Swing in java

Swing is the primary Java GUI toolkit. It is part of Oracle's Java Foundation Classes (JFC) — an API for providing a graphical user interface (GUI) for Java programs.

Swing was developed to provide a more sophisticated set of GUI components than the earlier Abstract Window Toolkit. Swing provides a native look and feel that emulates the look and feel of several platforms, and also supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform. It has more powerful and flexible components than AWT. In addition to familiar components such as buttons, check box and labels, Swing provides several advanced components such as tabbed panel, scroll panes, trees, tables and lists.

Unlike AWT components, Swing components are not implemented by platform-specific code. Instead they are written entirely in Java and therefore are platform-independent. The term "lightweight" is used to describe such an element.

Swing is built on top of AWT and is entirely written in Java, using AWT's lightweight component support. In particular, unlike AWT, t he architecture of Swing components makes it easy to customize both their appearance and behavior. Components from AWT and Swing can be mixed, allowing you to add Swing support to existing AWT-based programs. For example, swing components such as JSlider, JButton and JCheckbox could be used in the same program with standard AWT labels, textfields and scrollbars. You could subclass the existing Swing UI, model, or change listener classes without having to

reinvent the entire implementation. Swing also has the ability to replace these objects on-the-fly.

- 100% Java implementation of components
- Pluggable Look & Feel
- Lightweight components
- **Uses MVC Architecture**

  Model represents the data

  View as a visual representation of the data

  Controller takes input and translates it to changes in data
- **Three parts**

  Component set (subclasses of JComponent)

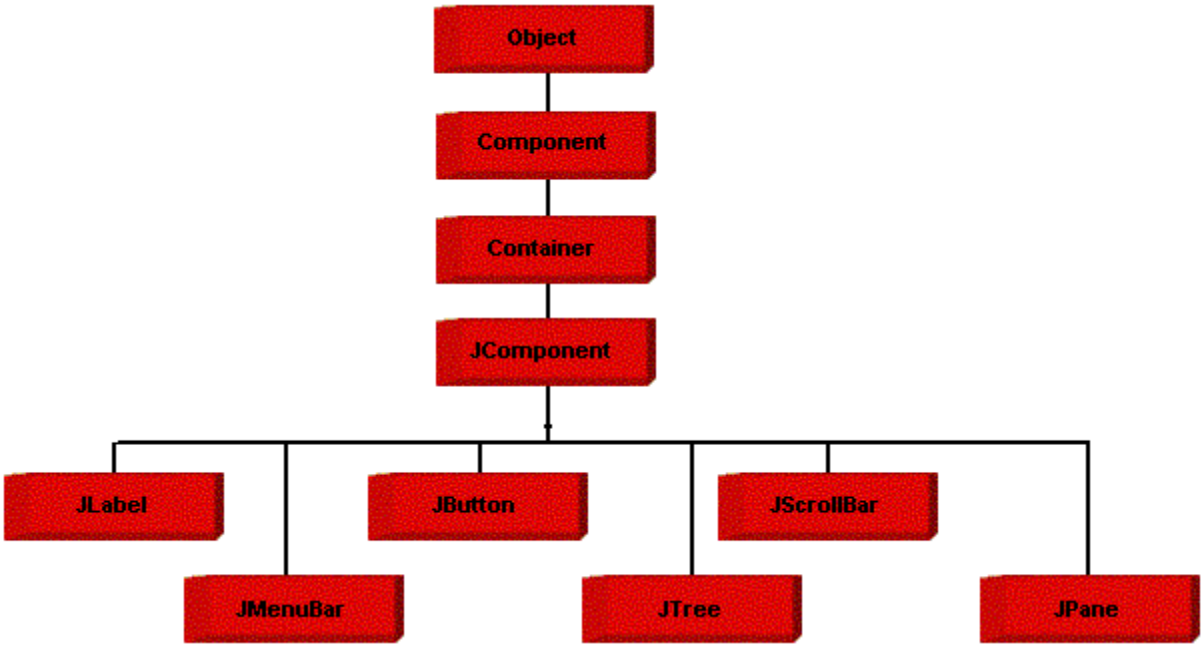
  Support classes

  Interfaces

In Swing, classes that represent GUI components have names beginning with the letter J. Some examples are JButton, JLabel, and JSlider. Altogether there are more than 250 new classes and 75 interfaces in Swing — twice as many as in AWT.

Java Swing class hierarchy

The class JComponent, descended directly from Container, is the root class for most of Swing's user interface components.



- JPanel is Swing's version of the AWT class Panel and uses the same default layout, FlowLayout. JPanel is descended directly from JComponent.
- **JFrame** is Swing's version of Frame and is descended directly from that class. The components added to the frame are referred to as its contents; these are managed by the contentPane. To add a component to a JFrame, we must use its contentPane instead.
- **JInternalFrame** is confined to a visible area of a container it is placed in. It can be iconified , maximized and layered.
- **JWindow** is Swing's version of Window and is descended directly from that class. Like Window, it uses BorderLayout by default.
- JDialog is Swing's version of Dialog and is descended directly from that class. Like Dialog, it uses BorderLayout by default. Like JFrame and JWindow, JDialog contains a rootPane hierarchy including a contentPane, and it allows layered and glass panes. All dialogs are modal, which means the current

thread is blocked until user interaction with it has been completed. JDialog class is intended as the basis for creating custom dialogs; however, some of the most common dialogs are provided through static methods in the class JOptionPane.

JLabel, descended from JComponent, is used to create text labels.

The abstract class AbstractButton extends class JComponent and provides a foundation for a family of button classes, including **JButton**.

JTextField allows editing of a single line of text. New features include the ability to justify the text left, right, or center, and to set the text's font.

JPasswordField (a direct subclass of JTextField) you can suppress the display of input. Each character entered can be replaced by an echo character. This allows confidential input for passwords, for example. By default, the echo character is the asterisk, *.

JTextArea allows editing of multiple lines of text. JTextArea can be used in conjunction with class JScrollPane to achieve scrolling. The underlying JScrollPane can be forced to always or never have either the vertical or horizontal scrollbar;

JButton is a component the user clicks to trigger a specific action.

JRadioButton is similar to JCheckbox, except for the default icon for each class. A set of radio buttons can be associated as a group in which only one button at a time can be selected.

JCheckBox is not a member of a checkbox group. A checkbox can be selected and deselected, and it also displays its current state.

JComboBox is like a drop down box. You can click a drop-down arrow and select an option from a list. For example, when the component has focus,

pressing a key that corresponds to the first character in some entry's name selects that entry. A vertical scrollbar is used for longer lists.

JList provides a scrollable set of items from which one or more may be selected. JList can be populated from an Array or Vector. JList does not support scrolling directly, instead, the list must be associated with a scrollpane. The view port used by the scroll pane can also have a user-defined border. JList actions are handled using ListSelectionListener.

JTabbedPane contains a tab that can have a tool tip and a mnemonic, and it can display both text and an image.

JToolbar contains a number of components whose type is usually some kind of button which can also include separators to group related components within the toolbar.

FlowLayout when used arranges swing components from left to right until there's no more space available. Then it begins a new row below it and moves from left to right again. Each component in a FlowLayout gets as much space as it needs and no more.

BorderLayout places swing components in the North, South, East, West and center of a container. You can add horizontal and vertical gaps between the areas.

GridLayout is a layout manager that lays out a container's components in a rectangular grid. The container is divided into equal-sized rectangles, and one component is placed in each rectangle.

GridBagLayout is a layout manager that lays out a container's components in a grid of cells with each component occupying one or more cells, called its display area. The display area aligns components vertically and horizontally, without requiring that the components be of the same size.

JMenubar can contain several JMenu's. Each of the JMenu's can contain a series of JMenuItem 's that you can select. Swing provides support for pull-down and popup menus.

Scrollable JPopupMenu is a scrollable popup menu that can be used whenever we have so many items in a popup menu that exceeds the screen visible height.

drawRect():

This is the method of the **Graphics** class (The **Graphics** class is used to drawing different-different type of shapes). This method draws the rectangle. It takes some integer value as parameter. This method is written like : **Graphics.**drawRect(x, y, height, width);.

x - This is the variable represents the row no. or the x - coordinate.
y - This is also a variable represents the column no. or the y - coordinate.

drawOval():

This is the method of the **Graphics** class which draws the oval on the frame. This method takes argument same as the drawRect() method. In this method first come the width and then height is specified.

fillRect():

This is the method of the **Graphics** class which is used to fill rectangle with the specified color which is set before using the setColor() method of the **Graphics** class. It also takes argument same as the drawRect() method.

fillOval():

This is also the method of the **Graphics** class which is used to fill the oval with color specified in the setColor() method before. This method also takes argument same as the drawOval() method.

JFrame and JApplet

JFrame and JApplet are top level containers. If you whish to create desktop application, you will use JFrame and if you plan to host your application in browser you will use JApplet.

JApplet - A base class that let's you write code that will run within the context of a browser, like for an interactive web page. This is cool and all but it brings limitations which is the price for it playing nice in the real world. Normally JApplet is used when you want to have your own UI in a web page. I've always wondered why people don't take advantage of applets to store state for a session so no database or cookies are needed.

JComponent - A base class for objects which intend to interact with Swing.

JFrame - Used to represent the stuff a window should have. This includes borders (resizeable y/n?), titlebar (App name or other message), controls (minimize/maximize allowed?), and event handlers for various system events like 'window close' (permit app to exit yet?).

JPanel - Generic class used to gather other elements together. This is more important with working with the visual layout or one of the provided layout managers e.g. gridbaglayout, etc. For example, you have a textbox that is bigger then the area you have reserved. Put the textbox in a scrolling pane and put that pane into a JPanel. Then when you place the JPanel, it will be more manageable in terms of layout.