**PRACTICAL # 08**

**OBJECT:**

Handling Events.

**THEORY:**

Procedures or functions help divide a program into logical and more manageable parts. They are the core for procedural programming paradigm.

The most commonly handled events are generated by the mouse, the keyboard, and various controls, such as a push button. Events are supported by the java.awt.event package.

**The Delegation Event Model**

This modern approach to handling events defines standard and consistent mechanisms to generate and process events. A source generates an event and sends it to one or more listeners. The listener simply waits until it receives an event. The listener processes the event and returns. The advantage of this design is the application logic processing events is cleanly separated from the user interface logic generating those events.

A UI element is able to “delegate” the processing of an event to a separate piece of code. The listeners register with a source to receive an event notification. The notifications are sent only to listeners that want to receive them.

Previously, an event was propagated up the containment hierarchy until it was handled by a component. The delegation event model eliminates this overhead.

**Events:**

An event is an object that describes a state change in a source. Its result of user interacting with the elements in GUI. Example: pressing a button, selecting a list item, and clicking mouse.

There are also non-user generated events.

For example, an event may be generated when a timer expires, a counter exceeds a

value, a software or hardware failure occurs, or an operation is completed. Events that are appropriate for your application can be defined.

**Event Sources**

A source is an object that generates an event. This occurs when the state of that object changes. Sources may generate more than one type of event. A source must register listeners to receive notifications about a specific type of event. Each type of event has its own registration method.

Here is the general form:

public void addTypeListener(TypeListener el)

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 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. The general form of such a method is:

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

When such an event occurs, the registered listener is notified. This is known as unicasting the event.

A source must also provide a method that allows a listener to unregister a specific type of event. The general form of such a method is:

public void removeTypeListener(TypeListener el)

For example, to remove a keyboard listener, call removeKeyListener( ). The methods that add or remove listeners are provided by the source that generates events. For example, the Component class provides methods to add and remove keyboard and mouse event listeners.

**Event Listeners**

A listener is an object that is notified when an event occurs.

It must be registered with one or more sources to receive notifications about specific events and 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. For example, the MouseMotionListener interface defines two methods to receive notifications when the mouse is dragged or moved.

**Event Classes**

Event classes provide a consistent means of encapsulating events. At the root of the Java event class hierarchy is EventObject, which is in java.util. It is the superclass for all events. Its one constructor is:

EventObject(Object src)

Here, src is the object that generates this event.

EventObject contains two methods: getSource( ) and toString( ). The getSource( ) method returns the source of the event. Its general form is:

Object getSource( )

toString( ) returns the string equivalent of the event.

The class AWTEvent, defined in java.awt package, is a subclass of EventObject. It is the superclass of all AWT-based events handled by the delegation event model. Its getID( ) method can be used to determine the type of the event. The signature of this method is:

int getID( )

The package java.awt.event defines several types of events generated UI elements.

Most important of these event classes

**java.awt.event Event Class Description**

**ActionEvent** Generated when a button is pressed, 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 super class for all component input event classes.

**ItemEvent** Generated when a check box or list item is clicked, 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.

**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.

**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 used to identify any modifiers associated with an action event: ALT\_MASK, CTRL\_MASK, META\_MASK, and SHIFT\_MASK. An integer constant, ACTION\_PERFORMED can be used to identify action events.

ActionEvent constructors:

ActionEvent(Object src, int type, String cmd)

ActionEvent(Object src, int type, String cmd, int modifiers)

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.

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 command name equal to the label on that button.

The getModifiers( ) method returns a value that indicates which modifier keys

( ALT , CTRL , META , and/or SHIFT ) were pressed when the event was generated. Its form

is shown here:

int getModifiers( )

getWhen( ) returns the time at which the event took place. This is called the event’s timestamp.

long getWhen( )

**AdjustmentEvent Class**

An AdjustmentEvent is generated by a scroll bar. There are five types of adjustment

events. The AdjustmentEvent class defines integer constants that can be used to identify them. The constants and their meanings:

**BLOCK\_DECREMENT:** user clicked inside the scroll bar to decrease its value.

**BLOCK\_INCREMENT:** user clicked inside the scroll bar to increase its value.

**TRACK:** The slider was dragged.

**UNIT\_DECREMENT:** The button at the end of the scroll bar was clicked to decrease its value.

**UNIT\_INCREMENT:** The button at the end of the scroll bar was clicked to increase its value.

**ADJUSTMENT\_VALUE\_CHANGED** indicates that a change has occurred.

AdjustmentEvent constructor:

AdjustmentEvent(Adjustable src, int id, int type, int data)

Here, src is a reference to the object that generated this event. The id equals ADJUSTMENT\_VALUE\_CHANGED. The type of the event is specified by type, and its associated data is data.

The getAdjustable( ) method returns the object that generated the event.

Adjustable getAdjustable( )

The type of the adjustment event may be obtained by the getAdjustmentType( ) method.

It returns one of the constants defined by AdjustmentEvent. The general form is shown here:

int getAdjustmentType( )

The amount of the adjustment can be obtained from the getValue( ) method:

int getValue( )

For example, when a scroll bar is manipulated, this method returns the value represented

by that change.

**ComponentEvent Class**

ComponentEvent is generated when size, position, or visibility of a component is changed. There are four types of component events. The ComponentEvent class defines integer constants used to identify them. The constants and their meanings:

**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 constructor:

ComponentEvent(Component src, int type)

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

ComponentEvent is the superclass either directly or indirectly of ContainerEvent, FocusEvent, KeyEvent, MouseEvent, and WindowEvent.

The getComponent( ) method returns the component that generated the event.

Component getComponent( )

**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 that can be used to identify them: COMPONENT\_ADDED and COMPONENT\_REMOVED.

ContainerEvent is a subclass of ComponentEvent with 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 specified by type, and the component that has been added to or removed from the container is comp.

A reference to the container that generated this event by using the getContainer( ) method:

Container getContainer( )

The getChild( ) method returns a reference to the component that was added to or removed from the container:

Component getChild( )

**FocusEvent Class**

A FocusEvent is generated when a component gains or loses input focus. These events are identified 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)

Focus Event(Component src, int type, boolean temporaryFlag, Component other)

Here, src is a reference to the component that generated this event. The type of the event is specified by type. The argument temporaryFlag is set to true if the focus event is temporary. Otherwise, it is false. (A temporary focus event occurs as a result of another user interface operation. For example, assume that the focus is in a text field. If user moves the mouse to adjust a scroll bar, the focus is temporarily lost.)

The opposite component, is passed in other. Therefore, if a FOCUS\_GAINED event occurred, other will refer to the component that lost focus.

To determine the other component by calling getOppositeComponent( ):

Component getOppositeComponent( )

The opposite component is returned.

The isTemporary( ) method indicates if this focus change is temporary. Its form is:

boolean isTemporary( )

The method returns true if the change is temporary. Otherwise, it returns false.

**InputEvent Class**

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

InputEvent defines several integer constants that represent any modifiers, such as the control key being pressed, that might be associated with the event. The InputEvent class defined the following eight values to represent the modifiers.

ALT\_MASK BUTTON2\_MASK META\_MASK

ALT\_GRAPH\_MASK BUTTON3\_MASK SHIFT\_MASK

BUTTON1\_MASK CTRL\_MASK

However, because of possible conflicts between the modifiers used by keyboard events and mouse events, and other issues, Java 2, version 1.4 added the following extended modifier values.

ALT\_DOWN\_MASK ALT\_GRAPH\_DOWN\_MASK BUTTON1\_DOWN\_MASK

BUTTON2\_DOWN\_MASK BUTTON3\_DOWN\_MASK CTRL\_DOWN\_MASK

META\_DOWN\_MASK SHIFT\_DOWN\_MASK

it is recommended to use the new, extended modifiers rather than the original modifiers. To test if a modifier was pressed at the time an event is generated, use isAltDown( ), isAltGraphDown( ), isControlDown( ), isMetaDown( ), and isShiftDown( ) methods.

The forms of these methods are shown here:

boolean isAltDown( )

boolean isAltGraphDown( )

boolean isControlDown( )

boolean isMetaDown( )

boolean isShiftDown( )

Obtain a value containing all of the original modifier flags by calling the getModifiers( ) method:

int getModifiers( )

Obtain the extended modifiers by called getModifiersEx( ):

int getModifiersEx( )

**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.

**Program:**

The program below defines a main function. It further goes to define two functions; the newlineProc that

**ACTIVITIES**

**Activity 1**

Try implementing the above program but without using the **ret** statement at the end of functions and observe the output.

**Activity 2**

Write a function that prints alphabets in reverse order, without using array.

**Activity 3**

Write a function to print a multi-digit number.

**REVIEW QUESTIONS**

1. What happens if RET statement is missed at the end of a function?
2. What happens in the system on some function call instruction like CALL someFunction?
3. How do functions help make a program manageable?