Swing

Introduction

Swing in Java is a part of Java's Foundation Classes (JFC) that provides a rich set of GUI (Graphical User Interface) components for building desktop applications. Swing is built on top of the Abstract Window Toolkit (AWT), but unlike AWT, which is platform-dependent, Swing is platform-independent and provides more sophisticated and flexible components.

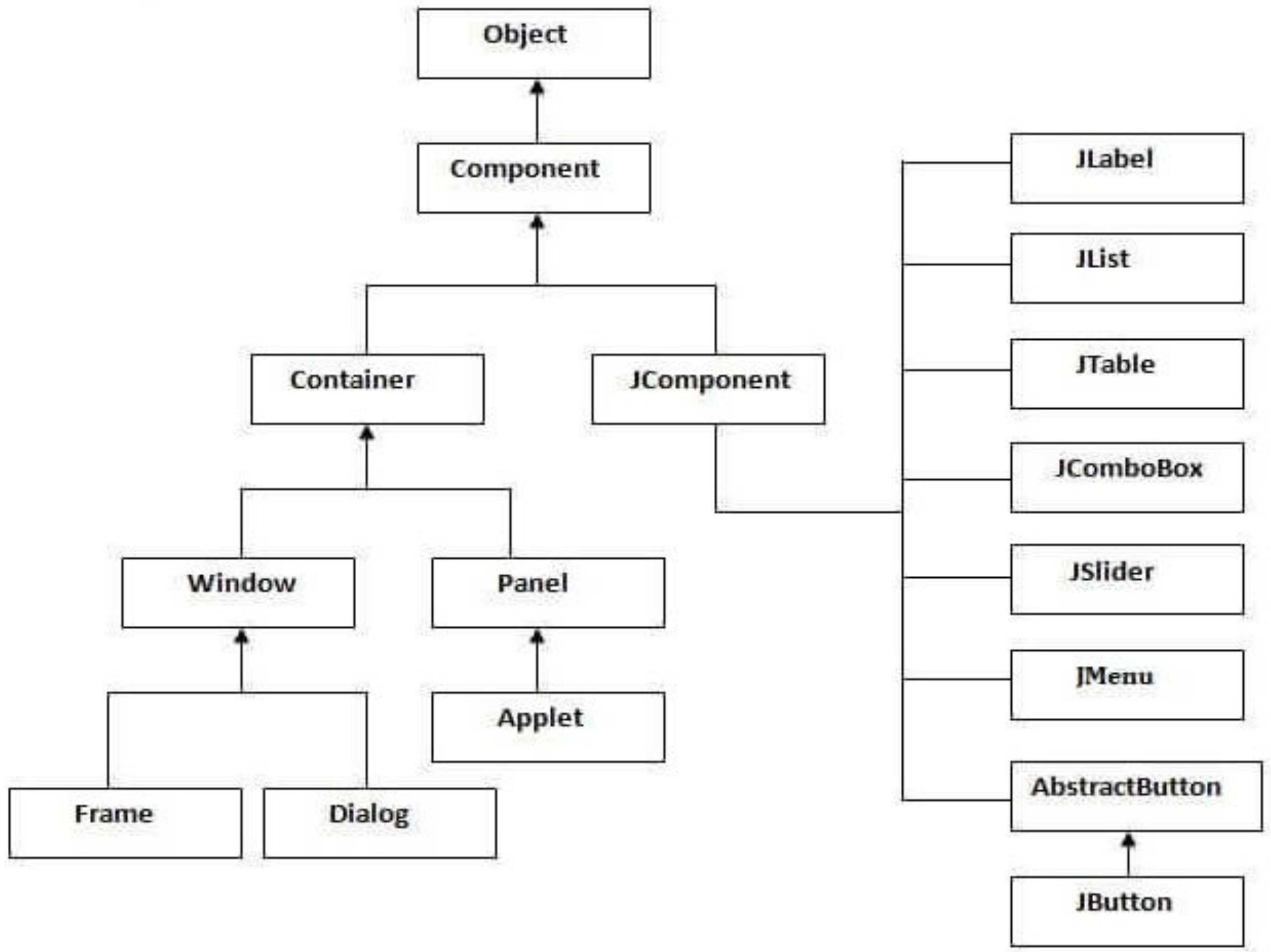
Features of Swing

Lightweight Components: Swing components are lightweight, meaning they are not dependent on the native operating system for rendering. This makes them portable across different platforms.

Pluggable Look and Feel: Swing allows you to change the look and feel (L&F) of the application. For example, you can make your application look like it belongs to Windows, macOS, or a custom L&F, regardless of the actual platform.

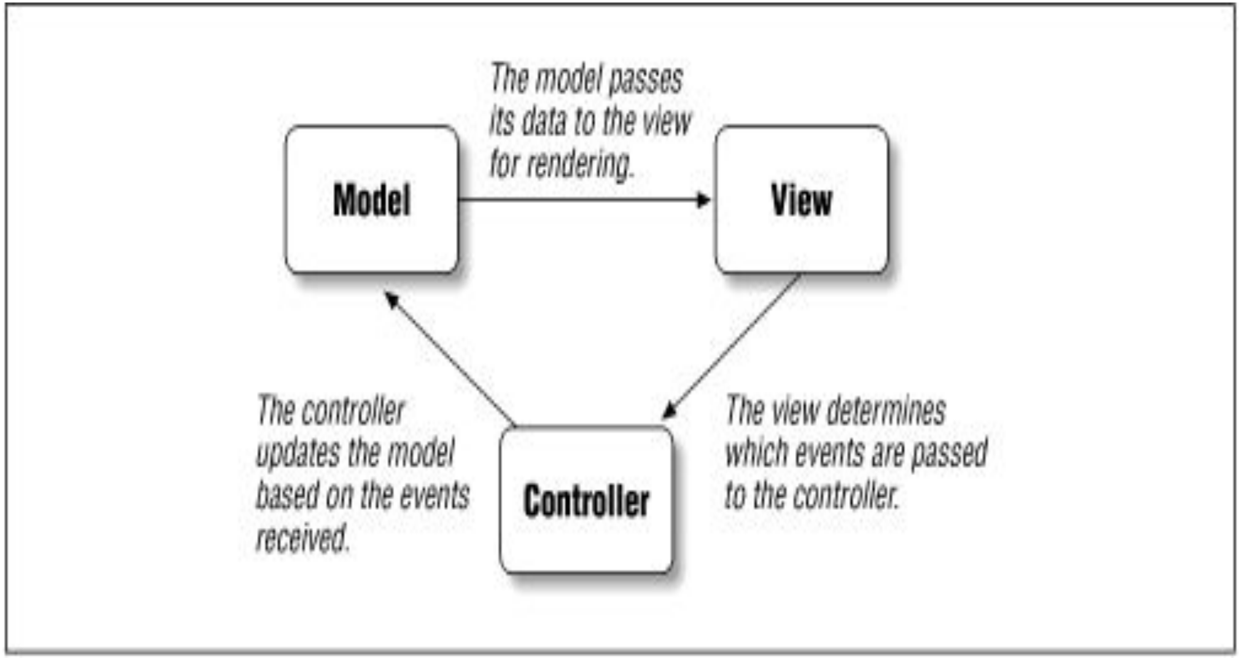
MVC Architecture: Swing components follow the Model-View-Controller (MVC) architecture. This separation of data (model), view (GUI), and controller (interaction logic) allows for better management of the application's interface and logic.

Event-Driven Programming: Swing is event-driven, meaning the program responds to events such as button clicks, mouse movements, and keyboard actions.


Object
Component
Container
JComponent
Window
Panel
Frame
Dialog
Applet
JLabel
JList
JTable
JComboBox
JSlider
JMenu
AbstractButton
JButton

Swing MVC Architecture

In Swing, the **Model-View-Controller (MVC)** architecture separates the internal representation of information (the **Model**) from how information is presented to or accepted from the user (the **View**) and the interaction that controls these processes (the **Controller**). This separation of concerns makes it easier to manage complex user interfaces and allows different components of the application to be developed independently.


The model passes its data to the view for rendering.
Model
View
The controller updates the model based on the events received.
Controller
The view determines which events are passed to the controller.

MVC

Model:

The Model represents the data or the state of the component. It holds all the information and logic related to the component.

View:

The View is responsible for how the data is displayed to the user. It is the visual part of the component.

Controller:

The Controller handles the user interactions, such as button clicks or text input, and updates the Model accordingly.

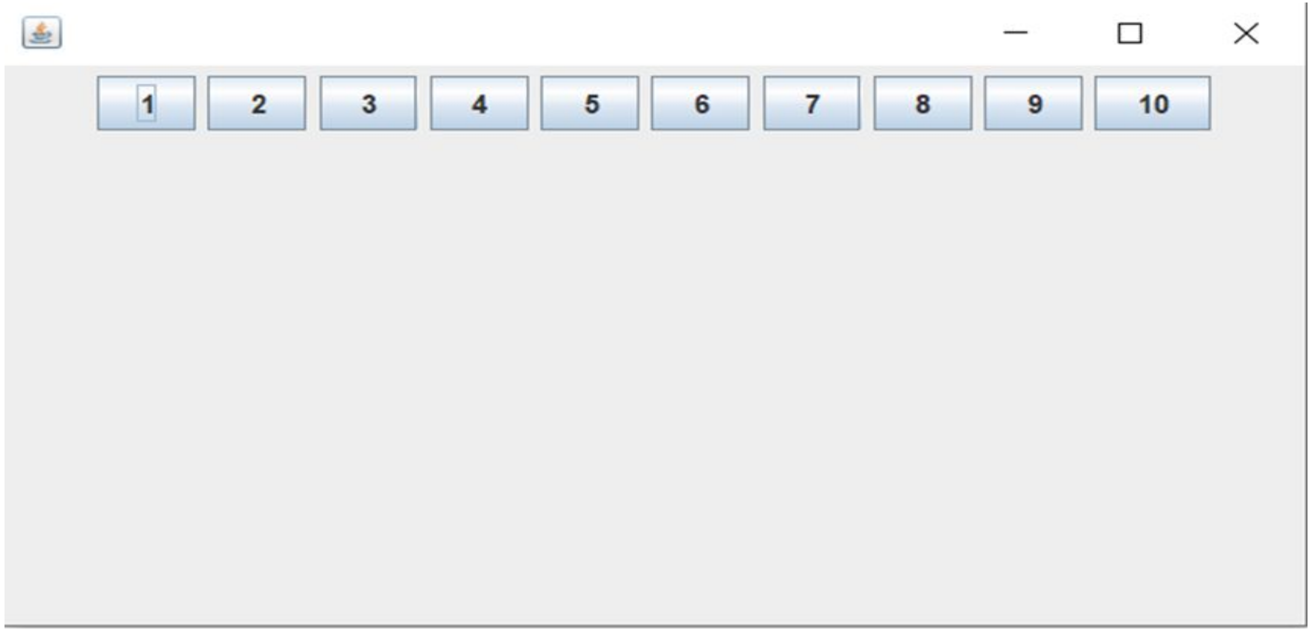
Layout Manager

- A **layout manager** is used to arrange components (like buttons, labels, text fields, etc.) within a container (such as a frame or panel).
- AWT provides several layout managers, each with its own strategy for arranging components.
- **Types of Layouts:**

1. FlowLayout
2. BorderLayout
3. GridLayout
4. CardLayout

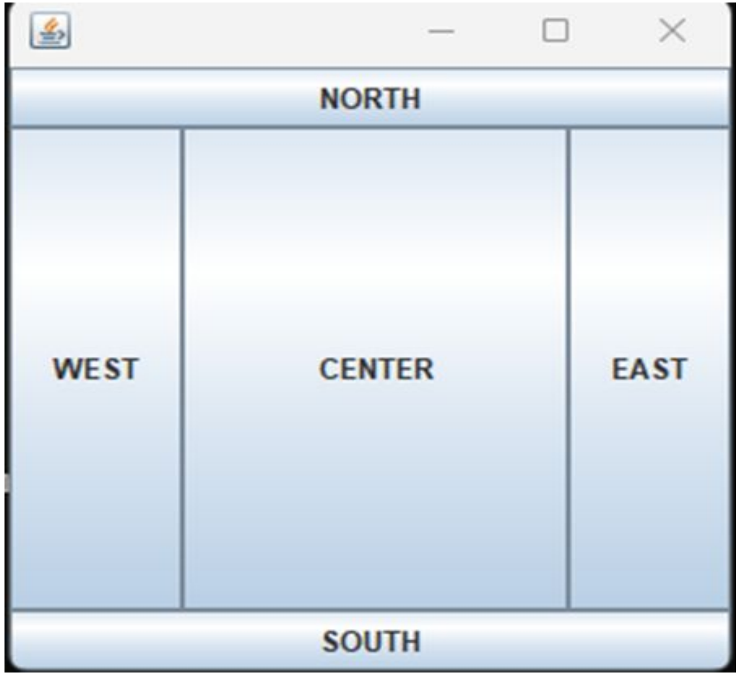
FlowLayout

Arranges components in a row, and if the row is full, it moves to the next row.



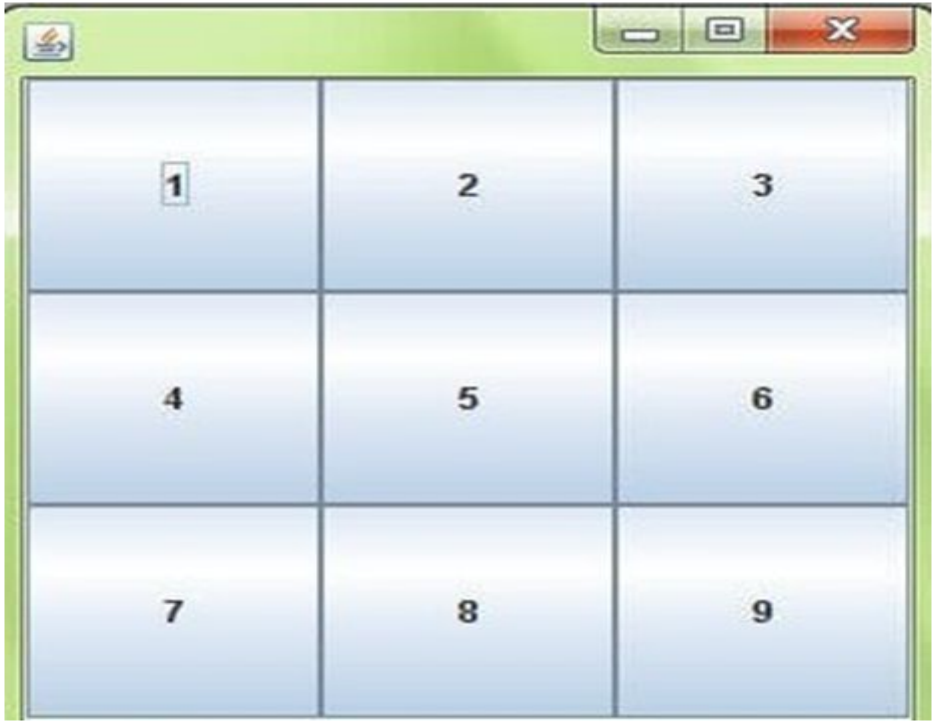
BorderLayout

Divides the container into five regions: **NORTH**, **SOUTH**, **EAST**, **WEST**, and **CENTER**. Each region can contain only one component.



GridLayout

Arranges components in a grid of rows and columns



CardLayout

Allows switching between different panels (cards). Only one component (card) is visible at a time.

JComponent

The **JComponent** class in Java is a fundamental part of the **Swing** library, which provides a richer set of components for building graphical user interfaces (GUIs) compared to AWT.

It is an abstract class that serves as the base class for all Swing components with visual representation.

Subclasses of JComponent

1. **JButton**: A button that can trigger an action when clicked.
2. **JLabel**: A component that displays text or an image.
3. **JTextField**: A single-line text input component.
4. **JTextArea**: A multi-line text area for text input.
5. **JPanel**: A general-purpose container used to group components.
6. **JTable**: A component for displaying tabular data.
7. **JComboBox**: A drop-down menu.

Methods:

- **void setBorder(Border border)**: Sets the border for the component.
- **void setToolTipText(String text)**: Sets a tooltip for the component.
- **void repaint():** Repaints the component.
- **void revalidate():** Requests that the component layout be redone.
- **void setOpaque(boolean isOpaque):** Defines whether the component is opaque.
- **void setPreferredSize(Dimension d):** Sets the preferred size of the component.
- **void addMouseListener(MouseListener l):** Adds a mouse listener to the component.

JFrame

The **JFrame** class in Java is a part of the **Swing** library and is used to create a window that contains other GUI components like buttons, labels, text fields, etc. It represents a **top-level window** with a title, a border, and a content area where components can be added.

Constructors:

- JFrame(): Creates a frame without a title.
- JFrame(String title): Creates a frame with the specified title.

Methods:

void setTitle(String title): Sets the title of the frame.

void setSize(int width, int height): Sets the size of the frame.

void setVisible(boolean visible): Shows or hides the frame.

void setDefaultCloseOperation(int operation): Specifies what happens when the user closes the frame.

void setResizable(boolean resizable): Sets whether the frame is resizable by the user.

Container getContentPane(): Returns the content pane for adding components.

void setJMenuBar(JMenuBar menuBar): Adds a menu bar to the frame.

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); to exit the application when the frame is closed.