
SWING**Q. What Are the Java Foundation Classes (JFC)?**

JFC is short for Java Foundation Classes, which encompass a group of features for building graphical user interfaces (GUIs) and adding rich graphics functionality and interactivity to Java applications. It is defined as containing the features shown in the table below.

Features of the Java Foundation Classes:

Feature	Description
Swing GUI Components	Includes everything from buttons to split panes to tables. Many components are capable of sorting, printing, and drag and drop, to name a few of the supported features.
Pluggable Look-and-Feel Support	The look and feel of Swing applications is pluggable, allowing a choice of look and feel. For example, the same program can use either the Java or the Windows look and feel. Additionally, the Java platform supports the GTK+ look and feel, which makes hundreds of existing look and feels available to Swing programs. Many more look-and-feel packages are available from various sources.
Accessibility API	Enables assistive technologies, such as screen readers and Braille displays, to get information from the user interface.
Java 2D API	Enables developers to easily incorporate high-quality 2D graphics, text, and images in applications and applets. Java 2D includes extensive APIs for generating and sending high-quality output to printing devices.
Drag and Drop	Drag and drop is one of the more common metaphors used in graphical interfaces today. The user is allowed to click and "hold" a GUI object, moving it to another window or frame in the desktop with predictable results. The Drag and Drop API allows users to implement droppable elements that transfer information between Java applications and native applications.
Internationalization	Allows developers to build applications that can interact with users worldwide in their own languages and cultural conventions. With the input method framework developers can build applications that accept text in languages that use thousands of different characters, such as Japanese, Chinese, or Korean.

Q. What is swing? Explain features of swing.

Java Swing is a lightweight Java graphical user interface (GUI) widget toolkit that includes a rich set of widgets. It is part of the Java Foundation Classes (JFC) and includes several packages for developing rich desktop applications in Java. Swing includes built-in controls such as trees, image buttons, tabbed

SWING

panes, sliders, toolbars, color choosers, tables, and text areas to display HTTP or rich text format (RTF). Swing components are written entirely in Java and thus are platform-independent.

Swing Features:

Swing provides many new features for those planning to write large-scale applications in Java. Here is an overview of some of the more popular features.

- ✓ **Pluggable Look-and-Feel :**

One of the unique features of Swing is its pluggable look-and-feel (PLAF) architecture, which allows a Swing application to change its entire appearance with one or two lines of code. The most common use of this feature is to give applications a choice between the native platform look-and-feel and a new platform-independent Java look-and-feel (also known as the Metal look-and-feel). Swing is distributed with three look-and-feels: Metal and two look-and-feels that mimic the appearance and behavior of the Windows and Motif (Unix/X) component toolkits. A look-and-feel that mimics the Macintosh platform is available as a separate download. While the Metal and Motif look-and-feels can be freely used.

- ✓ **Lightweight Components**

Most Swing components are lightweight means that components are not dependent on native peers to render themselves. Instead, they use simplified graphics primitives to paint themselves on the screen and can even allow portions to be transparent. Almost all of the Swing components are lightweight; only a few top-level containers are not. This design allows programmers to draw (and redraw) the look-and-feel of their application at runtime,

- ✓ A wide variety of new components, such as tables, trees, sliders, progress bars, internal frames, and text components is introduced in swing
- ✓ Swing components contain support for replacing their insets with an arbitrary number of concentric borders.
- ✓ Swing components can have tooltips placed over them. A tooltip is a textual popup that momentarily appears when the mouse cursor rests inside the component's painting region.
- ✓ You can arbitrarily bind keyboard events to components, defining how they will react to various keystrokes under given conditions.
- ✓ There is additional debugging support for the rendering of your own lightweight Swing components.
- ✓ A wide variety of new components, such as tables, trees, sliders, progress bars, internalframes, and text components.

SWING

Q. What are Swing Packages and Classes

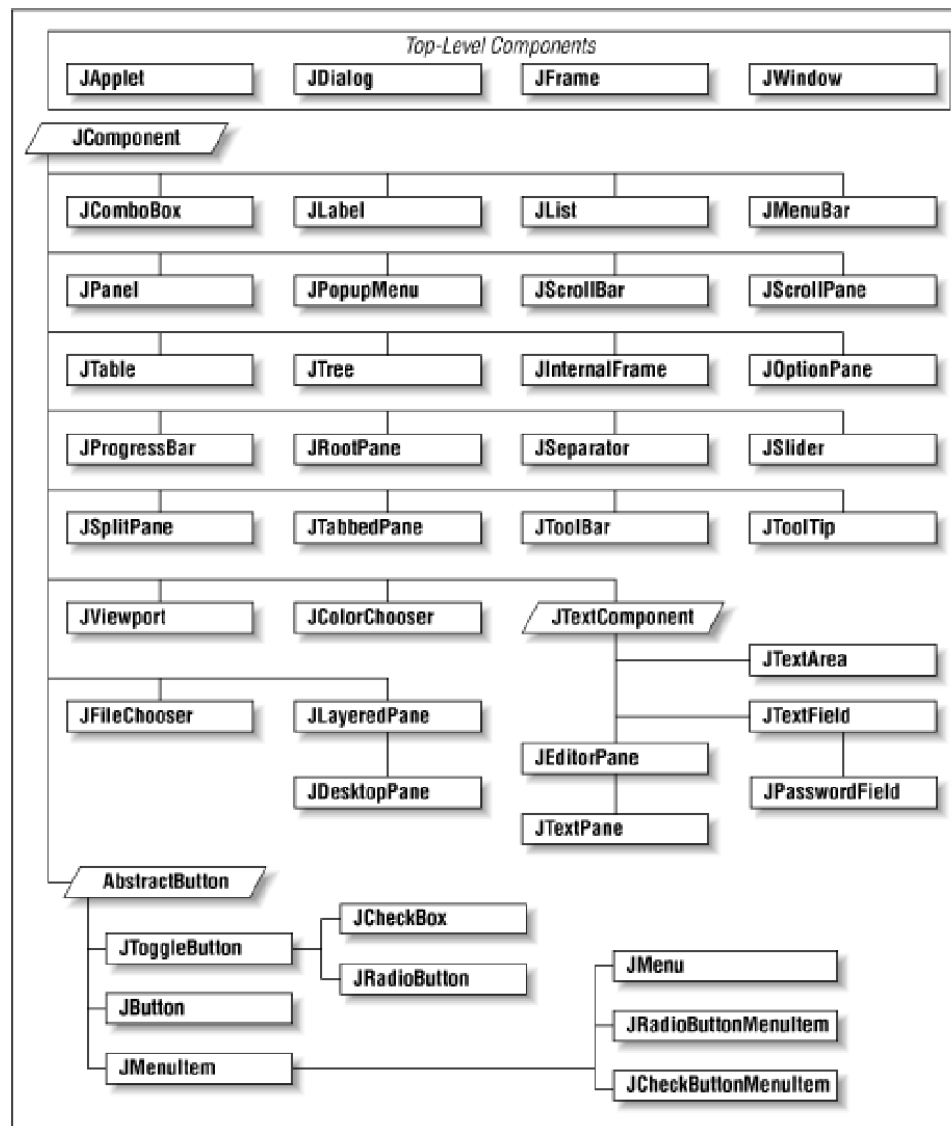
Here is a short description of each package in the Swing libraries.

<code>javax.accessibility</code>	Contains classes and interfaces that can be used to allow <i>assistive technologies</i> to interact with Swing components. Assistive technologies cover a broad range of items, from audible text readers to screen magnification. Although the accessibility classes are technically not part of Swing, they are used extensively throughout the Swing components.
<code>javax.swing</code>	Contains the core Swing components, including most of the model interfaces and support classes.
<code>javax.swing.border</code>	Contains the definitions for the abstract border class as well as eight predefined borders. Borders are not components; instead, they are special graphical elements that Swing treats as properties and places around components in place of their insets. If you wish to create your own border, you can subclass one of the existing borders in this package, or you can code a new one from scratch.
<code>javax.swing.colorchooser</code>	Contains support for the JColorChooser component,
<code>javax.swing.event</code>	Defines several new listeners and events that Swing components use to communicate asynchronous information between classes. To create your own events, you can subclass various events in this package or write your own event class.
<code>javax.swing.filechooser</code>	Contains support for the JFileChooser component,
<code>javax.swing.pending</code> <code>javax.swing.plaf</code>	Contains an assortment of components that aren't ready for prime time, but may be in the future. Defines the unique elements that make up the pluggable look-and-feel for each Swing component. Its various subpackages are devoted to rendering the individual look-and-feels for each component on a platform-by-platform basis. (Concrete implementations of the Windows and Motif L&Fs are in subpackages of com.sun.java.swing.plaf.)
<code>javax.swing.table</code>	Provides models and views for the table component. The table component allows you to arrange various information in a grid-based format with an appearance similar to a spreadsheet. Using the lower-level classes, you can manipulate how tables are viewed and selected, as well as how they display their information in each cell.
<code>javax.swing.text</code>	Provides scores of text-based classes and interfaces supporting a common design known as <i>document/view</i> .
<code>javax.swing.text.html</code>	Used specifically for reading and formatting HTML text through an ancillary editor kit.
<code>javax.swing.text.html.parser</code>	Contains support for parsing HTML.
<code>javax.swing.text.rtf</code>	Used specifically for reading and formatting the Rich Text Format (RTF) text through an ancillary editor kit.
<code>javax.swing.tree</code>	Defines models and views for a hierarchal tree component, such as you might see representing a file structure or a series of properties.
<code>javax.swing.undo</code>	Contains the necessary functionality for implementing undoable functions.

SWING

Q Explain Swing Components and the Containment Hierarchy.

This hierarchy that contains all the swing components is called as *Containment Hierarchy*. It states Swing components' position on the class hierarchy. Every containment hierarchy has its root which is a [top-level](#) container and all other swing components appear inside it.



The containment hierarchy, from top to bottom, is as follows:

SWING

1. Top-level Container(s)

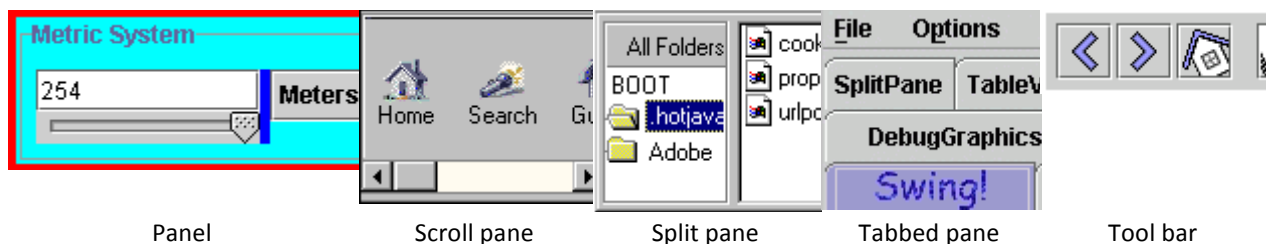
To appear onscreen, every GUI component must be part of a containment hierarchy. There is at least one containment hierarchy in every program that uses Swing components. Each containment hierarchy has a top-level container at its root. Swing provides four top-level container classes:

- ✓ **JApplet** - Enables applets to use Swing components.
- ✓ **JDialog** - The main class for creating a dialog window.
- ✓ **JFrame** - A top-level window with a title and a border.
- ✓ **JWindow** - As a rule, not very useful. Provides a window with no controls or title.

2. Intermediate Container(s)

Each top-level container must contain at least one intermediate container if there is to be anything useful displayed on the screen.

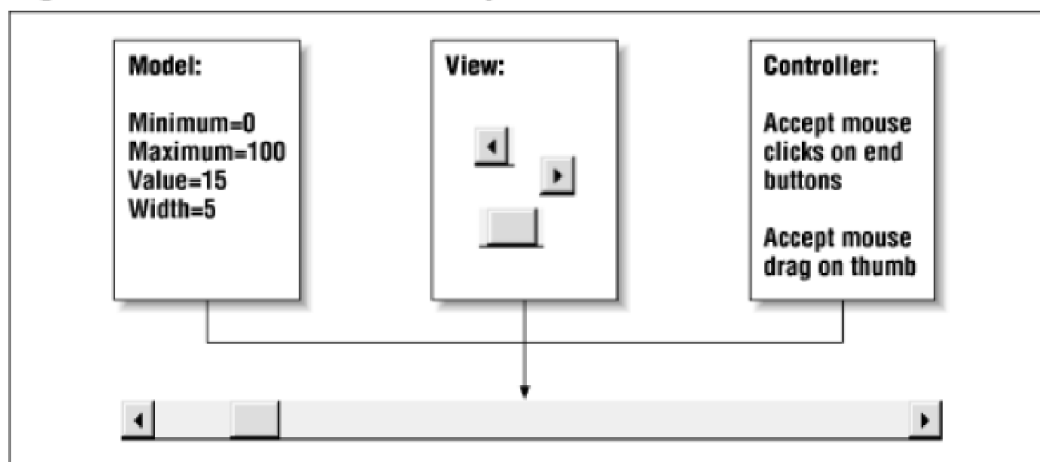
A panel, or pane, such as JPanel, is an intermediate container. Its only purpose is to simplify the positioning atomic components like buttons and labels. Other intermediate Swing containers, such as scroll panes (JScrollPane) and tabbed panes (JTabbedPane), typically play a more visible, interactive role in a program's GUI.



3. Atomic Component(s)

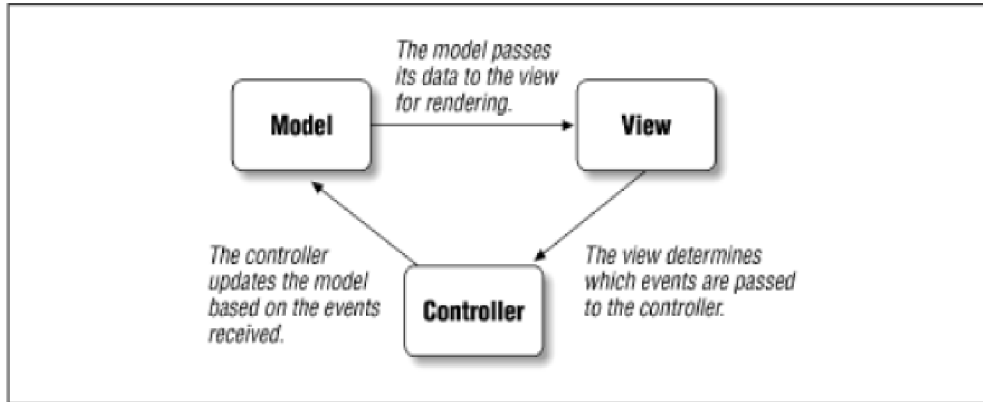
Atomic components are self-sufficient entities that present bits of information to the user. The Swing API provides many atomic components, including a button (JButton), a label (JLabel), combo boxes (JComboBox), text fields (JTextField), and tables (JTable).

Figure 1.5. The three elements of a model-view-controller architecture



SWING

1.6. Communication through the model-view-controller architecture



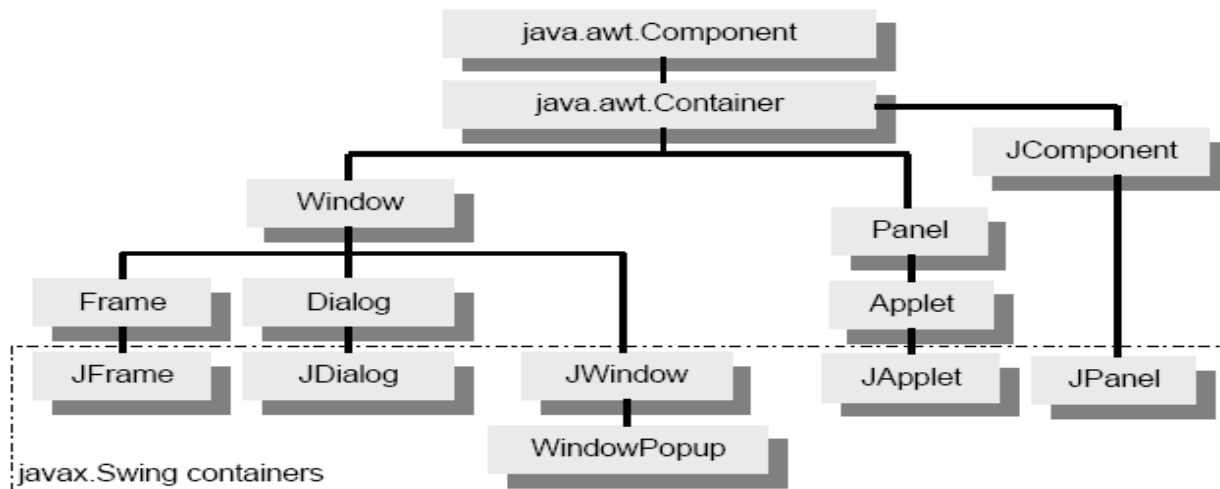
Q.Differentiate SWING and AWT or lightweight and heavyweight components

AWT	Swing
AWT stands for Abstract windows toolkit.	Swing is one of the package of JFC's (Java Foundation classes).
AWT components are called Heavyweight component.	Swings are called light weight component because swing components sits on the top of AWT components and do the work.
AWT components require java.awt package.	Swing components require javax.swing package.
AWT components are platform dependent.	Swing components are made in purely java and they are platform independent.
Look and feel feature is not supported in AWT.	We can have different look and feel in Swing.
These feature is not available in AWT.	Swing has many advanced features like JLabel, JTabbed pane which is not available in AWT. Also. Swing components are called "lightweight" because they do not require a native OS object to implement their functionality. JDialog and JFrame are heavyweight, because they do have a peer. So components like JButton, JTextArea, etc., are lightweight because they do not have an OS peer.
AWT is a thin layer of code on top of the OS.	Swing is much larger. Swing also has very much richer functionality.
Using AWT, you have to implement a lot of things yourself.	Swing has them built in.

SWING

Swing Container:

Containers are on screen windows that contain controls or other subcontainers.



Write Short notes on:

1. JFrame:

The javax.swing.JFrame class is used to create a "window". This window has a few characteristics of its own (title bar, etc), but most of the controls are placed in one of two subareas: *content pane* or *menu bar*.

Class constructors

S.N.	Constructor & Description
1	JFrame() Constructs a new frame that is initially invisible.
2	JFrame(String title) Creates a new, initially invisible Frame with the specified title.

Methods:

S.N	Method	Purpose
1	void setTitle(String)	Sets a JFrame's title using the String argument
2	void setSize(int ,int)	Sets a JFrame's size in pixels with the width and height as arguments

SWING

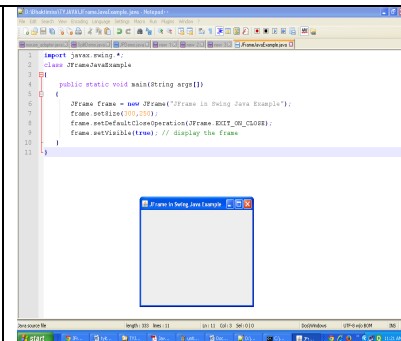
3	void setSize(Dimension)	Sets a JFrame's size using a Dimension class object, the Dimension(int ,int) constructor creates an object that represents both a width and a height
4	String (getTitle)	Returns a JFrame's title
5	void setResizable(boolean)	Sets the JFrame to be resizable by passing true to the method, or sets the JFrame not to be resizable by passing false to the method
6	boolean isResizable()	Returns true or false to indicate whether the JFrame is resizable
7	void setVisible(boolean)	Sets a JFrame to be visible using the boolean argument true and invisible using the boolean argument false
8	void setBounds(int,int, int,int)	Overrides the default behaviour for the JFrame to be positioned in the upper-left corner of the computerscreen's desktop. The first two arguments are the horizontal and vertical positions of the JFrame's upper-left corner on the desktop. The final two arguments set the width and height
9	void pack()	Causes this Window to be sized to fit the preferred size and layouts of its subcomponents. If the window and/or its owner are not yet displayable, both are made displayable before calculating the preferred size. The Window will be validated after the preferredSize is calculated.

The following steps are basically followed to get a JFrame window to appear on the screen.

1. Create an object of type JFrame.
2. Give the JFrame object a size or/and location using setSize () or setBounds () methods.
3. Make the JFrame object appear on the screen by calling setVisible () method.

Example:

```
import javax.swing.*;
class JFrameJavaExample
{
    public static void main(String args[])
    {
        JFrame frame = new JFrame("JFrame in
        Swing Java Example");
        frame.setSize(300,250);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true); // display the frame
    }
}
```



SWING

2. JApplet:

An extended version of `java.applet.Applet` that adds support for the JFC/Swing component architecture.

The `JApplet` class is slightly incompatible with `java.applet.Applet`. `JApplet` contains a `JRootPane` as it's only child.

The **contentPane** should be the parent of any children of the `JApplet`

Constructor:

`JApplet()`:Creates a swing applet instance.

Method	Description
<code>Container getContentPane()</code>	Returns the <code>contentPane</code> object for this applet.
<code>Component getGlassPane()</code>	Returns the <code>glassPane</code> object for this applet.
<code>JMenuBar getJMenuBar()</code>	Returns the <code>menubar</code> set on this applet.
<code>JLayeredPane getLayeredPane()</code>	Returns the <code>layeredPane</code> object for this applet.
<code>JRootPane getRootPane()</code>	Returns the <code>rootPane</code> object for this applet.
<code>void setContentPane(Container)</code>	Sets the <code>contentPane</code> property.
<code>void setGlassPane(Component)</code>	Sets the <code>glassPane</code> property.
<code>void setJMenuBar(JMenuBar)</code>	Sets the <code>menubar</code> for this applet.
<code>void setLayeredPane(JLayeredPane)</code>	Sets the <code>layeredPane</code> property.
<code>void setLayout(LayoutManager)</code>	By default the layout of this component may not be set, the layout of its <code>contentPane</code> should be set instead.
<code>void setRootPane(JRootPane)</code>	Sets the <code>rootPane</code> property.

3. JWindow:

The class **JWindow** is a container that can be displayed but does not have the title bar or window-management buttons.

constructors

S.N.	Constructor & Description
1	JWindow() Creates a window with no specified owner.
2	JWindow(Frame f) Creates a window with the specified owner frame.

`JPanel`:

SWING

Swing Components;

JLabel	JTextComponent
JButton	JTextField
JAbstractButton	JTextArea
JCheckbox	JTextPane
JRadioButton	JPasswordField
JList	JEditorPane
JComboBox	JToolTip
JRootPane	JOptionPane
JLayeredPane	

JLabel:

The class **JLabel** can display **either text, an image, or both**. Label's contents are aligned by setting the vertical and horizontal alignment in its display area. By default, labels are vertically centered in their display area. Text-only labels are leading edge aligned, by default; image-only labels are horizontally centered, by default.

constructors

S.N.	Constructor & Description
1	JLabel() Creates a JLabel instance with no image and with an empty string for the title.
2	JLabel(Icon image) Creates a JLabel instance with the specified image.
3	JLabel(String text) Creates a JLabel instance with the specified text.
4	JLabel(String text, Icon icon, int Alignment) Creates a JLabel instance with the specified text, image, and horizontal alignment. Alignment is an integer constant and it has following values: LEFT,CENTER,RIGHT,LEADING,TRAILING

Methods:

S. N.	Method & Description
1	Icon getIcon() Returns the graphic image (glyph, icon) that the label displays.
2	String getText() Returns the text string that the label displays.

SWING


3	void setIcon(Icon icon) Defines the icon this component will display.
4	void setText(String text) Defines the single line of text this component will display.
5	void setDisplayedMnemonic(char) char getDisplayedMnemonic() Sets or gets the letter that should look like a keyboard alternative. This is helpful when a label describes a component (such as a text field) that has a keyboard alternative but cannot display it. If the <code>labelFor</code> property is also set (using <code>setLabelFor</code>), then when the user activates the mnemonic, the keyboard focus is transferred to the component specified by the <code>labelFor</code> property.

JButton:

The class **JButton** is an implementation of a push button. This component has a label and generates an event when pressed. It can have Image also.

S.N.	Constructor & Description
1	JButton() Creates a button with no set text or icon.
2	JButton(Action a) Creates a button where properties are taken from the Action supplied.
3	JButton(Icon icon) Creates a button with an icon.
4	JButton(String text) Creates a button with text.
5	JButton(String text, Icon icon) Creates a button with initial text and an icon.
Methods:	
S.N.	Method & Description
1	<code>public void setText(String s):</code> is used to set specific text on button.
2	<code>public String getText():</code> is used to return the text of the button.
3	<code>public void setEnabled(boolean b):</code> is used to enable or disable the button.
4	<code>public void setIcon(Icon b):</code> is used to set the specified Icon on the button.

SWING

5	public Icon getIcon(): is used to get the Icon of the button.
6	public void setMnemonic(int a): is used to set the mnemonic on the button.
<p>Example:</p> <div style="display: flex; justify-content: space-between; align-items: flex-start;"> <div style="width: 60%;"> <pre> import javax.swing.*; import java.awt.*; class ButtonDemo extends JFrame { JButton b1,b2; public ButtonDemo() { setLayout(new FlowLayout()); b1=new JButton("YES"); b2=new JButton("No"); add(b1); add(b2); setVisible(true); pack(); setTitle("JButton"); } public static void main(String args[]) { ButtonDemo bd=new ButtonDemo(); } } </pre> </div> <div style="width: 35%; text-align: center;">  </div> </div>	

JCheckBox

A JCheckBox is a component that has two states: selected or unselected. The user can change the state of a check box by clicking on it. The state of a checkbox is represented by a boolean value that is true if the box is selected and is false if the box is unselected. A checkbox has a label, which is specified when the box is constructed:

Check Box Constructors

Constructor	Purpose
<pre> public JCheckBox() public JCheckBox(String title) public JCheckBox(String str, boolean state) public JCheckBox(Icon i) public JCheckBox(Icon i, boolean state) public JCheckBox(String title, Icon i) public JCheckBox(String title, Icon i, boolean state) </pre>	<p>Create a JCheckBox instance.</p> <p>Where,</p> <ul style="list-style-type: none"> > title: is the string argument specifies the text, if any, that the check box should display. > i: the Icon argument specifies the image that should be used instead of the look and feel default check box image. > State: is the initial state of the Check Box, Specifying the boolean argument as true initializes the check box to be selected. If the boolean argument is absent or false, then the check box is initially unselected.

SWING

Methods:

1	String getUIClassID() Returns a string that specifies the name of the L&F class that renders this component.
2	boolean isBorderPaintedFlat() Gets the value of the borderPaintedFlat property.
3	protected String paramString() Returns a string representation of this JCheckBox.
4	void setBorderPaintedFlat(boolean b) Sets the borderPaintedFlat property, which gives a hint to the look and feel as to the appearance of the check box border.

Example:

```
import javax.swing.*;
import java.awt.*;
class CheckBoxDemo extends JFrame
{
    JCheckBox b1,b2,b3;
    public CheckBoxDemo()
    {
        setLayout(new FlowLayout());
        b1=new JCheckBox("RED");
        b1.setMnemonic('R');
        b2=new JCheckBox("GREEN");
        b2.setMnemonic('G');
        b3=new JCheckBox("BLUE");
        b3.setMnemonic('B');

        add(b1);
        add(b2);
        add(b3);
        setVisible(true);
        pack();
        setTitle("JCheckBox");
    }
    public static void main(String args[])
    {
        CheckBoxDemo bd=new
CheckBoxDemo();

    }
}
```



SWING
JRadioButton:

Purpose:

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz. **It should be added in ButtonGroup to select one radio button only.**

Constructor	Purpose
public JRadioButton() public JRadioButton(String title) public JRadioButton(String str, boolean state) public JRadioButton(Icon i) public JRadioButton(Icon i, boolean state) public JRadioButton(String title, Icon i) public JRadioButton(String title, Icon i, boolean state)	Create a JRadioButton instance. Where, > title: is the string argument specifies the text, if any, that the radio button should display. > i: is the Icon argument specifies the image that should be used instead of the look and feel's default radio button image. > state: indicates the initial state of the radiobutton, Specifying the boolean argument as true initializes the radio button to be selected, subject to the approval of the ButtonGroup object. If the boolean argument is absent or false, then the radio button is initially unselected.

Methods:

AccessibleContext getAccessibleContext()	Gets the AccessibleContext associated with this JRadioButton.
String getUIClassID()	Returns the name of the L&F class that renders this component.
protected String paramString()	Returns a string representation of this JRadioButton.
void updateUI()	Resets the UI property to a value from the current look and feel

Example

```
import javax.swing.*;
import java.awt.*;
class RadioButtonDemo extends JFrame
{
    JRadioButton b1,b2,b3;
    public RadioButtonDemo()
    {
        setLayout(new FlowLayout());
        b1=new JRadioButton("RED");
```



SWING

<pre> b1.setMnemonic('R'); b2=new JRadioButton("GREEN"); b2.setMnemonic('G'); b3=new JRadioButton("BLUE"); b3.setMnemonic('B'); ButtonGroup bg=new ButtonGroup(); bg.add(b1);bg.add(b2);bg.add(b3); add(b1); add(b2); add(b3); setVisible(true); pack(); setTitle("JRadioButton"); } public static void main(String args[]) { RadioButtonDemo bd=new RadioButtonDemo(); } } </pre>	
--	--

JList:**Purpose:**

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 scrollpane can also have a user-defined border. JList actions are handled using ListSelectionListener.

Method or Constructor	Purpose
public JList() public JList(ListModel lm) public JList(Object[] item) public JList(Vector v)	Create a list with the initial list items specified. The second and third constructors implicitly create an immutable ListModel; Third constructor creates a JList that displays the elements in the specified array. Fourth constructor creates JList with specified list of Vector.

Methods:

Method	Purpose
void setVisibleRowCount(int) int getVisibleRowCount()	Set or get the visibleRowCount property. For a VERTICAL layout orientation, this sets or gets the preferred number of rows to display without requiring scrolling. For the HORIZONTAL_WRAP or VERTICAL_WRAP layout orientations, it defines how the cells wrap. See the setLayoutOrientation(int) for more information. The default value of this property is VERTICAL.
void setLayoutOrientation(int)	Set or get the way list cells are laid out. The possible layout formats are specified

SWING

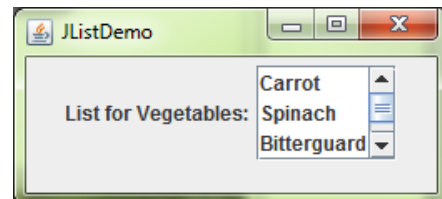
int getLayoutOrientation()	by the JList-defined values VERTICAL (a single column of cells; the default), HORIZONTAL_WRAP ("newspaper" style with the content flowing horizontally then vertically), and VERTICAL_WRAP ("newspaper" style with the content flowing vertically then horizontally).
int getFirstVisibleIndex() int getLastVisibleIndex()	Get the index of the first or last visible item.
void ensureIndexIsVisible(int)	Scroll so that the specified index is visible within the viewport that this list is in.

Managing the List's Selection

Method	Purpose
void addListSelectionListener(ListSelectionListener)	Register to receive notification of selection changes.
void setSelectedIndex(int) void setSelectedIndices(int[]) void setSelectedValue(Object, boolean) void setSelectionInterval(int, int)	Set the current selection as indicated. Use setSelectionMode to set what ranges of selections are acceptable. The boolean argument specifies whether the list should attempt to scroll itself so that the selected item is visible.
int getAnchorSelectionIndex() int getLeadSelectionIndex() int getSelectedIndex() int getMinSelectionIndex() int getMaxSelectionIndex() int[] getSelectedIndices() Object getSelectedValue() Object[] getSelectedValues()	Get information about the current selection as indicated.
void setSelectionMode(int) int getSelectionMode()	Set or get the selection mode. Acceptable values are: SINGLE_SELECTION, SINGLE_INTERVAL_SELECTION, or MULTIPLE_INTERVAL_SELECTION (the default), which are defined in ListSelectionModel.
void clearSelection() boolean isSelectionEmpty()	Set or get whether any items are selected.
boolean isSelectedIndex(int)	Determine whether the specified index is selected.

Example:

```
import javax.swing.*;
import java.awt.*;
public class JListDemo extends JFrame
{
    public JListDemo()
    {
        setLayout(new FlowLayout());
        add(new JLabel("List for Vegetables:"));
        Object vegetables[]={"Carrot","Spinach","Bitterguard","Okra"};
        JList jl=new JList(vegetables);
        jl.setVisibleRowCount(3);
        JScrollPane jsp=new JScrollPane(jl);
        add(jsp);
    }
}
```



SWING

<pre> setTitle("JListDemo"); setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); setSize(200,200); setVisible(true); } public static void main(String args[]) { new JListDemo(); } } </pre>	
---	--

JComboBox:

JComboBox is like a drop down box — you can click a drop-down arrow and select an option from a list. It generates ItemEvent. 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.

Constructor	Purpose
<pre> public JComboBox() JComboBox(ComboBoxModel) JComboBox(Object[]) JComboBox(Vector) </pre>	Create a combo box with the specified items in its menu. A combo box created with the default constructor has no items in the menu initially. Each of the other constructors initializes the menu from its argument: a model object, an array of objects, or a Vector of objects.

Methods:

<pre> void addItem(Object) void insertItemAt(Object, int) </pre>	Add or insert the specified object into the combo box's menu. The insert method places the specified object <i>at</i> the specified index, thus inserting it before the object currently at that index. These methods require that the combo box's data model be an instance of MutableComboBoxModel.
<pre> Object getItemAt(int) Object getSelectedItem() </pre>	Get an item from the combo box's menu.
<pre> void removeAllItems() void removeItemAt(int) void removeItem(Object) </pre>	Remove one or more items from the combo box's menu. These methods require that the combo box's data model be an instance of MutableComboBoxModel.
<pre> int getItemCount() </pre>	Get the number of items in the combo box's menu.

Example:

SWING

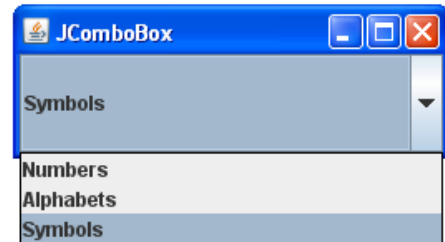
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class JComboBoxDemo extends JFrame {

    JLabel jlbPicture;
    public JComboBoxDemo() {

        String[] comboTypes = {
            "Numbers",                "Alphabets",
            "Symbols" };
        // Create the combo box, and set 2nd item as Default
        JComboBox cb = new JComboBox(comboTypes);
        cb.setSelectedIndex(2);
        add(cb, BorderLayout.CENTER);
        setVisible(true);
        pack();
        setTitle("JComboBox");
    }
    public static void main(String args[])
    {
        JComboBoxDemo bd=new JComboBoxDemo();

    }
}
```



JTextComponents

JTextComponent is the base class for swing text components. Swing provides six text components, in three groups:

1. **Text Controls** [[JTextField](#) and its subclasses [JPasswordField](#) and [JFormattedTextField](#)]

Also known simply as text fields, text controls can display only one line of editable text. Like buttons, they generate action events. Use them to get a small amount of textual information from the user and perform an action after the text entry is complete.

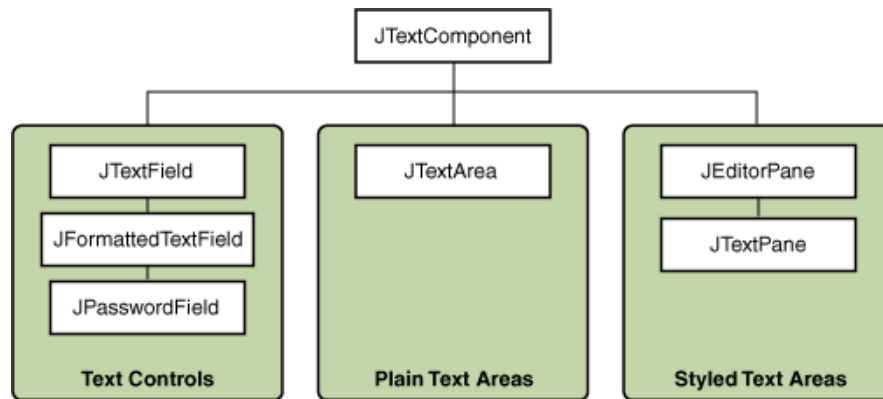
2. **Plain Text Areas** [[JTextArea](#)]

JTextArea can display multiple lines of editable text. Although a text area can display text in any font, all of the text is in the same font. Use a text area to allow the user to enter unformatted text of any length or to display unformatted help information.

3. **Styled Text Areas** [[JEditorPane](#) and its subclass [JTextPane](#)]

A styled text component can display editable text using more than one font. Some styled text components allow embedded images and even embedded components. Styled text components are powerful and multi-faceted components suitable for high-end needs, and offer more avenues for customization than the other text components. Because they are so powerful and flexible, styled text components typically require more initial programming to set up and use. One exception is that editor panes can be easily loaded with formatted text from a URL, which makes them useful for displaying uneditable help information.

SWING



JTextField:

A text field is a basic text control that enables the user to type a small amount of text. It is a lightweight component that allows the editing of a single line of text. When the user indicates that text entry is complete (usually by pressing Enter), the text field fires an action event.

Setting or Obtaining the Field's Contents

Method or Constructor	Purpose
JTextField() JTextField(String) JTextField(String, int) JTextField(int)	Creates a text field. When present, the int argument specifies the desired width in columns. The String argument contains the field's initial text.
void setText(String) String getText() (defined in JTextComponent)	Sets or obtains the text displayed by the text field.

Method	Purpose
void setEditable(boolean) boolean isEditable() (defined in JTextComponent)	Sets or indicates whether the user can edit the text in the text field.
void setColumns(int); int getColumns()	Sets or obtains the number of columns displayed by the text field. This is really just a hint for computing the field's preferred width.
void setHorizontalAlignment(int); int getHorizontalAlignment()	Sets or obtains how the text is aligned horizontally within its area. You can use JTextField.LEADING, JTextField.CENTER, and JTextField.TRAILING for arguments.

JPasswordField:

The JPasswordField class, a subclass of JTextField, provides specialized text fields for password entry. For security reasons, a password field does not show the characters that the user types. Instead, the field displays a character different from the one typed, such as an asterisk '*'. As another security precaution, a password field stores its value as an array of characters, rather than as a string.

Commonly Used JPasswordField Constructors and Methods

SWING

Constructor or Method	Purpose
JPasswordField() JPasswordField(String) JPasswordField(String, int) JPasswordField(int)	Creates a password field. When present, the int argument specifies the desired width in columns. The String argument contains the field's initial text.
char[] getPassword()	Returns the password as an array of characters.
void setEchoChar(char) char getEchoChar()	Sets or gets the echo character which is displayed instead of the actual characters typed by the user.
void addActionListener(ActionListener) void removeActionListener(ActionListener) (defined in JTextField)	Adds or removes an action listener.
void selectAll() (defined in JTextComponent)	Selects all characters in the password field.

JFormattedTextField:

Formatted text fields provide a way for developers to specify the valid set of characters that can be typed in a text field. Specifically, the JFormattedTextField class adds a *formatter* and an object *value* to the features inherited from the JTextField class. The formatter translates the field's value into the text it displays, and the text into the field's value.

Method or Constructor	Purpose
JFormattedTextField() JFormattedTextField(Object) JFormattedTextField(Format) JFormattedTextField(AbstractFormatter) JFormattedTextField(AbstractFormatterFactory) JFormattedTextField(AbstractFormatterFactory, Object)	Creates a new formatted text field. The Object argument, if present, specifies the initial value of the field and causes an appropriate formatter factory to be created. The Format or AbstractFormatter argument specifies the format or formatter to be used for the field, and causes an appropriate formatter factory to be created. The AbstractFormatterFactory argument specifies the formatter factory to be used, which determines which formatters are used for the field.
void setValue(Object) Object getValue()	Sets or obtains the value of the formatted text field. You must cast the return type based on how the JFormattedTextField has been configured. If the formatter has not been set yet, calling setValue sets the formatter to one returned by the field's formatter factory.

he following code formats the text in it as a date using the current locale format:

```
JFormattedTextField dobField = new JFormattedTextField();
dobField.setValue(new Date());
```

The following code formats a number in the current locale format:

```
JFormattedTextField salaryField = new JFormattedTextField();
salaryField.setValue(new Double(12345.98));
```

SWING

The following code creates a `JFormattedTextField` to format a date in mm/dd/yyyy format

```
DateFormat dateFormat = new SimpleDateFormat("mm/dd/yyyy");
DateFormatter dateFormatter = new DateFormatter(dateFormat);
dobField = new JFormattedTextField(dateFormatter);
```

The following code creates a `JFormattedTextField` to format a number in \$#0,000.00 format.

```
NumberFormat numFormat = new DecimalFormat("$#0,000.00");
NumberFormatter numFormatter = new NumberFormatter(numFormat);
salaryField = new JFormattedTextField(numFormatter);
```

JTextArea:

The `JTextArea` class provides a component that displays multiple lines of text and optionally allows the user to edit the text. If you need to obtain only one line of input from the user, you should use a text field. If you want the text area to display its text using multiple fonts or other styles, you should use an editor pane or text pane. If the displayed text has a limited length and is never edited by the user, use a label.

Setting or Obtaining Contents

Method or Constructor	Purpose
<code>JTextArea()</code> <code>JTextArea(String)</code> <code>JTextArea(String, int, int)</code> <code>JTextArea(int, int)</code>	Creates a text area. When present, the <code>String</code> argument contains the initial text. The <code>int</code> arguments specify the desired width in columns and height in rows, respectively.
<code>void setText(String)</code> <code>String getText()</code> <i>(defined in <code>JTextComponent</code>)</i>	Sets or obtains the text displayed by the text area.

Fine Tuning the Text Area's Appearance

Method	Purpose
<code>void setEditable(boolean)</code> <code>boolean isEditable()</code> <i>(defined in <code>JTextComponent</code>)</i>	Sets or indicates whether the user can edit the text in the text area.
<code>void setColumns(int);</code> <code>int getColumns()</code>	Sets or obtains the number of columns displayed by the text area. This is really just a hint for computing the area's preferred width.
<code>void setRows(int);</code> <code>int getRows()</code>	Sets or obtains the number of rows displayed by the text area. This is a hint for computing the area's preferred height.
<code>int setTabSize(int)</code>	Sets the number of characters a tab is equivalent to.

SWING

int setLineWrap(boolean)	Sets whether lines are wrapped if they are too long to fit within the allocated width. By default this property is false and lines are not wrapped.
int setWrapStyleWord(boolean)	Sets whether lines can be wrapped at white space (word boundaries) or at any character. By default this property is false, and lines can be wrapped (if line wrapping is turned on) at any character.

Implementing the Text Area's Functionality

Method	Purpose
void selectAll() (defined in JTextComponent)	Selects all characters in the text area.
void append(String)	Adds the specified text to the end of the text area.
void insert(String, int)	Inserts the specified text at the specified position.
void replaceRange(String, int, int)	Replaces the text between the indicated positions with the specified string.
int getLineCount() int getLineOfOffset(int) int getLineStartOffset(int) int getLineEndOffset(int)	Utilities for finding a line number or the position of the beginning or end of the specified line.

JEditorPane & JtextPane:

Two Swing classes support styled text: JEditorPane and its subclass JTextPane. The JEditorPane class is the foundation for Swing's styled text components and provides a mechanism through which you can add support for custom text formats. If you want unstyled text, use a text area instead.

JEditorPane API for Displaying Text from a URL

Method or Constructor	Description
JEditorPane(URL) JEditorPane(String)	Creates an editor pane loaded with the text at the specified URL.
setPage(URL) setPage(String)	Loads an editor pane (or text pane) with the text at the specified URL.
URL getPage()	Gets the URL for the editor pane's (or text pane's) current page.

JTextPane API

Method or Constructor	Description
JTextPane() JTextPane(StyledDocument)	Creates a text pane. The optional argument specifies the text pane's model.
StyledDocument getStyledDocument setStyledDocument(StyledDocument)	Gets or sets the text pane's model.

SWING