**UNIT – IV**

**AWT Controls: The AWT class hierarchy – user interface components- Labels - Button – Text Components - Check Box - Check Box Group - Choice - List Box - Panels – Scroll Pane - Menu - Scroll Bar**

**Working with Frame class - Colour - Fonts and layout managers**

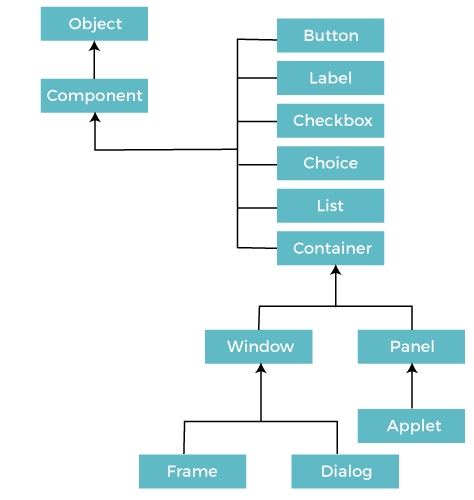
**Event Handling: Events - Event sources – Event Listeners - Event Delegation Model (EDM) – Handling Mouse and Keyboard Events - Adapter classes - Inner**

**Classes**

**AWT Controls in Java**

AWT (Abstract Window Toolkit) is a part of Java’s **java.awt** package, which provides components for building GUI applications. It includes a variety of controls like buttons, labels, text fields, checkboxes, choice menus, etc.

AWT Class Hierarchy



**Label (Label class)**

* **Constructor:**
  1. Label() → Creates an empty label.
  2. Label(String text) → Creates a label with the given text.
  3. Label(String text, int alignment) → Creates a label with text and alignment (Label.LEFT, Label.CENTER, Label.RIGHT).
* **Methods:**
  1. setText(String text) → Sets the label text.
  2. getText() → Returns the label text.
  3. setAlignment(int alignment) → Sets text alignment.

**2. TextField (TextField class)**

* **Constructor:**
  1. TextField() → Creates an empty text field.
  2. TextField(int columns) → Creates a text field with a specified width.
  3. TextField(String text) → Creates a text field with default text.
  4. TextField(String text, int columns) → Creates a text field with default text and width.
* **Methods:**
  1. setText(String text) → Sets text.
  2. getText() → Retrieves the text.
  3. setEchoChar(char c) → Sets a character mask for password fields.

**3. TextArea (TextArea class)**

* **Constructor:**
  1. TextArea() → Creates an empty multi-line text area.
  2. TextArea(String text) → Creates a text area with the given text.
  3. TextArea(int rows, int columns) → Creates a text area with specified dimensions.
  4. TextArea(String text, int rows, int columns) → Creates a text area with text and dimensions.
* **Methods:**
  1. setText(String text) → Sets text.
  2. getText() → Retrieves text.
  3. append(String str) → Appends text.

**4. Checkbox (Checkbox class)**

* **Constructor:**
  1. Checkbox() → Creates an empty checkbox.
  2. Checkbox(String label) → Creates a checkbox with a label.
  3. Checkbox(String label, boolean state) → Creates a checkbox with an initial state.
  4. Checkbox(String label, CheckboxGroup group, boolean state) → Creates a radio button-style checkbox.
* **Methods:**
  1. setState(boolean state) → Sets checkbox state.
  2. getState() → Returns checkbox state (true or false).
  3. setLabel(String label) → Sets checkbox label.

**5. CheckboxGroup (CheckboxGroup class)**

* **Constructor:**
  1. CheckboxGroup() → Creates a group of radio buttons.
* **Methods:**
  1. setSelectedCheckbox(Checkbox cb) → Selects a checkbox.
  2. getSelectedCheckbox() → Returns the currently selected checkbox.

**6. Choice (Choice class, Dropdown)**

* **Constructor:**
  1. Choice() → Creates an empty dropdown list.
* **Methods:**
  1. add(String item) → Adds an item.
  2. getItem(int index) → Gets an item at a specific index.
  3. getSelectedItem() → Returns the selected item.
  4. getSelectedIndex() → Returns the index of the selected item.

**7. List (List class)**

* **Constructor:**
  1. List() → Creates an empty list.
  2. List(int rows, boolean multipleSelection) → Creates a list with a specified number of visible rows and multiple selection option.
* **Methods:**
  1. add(String item) → Adds an item.
  2. getItem(int index) → Gets an item at a specific index.
  3. getSelectedItem() → Gets the selected item.
  4. getSelectedIndex() → Gets the selected index.

**8. Scrollbar (Scrollbar class)**

* **Constructor:**
  1. Scrollbar() → Creates a vertical scrollbar.
  2. Scrollbar(int orientation) → Creates a scrollbar (Scrollbar.HORIZONTAL or Scrollbar.VERTICAL).
  3. Scrollbar(int orientation, int value, int visible, int min, int max) → Creates a scrollbar with defined values.
* **Methods:**
  1. setValue(int value) → Sets scrollbar position.
  2. getValue() → Gets the current scrollbar value.

**9. Button (Button class)**

* **Constructor:**
  1. Button() → Creates an empty button.
  2. Button(String label) → Creates a button with a label.
* **Methods:**
  1. setLabel(String label) → Sets the button label.
  2. getLabel() → Retrieves the label.

**10. Panel (Panel class)**

* **Constructor:**
  1. Panel() → Creates an empty container to hold components.
* **Methods:**
  1. add(Component c) → Adds a component to the panel.

**11. Menu (Menu class and MenuBar class)**

* **Constructors:**
  1. MenuBar() → Creates a menu bar.
  2. Menu(String title) → Creates a menu with a title.
  3. MenuItem(String label) → Creates a menu item.
* **Methods:**
  1. add(MenuItem item) → Adds an item to the menu.
  2. add(Menu menu) → Adds a submenu.
  3. setMenuBar(MenuBar mb) → Sets a menu bar to a frame.

**Labels**

A **Label** is a non-editable text field used to display information.

**Example:**

import java.awt.\*;

public class LabelExample {

public static void main(String[] args) {

Frame f = new Frame("Label Example");

Label l1 = new Label("Username:");

l1.setBounds(50, 50, 100, 30);

f.add(l1);

f.setSize(300, 200);

f.setLayout(null);

f.setVisible(true);

}

}

### ****Button****

A **Button** is a clickable UI element that performs an action when clicked.

#### ****Example:****

import java.awt.\*;

import java.awt.event.\*;

public class ButtonExample {

public static void main(String[] args) {

Frame f = new Frame("Button Example");

Button b = new Button("Click Me");

b.setBounds(100, 100, 80, 30);

b.addActionListener(e -> System.out.println("Button Clicked"));

f.add(b);

f.setSize(300, 200);

f.setLayout(null);

f.setVisible(true);

}

}

### ****Text Components (TextField and TextArea)****

* **TextField**: A single-line text input field.
* **TextArea**: A multi-line text input field.

Example:

import java.awt.\*;

public class TextComponentExample {

public static void main(String[] args) {

Frame f = new Frame("Text Component Example");

TextField tf = new TextField();

tf.setBounds(50, 50, 150, 30);

TextArea ta = new TextArea("Enter text here...");

ta.setBounds(50, 100, 200, 100);

f.add(tf);

f.add(ta);

f.setSize(300, 250);

f.setLayout(null);

f.setVisible(true);

}

}

### ****Checkbox****

A **Checkbox** allows users to select or deselect an option.

#### ****Example:****

import java.awt.\*;

public class CheckboxExample {

public static void main(String[] args) {

Frame f = new Frame("Checkbox Example");

Checkbox cb1 = new Checkbox("Java");

cb1.setBounds(50, 50, 100, 30);

f.add(cb1);

f.setSize(300, 200);

f.setLayout(null);

f.setVisible(true);

}

}

### ****Checkbox Group****

A **CheckboxGroup** creates radio button-like behavior.

Example:

import java.awt.\*;

public class CheckboxGroupExample {

public static void main(String[] args) {

Frame f = new Frame("Checkbox Group Example");

CheckboxGroup cbg = new CheckboxGroup();

Checkbox male = new Checkbox("Male", cbg, false);

male.setBounds(50, 50, 100, 30);

Checkbox female = new Checkbox("Female", cbg, false);

female.setBounds(50, 100, 100, 30);

f.add(male);

f.add(female);

f.setSize(300, 200);

f.setLayout(null);

f.setVisible(true);

}

}

### ****Choice****

A **Choice** provides a dropdown list of options.

Example:

import java.awt.\*;

public class ChoiceExample {

public static void main(String[] args) {

Frame f = new Frame("Choice Example");

Choice c = new Choice();

c.setBounds(50, 50, 100, 30);

c.add("C");

c.add("C++");

c.add("Java");

c.add("Python");

f.add(c);

f.setSize(300, 200); import java.awt.\*;

public class ListExample {

public static void main(String[] args) {

Frame f = new Frame("List Example");

List l = new List(4, true);

l.setBounds(50, 50, 100, 100);

l.add("C");

l.add("C++");

l.add("Java");

l.add("Python");

f.add(l);

f.setSize(300, 200);

f.setLayout(null);

f.setVisible(true);

}

} f.setLayout(null);

f.setVisible(true);

}

}

### ****List Box****

A **List** allows users to select multiple items.

Example:

import java.awt.\*;

public class ListExample {

public static void main(String[] args) {

Frame f = new Frame("List Example");

List l = new List(4, true);

l.setBounds(50, 50, 100, 100);

l.add("C");

l.add("C++");

l.add("Java");

l.add("Python");

f.add(l);

f.setSize(300, 200);

f.setLayout(null);

f.setVisible(true);

}

}

### ****Panel****

A **Panel** is a container that holds components.

#### ****Example:****

import java.awt.\*;

public class PanelExample {

public static void main(String[] args) {

Frame f = new Frame("Panel Example");

Panel p = new Panel();

p.setBounds(50, 50, 200, 100);

p.setBackground(Color.LIGHT\_GRAY);

Button b = new Button("Inside Panel");

p.add(b);

f.add(p);

f.setSize(300, 200);

f.setLayout(null);

f.setVisible(true);

}

}

### ****ScrollPane****

A **ScrollPane** allows scrolling of large content.

Example:

import java.awt.\*;

public class ScrollPaneExample {

public static void main(String[] args) {

Frame f = new Frame("ScrollPane Example");

ScrollPane sp = new ScrollPane();

sp.setSize(200, 150);

TextArea ta = new TextArea("Scrollable content...");

sp.add(ta);

f.add(sp);

f.setSize(300, 200);

f.setLayout(null);

f.setVisible(true);

}

}

**Menu and ScrollBar**

* **Menu**: Used for menu bars.
* **ScrollBar**: Allows scrolling vertically/horizontally.

### ****Working with Frame Class****

A **Frame** is a window that serves as a container for components.

**Color, Fonts, and Layout Managers**

* **Color:** setBackground(Color.RED)
* **Fonts:** setFont(new Font("Serif", Font.BOLD, 14))
* **Layouts:** FlowLayout, BorderLayout, GridLayout.

**Complete Example**

import java.awt.\*;

import java.awt.event.\*;

public class AWTControlsExample extends Frame {

AWTControlsExample() {

setTitle("AWT Controls Example");

setSize(500, 500);

setLayout(null);

setVisible(true);

// Label

Label label = new Label("Enter Name:");

label.setBounds(50, 50, 100, 30);

add(label);

// TextField

TextField textField = new TextField();

textField.setBounds(160, 50, 150, 30);

add(textField);

// TextArea

TextArea textArea = new TextArea("Enter your text here...");

textArea.setBounds(50, 90, 260, 60);

add(textArea);

// Checkbox

Checkbox cb1 = new Checkbox("Java");

cb1.setBounds(50, 160, 60, 30);

Checkbox cb2 = new Checkbox("Python");

cb2.setBounds(120, 160, 70, 30);

add(cb1);

add(cb2);

// CheckboxGroup (Radio Buttons)

CheckboxGroup cbg = new CheckboxGroup();

Checkbox rb1 = new Checkbox("Male", cbg, false);

rb1.setBounds(50, 200, 80, 30);

Checkbox rb2 = new Checkbox("Female", cbg, false);

rb2.setBounds(140, 200, 80, 30);

add(rb1);

add(rb2);

// Choice (Dropdown)

Choice choice = new Choice();

choice.setBounds(50, 240, 100, 30);

choice.add("C");

choice.add("C++");

choice.add("Java");

choice.add("Python");

add(choice);

// List

List list = new List(4, true);

list.setBounds(180, 240, 100, 80);

list.add("Item 1");

list.add("Item 2");

list.add("Item 3");

add(list);

// Scrollbar

Scrollbar sb = new Scrollbar(Scrollbar.HORIZONTAL, 0, 20, 0, 100);

sb.setBounds(50, 330, 200, 20);

add(sb);

// Button

Button btn = new Button("Submit");

btn.setBounds(50, 370, 80, 30);

add(btn);

// Panel

Panel panel = new Panel();

panel.setBounds(300, 50, 150, 100);

panel.setBackground(Color.LIGHT\_GRAY);

Button panelButton = new Button("Panel Btn");

panel.add(panelButton);

add(panel);

// MenuBar and Menu

MenuBar mb = new MenuBar();

Menu menu = new Menu("File");

MenuItem mi1 = new MenuItem("Open");

MenuItem mi2 = new MenuItem("Save");

MenuItem mi3 = new MenuItem("Exit");

menu.add(mi1);

menu.add(mi2);

menu.add(mi3);

mb.add(menu);

setMenuBar(mb);

// Close Window Event

addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent e) {

dispose();

}

});

}

public static void main(String[] args) {

new AWTControlsExample();

}

}

## ****Working with Frame Class, Colors, Fonts, and Layout Managers in AWT****

### ****1. Frame Class****

The Frame class in AWT is a top-level window that contains GUI components like buttons, labels, text fields, etc.

#### ****Constructors:****

* Frame() → Creates an empty frame.
* Frame(String title) → Creates a frame with a specified title.

#### ****Methods:****

* setSize(int width, int height) → Sets the frame size.
* setTitle(String title) → Sets the frame title.
* setVisible(boolean state) → Makes the frame visible.
* setLayout(LayoutManager layout) → Sets a layout manager.
* setBackground(Color c) → Sets background color.
* setForeground(Color c) → Sets text color.
* add(Component c) → Adds a component to the frame.

### ****2. Working with Colors****

The Color class in AWT allows setting background and text colors.

#### ****Creating Colors:****

* Color(int r, int g, int b) → Custom color using RGB values (0-255).
* Color predefined colors:

**Color.RED, Color.BLUE, Color.GREEN, Color.YELLOW, Color.BLACK, Color.WHITE, etc.**

#### ****Applying Colors:****

frame.setBackground(Color.LIGHT\_GRAY);

button.setForeground(Color.RED);

### ****3. Working with Fonts****

The Font class in AWT allows setting fonts for text components.

#### ****Font Constructor:****

* Font(String name, int style, int size)
  + Font.PLAIN, Font.BOLD, Font.ITALIC

#### ****Applying Fonts:****

Font myFont = new Font("Arial", Font.BOLD, 16);

label.setFont(myFont);

### ****4. Layout Managers****

Layout managers control the arrangement of components in a container.

| **Layout Manager** | **Description** |
| --- | --- |
| **FlowLayout** | Places components in a row (default for Panel). |
| **BorderLayout** | Divides the container into NORTH, SOUTH, EAST, WEST, CENTER (default for Frame). |
| **GridLayout** | Arranges components in a grid of rows and columns. |
| **CardLayout** | Allows switching between different panels like a deck of cards. |
| **GridBagLayout** | Flexible grid-based layout with different component sizes. |
| **Null Layout (setLayout(null))** | Allows absolute positioning using setBounds(). |

#### ****Example: Setting Layout Managers****

setLayout(new FlowLayout()); // Components placed in a row

setLayout(new BorderLayout()); // Divides into five regions

setLayout(new GridLayout(2, 2)); // 2x2 grid

## ****Example Java Program Demonstrating Frame, Color, Font, and Layouts****

import java.awt.\*;

public class AWTExample extends Frame {

AWTExample() {

// Set Frame properties

setTitle("AWT Example");

setSize(400, 300);

setBackground(Color.LIGHT\_GRAY);

setLayout(new FlowLayout());

// Create and set font

Font myFont = new Font("Arial", Font.BOLD, 14);

// Label

Label label = new Label("Enter Name:");

label.setFont(myFont);

label.setForeground(Color.BLUE);

add(label);

// TextField

TextField textField = new TextField(20);

add(textField);

// Button

Button button = new Button("Submit");

button.setBackground(Color.YELLOW);

button.setFont(new Font("Verdana", Font.BOLD, 12));

add(button);

// Set Frame visibility

setVisible(true);

}

public static void main(String[] args) {

new AWTExample();

}

}

## ****Event Handling in AWT: Events and Event Sources****

### ****1. What is Event Handling?****

Event handling in Java AWT refers to how user interactions (such as clicking a button or typing in a text field) trigger specific actions in a program. It is implemented using the **Event Delegation Model**, which consists of:

* **Event Source** → The component that generates an event (e.g., Button, TextField).
* **Event Listener** → A class that listens for events and defines the response.
* **Event Object** → Contains details about the event.

### ****2. Events in Java AWT****

Events in AWT are represented as subclasses of java.awt.event.AWTEvent. Some commonly used events are:

| **Event Class** | **Description** |
| --- | --- |
| ActionEvent | Generated when a button is clicked, menu item selected, or text field action occurs. |
| MouseEvent | Triggered by mouse actions (click, press, release, move, enter, exit). |
| KeyEvent | Occurs when a key is pressed or released. |
| WindowEvent | Generated when a window is opened, closed, or minimized. |
| ItemEvent | Triggered when an item in a Choice, Checkbox, or List is selected/deselected. |
| TextEvent | Fired when the text in a TextField or TextArea changes. |
| AdjustmentEvent | Occurs when the value of a Scrollbar changes. |
| FocusEvent | Occurs when a component gains or loses focus. |

### ****3. Event Sources****

Event sources are AWT components that generate events. Some common sources include:

| **Component (Source)** | **Event Generated** |
| --- | --- |
| Button | ActionEvent |
| TextField | ActionEvent, TextEvent |
| Checkbox | ItemEvent |
| Choice | ItemEvent |
| List | ItemEvent, ActionEvent |
| Scrollbar | AdjustmentEvent |
| Window | WindowEvent |
| Mouse | MouseEvent |
| Keyboard | KeyEvent |

### ****4. Implementing Event Handling****

To handle an event, we follow **three steps**:

1. **Implement the Listener Interface** (e.g., ActionListener, MouseListener).
2. **Register the Listener** with the event source.
3. **Override the Event Handling Method** to define the response.

#### ****Example: Handling Button Click (ActionEvent)****

import java.awt.\*;

import java.awt.event.\*;

public class EventExample extends Frame implements ActionListener {

Button button;

EventExample() {

// Create Frame

setTitle("Event Handling Example");

setSize(300, 200);

setLayout(new FlowLayout());

// Create Button

button = new Button("Click Me");

add(button);

// Register Listener

button.addActionListener(this);

setVisible(true);

}

// Override actionPerformed method

public void actionPerformed(ActionEvent e) {

System.out.println("Button Clicked!");

}

public static void main(String[] args) {

new EventExample();

}

}

#### ****Explanation:****

* The class implements ActionListener, which listens for button clicks.
* button.addActionListener(this); registers the listener.
* actionPerformed(ActionEvent e) prints a message when the button is clicked.

### ****5. Different Event Listeners and Their Methods****

| **Listener Interface** | **Event Methods** |
| --- | --- |
| ActionListener | actionPerformed(ActionEvent e) |
| MouseListener | mouseClicked(MouseEvent e), mousePressed(MouseEvent e), mouseReleased(MouseEvent e), mouseEntered(MouseEvent e), mouseExited(MouseEvent e) |
| MouseMotionListener | mouseMoved(MouseEvent e), mouseDragged(MouseEvent e) |
| KeyListener | keyPressed(KeyEvent e), keyReleased(KeyEvent e), keyTyped(KeyEvent e) |
| WindowListener | windowOpened(WindowEvent e), windowClosing(WindowEvent e), windowClosed(WindowEvent e), windowIconified(WindowEvent e), windowDeiconified(WindowEvent e), windowActivated(WindowEvent e), windowDeactivated(WindowEvent e) |
| ItemListener | itemStateChanged(ItemEvent e) |
| FocusListener | focusGained(FocusEvent e), focusLost(FocusEvent e) |

### ****6. Example: Handling Multiple Events****

import java.awt.\*;

import java.awt.event.\*;

public class MultiEventExample extends Frame implements ActionListener, MouseListener {

Button button;

Label label;

MultiEventExample() {

setTitle("Multi-Event Handling");

setSize(400, 250);

setLayout(new FlowLayout());

label = new Label("Click the button or the frame");

add(label);

button = new Button("Click Me");

add(button);

// Registering Event Listeners

button.addActionListener(this);

addMouseListener(this);

setVisible(true);

}

// Button Click Event

public void actionPerformed(ActionEvent e) {

label.setText("Button Clicked!");

}

// Mouse Events

public void mouseClicked(MouseEvent e) { label.setText("Mouse Clicked!"); }

public void mousePressed(MouseEvent e) { label.setText("Mouse Pressed!"); }

public void mouseReleased(MouseEvent e) { label.setText("Mouse Released!"); }

public void mouseEntered(MouseEvent e) { label.setText("Mouse Entered!"); }

public void mouseExited(MouseEvent e) { label.setText("Mouse Exited!"); }

public static void main(String[] args) {

new MultiEventExample();

}

}

## ****Summary****

* **Events** are triggered by user actions (e.g., button click, mouse movement).
* **Event Sources** generate these events (e.g., Button, TextField).
* **Event Listeners** handle these events by implementing interfaces like ActionListener or MouseListener.
* **Event Handling** follows a process:
  1. Implement the listener interface.
  2. Register the listener with the source.
  3. Override the appropriate method to define behavior.

## ****Event Listeners****

Event listeners are interfaces in java.awt.event that define methods to handle various events. A listener is registered with an event source, and when the event occurs, the respective method is invoked.

### ****Common Event Listeners in AWT****

| **Listener Interface** | **Events Handled** | **Methods to Implement** |
| --- | --- | --- |
| ActionListener | Button click, MenuItem selection, TextField action | actionPerformed(ActionEvent e) |
| MouseListener | Mouse click, press, release, enter, exit | mouseClicked(), mousePressed(), mouseReleased(), mouseEntered(), mouseExited() |
| MouseMotionListener | Mouse move and drag | mouseMoved(), mouseDragged() |
| KeyListener | Key press, release, and type | keyPressed(), keyReleased(), keyTyped() |
| ItemListener | Checkbox, Choice, List item selection | itemStateChanged(ItemEvent e) |
| WindowListener | Window open, close, minimize, maximize | windowOpened(), windowClosing(), windowClosed(), windowIconified(), windowDeiconified(), windowActivated(), windowDeactivated() |

## ****2. Event Delegation Model (EDM)****

The **Event Delegation Model (EDM)** is Java’s approach to handling events efficiently. Instead of components processing their own events, they delegate them to an external listener.

### ****How EDM Works?****

1. **Event Source** → Generates the event (e.g., Button, TextField).
2. **Event Object** → Encapsulates details of the event (e.g., ActionEvent, MouseEvent).
3. **Event Listener** → Listens for the event and executes appropriate code.
4. **Event Registration** → The event source registers the listener using methods like addActionListener(), addMouseListener(), etc.

### ****Example: Button Click Handling Using EDM****

import java.awt.\*;

import java.awt.event.\*;

public class EDMExample extends Frame implements ActionListener {

Button button;

EDMExample() {

setTitle("Event Delegation Model Example");

setSize(300, 200);

setLayout(new FlowLayout());

button = new Button("Click Me");

add(button);

// Register listener

button.addActionListener(this);

setVisible(true);

}

// Event handling method

public void actionPerformed(ActionEvent e) {

System.out.println("Button Clicked!");

}

public static void main(String[] args) {

new EDMExample();

}

}

## ****3. Handling Mouse and Keyboard Events****

### ****Mouse Events****

Mouse events are handled using MouseListener and MouseMotionListener.

#### ****Example: Handling Mouse Events****

import java.awt.\*;

import java.awt.event.\*;

public class MouseEventExample extends Frame implements MouseListener {

Label label;

MouseEventExample() {

setTitle("Mouse Event Example");

setSize(300, 200);

setLayout(new FlowLayout());

label = new Label("Click or move the mouse");

add(label);

addMouseListener(this); // Registering Mouse Listener

setVisible(true);

}

public void mouseClicked(MouseEvent e) { label.setText("Mouse Clicked!"); }

public void mousePressed(MouseEvent e) { label.setText("Mouse Pressed!"); }

public void mouseReleased(MouseEvent e) { label.setText("Mouse Released!"); }

public void mouseEntered(MouseEvent e) { label.setText("Mouse Entered!"); }

public void mouseExited(MouseEvent e) { label.setText("Mouse Exited!"); }

public static void main(String[] args) {

new MouseEventExample();

}

}

### ****Keyboard Events****

Keyboard events are handled using KeyListener.

#### ****Example: Handling Keyboard Events****

import java.awt.\*;

import java.awt.event.\*;

public class KeyEventExample extends Frame implements KeyListener {

Label label;

KeyEventExample() {

setTitle("Keyboard Event Example");

setSize(300, 200);

setLayout(new FlowLayout());

label = new Label("Press any key");

add(label);

addKeyListener(this); // Registering Key Listener

setVisible(true);

}

public void keyPressed(KeyEvent e) { label.setText("Key Pressed: " + e.getKeyChar()); }

public void keyReleased(KeyEvent e) { label.setText("Key Released: " + e.getKeyChar()); }

public void keyTyped(KeyEvent e) { label.setText("Key Typed: " + e.getKeyChar()); }

public static void main(String[] args) {

new KeyEventExample();

}

}

## ****4. Adapter Classes****

Adapter classes simplify event handling by providing empty method implementations of listener interfaces. Instead of implementing all methods, we override only the ones needed.

### ****Common Adapter Classes****

| **Adapter Class** | **Implements** |
| --- | --- |
| MouseAdapter | MouseListener |
| MouseMotionAdapter | MouseMotionListener |
| KeyAdapter | KeyListener |
| WindowAdapter | WindowListener |

### ****Example: Using MouseAdapter****

import java.awt.\*;

import java.awt.event.\*;

public class AdapterExample extends Frame {

Label label;

AdapterExample() {

setTitle("Adapter Example");

setSize(300, 200);

setLayout(new FlowLayout());

label = new Label("Move the mouse inside frame");

add(label);

addMouseListener(new MouseAdapter() {

public void mouseClicked(MouseEvent e) {

label.setText("Mouse Clicked!");

}

});

setVisible(true);

}

public static void main(String[] args) {

new AdapterExample();

}

}

* **Why use adapters?**
  + Avoids implementing unused methods.
  + Simplifies code and improves readability.

## ****5. Inner Classes for Event Handling****

Instead of implementing listeners in the main class, we can use **inner classes** to keep the code more structured.

### ****Example: Using an Inner Class for Event Handling****

import java.awt.\*;

import java.awt.event.\*;

public class InnerClassExample extends Frame {

Button button;

InnerClassExample() {

setTitle("Inner Class Event Handling");

setSize(300, 200);

setLayout(new FlowLayout());

button = new Button("Click Me");

add(button);

button.addActionListener(new ButtonClickHandler());

setVisible(true);

}

// Inner class handling event

class ButtonClickHandler implements ActionListener {

public void actionPerformed(ActionEvent e) {

System.out.println("Button Clicked using Inner Class!");

}

}

public static void main(String[] args) {

new InnerClassExample();

}

}

* **Advantages of Inner Classes:**
  1. Keeps event-handling code closer to the component.
  2. Reduces complexity in large applications.

## ****Summary****

1. **Event Listeners**: Interfaces that handle events (ActionListener, MouseListener, KeyListener, etc.).
2. **Event Delegation Model (EDM)**: Components delegate events to external listeners.
3. **Handling Mouse & Keyboard Events**: Implement MouseListener, MouseMotionListener, and KeyListener.
4. **Adapter Classes**: Provide default implementations, reducing unnecessary method overrides.
5. **Inner Classes**: Helps keep event handling code organized.