Introduction to Swing

Swing API is a set of extensible GUI Components to ease the developer's life to create JAVA based Front End/GUI Applications. It is build on top of AWT API and acts as a replacement of AWT API, since it has almost every control corresponding to AWT controls. Swing component follows a Model-View-Controller architecture to fulfill the following criteria:-

- A single API is to be sufficient to support multiple look and feel.
- API is to be model driven so that the highest level API is not required to have data.
- API is to use the Java Bean model so that Builder Tools and IDE can provide better services to the developers for use.

MVC Architecture

Swing API architecture follows loosely based MVC architecture in the following manner.

- Model represents component's data.
- View represents visual representation of the component's data.
- Controller takes the input from the user on the view and reflects the changes in Component's data.
- Swing component has Model as a seperate element, while the View and Controller part are clubbed in the User Interface elements. Because of which, Swing has a pluggable look-and-feel architecture.

Swing Features

- **Light Weight** − Swing components are independent of native Operating System's API as Swing API controls are rendered mostly using pure JAVA code instead of underlying operating system calls.
- **Rich Controls** − Swing provides a rich set of advanced controls like Tree, TabbedPane, slider, colorpicker, and table controls.
- **Highly Customizable** − Swing controls can be customized in a very easy way as visual apperance is independent of internal representation.
- **Pluggable look-and-feel** − SWING based GUI Application look and feel can be changed at run-time, based on available values.

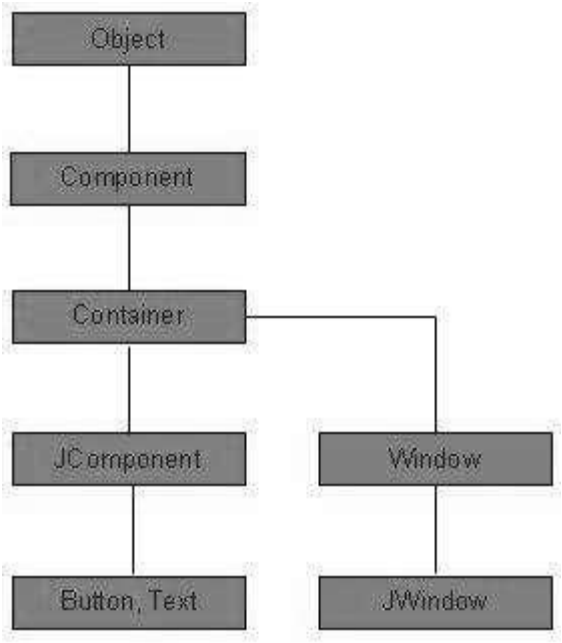
SWING - Controls

Every user interface considers the following three main aspects −

- **UI Elements** − These are the core visual elements the user eventually sees and interacts with. GWT provides a huge list of widely used and common elements varying from basic to complex, which we will cover in this tutorial.
- **Layouts** − They define how UI elements should be organized on the screen and provide a final look and feel to the GUI (Graphical User Interface). This part will be covered in the Layout chapter.

Reference:- Herbert Schildt - Java_ The Complete Reference, Eleventh Edition 11(2019, McGraw-Hill Education)

- **Behavior** − These are the events which occur when the user interacts with UI elements. This part will be covered in the Event Handling chapter.



Every SWING controls inherits properties from the following Component class hiearchy.

S.No.	Class & Description
1	Component A Component is the abstract base class for the non menu user-interface controls of SWING. Component represents an object with graphical representation
2	Container A Container is a component that can contain other SWING components
3	JComponent A JComponent is a base class for all SWING UI components. In order to use a SWING component that inherits from JComponent, the component must be in a containment hierarchy whose root is a top-level SWING container

SWING UI Elements

Following is the list of commonly used controls while designing GUI using SWING.

Reference:- Herbert Schildt - Java_ The Complete Reference, Eleventh Edition 11(2019, McGraw-Hill Education)

S.No.	Class & Description
1	JLabel A JLabel object is a component for placing text in a container.
2	JButton This class creates a labeled button.
3	JColorChooser A JColorChooser provides a pane of controls designed to allow a user to manipulate and select a color.
4	JCheck Box A JCheckBox is a graphical component that can be in either an **on** (true) or **off** (false) state.
5	JRadioButton The JRadioButton class is a graphical component that can be in either an **on** (true) or **off** (false) state. in a group.
6	JList A JList component presents the user with a scrolling list of text items.
7	JComboBox A JComboBox component presents the user with a to show up menu of choices.
8	JTextField A JTextField object is a text component that allows for the editing of a single line of text.
9	JPasswordField A JPasswordField object is a text component specialized for password entry.
10	JTextArea A JTextArea object is a text component that allows editing of a multiple lines of text.

Reference:- Herbert Schildt - Java_ The Complete Reference, Eleventh Edition 11(2019, McGraw-Hill Education)

11	ImageIcon A ImageIcon control is an implementation of the Icon interface that paints Icons from Images
12	JScrollbar A Scrollbar control represents a scroll bar component in order to enable the user to select from range of values.
13	JOptionPane JOptionPane provides set of standard dialog boxes that prompt users for a value or informs them of something.
14	JFileChooser A JFileChooser control represents a dialog window from which the user can select a file.
15	JProgressBar As the task progresses towards completion, the progress bar displays the task's percentage of completion.
16	JSlider A JSlider lets the user graphically select a value by sliding a knob within a bounded interval.
17	JSpinner A JSpinner is a single line input field that lets the user select a number or an object value from an ordered sequence.

The Origins of Swing

Swing was a response to deficiencies present in Java's original GUI subsystem: the Abstract Window Toolkit. The AWT defines a basic set of controls, windows, and dialog boxes that support a usable, but limited graphical interface. One reason for the limited nature of the AWT is that it translates its various visual components into their corresponding, platform-specific equivalents, or peers. This means that the look and feel of a component is defined by the platform, not by Java. Because the AWT components use native code resources, they are referred to as heavyweight.

AWT led to several problems:

1. Due to variations between operating systems, a component might look, or even act, differently on different platforms. This potential variability threatened the overarching philosophy of Java: write once, run anywhere.
2. The look and feel of each component was fixed (because it is defined by the platform) and could not be (easily) changed.
3. The use of heavyweight components caused some frustrating restrictions.

Swing Is Built on the AWT

Swing is built on the foundation of the AWT. This is why the AWT is still a crucial part of Java. Swing also uses the same event handling mechanism as the AWT.

Two Key Swing Features

1. Swing Components Are Lightweight

They are written entirely in Java and do not map directly to platform-specific peers. Thus, lightweight components are more efficient and more flexible. Lightweight components do not translate into native peers. As a result, each component will work in a consistent manner across all platforms.

2. Swing Supports a Pluggable Look and Feel

Swing supports a *pluggable look and feel* (PLAF). Because each Swing component is rendered by Java code rather than by native peers.

This fact means that it is possible to separate the look and feel of a component from the logic of the component, and this is what Swing does. Separating out the look and feel provides a significant advantage: it becomes possible to change the way that a component is rendered without affecting any of its other aspects.

The MVC Connection

In general, a visual component is a composite of three distinct aspects:

• The way that the component looks when rendered on the screen
• The way that the component reacts to the user
• The state information associated with the component

In MVC terminology,

Model corresponds to the state information associated with the component. For example, in the case of a check box, the model contains a field that indicates if the box is checked or unchecked.

View determines how the component is displayed on the screen,

Controller determines how the component reacts to the user. For example, when the user clicks a check box, the controller reacts by changing the model to reflect the user's choice (checked or unchecked).

Swing uses a modified version of MVC that combines the view and the controller into a single logical entity called the *UI delegate*. For this reason, Swing's approach is called either the ***Model-Delegate*** **architecture or the** ***Separable Model*** **architecture.**

Because the view (look) and controller (feel) are separate from the model, the look and feel can be changed without affecting how the component is used within a program.

Reference:- Herbert Schildt - Java_ The Complete Reference, Eleventh Edition 11(2019, McGraw-Hill Education)