1. JButton :

Description: Represents a push-button component that triggers an action when clicked.

- **Constructors:**

a) JButton(): Creates a button with no text or icon.
b) JButton(String text): Creates a button with the specified text.
c) JButton(Icon icon): Creates a button with the specified icon.
d) JButton(String text, Icon icon): Creates a button with the specified text and icon.

- **Methods:**

a) void setText(String text): Sets the text of the button.
b) String getText(): Returns the text of the button.
c) void setIcon(Icon icon): Sets the icon of the button.
d) Icon getIcon(): Returns the icon of the button.
e) void addActionListener(ActionListener listener): Registers an action listener to be notified when the button is clicked.
f) void setEnabled(boolean enabled): Sets whether the button is enabled or disabled.
g) boolean isEnabled(): Returns true if the button is enabled; otherwise, false.

2. JLabel :

Description: Displays a short text string or an image.

- **Constructors**:

a) JLabel(): Creates a label with no text or image.
b) JLabel(String text): Creates a label with the specified text.
c) JLabel(Icon image): Creates a label with the specified image.
d) JLabel(String text, Icon image, int horizontalAlignment): Creates a label with the specified text, image, and alignment.

- **Methods:**

a) void setText(String text): Sets the text of the label.
b) String getText(): Returns the text of the label.
c) void setIcon(Icon icon): Sets the icon of the label.
d) Icon getIcon(): Returns the icon of the label.
e) void setHorizontalAlignment(int alignment): Sets the horizontal alignment of the label.
f) int getHorizontalAlignment(): Returns the horizontal alignment of the label.

3. JCheckBox :

Description: Represents a check box that can be selected or deselected.

- **Constructors:**

a) JCheckBox(): Creates an initially unselected check box with no text.
b) JCheckBox(String text): Creates an initially unselected check box with the specified text.
c) JCheckBox(String text, boolean selected): Creates a check box with the specified text and selection state.

- **Methods:**

a) void setSelected(boolean selected): Sets the selection state of the check box.
b) boolean isSelected(): Returns true if the check box is selected; otherwise, false.
c) void setText(String text): Sets the text of the check box.
d) String getText(): Returns the text of the check box.
e) void addItemListener(ItemListener listener): Registers an item listener to be notified when the check box's state changes.
f) void setEnabled(boolean enabled): Sets whether the check box is enabled or disabled.
g) boolean isEnabled(): Returns true if the check box is enabled; otherwise, false.

4. JComboBox :

Description: Represents a drop-down list of selectable items.

- **Constructors:**

a) JComboBox(): Creates a combo box with no items.
b) JComboBox(Object[] items): Creates a combo box with the specified array of items.
c) JComboBox(Vector<?> items): Creates a combo box with the specified vector of items.

- **Methods:**

a) void addItem(Object item): Adds an item to the combo box.
b) void removeItem(Object item): Removes an item from the combo box.
c) Object getSelectedItem(): Returns the currently selected item.
d) void setSelectedItem(Object item): Sets the selected item in the combo box.
e) void addActionListener(ActionListener listener): Registers an action listener to be notified when an item is selected.
f) void setEnabled(boolean enabled): Sets whether the combo box is enabled or disabled.
g) boolean isEnabled(): Returns true if the combo box is enabled; otherwise, false.

5. JList :

Description: Displays a list of items from which the user can select one or more.

- **Constructors:**

a) JList(): Creates a list with an empty model.
b) JList(ListModel<E> dataModel): Creates a list that displays the elements in the specified model.
c) JList(E[] listData): Creates a list that displays the elements in the specified array.

- **Methods:**

a) void setListData(E[] listData): Sets the data model to display the specified array of elements.
b) void setModel(ListModel<E> model): Sets the data model for the list.
c) ListModel<E> getModel(): Returns the data model for the list.
d) void setSelectedIndex(int index): Selects the specified index.
e) int getSelectedIndex(): Returns the index of the first selected item; returns -1 if no item is selected.
f) Object[] getSelectedValues(): Returns an array of all the selected values.
g) void addListSelectionListener(ListSelectionListener listener): Registers a listener to be notified when the selection changes.

6. JMenuBar :

Description: Represents a horizontal bar containing menus.

- **Constructors:**

a) JMenuBar(): Creates a new menu bar.

- **Methods:**

a) void add(JMenu menu): Adds the specified menu to the menu bar.
b) void remove(int index): Removes the menu at the specified index from the menu bar.
c) int getMenuCount(): Returns the number of menus in the menu bar.
d) JMenu getMenu(int index): Returns the menu at the specified index.
e) void setEnabled(boolean enabled): Sets whether the menu bar is enabled or disabled.
f) boolean isEnabled(): Returns true if the menu bar is enabled; otherwise, false.

7. JMenu :

Description: Represents a menu, which is a pull-down or pop-up list of items.

- **Constructors:**

a) JMenu(): Creates a new menu with no text.
b) JMenu(String text): Creates a new menu with the specified text.

- **Methods:**

a) void add(JMenuItem menuItem): Adds the specified menu item to the menu.
b) void addSeparator(): Adds a separator to the menu.
c) void setEnabled(boolean enabled): Sets whether the menu is enabled or disabled.
d) boolean isEnabled(): Returns true if the menu is enabled; otherwise, false.
e) void setText(String text): Sets the text of the menu.
f) String getText(): Returns the text of the menu.

8. JMenuItem :

Description: Represents an item in a menu, which can be selected by the user.

- **Constructors**:

a) JMenuItem(): Creates a new menu item with no text or icon.
b) JMenuItem(String text): Creates a new menu item with the specified text.
c) JMenuItem(Icon icon): Creates a new menu item with the specified icon.
d) JMenuItem(String text, Icon icon): Creates a new menu item with the specified text and icon.

- **Methods**:

a) void addActionListener(ActionListener listener): Registers an action listener to be notified when the menu item is selected.
b) void setEnabled(boolean enabled): Sets whether the menu item is enabled or disabled.
c) boolean isEnabled(): Returns true if the menu item is enabled; otherwise, false.
d) void setText(String text): Sets the text of the menu item.
e) String getText(): Returns the text of the menu item.
f) void setIcon(Icon icon): Sets the icon of the menu item.
g) Icon getIcon(): Returns the icon of the menu item.

9. JPasswordField :

Description: Provides a field for entering a password, where the characters are masked.

- **Constructors:**

a) JPasswordField(): Creates a new password field with no text.

- **Methods:**

a) void setText(String text): Sets the text of the password field.
b) String getText(): Returns the text of the password field.
c) void setEchoChar(char echoChar): Sets the character to be used for echoing.
d) char getEchoChar(): Returns the character used for echoing.
e) void setEnabled(boolean enabled): Sets whether the password field is enabled or disabled.
f) boolean isEnabled(): Returns true if the password field is enabled; otherwise, false.

10. JRadioButton :

Description: Represents a radio button that can be selected in a group of mutually exclusive options.

- **Constructors:**

a) JRadioButton(): Creates an initially unselected radio button with no text.
b) JRadioButton(String text): Creates an initially unselected radio button with the specified text.
c) JRadioButton(String text, boolean selected): Creates a radio button with the specified text and selection state.

- **Methods:**

a) void setSelected(boolean selected): Sets the selection state of the radio button.
b) boolean isSelected(): Returns true if the radio button is selected; otherwise, false.
c) void setText(String text): Sets the text of the radio button.
d) String getText(): Returns the text of the radio button.
e) void addItemListener(ItemListener listener): Registers an item listener to be notified when the radio button's state changes.
f) void setEnabled(boolean enabled): Sets whether the radio button is enabled or disabled.
g) boolean isEnabled(): Returns true if the radio button is enabled; otherwise, false.

11. ButtonGroup :

Description: Manages a group of buttons, where only one button at a time can be selected within the group.

- **Constructors**:

a) ButtonGroup(): Creates a new button group.

- **Methods**:

a) void add(AbstractButton b): Adds the specified button to the group.
b) void remove(AbstractButton b): Removes the specified button from the group.
c) void clearSelection(): Clears the selection from all buttons in the group.
d) ButtonModel getSelection(): Returns the model of the selected button in the group, or null if no button is selected.
e) Enumeration<AbstractButton> getElements(): Returns an enumeration of all the buttons in the group.
f) int getButtonCount(): Returns the number of buttons in the group.
g) boolean isSelected(ButtonModel m): Returns true if the specified button model is selected in the group; otherwise, false.
h) void setSelected(ButtonModel m, boolean b): Sets the selection state of the specified button model in the group.

12. JScrollBar :

Description: Represents a scroll bar component for scrolling through content.

- **Constructors:**

a) JScrollBar(): Creates a vertical scroll bar.
b) JScrollBar(int orientation): Creates a scroll bar with the specified orientation.

- **Methods:**

a) void setValue(int value): Sets the current value of the scroll bar.
b) int getValue(): Returns the current value of the scroll bar.
c) void setMinimum(int min): Sets the minimum value of the scroll bar.
d) int getMinimum(): Returns the minimum value of the scroll bar.
e) void setMaximum(int max): Sets the maximum value of the scroll bar.
f) int getMaximum(): Returns the maximum value of the scroll bar.
g) void setUnitIncrement(int unitIncrement): Sets the unit increment of the scroll bar.
h) int getUnitIncrement(): Returns the unit increment of the scroll bar.
i) void setBlockIncrement(int blockIncrement): Sets the block increment of the scroll bar.
j) int getBlockIncrement(): Returns the block increment of the scroll bar.

13. JScrollPane :

Description: Provides a scrollable view of a component.

- **Constructors:**

a) JScrollPane(): Creates a new scroll pane.
b) JScrollPane(Component view): Creates a new scroll pane with the specified view component.

- **Methods:**

a) void setViewportView(Component view): Sets the component to be displayed in the viewport.
b) Component getViewportView(): Returns the component displayed in the viewport.
c) void setHorizontalScrollBarPolicy(int policy): Sets the horizontal scroll bar policy.
d) int getHorizontalScrollBarPolicy(): Returns the horizontal scroll bar policy.
e) void setVerticalScrollBarPolicy(int policy): Sets the vertical scroll bar policy.
f) int getVerticalScrollBarPolicy(): Returns the vertical scroll bar policy.
g) void setBorder(Border border): Sets the border of the scroll pane.
h) Border getBorder(): Returns the border of the scroll pane.

14. JFrame :

Description: Represents a top-level window with a title and borders.

- **Constructors**:

a) JFrame(): Creates a new, initially invisible frame with no title.
b) JFrame(String title): Creates a new, initially invisible frame with the specified title.

- **Methods**:

a) void setDefaultCloseOperation(int operation): Sets the default close operation for the frame.
b) int getDefaultCloseOperation(): Returns the default close operation for the frame.
c) void setVisible(boolean visible): Sets the visibility of the frame.
d) boolean isVisible(): Returns true if the frame is visible; otherwise, false.
e) void setTitle(String title): Sets the title of the frame.
f) String getTitle(): Returns the title of the frame.
g) void setSize(int width, int height): Sets the size of the frame.
h) Dimension getSize(): Returns the size of the frame.
i) void setLocationRelativeTo(Component c): Sets the location of the frame relative to the specified component.

j) void add(Component comp): Adds the specified component to the frame.

15. JPanel :

Description: A lightweight container that provides a way to group and organize other components.

- **Constructors**:

a) JPanel(): Creates a new panel with a double buffer and a flow layout.
b) JPanel(LayoutManager layout): Creates a new panel with the specified layout manager.

- **Methods**:

a) void setLayout(LayoutManager layout): Sets the layout manager for the panel.
b) LayoutManager getLayout(): Returns the layout manager for the panel.
c) void add(Component comp): Adds the specified component to the panel.
d) Component[] getComponents(): Returns an array of all the components in the panel.
e) void remove(Component comp): Removes the specified component from the panel.

16. JFileChooser :

Description: Provides a dialog for selecting files from the file system.

- **Constructors**:

a) JFileChooser(): Creates a new file chooser pointing to the user's default directory.

- **Methods**:

a) int showOpenDialog(Component parent): Shows an "Open File" dialog.
b) int showSaveDialog(Component parent): Shows a "Save File" dialog.
c) File getSelectedFile(): Returns the selected file.
d) void setSelectedFile(File file): Sets the selected file.
e) void setFileSelectionMode(int mode): Sets the file selection mode (FILES_ONLY, DIRECTORIES_ONLY, or FILES_AND_DIRECTORIES).

17. JProgressBar :

Description: Displays the progress of a task over time.

- **Constructors**:

a) JProgressBar(): Creates a new progress bar with default minimum and maximum values.
b) JProgressBar(int orient): Creates a new progress bar with the specified orientation (HORIZONTAL or VERTICAL).

- **Methods**:

a) void setValue(int n): Sets the current value of the progress bar.
b) int getValue(): Returns the current value of the progress bar.
c) void setMinimum(int n): Sets the minimum value of the progress bar.
d) int getMinimum(): Returns the minimum value of the progress bar.
e) void setMaximum(int n): Sets the maximum value of the progress bar.
f) int getMaximum(): Returns the maximum value of the progress bar.
g) void setStringPainted(boolean b): Sets whether the progress bar should display a string representation of its current value.
h) boolean isStringPainted(): Returns true if the progress bar displays a string representation of its current value; otherwise, false.

18. JSlider :

Description: Represents a slider component for selecting a value from a range.

- **Constructors:**

a) JSlider(): Creates a horizontal slider with default minimum, maximum, and initial values.
b) JSlider(int orientation): Creates a slider with the specified orientation.
c) JSlider(int min, int max): Creates a horizontal slider with the specified minimum and maximum values.
d) JSlider(int orientation, int min, int max): Creates a slider with the specified orientation, minimum, and maximum values.
e) JSlider(BoundedRangeModel brm): Creates a slider using the specified bounded range model.

- **Methods:**

a) void setValue(int value): Sets the current value of the slider.
b) int getValue(): Returns the current value of the slider.

c) void setMinimum(int min): Sets the minimum value of the slider.
d) int getMinimum(): Returns the minimum value of the slider.
e) void setMaximum(int max): Sets the maximum value of the slider.
f) int getMaximum(): Returns the maximum value of the slider.
g) void setMajorTickSpacing(int spacing): Sets the spacing between major tick marks.
h) int getMajorTickSpacing(): Returns the spacing between major tick marks.
i) void setMinorTickSpacing(int spacing): Sets the spacing between minor tick marks.
j) int getMinorTickSpacing(): Returns the spacing between minor tick marks.

19. JSpinner :

Description: Provides a spin box component for selecting numeric or textual values from a sequence.

- **Constructors:**

a) JSpinner(): Creates a spinner with an integer model and initial value 0.
b) JSpinner(SpinnerModel model): Creates a spinner using the specified model.

- **Methods:**

a) void setValue(Object value): Sets the current value of the spinner.
b) Object getValue(): Returns the current value of the spinner.
c) void setModel(SpinnerModel model): Sets the model that the spinner uses to display and change its values.
d) SpinnerModel getModel(): Returns the model that the spinner uses to display and change its values.
e) void setEnabled(boolean enabled): Sets whether the spinner is enabled or disabled.
f) boolean isEnabled(): Returns true if the spinner is enabled; otherwise, false.

20. JTabbedPane :

Description: Represents a container with multiple tabs, each containing a different component.

- **Constructors:**

a) JTabbedPane(): Creates a new tabbed pane.

- **Methods:**

a) void addTab(String title, Component component): Adds a component with the specified title to the tabbed pane.

b) void insertTab(String title, Icon icon, Component component, String tip, int index): Inserts a component with the specified title and icon into the tabbed pane at the specified index.
c) void removeTabAt(int index): Removes the tab at the specified index.
d) Component getComponentAt(int index): Returns the component at the specified tab index.
e) int getTabCount(): Returns the number of tabs in the tabbed pane.
f) void setEnabled(boolean enabled): Sets whether the tabbed pane is enabled or disabled.
g) boolean isEnabled(): Returns true if the tabbed pane is enabled; otherwise, false.

21. JTable :

Description: Displays data in a tabular format with rows and columns.

- **Constructors:**

a) JTable(): Creates a table with an empty data model.
b) JTable(int rows, int cols): Creates a table with the specified number of rows and columns.
c) JTable(Object[][] rowData, Object[] columnNames): Creates a table with the specified data and column names.
d) JTable(TableModel dm): Creates a table with the specified data model.

- **Methods:**

a) void setModel(TableModel dataModel): Sets the data model for the table.
b) TableModel getModel(): Returns the data model for the table.
c) void setRowSelectionInterval(int index0, int index1): Selects the rows from index0 to index1.
d) int getSelectedRow(): Returns the index of the first selected row; returns -1 if no row is selected.
e) Object getValueAt(int row, int column): Returns the value at the specified row and column.
f) void setValueAt(Object value, int row, int column): Sets the value at the specified row and column.
g) void addMouseListener(MouseListener listener): Registers a mouse listener to be notified of mouse events.
h) void setEnabled(boolean enabled): Sets whether the table is enabled or disabled.
i) boolean isEnabled(): Returns true if the table is enabled; otherwise, false.

22. JTextArea:

Description: Provides a multi-line text input area.

- **Constructors:**

a) JTextArea(): Creates a new text area with no text.
b) JTextArea(String text): Creates a new text area with the specified text.
c) JTextArea(int rows, int cols): Creates a new text area with the specified number of rows and columns.
d) JTextArea(String text, int rows, int cols): Creates a new text area with the specified text, number of rows, and columns.

- **Methods:**

a) void setText(String text): Sets the text content of the text area.
b) String getText(): Returns the text content of the text area.
c) void append(String text): Appends the specified text to the end of the text area.
d) void setEditable(boolean editable): Sets whether the text area is editable or not.
e) boolean isEditable(): Returns true if the text area is editable; otherwise, false.
f) void setLineWrap(boolean wrap): Sets whether the text should wrap to the next line if it exceeds the width of the text area.
g) boolean getLineWrap(): Returns true if line wrapping is enabled; otherwise, false.

23. JTextField :

Description: Provides a single-line text input field.

- **Constructors:**

a) JTextField(): Creates a new text field with no initial text.
b) JTextField(String text): Creates a new text field with the specified initial text.
c) JTextField(int columns): Creates a new text field with the specified number of columns to display.
d) JTextField(String text, int columns): Creates a new text field with the specified initial text and number of columns to display.

- **Methods:**

a) void setText(String text): Sets the text content of the text field.
b) String getText(): Returns the text content of the text field.
c) void setEditable(boolean editable): Sets whether the text field is editable or not.
d) boolean isEditable(): Returns true if the text field is editable; otherwise, false.
e) void addActionListener(ActionListener listener): Registers an action listener to be notified when the user presses Enter in the text field.

f) void setHorizontalAlignment(int alignment): Sets the horizontal alignment of the text field content.

24. JToggleButton :

Description: Represents a button that can be toggled between selected and unselected states.

- **Constructors:**

a) JToggleButton(): Creates a toggle button with no text or icon.
b) JToggleButton(String text): Creates a toggle button with the specified text.
c) JToggleButton(Icon icon): Creates a toggle button with the specified icon.
d) JToggleButton(String text, Icon icon): Creates a toggle button with the specified text and icon.

- **Methods:**

a) void setSelected(boolean selected): Sets the selection state of the toggle button.
b) boolean isSelected(): Returns true if the toggle button is selected; otherwise, false.
c) void setText(String text): Sets the text of the toggle button.
d) String getText(): Returns the text of the toggle button.
e) void setIcon(Icon icon): Sets the icon of the toggle button.
f) Icon getIcon(): Returns the icon of the toggle button.
g) void addActionListener(ActionListener listener): Registers an action listener to be notified when the toggle button is clicked.

25. JToolBar :

Description: A container that groups together related components, typically buttons, in a horizontal or vertical bar.

- **Constructors:**

a) JToolBar(): Creates a new toolbar with horizontal orientation.

- **Methods:**

a) void add(Component comp): Adds the specified component to the toolbar.
b) void addSeparator(): Adds a separator to the toolbar.
c) void setFloatable(boolean floatable): Sets whether the toolbar is floatable or not.
d) boolean isFloatable(): Returns true if the toolbar is floatable; otherwise, false.
e) void setOrientation(int orientation): Sets the orientation of the toolbar (JToolBar.HORIZONTAL or JToolBar.VERTICAL).
f) int getOrientation(): Returns the orientation of the toolbar.

26. JTree :

Description: Displays hierarchical data in a tree structure, where each node can have children nodes.

- **Constructors:**

a) JTree(): Creates a new tree with no root.
b) JTree(TreeNode root): Creates a new tree with the specified root node.

- **Methods:**

a) void setModel(TreeModel newModel): Sets the data model for the tree.
b) TreeModel getModel(): Returns the data model for the tree.
c) void setRootVisible(boolean visible): Sets whether the root node is visible.
d) boolean isRootVisible(): Returns true if the root node is visible; otherwise, false.
e) void setEditable(boolean editable): Sets whether the tree is editable or not.
f) boolean isEditable(): Returns true if the tree is editable; otherwise, false.

27. JTreeItem :

Description: Represents an item in a tree structure, typically used with the Java Swing JTree component.

- **Properties:**

a) Object value: The value associated with the tree item.
b) List<JTreeItem> children: The list of child tree items.

- **Methods:**

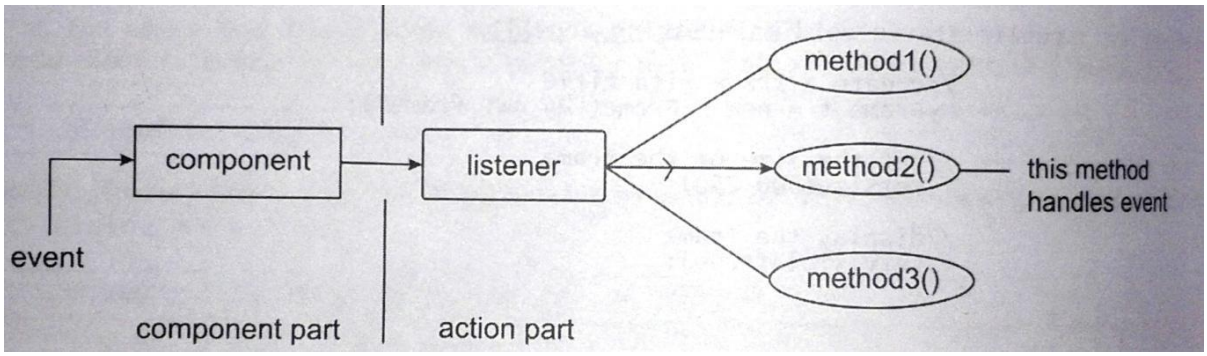
a) Object getValue(): Returns the value associated with the tree item.
b) void setValue(Object value): Sets the value associated with the tree item.
c) void addChild(JTreeItem child): Adds a child tree item to the current tree item.
d) void removeChild(JTreeItem child): Removes a child tree item from the current tree item.

Window: A window represents a rectangular area on the screen without any borders or title bar. The Window class creates a top-level window.

Frame: It is a subclass of Window and it has title bar, menu bar, border and resizing windows.

Delegation Event Model:

The modern approach (from version 1.1 onwards) to handle events is based on the delegation event model. 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 receives an event. Once an event is received, the listener processes the event and then returns. The advantage of this design is that the application logic that processes events is cleanly separated from the user interface logic that generates those events.

A user interface element is able to ‑delegate the processing of an event to a separate piece of code. In the delegation 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.

Events: An *event* is an object that describes a state change in a source. It can be generated as a consequence of a person interacting with the elements in a GUI. 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.

Event Sources: A source is an object that generates an event. Generally sources are components. Sources may generate more than one type of event.

A source must register listeners in order for the 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)

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( ).

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)

Event Listeners: A listener is an object that is notified when an event occurs. It has two major requirements.

1. It must have been registered with one or more sources to receive notifications about specific types of events.
2. 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* package.

Sources of Events:

Event Source	Description
Button	Generates action events when the button is pressed.
Check box	Generates item events when the check box is selected or deselected.
Choice	Generates item events when the choice is changed.
List	Generates action events when an item is double-clicked;
Menu item	Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected.
Scroll bar	Generates adjustment events when the scroll bar is manipulated.
Text components	Generates text events when the user enters a character.
Window	Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Event Classes and Listener Interfaces:

The java.awt.event package provides many event classes and Listener interfaces for event handling. At the root of the Java event class hierarchy is **EventObject**, which is in **java.util**. It is the super class for all events. It‘s one constructor is shown here:

EventObject(Object *src*) - Here, *src* is the object that generates this event.

EventObject contains two methods:

Object getSource( ) - Object on which event initially occurred.

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

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

int getID( )

.

The package **java.awt.event** defines many types of events that are generated by various user interface elements

Event Class	Description	Listener Interface
ActionEvent	Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.	ActionListener
AdjustmentEvent	Generated when a scroll bar is manipulated.	AdjustmentListener
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible.	ComponentListener
ContainerEvent	Generated when a component is added to or removed from a container.	ContainerListener
FocusEvent	Generated when a component gains or losses keyboard focus.	FocusListener
InputEvent	Abstract super class for all component input event classes.	
ItemEvent	Generated when a check box or list item is Clicked	ItemListener
KeyEvent	Generated when input is received from the keyboard.	KeyListener
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.	MouseListener and MouseMotionListener
TextEvent	Generated when the value of a text area or text field is changed.	TextListener
WindowEvent	Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.	WindowListener

Useful Methods of Component class:

Method	Description
public void add(Component c)	inserts a component.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.

EventListener Interfaces:

An event listener registers with an event source to receive notifications about the events of a particular type. Various event listener interfaces defined in the `java.awt.event` package are given below:

Interface	Description
ActionListener	Defines the actionPerformed() method to receive and process action events. *void actionPerformed(ActionEvent ae)*
MouseListener	Defines five methods to receive mouse events, such as when a mouse is clicked, pressed, released, enters, or exits a component *void mouseClicked(MouseEvent me)* *void mouseEntered(MouseEvent me)* *void mouseExited(MouseEvent me)* *void mousePressed(MouseEvent me)* *void mouseReleased(MouseEvent me)*
MouseMotionListener	Defines two methods to receive events, such as when a mouse is dragged or moved. *void mouseDragged(MouseEvent me)* *void mouseMoved(MouseEvent me)*
AdjustmentListner	Defines the adjustmentValueChanged() method to receive and process the adjustment events. *void adjustmentValueChanged(AdjustmentEvent ae)*
TextListener	Defines the textValueChanged() method to receive and process an event when the text value changes. *void textValueChanged(TextEvent te)*
WindowListener	Defines seven window methods to receive events. *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)*
ItemListener	Defines the itemStateChanged() method when an item has been *void itemStateChanged(ItemEvent ie)*
WindowFocusListener	This interface defines two methods: **windowGainedFocus( )** and **windowLostFocus( )**. These are called when a window gains or loses input focus. Their general forms are shown here: *void windowGainedFocus(WindowEvent we)* *void windowLostFocus(WindowEvent we)*
ComponentListener	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)*

ContainerListener	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)*
FocusListener	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 shown here: *void focusGained(FocusEvent fe)* *void focusLost(FocusEvent fe)*
KeyListener	This interface defines three methods. *void keyPressed(KeyEvent ke)* *void keyReleased(KeyEvent ke)* *void keyTyped(KeyEvent ke)*

Steps to perform Event Handling

Following steps are required to perform event handling:

1. Register the component with the Listener
2. Implement the concerned interface

Registration Methods:

For registering the component with the Listener, many classes provide the registration methods. For example:

- **Button**
  - public void addActionListener(ActionListener a){}
- **MenuItem**
  - public void addActionListener(ActionListener a){}
- **TextField**
  - public void addActionListener(ActionListener a){}
  - public void addTextListener(TextListener a){}
- **TextArea**
  - public void addTextListener(TextListener a){}
- **Checkbox**
  - public void addItemListener(ItemListener a){}
- **Choice**
  - public void addItemListener(ItemListener a){}
- **List**
  - public void addActionListener(ActionListener a){}
  - public void addItemListener(ItemListener a){}
- **Mouse**
  - public void addMouseListener(MouseListener a){}

Event Classes in Java Swing

1. **ActionEvent Class:** Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.

- **Contructors:**
  a. ActionEvent(Object src, int type, String cmd)
  b. ActionEvent(Object src, int type, String cmd, int modifiers)
  c. ActionEvent(Object src, int type, String cmd, long when, int modifiers)
  { src: Source on which event occurred **| type:** type of event **| modifiers:** indicates which modifier keys (ALT, CTRL, META, and/or SHIFT) were pressed when the event was generated **}**

- **Methods:**
  a. String getActionCommand(): Obtain the command name for the invoking ActionEvent object.
  b. int getModifiers(): Returns a value that indicates which modifier keys (ALT, CTRL, META, and/or SHIFT) were pressed when the event was generated.
  c. long getWhen(): Returns event's timestamp i.e. when did it occur.

2. **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.

- **Constructors:**
  a. AdjustmentEvent(Object src, int id, int type, int value)
  b. AdjustmentEvent(Object src, int id, int type, int value, boolean isAdjusting)
  { src: Source of the event. **| id:** Identifier for the event. **| type:** Type of adjustment event. **| value:** Value associated with the adjustment. **| isAdjusting:** Indicates whether the adjustment is still in progress.**}**

- **Methods:**
  a. int getAdjustmentType(): Retrieves the type of adjustment associated with the event.
  b. Adjustable getAdjustable():Obtains the adjustable component associated with the event.
  c. int getValue():Retrieves the current value of the adjustable component.
  d. boolean getValueIsAdjusting(): Determines if the value of the adjustable component is still in the process of being adjusted.

3. **ComponentEvent Class:** A ComponentEvent is generated when the size, position, or visibility of a component is changed.

- **Constructors:**
  a. ComponentEvent(Component src, int id) **{ src**: Source of Event | **id**: Identifier of event **}**

- **Methods:**
  a. Component getComponent(): Returns the component associated with the event.

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

- **Constructors:**

a. ContainerEvent(Container src, int id, Component child)
{ src: Source of event **| id:** identifier of the event **| child:** Child component affected by event **}**

- **Methods:**

a. Component getChild(): Returns a reference to the component that was added to or removed from the container

5. **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.

- **Constructors:**

a. FocusEvent(Component src, int type)
b. FocusEvent(Component src, int type, boolean temporaryFlag)
c. Focus Event(Component src, int type, boolean temporaryFlag, Component other)
{ src: Event Source | **type**: Type of event | **tempFlag**: 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.) **}**

- **Methods:**

a. Component getComponent(): Return component reference on which event occurred.
b. boolean isTemporary(): The isTemporary( ) method indicates if this focus change is temporary.

6. **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.

- **Constructors:**
None (Abstract class)

- **Methods:**

a. boolean isAltDown()
b. boolean isControlDown()
c. boolean isMetaDown()
d. boolean isShiftDown()
{ These methods returns the Boolean value whether given type of modifier key was pressed or not **}**
* **Modifers**: [ALT_MASK, META_MASK, ALT_GRAPH_MASK, SHIFT_MASK, CTRL_MASK]

7. **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.

- **Constructors:**

a. ItemEvent(ItemSelectable src, int id, Object item, int stateChange)
{ state: DESELECTED, SELECTED **}**

- **Methods:**

a. Object getItem(): Return object reference that generated item event.
b. ItemSelectable getItemSelectable(): Returns **ItemSelectable** object reference
c. int getStateChange(): Returns the state change (i.e., SELECTED or DESELECTED).

8. **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.

- **Constructors:**

a. KeyEvent(Component src, int id, long when, int modifiers, int keyCode, char keyChar)
{ when: Timestamp when key was pressed | **keyCode:** The virtual key code, such as VK_UP, VK_A, and so forth, is passed in code | **keyChar:** The character equivalent (if one exists) is passed in ch. If no valid character exists, then ch contains CHAR_UNDEFINED **}**

- **Methods:**

a. char getKeyChar(): Returns character that was entered
b. int getKeyCode(): Returns keyCode as described above..

9. **MouseEvent Class:** (Subclass of InputEvent) 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.

- **Constructors:**

a. MouseEvent(Component src, int id, long when, int modifiers, int x, int y, int clickCount, boolean popupTrigger)
{ **popipTrigger**: The triggersPopup flag indicates if this event causes a pop-up menu to appear on this platform.. Other parameters are already explained above }

- **Methods:**

a. int getButton(): Returns button reference on which event occurred
b. int getClickCount(): Returns number of times the mouse was clicked
c. Point getPoint(): Returns x & y co-ordinate values where the event occurred.

10. **MouseWheelEvent Class:** The MouseWheelEvent class encapsulates a mouse wheel event. It is a subclass of MouseEvent

- **Constructors:**

a. MouseWheelEvent(Component src, int id, long when, int modifiers, int x, int y, int clickCount, boolean popupTrigger, int scrollType, int scrollAmount, int wheelRotation)
{ scrollType: Type of scroll **| scrollAmount:** Amount of scroll **| wheelRotation:** Amount of wheel rotation **}**

- **Methods:**

a. int getScrollType(): It returns either WHEEL_UNIT_SCROLL or WHEEL_BLOCK_SCROLL
b. int getScrollAmount():If the scroll type is WHEEL_UNIT_SCROLL, you can obtain the number of units
c. int getWheelRotation(): Returns the number of rotational units.

11. **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.

- **Constructors:**

a. TextEvent(Object src, int id)
{ src: Source of event | **id:** Type of event **}**

12. **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 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.

- **Constructors:**

a. WindowEvent(Window src, int id)
b. WindowEvent(Window src, int id, Window oppositeWindow)
{ oppositeWindow: Opposite window related to this event **}**

- **Methods:**

a. Window getWindow(): Returns window object on which event occurred
b. Window getOppositeWindow(): Returns opposite window object that of which event occurred.
c. int getOldState(): Returns static variable value of state from which window entered new state.
d. int getNewState(): Returns static variable value of state to which window entered after event occurred.