

Introduction to AWT

AWT (Abstract Window Toolkit) is a **Java library** used to create **Graphical User Interface (GUI)** applications. It provides a set of classes for building windows, buttons, menus, text boxes, and other GUI components. AWT is part of the Java Foundation Classes (JFC) and allows Java programs to interact with the **native GUI** of the operating system (like Windows, macOS, or Linux).

Features of AWT

Feature	Description
GUI Development	Provides ready-made components (Label , Button , TextField , etc.)
Event Handling	Supports user interaction using the event model (java.awt.event package)
Graphics Support	Allows drawing of shapes, colors, and images
Layout Management	Automatically arranges components within containers
Platform Integration	Uses native GUI components for better OS-level performance

AWT Architecture Overview

AWT is based on three main categories:

1. **Components**: Individual GUI elements (e.g., Button, Label, TextField).
2. **Containers**: Hold and organize components (e.g., Frame, Panel, Applet).
3. **Event Handling**: Mechanism for responding to user actions (e.g., button clicks, key presses).

AWT Packages

AWT classes and interfaces are mainly found in two important packages:

a) **java.awt Package**

Contains classes for GUI components, containers, layout managers, colors, and fonts.

Class	Description
Button	Creates a push button
Label	Displays text on the screen
TextField	Single-line input box
TextArea	Multi-line input area
Frame	Main window for application
Panel	Generic container for grouping components
List, Checkbox, Choice	Provide list and selection controls
Color, Font, Graphics	Used for styling and drawing
LayoutManager	Controls component arrangement

b) java.awt.event Package

Contains classes and interfaces for event handling (user interactions).

Interface / Class	Purpose
ActionListener	Handles button clicks
WindowListener	Handles window events (open, close, minimize)
MouseListener, MouseMotionListener	Handles mouse events
KeyListener	Handles keyboard input
ItemListener	Handles item selection
ActionEvent, KeyEvent, MouseEvent	Event classes representing the action

Platform Dependence of AWT

AWT is **platform-dependent** because its components are **heavyweight**—they rely on the **native GUI components** of the operating system.

When you create an AWT Button, Java internally calls the underlying OS's button control. Therefore, the appearance and behavior of AWT components can **vary** between platforms (Windows, Linux, macOS).

Difference between Heavyweight and Lightweight Components

Type	Description	Example
Heavyweight	Uses native OS GUI components	AWT components (Button, TextField)
Lightweight	Pure Java components; not	Swing components (JButton,

Type	Description	Example
	dependent on OS	JTextField)

Example Program

```
import java.awt.*;

public class AWTExample {
    public static void main(String[] args) {
        Frame f = new Frame("AWT Example");

        Label l = new Label("Welcome to Java AWT!");
        l.setBounds(80, 100, 200, 30);

        f.add(l);
        f.setSize(350, 250);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```



Output: A small window (Frame) appears with a label “Welcome to Java AWT!”

AWT Components and Containers

Introduction to AWT Components

AWT (Abstract Window Toolkit) is a foundational part of Java's GUI (Graphical User Interface) framework. It provides classes for creating essential GUI elements like windows, buttons, text fields, lists, and more.

Package:

```
import java.awt.*;  
import java.awt.event.*;
```

Commonly Used AWT Components

Components are the basic, interactive GUI elements. Each is a subclass of the **Component** class.

Component	Description	Example Code Snippet
Label	Displays a non-editable, static text string, often used to show information or prompt for input.	Label lbl = new Label("Enter your name.");
Button	Creates a push button that initiates an action when clicked.	Button b = new Button("Click Me");
TextField	A single-line box used for accepting text input from the user.	TextField tf = new TextField("Type here");
TextArea	Used for accepting multi-line text input (e.g., comments or messages).	TextArea ta = new TextArea("Enter your message here");
Checkbox	Represents a binary option that can be checked or unchecked.	Checkbox cb = new Checkbox("I agree");
CheckboxGroup	Used to group Checkboxes so that only one can be selected at a time (Radio Buttons).	CheckboxGroup cg = new CheckboxGroup();
List	Displays a scrollable list of items, allowing the user to select one or multiple options.	List list = new List(3); // shows 3 items at a time

3. AWT Containers

A **Container** is a component specifically designed to hold and manage the layout of other components (including other containers).

Container	Description	Key Characteristic
Panel	A simple, generic container for organizing components.	Cannot exist independently; must be placed inside another top-level container (like a Frame).
Frame	A top-level window with a title bar, border, and control buttons (close/minimize).	The main window for AWT applications.
Dialog	A popup window used for focused interactions, such as displaying messages, warnings, or getting input.	Typically <i>modal</i> (blocks interaction with the main window until closed).
Applet	A Java program that runs embedded inside a web browser or applet viewer.	Extends the Applet class (from <code>java.applet</code>).

Event Handling in AWT

Introduction to Event Handling

Event handling in Java **AWT** (Abstract Window Toolkit) is a mechanism that controls the behavior of a program based on user interactions—like clicking a button, typing a key, or closing a window.

Java uses the **Event Delegation Model**, which defines three main parts:

- **Event Source:** The component that generates an event (e.g., a **Button**).
- **Event Object:** An object that encapsulates details about the event (e.g., **ActionEvent**, **MouseEvent**).
- **Event Listener:** An interface that receives and handles the event by defining specific methods.

Event Handling Steps

1. **Import** required packages (`java.awt.*` and `java.awt.event.*`).
2. **Implement** the appropriate listener interface (e.g., `ActionListener`).
3. **Register** the listener with the event source component using a method like `addActionListener()`.
4. **Override** the listener method(s) to define the action that should be performed.

Major Event Types in AWT

Event Type	Generated When	Listener Interface	Important Method
ActionEvent	Button clicked, Menu item chosen, Enter pressed in Text Field.	ActionListener	<code>public void actionPerformed(ActionEvent e)</code>
WindowEvent	Window opened, closing, closed, minimized, etc.	WindowListener	<code>public void windowClosing(WindowEvent e)</code>
MouseEvent	Mouse click, press, release, enter, or exit.	MouseListener	<code>public void mouseClicked(MouseEvent e)</code>
KeyEvent	Key pressed, released, or typed.	KeyListener	<code>public void keyTyped(KeyEvent e)</code>
ItemEvent	Item selected or deselected (e.g., Checkbox, List).	ItemListener	<code>public void itemStateChanged(ItemEvent e)</code>

Example: ActionEvent Implementation

This example demonstrates how to handle a button click using ActionListener.

```
import java.awt.*;
import java.awt.event.*;

public class ActionEventExample extends Frame implements
ActionListener {
    TextField tf;
    Button b;

    ActionEventExample() {
        // Component setup...
        tf = new TextField();
        b = new Button("Click Me");

        // 3. Register the listener
        b.addActionListener(this);

        // Add components and set up frame...
        // ...
    }

    // 4. Override the listener method
    public void actionPerformed(ActionEvent e) {
        tf.setText("Button Clicked!");
    }

    public static void main(String[] args) {
        new ActionEventExample();
    }
}
```

I cannot convert this document into a PDF file. I am a text-based AI and don't have the capability to generate or save files in proprietary formats like PDF.

However, I can provide the content in a **clearly formatted and easily copy-pastable** **Markdown format** which you can then paste into a word processor (like Google Docs, Microsoft Word, or LibreOffice) or a dedicated online Markdown-to-PDF converter to generate a PDF yourself.

Here is the structured content:

EVENT LISTENERS IN JAVA (AWT)

Introduction

In Java AWT, **event listeners** are interfaces that handle various user actions (events) like clicking a button, moving the mouse, typing keys, or closing a window.

- They are part of the `java.awt.event` package.
- Each listener interface has one or more abstract methods that must be implemented to respond to specific events.

Event Handling Mechanism

In Java AWT, the process involves three main components:

- **Event Source:** The GUI component (like Button, Frame, TextField).
- **Event Object:** Represents the event (ActionEvent, MouseEvent, etc.).
- **Event Listener:** The interface that receives and handles the event.

Steps to Handle an Event:

1. Import `java.awt.event.*`.
2. Implement the listener interface.
3. **Register the listener** with the component using methods like `addActionListener()`, `addKeyListener()`, etc.
4. Override required event-handling methods.

1. ActionListener

Detail	Description
Used For	Handling action events, most commonly button clicks .
Package	<code>java.awt.event.ActionListener</code>
Key Method	<code>public void actionPerformed(ActionEvent e)</code>

Example Snippet:

```
// ...
b.addActionListener(this); // Registration
// ...
public void actionPerformed(ActionEvent e) {
    System.out.println("Button Clicked!");
}
```

```
}
```

2. WindowListener

Detail	Description
Used For	Handling window-related events (opening, closing, minimizing, etc.).
Package	java.awt.event.WindowListener
Key Methods	windowOpened(), windowClosing() , windowClosed(), windowActivated(), windowDeactivated(), windowIconified(), windowDeiconified().

✓ Example Snippet:

```
// ...
addWindowListener(this); // Registration
// ...
public void windowClosing(WindowEvent e) {
    System.out.println("Window Closing...");
    dispose(); // Releases resources and closes the window
}
// All 7 methods must be implemented (even if empty)
// ...
```

3. MouseListener / MouseMotionListener

These are two separate interfaces for handling mouse activity.

Listener	Used For	Key Method(s)
MouseListener	Mouse clicks, presses, releases, entering, exiting component area.	mouseClicked(), mousePressed(), mouseReleased(), mouseEntered(), mouseExited().
MouseMotionListener	Mouse movement within the component's area.	mouseMoved(), mouseDragged().

✓ Example Snippet:

```
// ...
addMouseListener(this);
addMouseMotionListener(this); // Registration
// ...
public void mouseClicked(MouseEvent e) {
    System.out.println("Mouse Clicked at: " + e.getX() + ", " +
e.getY());
}
```

```

public void mouseMoved(MouseEvent e) {
    System.out.println("Mouse Moved");
}
// All required methods from both interfaces must be implemented
// ...

```

4. KeyListener

Detail	Description
Used For	Handling keyboard events (key press, release, and typing).
Package	java.awt.event.KeyListener
Key Methods	keyPressed() , keyReleased() , keyTyped() .

Example Snippet:

```

// ...
tf.addKeyListener(this); // Registration on a TextField
// ...
public void keyPressed(KeyEvent e) {
    System.out.println("Key Pressed: " + e.getKeyChar());
}
// ...

```

Summary Table

Listener Interface	Used For	Key Method(s)	Event Class
ActionListener	Button clicks, item selection (menu, list, etc.)	actionPerformed()	ActionEvent
WindowListener	Window actions (closing, opening, etc.)	windowClosing() etc. (7 methods)	WindowEvent
MouseListener	Mouse clicks & component area entry/exit	mouseClicked() etc. (5 methods)	MouseEvent
MouseMotionListener	Mouse drag or move actions	mouseMoved(), mouseDragged()	MouseEvent
KeyListener	Keyboard input	keyPressed(), keyReleased(), keyTyped()	KeyEvent

Would you like me to clarify the purpose of the **Adapter classes** in AWT, which help simplify implementing listeners like WindowListener and MouseListener?