International Journal of Scientific Research & Engineering Trends
Volume 6, Issue 4, July-Aug-2020, ISSN (Online): 2395-566X

Event Handling in Java

M. Amitha Reddy
Hyderabad, India
amithareddy0909@gmail.com

***Abstract*-This paper explores the definition and importance of event handling in Java with focus on different event handling classes. We will aslo see that we cannot write a GUI program without event handlers. Most events to which the program will respond are generated when the user interacts with a GUI-based program. These are the types of events examined in this paper. They are passed into the program in a variety of ways, with the specific method dependent upon the actual event. There are several types of events, including those generated by the mouse, the keyboard and various GUI controls, such as a push button, scroll bar or check box. This paper begins with an overview of Java's event handling mechanism. It then examines the main event classes and interfaces used by the AWT and develops several examples that demonstrate the fundamentals of event processing.**

***Keywords*-Graphical User Interface Event, Source, Listener, Component, Container, Object .**

I.INTRODUCTION

Event handling is fundamental to Java Programming because it is integral to the creation of many kinds of applications, including applets and other types of GUI-based programs. The deligation event model defines standard and consistent mechanisms to generate and process events. Its concept is quite simple: a source generates an event and sends it to one or more listeners. In this scheme, the listener simply waits until it revceives an event. Once an event is received, the listener processes the event and then returns. In the deligation event model, listeners must register with a source in order to receive an event notification. This provides an important benefit: notifications are sent only to listeners that want to receive them. This is a more efficient way to handle events than the design used by the original java 1.0 approach. Previously, an event was propagated up the containment hierarchy until it was handled by a component. This required components to receive events that they did not process and it wasted time. The deligation event model eliminates this overhead. We cannot write a GUI program without event handlers.

II. EVENTS

In the deligation event model, an event is an object that describes a state change in a source. Among another causes, an event can be gnerated as a consequence of a person interacting with the elements in a graphical user interface. Some of the activities that cause events to be generated are pressing a button, entering a character via the keyboard, selecting an item in a list and clicking the mouse. Many other user operations could also be sited as examples. 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, a software or hardware failure occurs, or an operation is completed.

III. EVENT SOURCES

A source is an object that generates an event. This occurs when the internal state of that object changes in some way. Sources may generate more than one type of event. A source must register listeners in order for the listeners to receive notifications about a spectific type of event. Each type of event has it own registration method. Here is the general form:

public void addTypeListener (TypeListener el)

Here, Type is the name of the event, and el is a reference to event listener. For example, the method that registers a keyboard event listener is called add Key Listener(). The method that registers a mouse motion listener is called add Mouse MotionListener(). When an event occurs, all the registered listeners are notified and receive a copy of the event object. This is known as multicasting the event. In all cases, notifications are sent only to listeners that register to receive them. Some sources me allow only one listener to register. The general form of such a method is this:

public void addTypeListener (TypeListener el)

throws java. util. TooManyListenersException

Here, Type is the name of the event, and el is a reference to event listener. When such an event occurs, the registered listener is notified. This is known as uncasting the event. A source must also provide a method that allows a listener to unregister an interest in a specific type of event. The general form of such a method is this:

public void removeTypeListener (TypeListener el)

Here, Type is the name of the event, and el is a reference to 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 generates events. For example, the Component class provides methods to add and remove keyboard and mouse event listeners.

III. EVENT LISTENERS

A listener is an object that is notified when an event occurs. It has two major requirements. First, it must have been registered with one or more sources to receive notifications about specific type of events. Second, it must implement methods to receive and process these notifications.

The methods that receive and process events are defined in a set of interfaces, such as those found in java.awt.event. for example, the MouseMotionListener interface defines two methods to receive notification when the mouse is dragged or moved. Any object may receive and process one or both of these events if it provides an implementation of this interface.

IV. EVENT CLASSES

The classes that represent events are at the core of Java's event handling mechanism. Thus, a discussion of event handling must begin with the event classes. This report focuses on AWT 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 shown here:

EventObject(Object src)

Here, src is the object that generates this event. EventObject defines two methods getSource() and ToString(). The method returns the source of the evnt. Its general form is shown here:

Object getSource()

As expected, EventObject returns the string equivalent of the event.

The class AWTEvent, defined within java.awt package, is a subclass of EventObject. It is the superclass of all AWT-based events used by the deligation 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 many types of events that are generated by various user interface elements. The following table shows several commonly used event classes and provides a brief description of when they are generated.

Table1: Event classes and their description.

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

1. The ActionEvent Class

An ActionEvent is generated when a button is pressed, a list item is double-clicked, or a menu item is selected. The ActionEvent class defines four integer constants that can be used to identify any modifiers associated with an action event: ALT_MASK, CTRL_MASK, META_MASK, and SHIFT_MASK. In addition, there is an integer constant, ACTION_PERFORMED, which can be used to identify action events.

1.1 ActionEvent has these three 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 indicate which modifier keys(ALT, CTRL, META, and/or SHIFT) were pressed when the event was generated. The when parameter specifies when the event occurred.

We can obtain the command name for the invoking ActionEvent object by using the getActionCommand() method, shown below:

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() returns a value that indicates which modifier keys where pressed when the event was generated. Its form is:

int getModifiers()

The method getWhen() returns the time at which the event took place. This is called the event's timestamp. The getWhen() method is shown below:

Long getWhen()

2. The 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 are shown below:

Table-2: Integer constantsand their meanings.

Block_Decrement	The user clicked inside the scroll bar to decrease its value.
Block_Increment	The 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.

In addition, there is an integer constant, ADJUSTMENT_VALUE_CHANGED, that indicates that a change has occurred. Here is one AdjustmentEvent constructor:AdjustmentEvent(Adjustable src, int id, int type, int val)

Here, src is a reference to the object that generated this event. the id specifies the event. the type of adjustment is specified by type, and its associated value is val.

The getAdjustable() method returns the object that generated the event. its form is shown below:

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 adjustment can be obtained from the getValue() method, shown below:

Int getValue()

For example, when a scroll bar is manipulated, this method returns the value represented by that change.

3. The ComponentEvent Class

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

Table-3: Integer constantsand 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.

3.1. ComponentEvent has its constructor:

Component Event(Component src, int type)

Here, src is a reference to the object that generated this event. the id specifies the 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, among others.

The getComponent() method returns the component that generated the event. it is shown below:

Component get Component()

4. 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 that can be used to identify them: COMPONENT_ADDED and COMPONENT_REMOVED. They indicate that a

component has been added to or removed from a 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 object 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.

You can obtain a reference to the container that generated this event by using the getContainer() method, shown below:

Container getContainer()

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

Component getChild()

5. The 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, booleantemporaryFlag)
FocusEvent(Component src, int type, booleantemporaryFlag, Component other)

Here, src is a reference to the object 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 set to 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 the user moves the mouse to adjust a scroll bar, the focus is temporarily lost.)

The other component involved in the focus change, called the opposite component, is passed in other. Therefore, if a FOCUS_GAINED event occurred, other will refer to component that lost focus. Conversely, if a FOCUS_LOST event occurred, other will refer to the component that gains focus.

We can determine the other component by calling getOppositeComponent(), shown here:

Component getOppositeComponent()

The opposite component is returned.

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

Boolean isTemporary()

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

6. The 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. originally, the InputEvent class defined the following eight values to represent the modifiers:

Table-4: Values to represent the modifiers.

Alt_Mask	Button2_Mask	Meta_Mask
Alt_Graph_Mask	Button3_Mask	Shift_Mask
Button1_Mask	Crtl_Mask	

However, because of possible conflicts between the modifiers used by the keyboard events and mouse events, and other issues, the following extended modifier values were added:

Table-5: Extended modifier values.

Alt_Down_Mask	Button2_Down_Mask	Meta_Down_Mask
Alt_Graph_Down_Mask	Button3_Down_Mask	Shift_Down_Mask
Button1_Down_Mask	Crtl_Down_Mask	

7. The 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 events, which are identified by the following integer constants:

Table-6: extended modifier values.

Deselected	The User Deselected An Item.
Selected	The User Selected An Item.

In addition, ItemEvent defines one integer constant, ITEM_STATE_CHANGED, that signifies a change of state. ItemEvent has this constructor:

ItemEvent(ItemSelectablesrc, int type, Object entry, int state)

Here, src is a reference to the object that generated this event. For example, this might be a list or choice element. The type of the event is specified by type. The specific item that generated the item event is passed in entry. The current state of that item is in state.

The getItem() method can be used to obtain a reference to the item that changed. Its signature is shown here:

Object getItem()

The getItemSelectable() method can be used to obtain a reference to the ItemSelectable object that generated an event. its general form is shown here:

ItemSelectablegetItemSelectable()

Lists and choices are examples of user interface elements that implement the ItemSelectable interface. The getStateChange() method returns the state change (that is, SELECTED or DESELECTED) for the event. Its shown here:

int getStateChange()

8. 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. The last event occurs only when a character is generated. There are many other integer constants that are defined by KeyEvent. For example, VK_0 through VK_9 and VK_A through VK_Z define the ASCII equivalents of the numbers and letters. Here are some others:

Table-7: KeyEventinteger constants.

Vk_Alt Vk_Down	Vk_Left	Vk_Right	Vk_Cancel
Vk_Enter	Vk_Page_Down	Vk_Shift	Vk_Control
Vk_Escape	Vk_Page_Up	Vk_Up	

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

KeyEvent is subclass of InputEvent. Here is one of its costructors:

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

Here, src is a reference to the object that generated this event. The type of the event is specified by type. The system time at which the key was pressed is passed in when. The modifiers argument indicates which modifiers where pressed when this key event occurred. The virtual key code, such as VK_UP, VK_A, and so forth, is passed in code. The character equivalent (if one exists) is passed in ch. If no valid character exists, then ch contains CHAR_UNDEFINED. For KEY_TYPED events, code will contain VK_UNDEFINED.

The KeyEvent class defines several methods, but probably the most commonly used ones are getKeyChar(), wich returns the character that was entered, and getKeyCode(), which returns the key code. The general forms are shown below:

Char getKeyChar()

int getKeyCode()

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

9. The MouseEvent Class

There are eight types of mouse events. The MouseEvent class defines the following integer constants that can be used to identify them.

Table-8: MouseEvent integer constants.

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. Here is one of the constructors:

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

Here, src is a reference to the object that generated this event. The type of the event is specified by type. The system time at which the mouse event occurred is passed in when. The modifiers argument indicates which modifiers where passed when mouse event occurred. The coordinates of the mouse are passed in x and y. The click count is passed in clicks. ThetriggersPopup flag indicates if this event causes a pop-up menu to appear on this platform.

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

int getX()
int geY()

The translatePoint() method changes the location of the event. its form is shown here:

void translatePoint(int x, int y)

Here, the arguments x and y are added to the coordinates of the event.

The getClickCount() method obtains the number of mouse clicks for this event. Its signature is shown here:

int getClickCount()

The isPopupTrigger() method tests if this event causes a pop-up menu to appear on this platform. Itsnform is shown here:

Boolean isPopupTrigger()

10. 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 typically located between the left and right buttons. Mouse wheels are used for scrolling. MouseWheelEvent defines these two integer constants.

Table-9: MouseWheelEventinteger constants.

Wheel_Block_Scroll	A page-up or page-down scroll event occurred.
Wheel_Unit_Scroll	A line-up or line-down scroll event occurred.

Here is one of the constructors defined by

11. MouseWheelEvent:

MouseWheelEvent(Component src, int type, long when, int modifiers, int x, int y, int clicks, booleantriggersPopup, int scrollHow, int amount, int count)

Here, src is a reference to the object that generated this event. The type of the event is specified by type. The system time at which the mouse event occurred is passed in when. The modifiers argument indicates which modifiers where passed when mouse event occurred. The coordinates of the mouse are passed in x and y. The click count is passed in clicks. ThetriggersPopup flag indicates if this event causes a pop-up menu to appear on this platform. The scrollhow value must be either WHEEL_UNIT_SCROLLor
WHEEL_BLOCK_SCROLL. The number of units to scroll is passed in amount. The count parameter indicates the number of rotational units that the wheel moved.

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

int getWheelRotation()

It returns the number of rotational units. If the value is positive, the wheel moved counterclockwise else, the wheel moved clockwise. JDK 7 added a method called getPreciseWheelRotation(), which supports high-resolution wheels. It works like getWheelRotation(), but returns a double.

12. The TextEvent Class

Instances of this class describe text events. These are generated by text fields and text areas when characters are entered by a user or program. TextEvent defines the integer constant TEXT_VALUE_CHANGED.

The one constructor for this class is shown here:

TextEvent(Object src, int type)

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

The TextEvent object does not include the characters currently in the text component that generated the event. Instead, your program must use other methods associated with the text components to retrieve that information.

13. The WindowEvent Class

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

Table-10: WindowEventinteger constants**.**

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. It defines several constructors. The first is

WindowEvent(Window src, int type)

Here, src is a reference to the object that generated this event. The type of the event is specified by type the next three constructors offer more detailed control:

WindowEvent(Window src, int type, Window other)

WindowEvent(Window src, int type, int fromState, int toState)

WindowEvent(Window src, int type, Window other, , int fromState, int toState)

Here, other specifies the opposite window when a focus or activation event occurs. The fromState specifies the prior state of the window, and toState specifies the new state that the window will have when a window state change occurs.

A commonly used method in this class is getWindow(). It returns the Window object that generated the event.

14. Event Listener Interfaces

The deligation event model has two parts: sources and listeners. Listeners are created by implementing one or more of the interfaces defined by the java.awt. event package. When an event occurs, the event source invokes the appropriate method defined by the listener and provides an event object as its argument.

Table-11: Event Listener Interfaces.

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

15. The ActionListener Interface

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

void actionPerformed(ActionEvent ae)

16. The AdjustmentListener Interface

This interface defines the adjustmentValueChanged() method that is invoked when an adjustment event occurs. Its general form is:

void adjustmentValueChanged(AdjustmentEvent ae)

17. The ComponentListener Interface

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

void componentResized(ComponentEventce)
void componentMoved(ComponentEventce)
void componentShown(ComponentEventce)
void componentHidden(ComponentEventce)

18. The ContainerListener Interface

This interface contains two methods. When a component is added to a container, componentAdded() is invoked. When a component isremoved from a container, componentRemoved() is invoked. Their general forms are:

void componentAdded(ContainerEventce)
void componentRemoved(ContainerEventce)

19. 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 general forms are:

void focusGained(FocusEventfe)
void focusLost(FocusEventfe)

20. The ItemListener Interface

This interface defines the itemStateChanged() method that is invoked when the state of an item changes. Its general form is:

void itemStateChanged(ItemEventie)

21. The KeyListener Interface

This interface defines three methods. The KeyPressed() and KeyReleased() methods are invoked when a key is pressed and released, respectively. The keyTyped() method is invoked when a character has been entered. The general forms of these methods are:

void KeyPressed(KeyEventke)
void KeyReleased(KeyEventke)
void KeyTyped(KeyEventke)

22. 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 form of these methoda are:

void mouseClicked(MouseEvent me)

void mouseEntered(MouseEvent me)
void mouseExited(MouseEvent me)
void mousePressed(MouseEvent me)
void mouseReleased(MouseEvent me)

23. The MouseMotionListener Interface

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

void mouseDragged(MouseEvent me)
void mouseMoved(MouseEvent me)

24. The MouseWheelListener Interface

This interface defines the mouseWheelMoved() method that is invoked when the mouse wheel is moved. Its general form is:

void mouseWheelMoved(MouseWheelEventmwe)

25. The TextListener Interface

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

void textValueChanged(TextEventte)

26. The WindowFocusListener Interface

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

void windowGainedFocus(WindowEvent we)
void windowLostFocus(WindowEvent we)

27. 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 window is iconified, the windowIconified() method is called. When a window is deiconified, the windowDeiconified() method is called. When a window is opened or closed, the windowOpened() or windowClosed() mwthoda 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)

V. FINDINGS

Event handling is fundamental to Java programming because it is integral to the creation of many kinds of applications, including applets and other types of GUI-based programs. Applets are event-driven programs that use a graphical user interface to interact with the user. Any program that uses a graphical user interface is event-driven. Thus, these types of program cannot be written without a solid command of event handling.

REFERENCES

[1] An introduction to programming and OO design using Java, J. Nino and F. A. Hosch, John Wiley & Sons.
[2] Core Java, Volume 1, 9th edition, Cay S. Horstmann and G. Cornell, Pearson.
[3] Event Handling, https://en.m.wikibooks.org
[4] Event Listner types, books.gigatux.nl
[5] Introduction to Event Listeners, https://docs.oracle.com
[6] Introduction to Java programming, Y. Daniel Liang, Pearson Education.
[7] Java Event Handling, Grant Palmer.
[8] Java Events, https://www.javatpoint.com
[9] Java for Programmers, P. J. Deitel, 10th Edition Pearson education.
[10] Java foundation classes, https://docstore.mik.ua
[11] Java Programming and Application Development, R.A. Johnson, Cengage Learning.
[12] Java Programming, D. S. Malik and P. S. Nair, Cengage Learning.
[13] Java, the complete reference, 9th edition, Herbert Schildt.
[14] Listners in Java, https://www.bookofnetwork.com
[15] Modelling of Java GUI Event Handling, https://researchgate.net
[16] Programming in Java, S. Malhotra, S. Chaudary, 2nd edition, Oxford Univ. Press.
[17] Research on Event Handling Models of Java, www.ccs.asia.edu.tw.com
[18] Thinking for Java, Bruce Eckel, Pearson education.
[19] Event handling in Java, https://www.edureka.com
[20] Java and Event handling, https://www.infoworld.com
[21] Core Java: An integrated Approach by R. Nageshwara Rao.
[22] Programming with Java, 6th edition, E. Balaguruswamy.
[23] Java: the complete reference, 11th edition, Herbert Schildt.
[24] Computer programming, Steve Tudor.