```
    }
  public void paint(Graphics gp)
    {
    gp.drawString(txt, 20, 20);
    gp.drawString(txt1, 20, 40);
    gp.drawString(txt2, 20, 60);
    gp.drawString(txt3, 20, 80);
    showStatus("Key events");
    }
}
```

The above program gives the following output:

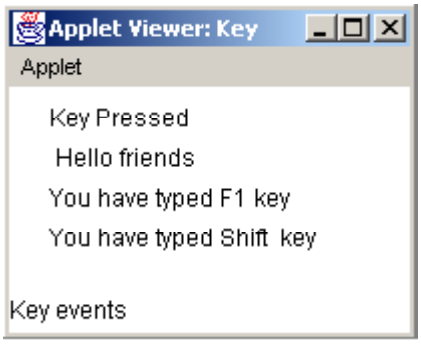

Fig.18.6 Output Screen for Program 18.2

In program 18.2, the method **requestFocus()** is required to receive the key strokes.

18.5.3 Window Event Handling

The following program 18.3 illustrates the use of methods in **WindowListener** interface and **WindowEvent** class. For handling window events, there are seven interface methods. All methods of the interface are to be implemented whether they are used in the program or not. In the following program, window is created using **Frame** class. A window created from **Frame** does not have a default window closing method. Hence, window closing event is captured and **System.exit(0)** is called to close the window.

Program 18.3

```
/* This program illustrates the use of methods in the
   WIndowListener interface. The window is closed
   by capturing the WindowClosing event. Whether you need
   one or all methods, you must implement all interface
  methods. */
import java.awt.Graphics;
import java.awt.Frame;
import java.awt.event.WindowEvent;
```

```
import java.awt.event.WindowListener;
class Myframe
    extends Frame
    implements WindowListener
    {
    String txt1 = " ";
    String txt2 = " ";
    String txt3 = " ";
    Myframe(String title)
        {
        super(title);
        setSize(200, 150);
        addWindowListener(this);
        }
    public void windowActivated(WindowEvent we)
        {
        txt1 = "Window activated";
        }
    public void windowClosed(WindowEvent we)
        {
        ;
        }
    public void windowClosing(WindowEvent we)
        {
        System.exit(0);
        }
    public void windowDeactivated(WindowEvent we)
        {
        txt2 = "Window deactivated";
        }
    public void windowDeiconified(WindowEvent we)
        {
        txt3 = "Window deiconified";
        }
    public void windowIconified(WindowEvent we)
        {
        ;
        }
    public void windowOpened(WindowEvent we)
        {
        ;
        }
    public void paint(Graphics gp)
       {
       gp.drawRoundRect(15, 25, 90, 20, 10, 5);
       gp.drawString("Window Event", 20, 40);
       gp.drawString("Click X in the window to close",
                                              20, 60);
       gp.drawString(txt1, 20, 80);
        gp.drawString(txt2, 20, 100);
```

```
        gp.drawString(txt3, 20, 120);
        }
    }
public class Winevent
    {
    public static void main(String args [])
        {
        Myframe mframe = new Myframe("Window Events");
        mframe.setVisible(true);
        }
    }
```

The above program gives the following screen output:

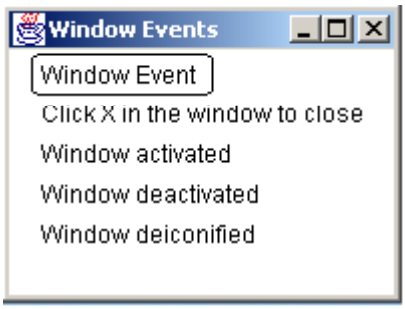

Fig.18.7 Output Screen for Program 18.3

Frame window does not have the conventional window closing method. Hence, manual coding is to be used (like system.exit(0)) to close the window.

18.6 Adapter Classes

In the previous section, we have seen that interface listeners are used to handle various types of events. Each interface has one or more methods. To use any one method in the interface, all methods are to be implemented. In many occasions, one or two methods may be needed out of several methods in an interface. For example, to use a **windowClosing** method in **windowListener** interface, all other six methods are to be implemented with null statements. This is a cumbersome process. To solve this problem, classes have been designed which implement all the methods of an interface, each method with a do-nothing code. Such classes are called **Adapter** classes. All interfaces that have more than one method have a campanion **Adapter** class. Subclasses of these **Adapter** classes can be created, in which the required method can be implemented. Subclasses of **Adapter** classes can implement one or more methods of the interface method, but not all of them.

An adapter class implements all the methods of an interface with do-nothing codes. When a few out of several interface methods are needed in an application, the adapter classes can be subclassed. This avoids implementing all unwanted interface methods. Interfaces having two or more methods have corresponding adapter classes.

The following program 18.4, illustrates the use of **WindowAdapter** of the **windowListener** interface:

Program 18.4

```
/*  This program illustrates the use of WindowAdapter.
    Adapter class helps to reduce the burden of implementing
    all interface methods other than the required ones.
 */
import java.awt.Graphics;
import java.awt.Frame;
import java.awt.event.WindowEvent;
import java.awt.event.WindowAdapter;
class winclose
    extends WindowAdapter
    {
    public void windowClosing(WindowEvent we)
        {
        System.exit(0);
        }
    }
class Myframe
    extends Frame
    {
    Myframe(String title)
      {
      super(title);
      setSize(200, 150);
      winclose wc = new winclose();
      addWindowListener(wc);
      }
    public void paint(Graphics gp)
       {
       gp.drawRoundRect(15, 25, 150, 20, 10, 5);
       gp.drawString("WindowAdapter demo", 20, 40);
       gp.drawString("Click x in the window to close",
                                          20, 60);
```

```
        }
    }
public class Winadapt
    {
    public static void main(String args [])
        {
        Myframe mframe = new Myframe("Window Events");
        mframe.setVisible(true);
        }
    }
```

The above program gives the following screen output:

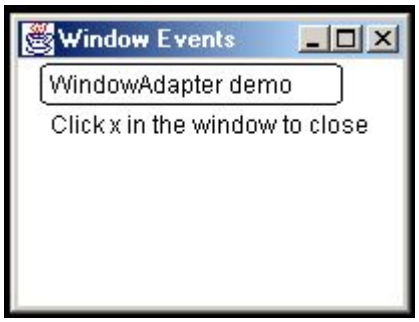

Fig.18.8 Output Screen for Program 18.4

After reading this chapter, you should have learned the following:

- Events are change of state of a source.
- Mouse, keyboard, buttons, checkboxes, scroll bar, text field act as source of events.
- Methods in listener interfaces can be used to capture events and take appropriate action.
- An adapter class implements all methods of an interface with a do-nothing code.
- Adapter classes provide an alternative and short-cut method to implement interfaces.

In the next chapter, you will learn about Swing and GUI components.

Worked Out Problem–18

Problem 18.1w

Write an applet that will accept a text from a keyboard and display it on the screen as a moving text from right to left. Use key event to input the text.

Program 18.1w

```
/* ------------------------------------------------------------
This program accepts the keys typed on the keyboard, makes a
string and displays it as a moving text.

somasundaramk@yahoo.com

<applet code = Prob181 width = 500 height =100 >
</applet>
------------------------------------------------------------ */

import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class Prob181
    extends Applet
    implements KeyListener, Runnable
     {
     String msg = " ";
     String submsg = " ";
     char fchr;
     int kcode;
     char ch;
     Font ft;
     boolean bigsize = false;
     boolean thstart = false;
     Thread th;
     public void init()
           {
           ft = new Font("Serif", Font.PLAIN, 65);
           addKeyListener(this);
           requestFocus();
           }
    public void keyTyped(KeyEvent ke)
        {
        ch = ke.getKeyChar();
        msg += ch;
        if (msg.length() > 45)
            msg = " ";
        repaint();
        }
```

```
public void keyPressed(KeyEvent ke)
    {
    kcode = ke.getKeyCode();
    if (kcode == ke.VK_ENTER)
        {
        bigsize = true;
        thstart = true;
        repaint();
        }
    }
public void start()
    {
    th = new Thread(this);
    }
public void run()
    {
    while (true)
      {
      fchr = msg.charAt(0);
      submsg = msg.substring(1);
      msg = submsg + fchr;
      try
           {
            Thread.currentThread().sleep(300);
            }
        catch (InterruptedException ie)
            {
            System.out.println("Interrupt error");
            }
        repaint();
        }
    }
public void keyReleased(KeyEvent ke)
    {
    repaint();
    }
public void paint(Graphics gp)
    {
    gp.drawString("Type a text for display and Press
                           Enter Key", 20, 90);
    if (bigsize)
        {
        gp.setFont(ft);
        }
    if (thstart)
        {
        th.start();

        thstart = false;
        }
```

```
        gp.drawString(msg, 20, 60);
        showStatus("Moving Text");
      }
   }
```

The above program gives the following output:

(a)

(b)

Fig.18.9 Output Screens for Program 18.1w

Exercise-18

I. Fill in the blanks:

18.1. Pressing a button of a mouse causes _____ event.

18.2. An event is generated by a _____ .

18.3. Java implements _____ model to handle events.

18.4. _____ is a consumer of events.

18.5. To get an event notification, a listener has to _____ with the _____.

18.6. To receive key strokes, the _____ method is to be called.

18.7. A class with do-nothing methods as an alternative to listener interface is called _____ class.

18.8. A user can implement (override) one or more number of methods of an adapter class but not _____ .

18.9. The general form of the method to register a listener with the event source is _____ .

II. Write a Java program for the following:

18.10. Write a program in which a sound clip is played when mouse is dragged.

18.11. Write a program to capture key events. When a (or A) is typed, APPLE is to be displayed on the applet window.

18.12. Write a program to implement mouseClicked method using MouseAdapter Class.

* * * * * *