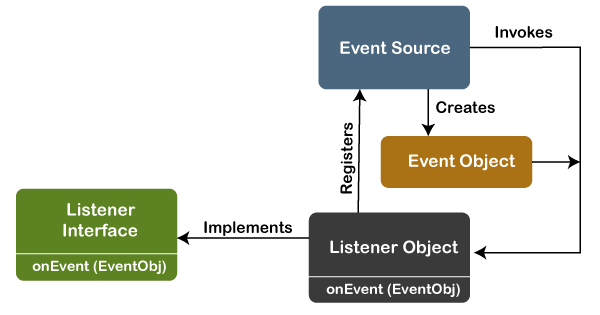
**Event Handling**

* **Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism have the code which is known as event handler that is executed when an event occurs. Java Uses the Delegation Event Model to handle the events.**
* Event handling isfundamental to Java programming because it is integral to the creation of applets andother types of GUI-based programs. Applets are event-drivenprograms that use a graphical user interface to interact with the user.
* Events are supported by a number of packages, including **java.awt**, and**java.awt.event**.
* Most events to which your program will respond are generated when the user interactswith a GUI-based program. There are several types of events, including those generated by the mouse, thekeyboard, and various GUI controls, such as a push button, scroll bar, or check box.

**The Delegation Event Model**



* The modern approach for event processing is based on the Delegation Model. It defines a standard and compatible mechanism to generate and process events. In this model, a source generates an event and forwards it to one or more listeners. The listener waits until it receives an event. Once it receives the event, it is processed by the listener and returns it. The UI elements are able to delegate the processing of an event to a separate function.
* The key advantage of the Delegation Event Model is that the application logic is completely separated from the interface logic.
* In this model, the listener must be connected with a source to receive the event notifications. Thus, the events will only be received by the listeners who wish to receive them. So, this approach is more convenient than the inheritance-based event model (in Java 1.0).
* In the older model, an event was propagated up the containment until a component was handled. This needed components to receive events that were not processed, and it took lots of time. The Delegation Event model overcame this issue.
* Basically, an Event Model is based on the following three components:
* Events
* Sources
* Listeners

**Events:**

* **An event is an object that describes a state change in a source.**
* It canbe generated as a consequence of a person interacting with the elements in a graphical userinterface. Some of the activities that cause events to be generated are pressing a button, enteringa character via the keyboard, selecting an item in a list, and clicking the 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.

**Event Sources**

* **A source generates an event and sends it to one or more listeners.**
* **Asource is an object that generates an event.**
* This occurs when the internal state of that objectchanges in some way. Sources may generate more than one type of event.Asource must register listeners in order for the listeners to receive notifications abouta specific type of event. Each type of event has its own registration method. Here is thegeneral form:

public void addTypeListener(TypeListenerel)

* 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 methodthat registers a mouse motion listener is called addMouseMotionListener( ). When an eventoccurs, all registered listeners are notified and receive a copy of the event object.
* A source must also provide a method that allows a listener to unregister an interestin a specific type of event.
* The general form of such a method is this:

public void removeTypeListener(TypeListenerel)

* Here, Type is the name of the event, and el is a reference to the event listener. For example,to remove a keyboard listener, you would call

removeKeyListener( ).

* The methods that add or remove listeners are provided by the source that generatesevents.
* For example, the Component class provides methods to add and remove keyboardand mouse event listeners.

**Event Listeners**

* **Alistener 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 aboutspecific types of events. Second, it must implement methods to receive and process thesenotifications.
* The methods that receive and process events are defined in a set of interfaces found injava.awt.event.
* For example, the MouseMotionListenerinterface defines two methods toreceive notifications when the mouse is dragged or moved.

**Event Classes**

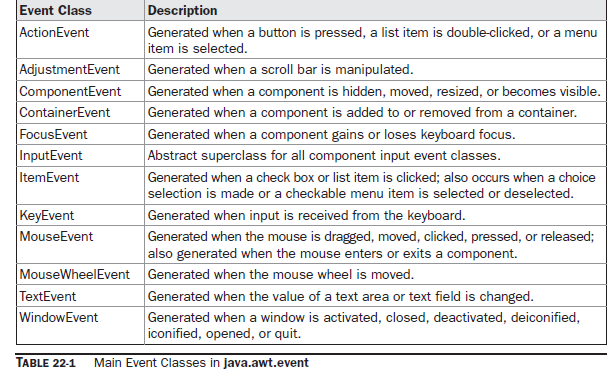
* **The classes that represent events are at the core of Java’s event handling mechanism. The most widely used events are those defined by the AWT and those defined by Swing.**

**COMPLETE DESCRIPTION OF delegation event model**

**Sources of Events**



**Event Classes**



**The ActionEvent Class**

An **ActionEvent**is generated when a button is pressed, a list item is double-clicked, or amenu item is selected. There is an integer constant, **ACTION\_PERFORMED**, which can be used to identify action events.

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

String getActionCommand( )

For example, when a button is pressed, an action event is generated that has a commandname equal to the label on that button.

**The ComponentEvent Class**

A**ComponentEvent**is generated when the size, position, or visibility of a component ischanged. There are four types of component events. The **ComponentEvent**class definesinteger constants that can be used to identify them. The constants and their meanings areshown here:

COMPONENT\_HIDDEN The component was hidden.

COMPONENT\_MOVED The component was moved.

COMPONENT\_RESIZED The component was resized.

COMPONENT\_SHOWN The component became visible.

**ComponentEvent**has this constructor:

ComponentEvent(Component *src*, int*type*)

Here, *src*is a reference to the object that generated this event. The type of the event isspecified by *type.*

**ComponentEvent**is the superclass either directly or indirectly of **ontainerEvent**,

**FocusEvent**, **KeyEvent**, **MouseEvent**, and **WindowEvent**.

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

**The ContainerEvent Class**

A**ContainerEvent**is generated when a component is added to or removed from a container.

There are two types of container events. The **ContainerEvent**class defines **int**constants thatcan be used to identify them:

**COMPONENT\_ADDED** and **COMPONENT\_REMOVED**.

They indicate that a component has been added to or removed from the container.

**ContainerEvent**is a subclass of **ComponentEvent**and has this constructor:

ContainerEvent(Component *src*, int*type*, Component *comp*)

Here, *src*is a reference to the container that generated this event. The type of the event is specifiedby *type,* and the component that has been added to or removed from the container is *comp.*

You can obtain a reference to the container that generated this event by using the

**getContainer( )** method, shown here:

Container getContainer( )

The **getChild( )** method returns a reference to the component that was added to or removedfrom the container. Its general form is shown here:

Component getChild( )

**The FocusEvent Class**

A**FocusEvent**is generated when a component gains or loses input focus. These events areidentified by the integer constants **FOCUS\_GAINED** and **FOCUS\_LOST**.

**FocusEvent**is a subclass of **ComponentEvent**and has these constructors:

FocusEvent(Component *src*, int*type*)

FocusEvent(Component *src*, int*type*, boolean*temporaryFlag*)

FocusEvent(Component *src*, int*type*, boolean*temporaryFlag*, Component *other*)

**The InputEvent Class**

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

**The KeyEvent Class**

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

**KEY\_PRESSED**, **KEY\_RELEASED**, and**KEY\_TYPED**.

The first two events are generated when any key is pressed or released. Thelast event occurs only when a character is generated. Remember, not all keypresses resultin characters. For example, pressing SHIFT does not generate a character.

There are many other integer constants that are defined by **KeyEvent**. For example, **VK\_0**through **VK\_9** and **VK\_A** through **VK\_Z** define the ASCII equivalents of the numbers andletters.

**KeyEvent**is a subclass of **InputEvent**. The **KeyEvent**class defines several methods, but the most commonly used ones are**getKeyChar( )**, which returns the character that was entered, and **getKeyCode( )**, whichreturns the key code. Their general forms are shown here:

char getKeyChar( )

intgetKeyCode( )

If no valid character is aavailable, then **getKeyChar( )** returns **CHAR\_UNDEFINED**. Whena **KEY\_TYPED** event occurs, **getKeyCode( )** returns **VK\_UNDEFINED**.

**The MouseEvent Class**

There are eight types of mouse events. The **MouseEvent**class defines the following integerconstants 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**is a subclass of **InputEvent**.

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

intgetX( )

intgetY( )

Alternatively, you can use the **getPoint( )** method to obtain the coordinates of the mouse.It is shown here:

Point getPoint( )

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

The **getClickCount( )** method obtains the number of mouse clicks for this event.

Its signature is shown here:

IntgetClickCount( )

**The MouseWheelEvent Class**

The **MouseWheelEvent**class encapsulates a mouse wheel event. It is a subclass of **MouseEvent**.Not all mice have wheels. If a mouse has a wheel, it is located between the left and rightbuttons. Mouse wheels are used for scrolling.

**MouseWheelEvent**defines these two integerconstants:

WHEEL\_BLOCK\_SCROLL A page-up or page-down scroll event occurred.

WHEEL\_UNIT\_SCROLL A line-up or line-down scroll event occurred.

**MouseWheelEvent**defines methods that give you access to the wheel event. To obtainthe number of rotational units, call **getWheelRotation( )**, shown here:

IntgetWheelRotation( )

It returns the number of rotational units. If the value is positive, the wheel moved

Counterclockwise. If the value is negative, the wheel moved clockwise.

**The TextEvent Class**

Instances of this class describe text events. These are generated by text fields and text areaswhen characters are entered by a user or program.

**TextEvent**defines the integer constant

**TEXT\_VALUE\_CHANGED**.

**The WindowEvent Class**

There are ten types of window events. The**WindowEvent**class defines integer constants thatcan 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**is a subclass of **ComponentEvent**.

Acommonly used method in this class is **getWindow( )**. It returns the **Window** objectthat generated the event. Its general form is shown here:

Window getWindow( ).

**Event Listener Interfaces**

The delegation event model has two parts: sources and listeners. Listeners arecreated 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 thelistener and provides an event object as its argument. The following table lists commonly used listenerinterfaces and provides a brief description of the methods that they define.



**The ActionListener Interface**

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

void actionPerformed(ActionEvent*ae*)

**The AdjustmentListener Interface**

This interface defines the **adjustmentValueChanged( )** method that is invoked when anadjustment event occurs. Its general form is shown here:

void adjustmentValueChanged(AdjustmentEvent*ae*)

**The ComponentListener Interface**

This interface defines four methods that are invoked when a component is resized, moved,shown, or hidden. Their general forms are shown here:

void componentResized(ComponentEvent*ce*)

void componentMoved(ComponentEvent*ce*)

void componentShown(ComponentEvent*ce*)

void componentHidden(ComponentEvent*ce*)

**The ContainerListener Interface**

This interface contains two methods. When a component is added to a container,

**componentAdded( )** is invoked. When a component is removed from a container,

**componentRemoved( )** is invoked. Their general forms are shown here:

void componentAdded(ContainerEvent*ce*)

void componentRemoved(ContainerEvent*ce*)

**The FocusListener Interface**

This interface defines two methods. When a component obtains keyboard focus, **focusGained( )**is invoked. When a component loses keyboard focus, **focusLost( )** is called. Their generalforms are shown here:

void focusGained(FocusEvent*fe*)

void focusLost(FocusEvent*fe*)

**The ItemListener Interface**

This interface defines the **itemStateChanged( )** method that is invoked when the state of anitem changes. Its general form is shown here:

void itemStateChanged(ItemEvent*ie*)

**The KeyListener Interface**

This interface defines three methods. The **keyPressed( )** and **keyReleased( )** methods areinvoked when a key is pressed and released, respectively. The **keyTyped( )** method is invokedwhen 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 general forms of these methods are shown here:

void keyPressed(KeyEvent*ke*)

void keyReleased(KeyEvent*ke*)

void keyTyped(KeyEvent*ke*)

**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:

void mouseClicked(MouseEvent*me*)

void mouseEntered(MouseEvent*me*)

void mouseExited(MouseEvent*me*)

void mousePressed(MouseEvent*me*)

void mouseReleased(MouseEvent*me*)

**The MouseMotionListener Interface**

This interface defines two methods. The **mouseDragged( )** method is called multiple timesas the mouse is dragged. The **mouseMoved( )** method is called multiple times as the mouseis moved. Their general forms are shown here:

void mouseDragged(MouseEvent*me*)

void mouseMoved(MouseEvent*me*)

**The MouseWheelListener Interface**

This interface defines the **mouseWheelMoved( )** method that is invoked when the mousewheel is moved. Its general form is shown here:

void mouseWheelMoved(MouseWheelEvent*mwe*)

**The TextListener Interface**

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

void textChanged(TextEvent*te*)

**The WindowFocusListener Interface**

This interface defines two methods: **windowGainedFocus( )** and **windowLostFocus( )**. Theseare called when a window gains or loses input focus. Their general forms are shown here:

void windowGainedFocus(WindowEvent*we*)

void windowLostFocus(WindowEvent*we*)

**The WindowListener Interface**

This interface defines seven methods. The **windowActivated( )** and **windowDeactivated( )**methods are invoked when a window is activated or deactivated, respectively. If a windowis 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. The general forms of these methods are

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*)

**Handling Mouse Events**

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.

To handle mouse events, you must implement the **MouseListener** and the **MouseMotionListener** interfaces. The following applet demonstrates the process. It displays the current coordinates of the mouse in the applet’s status window. Each time a button is pressed, the word “Down” is displayed at the location of the mouse pointer. Each time the button is released, the word “Up” is shown. If a button is clicked, the message “Mouse clicked” is displayed in the upper left corner of the applet display area.

As the mouse enters or exits the applet window, a message is displayed in the upper-left corner of the applet display area. When dragging the mouse, a \* is shown, which tracks with the mouse pointer as it is dragged. Notice that the two variables, **mouseX** and **mouseY**, store the location of the mouse when a mouse pressed, released, or dragged event occurs. These coordinates are then used by **paint( )** to display output at the point of these occurrences.

// Demonstrate the mouse event handlers.

import java.awt.\*;

import java.awt.event.\*;

import java.applet.\*;

/\*<applet code="MouseEvents" width=300 height=100>

</applet>

\*/

public class MouseEvents extends Applet

implements MouseListener, MouseMotionListener

{

String msg = "";

intmouseX = 0, mouseY = 0; // coordinates of mouse

public void init() {

addMouseListener(this);

addMouseMotionListener(this);

}

// Handle mouse clicked.

public void mouseClicked(MouseEvent me)

{

// save coordinates

mouseX = 0;

mouseY = 10;

msg = "Mouse clicked.";

repaint();

}

// Handle mouse entered.

public void mouseEntered(MouseEvent me)

{

// save coordinates

mouseX = 0;

mouseY = 10;

msg = "Mouse entered.";

repaint();

}

// Handle mouse exited.

public void mouseExited(MouseEvent me)

{

// save coordinates

mouseX = 0;

mouseY = 10;

msg = "Mouse exited.";

repaint();

}

// Handle button pressed.

public void mousePressed(MouseEvent me)

{

// save coordinates

mouseX = me.getX();

mouseY = me.getY();

msg = "Down";

repaint();

}

// Handle button released.

public void mouseReleased(MouseEvent me)

{

// save coordinates

mouseX = me.getX();

mouseY = me.getY();

msg = "Up";

repaint();

}

// Handle mouse dragged.

public void mouseDragged(MouseEvent me)

{

// save coordinates

mouseX = me.getX();

mouseY = me.getY();

msg = "\*";

showStatus("Dragging mouse at " + mouseX + ", " + mouseY);

repaint();

}

// Handle mouse moved.

public void mouseMoved(MouseEvent me)

// show status

showStatus("Moving mouse at " + me.getX() + ", " + me.getY());

}

// Display msg in applet window at current X,Y location.

public void paint(Graphics g)

{

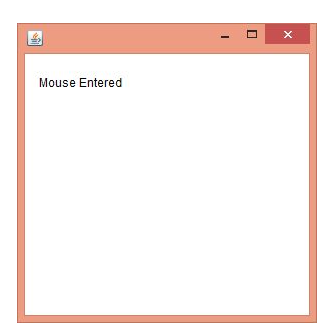
g.drawString(msg, mouseX, mouseY);

}

}

**Output :**

****



The **MouseEvents** class extends **Applet** and implements both the **MouseListener** and **MouseMotionListener** interfaces. These two interfaces contain methods that receive and process the various types of mouse events. Notice that the applet is both the source and the listener for these events. This works because **Component**, which supplies the **addMouseListener()** and **addMouseMotionListener( )** methods, is a super class of **Applet**. Being both the source and the listener for events is a common situation for applets. Inside **init( )**, the applet registers itself as a listener for mouse events. This is done by using

**addMouseListener( )** and **addMouseMotionListener( )**, which, as mentioned, are members of **Component**.

They are shown here:

void addMouseListener(MouseListener *ml*)

voidaddMouseMotionListener(MouseMotionListener *mml*)

**Handling Keyboard Events**

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

**KEY\_PRESSED**, **KEY\_RELEASED**, and **KEY\_TYPED**.

To handle keyboard events, you use the same general architecture as that shown in the mouse event example in the preceding section. The difference, of course, is that you will be implementing the **KeyListener** interface.

Before looking at an example, it is useful to review how key events are generated. When a key is pressed, a **KEY\_PRESSED** event is generated. This results in a call to the **keyPressed( )**event handler. When the key is released, a **EY\_RELEASED** event is generated and the **keyReleased( )** handler is executed. If a character is generated by the keystroke, then a **KEY\_TYPED** event is sent and the **keyTyped( )** handler is invoked. Thus, each time the user presses a key, at least two and often three events are generated. If all you care about are actual characters, then you can ignore the information passed by the key press and release events.

However, if your program needs to handle special keys, such as the arrow or function keys, then it must watch for them through the **keyPressed( )** handler.

The following program demonstrates keyboard input. It echoes keystrokes to the applet window and shows the pressed/released status of each key in the status window.

// Demonstrate the key event handlers.

import java.awt.\*;

import java.awt.event.\*;

import java.applet.\*;

/\*

<applet code="SimpleKey" width=300 height=100>

</applet>

\*/

public class SimpleKey extends Applet

implements KeyListener {

String msg = "";

int X = 10, Y = 20; // output coordinates

public void init() {

addKeyListener(this);

}

public void keyPressed(KeyEventke)

{

showStatus("Key Down");

}

public void keyReleased(KeyEventke)

{

showStatus("Key Up");

}

public void keyTyped(KeyEventke)

{

msg += ke.getKeyChar();

repaint();

}

// Display keystrokes.

public void paint(Graphics g)

{

g.drawString(msg, X, Y);

}

}

**Output:**



**Adapter Classes**

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



You can define a new class to act as an event listener by extending one of the adapter classes and implementing only those events in which you are interested.

For example, the **MouseMotionAdapter** class has two methods, **mouseDragged( )**and **mouseMoved( )**, which are the methods defined by the **MouseMotionListener** interface.

If you were interested in only mouse drag events, then you could simply extend **MouseMotionAdapter** and override **mouseDragged( )**. The empty implementation of **mouseMoved( )** would handle the mouse motion events for you.

The above table lists the commonly used adapter classes in **java.awt.event** and notes the interface that each implements.

The following example demonstrates an adapter. It displays a message in the status bar of an applet viewer or browser when the mouse is clicked or dragged. However, all other mouse events are silently ignored. The program has three classes. **AdapterDemo** extends **Applet**. Its **init( )** method creates an instance of **MyMouseAdapter** and registers that object to receive notifications of mouse events. It also creates an instance of **MyMouseMotionAdapter** and registers that object to receive notifications of mouse motion events. Both of the constructors take a reference to the applet as an argument.

**MyMouseAdapter** extends **MouseAdapter** and overrides the **mouseClicked( )** method. The other mouse events are silently ignored by code inherited from the **MouseAdapter** class. **MyMouseMotionAdapter** extends **MouseMotionAdapter** and overrides the **mouseDragged( )** method. The other mouse motion event is silently ignored by code inherited from the **MouseMotionAdapter** class.

Note that both of the event listener classes save a reference to the applet. This information is provided as an argument to their constructors and is used later to invoke the **showStatus( )**

method.

// Demonstrate an adapter.

import java.awt.\*;

import java.awt.event.\*;

import java.applet.\*;

/\*<applet code="AdapterDemo" width=300 height=100>

</applet>

\*/

public class AdapterDemo extends Applet

{

public void init() {

addMouseListener(new MyMouseAdapter(this));

addMouseMotionListener(new MyMouseMotionAdapter(this));

}

}

class MyMouseAdapter extends MouseAdapter

{

AdapterDemoadapterDemo;

public MyMouseAdapter(AdapterDemoadapterDemo)

{

this.adapterDemo = adapterDemo;

}

// Handle mouse clicked.

public void mouseClicked(MouseEvent me)

{

adapterDemo.showStatus("Mouse clicked");

}

}

class MyMouseMotionAdapter extends MouseMotionAdapter

{

AdapterDemoadapterDemo;

public MyMouseMotionAdapter(AdapterDemoadapterDemo)

{

this.adapterDemo = adapterDemo;

}

// Handle mouse dragged.

public void mouseDragged(MouseEvent me)

{

adapterDemo.showStatus("Mouse dragged");

}

}

As you can see by looking at the program, not having to implement all of the methodsdefined by the **MouseMotionListener**and **MouseListener**interfaces saves you a considerableamount of effort and prevents your code from becoming cluttered with empty methods. Asan exercise, you might want to try rewriting one of the keyboard input examples shown

earlier so that it uses a **KeyAdapter**.

**Output :**

****

