**Java's UI frameworks**include**Java AWT, Java Swing, and JavaFX.**Thisplays a very important role in creating the user experience of **Java applications**. These frameworks provide a range of tools and components for creating **graphical user interfaces (GUIs**) that are not only functional but also visually appealing

**Java AWT**

The[**Abstract Window Toolkit (AWT)**](https://www.geeksforgeeks.org/java-awt-tutorial/) **is like the reliable foundation of Java's GUI development.** It's been around for a while and is like the sturdy base that helps developers build interfaces that look and feel the same, no matter which operating system they're on. Basically, AWT uses the special features of each operating system to make sure the interface feels familiar to users.

Even though there are newer frameworks with more advanced features, AWT is still a great choice for some Java applications. It is simple and very trustworthy which makes it perfect for projects where a straightforward approach to building interfaces is preferred.

**Features and Components of Java AWT**

* **Components for Building Interfaces:**AWT provides various different components like buttons, text fields, and checkboxes that you can basically use to build the visual part of your application.
* **Interaction with Users:**These components are designed to interact with users and simply allow them to click buttons, enter text in fields, and select particular options from checkboxes.
* **Handling Complex Elements:** AWT also offers more complex and multiple advanced elements like menus, dialogs, and windows that basically enable you to create innovatory interfaces that can simply display information and respond to user actions.

**Java Swing**

[**Java Swing**](https://www.geeksforgeeks.org/introduction-to-java-swing/) **is a very powerful GUI toolkit for Java applications**, introduced as an extension of AWT. Unlike AWT, Swing provides a rich set of components and features that are all implemented in [Java](https://www.geeksforgeeks.org/java/). While AWT components are based on the native platform, Swing components are simply entirely written in Java which provides a consistent look and feel across different platforms. And with this feature Swing simply becomes a very popular choice for cross-platform applications. Despite the emergence of newer frameworks like JavaFX, Swing remains relevant and widely used in Java GUI development.

**Features and Components of Java Swing**

* **Rich Component Library:**The design of complicated and feature-rich user interfaces is essentially made possible by Swing's vast library consisting multiple components, which includes buttons, text fields, lists, tables, and many more items.
* **Customization and Look-and-Feel:** Swing components are basically highly customizable, and they provide a consistent look and feel across various different platforms that gives developers greater control over the visual appearance of their applications.
* **Advanced Features:**Numerous advanced functions offered by Swing like support for drag and drop, integrated undo/redo capabilities, and pluggable style and feel, simply improve both the user experience and the productivity of developers.
* **Support for MVC Architecture:** Swing is basically designed on the [**Model-View-Controller (MVC)**](https://www.geeksforgeeks.org/mvc-framework-introduction/)architecture, which separates the data model, user interface, and control logic which simply makes it easier to maintain and extend applications.
* **Integration with AWT:** While Swing is generally an independent framework, it can seamlessly integrate with AWT that allows developers to combine AWT and Swing components in their applications for added flexibility.
* **Event-Driven Programming:**Swing simply follows an event-driven programming model in which, user actions or system events trigger responses in the application that simply enables the instructiveness and responsiveness in interfaces.

**JavaFX**

[**JavaFX**](https://www.geeksforgeeks.org/javafx-tutorial/)**is a modern UI toolkit for Java applications which is designed to replace Swing as the standard GUI library.** A rich set is offered by JavaFX for creating cutting-edge, visually attractive[user interfaces (UI).](https://www.geeksforgeeks.org/user-interface-ui/)Unlike Swing, JavaFX is built entirely in Java and offers extensive support for modern UI elements, multimedia, 2D and 3D graphics, and animation. JavaFX is a flexible option for cross-platform development because its programs may be run on PCs, mobile devices, and browsers. JavaFX has generally become the preferred choice for many Java developers when creating next-generation program due of its focus on rich user experiences and innovative design principles.

**Features and Components of JavaFX**

* **Modern UI Components:** A wide range of contemporary UI components, including as tables, charts, trees, and more, are typically provided by JavaFX, enabling the development of visually appealing and feature-rich user interfaces.
* **Hardware Acceleration:** JavaFX can also produce fluid animations, seamless transitions, and high-performance visual effects by rendering visuals via hardware acceleration.
* **Multimedia Support:**For applications that basically contains rich media content, JavaFX often provides integrated support for multimedia components like audio, video, and 3D graphics.
* **Cross-Platform Deployment:**Due to JavaFX's support for the[Java Virtual Machine (JVM)](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/) and integration with web technologies like WebView, applications can be used on a variety of platforms, including desktop, online, and mobile.
* **Highly Customizable:** JavaFX components are highly customizable, with the support for[CSS](https://www.geeksforgeeks.org/css-introduction/)styling and FXML for defining UI layouts that simply enables the developers to create unique and stylish interfaces.
* **FXML for UI Design:** JavaFX supports **FXML**, an[XML](https://www.geeksforgeeks.org/xml-basics/)-based markup language for defining UI layouts, separating the UI design from application logic, and promoting a more declarative approach to UI development.

**Java AWT**

import java.awt.\*;

class FrameDemo

{

public static void main(String args[])

{

Button b1,b2;

TextField t1,t2;

Label l1,l2;

//create object for Frame

Frame f=new Frame();

//set title

f.setTitle("My III SEM CSE-A");

f.setLayout(new FlowLayout());

f.setBounds(400,400,400,400);

//create objects

l1=new Label("Name");

l2=new Label("Dept");

t1=new TextField(20);

t2=new TextField(20);

b1=new Button("SUBMIT");

b2=new Button("Reset");

//add componets to Frame using add()

f.add(l1);//label added

f.add(t1);

f.add(l2);

f.add(t2);

f.add(b1);

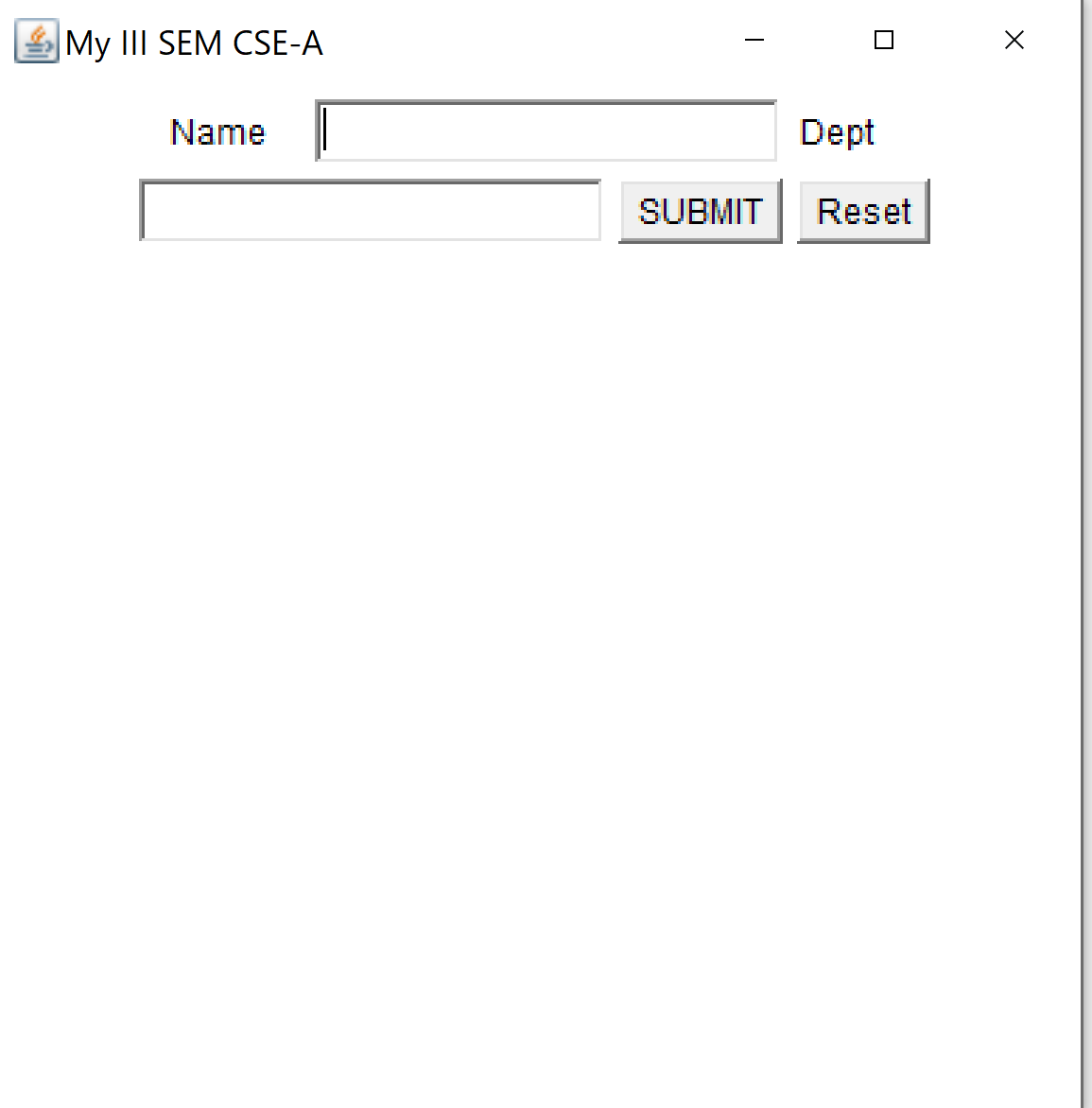
f.add(b2);

//visible frame to user by setVisible(true);

f.setVisible(true);

}//main

}//class



**Java Swing**

import javax.swing.\*;

import java.awt.FlowLayout;

class SwingDemo

{

public static void main(String args[])

{

JButton b1,b2;

JTextField t1,t2;

JLabel l1,l2;

//create object for Frame

JFrame f=new JFrame();

//set title

f.setTitle("My III SEM CSE-A");

f.setLayout(new FlowLayout());

//create objects

l1=new JLabel("Name");

l2=new JLabel("Dept");

t1=new JTextField(20);

t2=new JTextField(20);

b1=new JButton("SUBMIT");

b2=new JButton("Reset");

//add componets to Frame using add()

f.add(l1);//label added

f.add(t1);

f.add(l2);

f.add(t2);

f.add(b1);

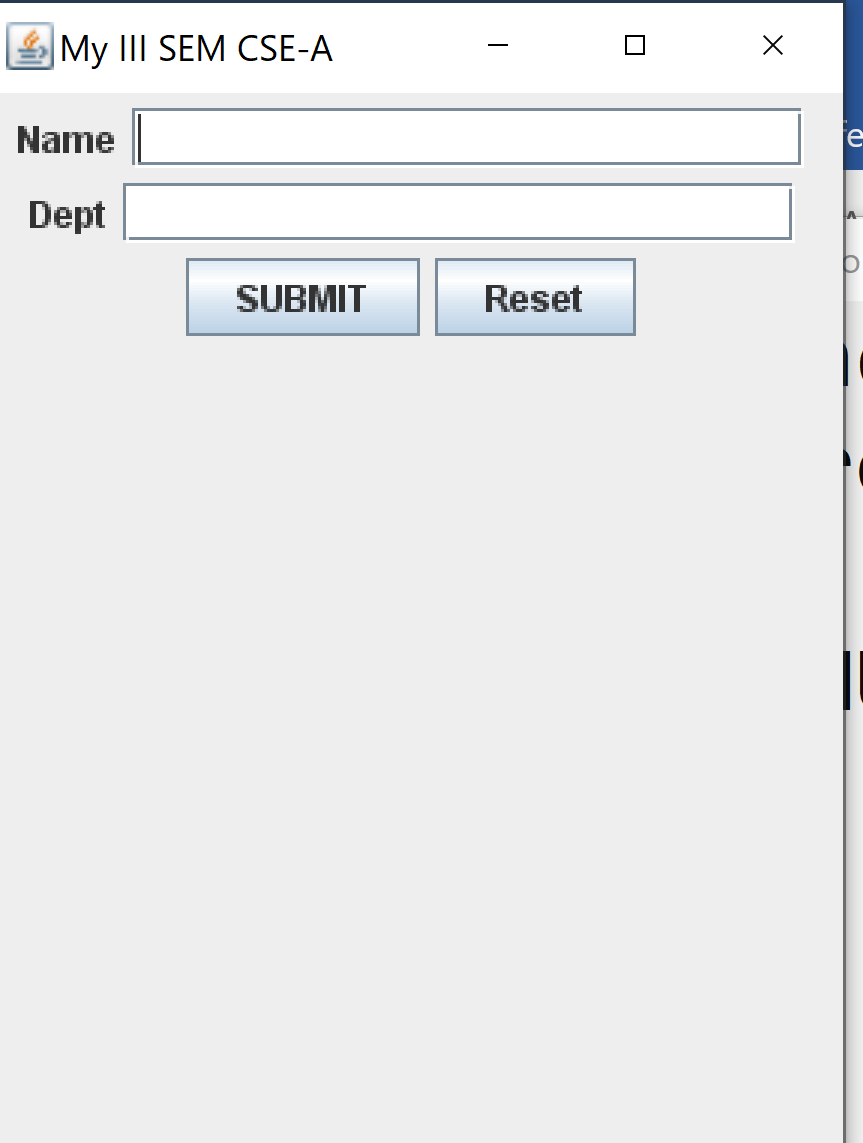
f.add(b2);

//visible frame to user by setVisible(true);

f.setVisible(true);

}//main

}//class



**JavaFX**

Java

**import** **javafx.application.Application**;

**import** **javafx.scene.Scene**;

**import** **javafx.scene.control.Label**;

**import** **javafx.stage.Stage**;

**public** **class** **HelloWorldJavaFX** **extends** Application {

@Override

**public** void start(Stage stage) {

Label label = **new** Label("Hello, World!");

Scene scene = **new** Scene(label, 300, 100);

stage.setTitle("Hello, World (JavaFX)");

stage.setScene(scene);

stage.show();

}

**public** **static** void main(String[] args) {

launch(args);

}

}

**Comparison Table: Java AWT vs Java Swing vs Java FX**

| **Aspect** | **Java AWT** | **Java Swing** | **JavaFX** |
| --- | --- | --- | --- |
| **Performance** | Lightweight, minimal overhead | Optimized for complex UIs, efficient rendering | Hardware-accelerated, excellent performance. |
| **Richness of UI Components** | Basic set of UI components | Rich library with advanced components | Modern and extensive UI component set |
| **Ease of Use and Learning Curve** | Simple and straightforward | More extensive, may have a steeper learning curve. | Modern UI components, may be more complex. |
| **Compatibility with Modern Java** | Compatible, lacks advanced features. | Widely compatible, well-supported | Seamlessly integrated with modern Java. |
| **Community Support and Maintenance** | Large community, part of Java platform | Strong community, actively maintained | Growing community, actively maintained. |
| **Platform Compatibility** | Primarily for desktop, limited web/mobile support | Desktop-focused, limited web/mobile support | Cross-platform (desktop, web, mobile) |

Note:Layouts and event handling classes listeners are common for AWT,Swing and JavaFx

The GUI in [**Java**](https://www.tutorialspoint.com/java/index.htm)processes the interactions with users via **mouse**, **keyboard,**and various user controls such as **buttons**, **checkboxes**, **text fields**, etc., as events. These events are to be handled properly to implement Java as an **Event-Driven Programming.**

## What is Event Handling?

[**Event handling**](https://www.tutorialspoint.com/awt/awt_event_handling.htm) refers to the mechanism that controls events and determines the actions taken when an event occurs. This mechanism includes code known as an event handler, which is executed in response to an event.

## Components in Event Handling

The following are the three main components of event handling in Java:

* [Events](https://www.tutorialspoint.com/what-is-an-event-handling-and-describe-the-components-in-event-handling-in-java#events)
* [Event Sources](https://www.tutorialspoint.com/what-is-an-event-handling-and-describe-the-components-in-event-handling-in-java#sources)
* [Event Listeners/Handlers](https://www.tutorialspoint.com/what-is-an-event-handling-and-describe-the-components-in-event-handling-in-java#listeners)

## Events

The [**events**](https://www.tutorialspoint.com/awt/awt_event_classes.htm)are defined as an object that describes a change in the state of a source object. Java defines a number of such Event Classes inside**java.awt.event package**.  
Some of the events are as follows:

* [ActionEvent:](https://www.tutorialspoint.com/awt/awt_action_event.htm) Occurs when a button is clicked or an action is performed.
* [MouseEvent:](https://www.tutorialspoint.com/awt/awt_mouse_event.htm) Triggered by mouse actions (click, drag, move).
* [KeyEvent:](https://www.tutorialspoint.com/awt/awt_key_event.htm)Generated when a key is pressed, released, or typed.
* **FocusEvent:** Occurs when a component gains or loses focus.
* **ItemEvent:** Fired when an item is selected/deselected (e.g., checkbox).

## Event Sources

A **source**is an [**object**](https://www.tutorialspoint.com/java/lang/java_lang_object.htm)that generates **an event**. An event generation occurs when an **internal state** of that object **changes in some way**. A source must register listeners in order for the listeners to receive notifications about a specific type of event.  
Some of the event sources are as follows:

* **Button:** Generates an ActionEvent when clicked.
* **CheckBox:** Generates an ItemEvent on selection change.
* **List:**Fire ActionEvent or ItemEvent on item selection.
* **Choice:** Triggers ItemEvent when an option is chosen.
* **Window:**Shows WindowEvent on close/minimize/maximize.

## Event Listeners

A [**listener**](https://www.tutorialspoint.com/awt/awt_event_listeners.htm)is an object that is**notified when an event occurs**. A Listener has two major requirements: it should be registered to one or more source objects to receive event notifications, and it must implement methods to receive and process those notifications. Java has defined a set of interfaces for receiving and processing the events under the **java.awt.event package**.

Some of the listeners are as follows:

* [ActionListener:](https://www.tutorialspoint.com/awt/awt_action_listener.htm) Handles button clicks like **actionPerformed()**.
* [MouseListener:](https://www.tutorialspoint.com/awt/awt_mouse_listener.htm) Handles mouse events like **mouseClicked()** and **mousePressed()**.
* [ItemListener:](https://www.tutorialspoint.com/awt/awt_item_listener.htm)Manages ItemEvent like **checkbox**and **radio button** toggles.
* [KeyListener:](https://www.tutorialspoint.com/awt/awt_key_listener.htm) Handles keyboard input like **keyPressed()** and **keyReleased()**.
* [WindowListener:](https://www.tutorialspoint.com/awt/awt_window_listener.htm) Handles window operations like **windowOpened()** and**windowClosed()**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |