Swing

Swing

Swing is a GUI (Graphical User Interface) toolkit for Java, providing a comprehensive set of components for building desktop applications.

It is part of the Java Foundation Classes (JFC) and has been the primary GUI toolkit for Java developers since its introduction in Java 2 (JDK 1.2).

Java Swing provides platform-independent and lightweight components. The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Need for Swing

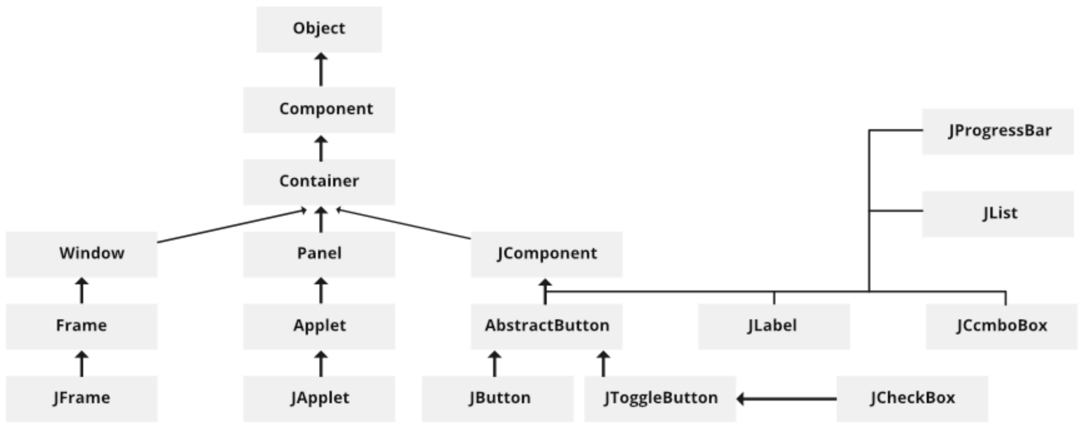
1. **Platform Independence:** Swing is part of the Java Foundation Classes (JFC) and provides a set of lightweight components that can be used to create platform-independent GUI applications. This means that applications developed using Swing can run on any platform that supports Java, without requiring modifications.
2. **Rich Component Library:** Swing offers a wide range of components such as buttons, labels, text fields, combo boxes, lists, tables, and more. These components can be easily customized and combined to create sophisticated user interfaces.
3. **Event-Driven Programming:** Swing follows an event-driven programming model, where user interactions (e.g., button clicks, mouse movements) trigger events that are handled by event listeners. This allows developers to create interactive applications that respond to user input in real-time.
4. **Layout Management:** Swing provides layout managers that help arrange components within a container according to specific layout rules. This makes it easier to create GUIs that adapt to different screen sizes and resolutions.
5. **Customization and Pluggable Look and Feel:** Swing allows developers to customize the appearance and behavior of components to suit the requirements of their application. Additionally, Swing supports different look and feel options, allowing developers to choose the visual style that matches the platform or their design preferences.
6. **Integration with Other Java Technologies:** Swing seamlessly integrates with other Java technologies such as AWT (Abstract Window Toolkit), JavaFX, and JavaBeans.

This allows developers to leverage the strengths of these technologies while building GUI applications.

Difference between AWT and Swing

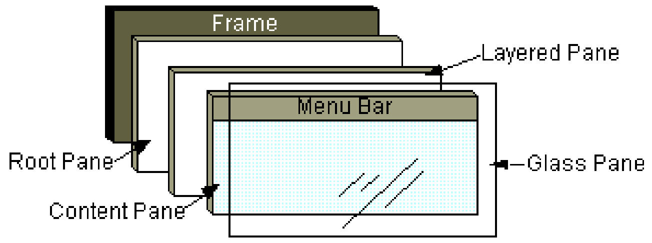
S.NO	**AWT**	**Swing**
1.	Java AWT is an API to develop GUI applications in Java	Swing is a part of Java Foundation Classes and is used to create various applications.
2.	The components of Java AWT are heavy weighted.	The components of Java Swing are light weighted.
3.	Java AWT has comparatively less functionality as compared to Swing.	Java Swing has more functionality as compared to AWT.
4.	The execution time of AWT is more than Swing.	The execution time of Swing is less than AWT.
5.	The components of Java AWT are platform dependent.	The components of Java Swing are platform independent.
6.	MVC pattern is not supported by AWT.	MVC pattern is supported by Swing.
7.	AWT provides comparatively less powerful components.	Swing provides more powerful components.
8	AWT components require java.awt package	Swing components requires javax.swing package
9	AWT is a thin layer of code on top of the operating system.	Swing is much larger swing also has very much richer functionality.
10	AWT stands for Abstract windows toolkit .	Swing is also called as JFC(java Foundation classes). It is part of oracle's JFC.
11	Using AWT , you have to implement a lot of things yourself .	Swing has them built in.

Components Hierarchy



Panes

In general, you don't directly create a `JRootPane` object. Instead, you get a `JRootPane` (whether you want it or not!) when you instantiate `JInternalFrame` or one of the top-level Swing containers, such as `JApplet`, `JDialog`, and `JFrame`



The glass pane

Hidden, by default. If you make the glass pane visible, then it's like a sheet of glass over all the other parts of the root pane. It's completely transparent unless you implement the glass pane's `paintComponent` method so that it does something, and it can intercept input events for the root pane. The glass pane is useful when you want to be able to catch events or paint over an area that already contains one or more components. For example, you can deactivate mouse events for a multi-component region by having the glass pane intercept the events. Or you can display an image over multiple components using the glass pane.

```
Component glassPane = jframe.getGlassPane();
```

The layered pane

Serves to position its contents, which consist of the content pane and the optional menu bar. Can also hold other components in a specified Z order.

```
LayeredPane layeredPane = frame.getLayeredPane();
```

The content pane

The container of the root pane's visible components, excluding the menu bar.

```
Container contentPane = jframe.getContentPane();
```

The optional menu bar

The home for the root pane's container's menus. If the container has a menu bar, you generally use the container's `setJMenuBar` method to put the menu bar in the appropriate place.

Swing Components

Component	Description
`JLabel`	A non-interactive component used to display text, icons, or images. It is commonly used to provide descriptions or titles in a GUI.
`JTextField`	An input component that allows users to enter and edit a single line of text. It's useful for accepting user input, such as names, addresses, or search queries.
`JPasswordField`	Similar to `JTextField`, but used specifically for entering passwords. The text entered is typically masked (e.g., with asterisks) for security purposes.
`JTextArea`	A multi-line text area component that allows users to enter and edit multiple lines of text. It's suitable for larger blocks of text, such as comments or messages.
`JButton`	An interactive component that triggers an action when clicked. It's commonly used to perform tasks or submit forms in a GUI.
`JCheckBox`	A component that represents a checkbox, allowing users to select or deselect an option. It's useful for presenting multiple choices where more than one can be selected.
`JRadioButton`	A component that represents a radio button, allowing users to select one option from a group of mutually exclusive choices.
`JComboBox`	A component that provides a drop-down list of items from which users can select one option. It's useful for presenting a list of choices in a compact format.
`JList`	A component that displays a list of items, either vertically or horizontally. Users can select one or more items from the list.

Swing Components (Syntax & Methods)

Component	Basic Syntax	Important Methods
JLabel	`JLabel label = new JLabel("Text");`	`setText(String text)`, `getText()`, `setIcon(Icon icon)`
JTextField	`JTextField textField = new JTextField(20);`	`setText(String text)`, `getText()`, `addActionListener(ActionListener l)`
JPasswordField	`JPasswordField passwordField = new JPasswordField(20);`	`setText(String text)`, `getText()`, `addActionListener(ActionListener l)`
JTextArea	`JTextArea textArea = new JTextArea(5, 20);`	`setText(String text)`, `getText()`, `append(String text)`
JButton	`JButton button = new JButton("Text");`	`setText(String text)`, `getText()`, `addActionListener(ActionListener l)`
JCheckBox	`JCheckBox checkBox = new JCheckBox("Text");`	`setText(String text)`, `getText()`, `isSelected()`, `setSelected(boolean b)`
JRadioButton	`JRadioButton radioButton = new JRadioButton("Text");`	`setText(String text)`, `getText()`, `isSelected()`, `setSelected(boolean b)`
JComboBox	`JComboBox comboBox = new JComboBox(new String[]{"Item1", "Item2"});`	`addItem(Object item)`, `removeItem(Object item)`, `getSelectedItem()`
JList	`JList list = new JList(new String[] {"Item1", "Item2"});`	`setListData(Object[] listData)`, `getSelectedValue()`