Chapter -1

Swing Components: JLable and ImageIcon, JTextField, The Swing Buttons: J Button, CheckBoxes, RadioButtons, JTabbedPane, JList, JComboBox, JTable, Menu Bars and Menu

Java Swing Introduction

To create a Java program with a graphical user interface (GUI). Swing is the primary Java GUI widget toolkit. JAVA provides a rich set of libraries to create Graphical User Interface in platform independent way. Swing API is set of extensible GUI Components to ease developer's life to create JAVA based Front End/ GUI Applications. It is build upon top of AWT API and acts as replacement of AWT API as it has almost every control corresponding to AWT controls. Component is a user interface element that occupies screen space. E.g., button, text field, scrollbar.

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 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.

It is part of Java Foundation Classes (JFC) — an API for providing a graphical user interface (GUI) for Java programs. The JFC also include other features important to a GUI program, such as the ability to add rich graphics functionality and the ability to create a program that can work in different languages and by users with different input devices. Swing is a set of program components for Java programmers that provide the ability to create graphical user interface (GUI) components, such as buttons and scroll bars, that are independent of the windowing system for specific operating system .

Swing is important to develop Java programs with a graphical user interface (GUI). There are many components which are used for the building of GUI in Swing. The Swing Toolkit consists of many components for the building of GUI. These components are also helpful in providing interactivity to Java applications.

The Swing toolkit includes a rich set of components for building GUIs and adding interactivity to Java applications. Swing includes all the components you would expect from a modern toolkit: table controls, list controls, tree controls, buttons, and labels.

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.

Features of Swing

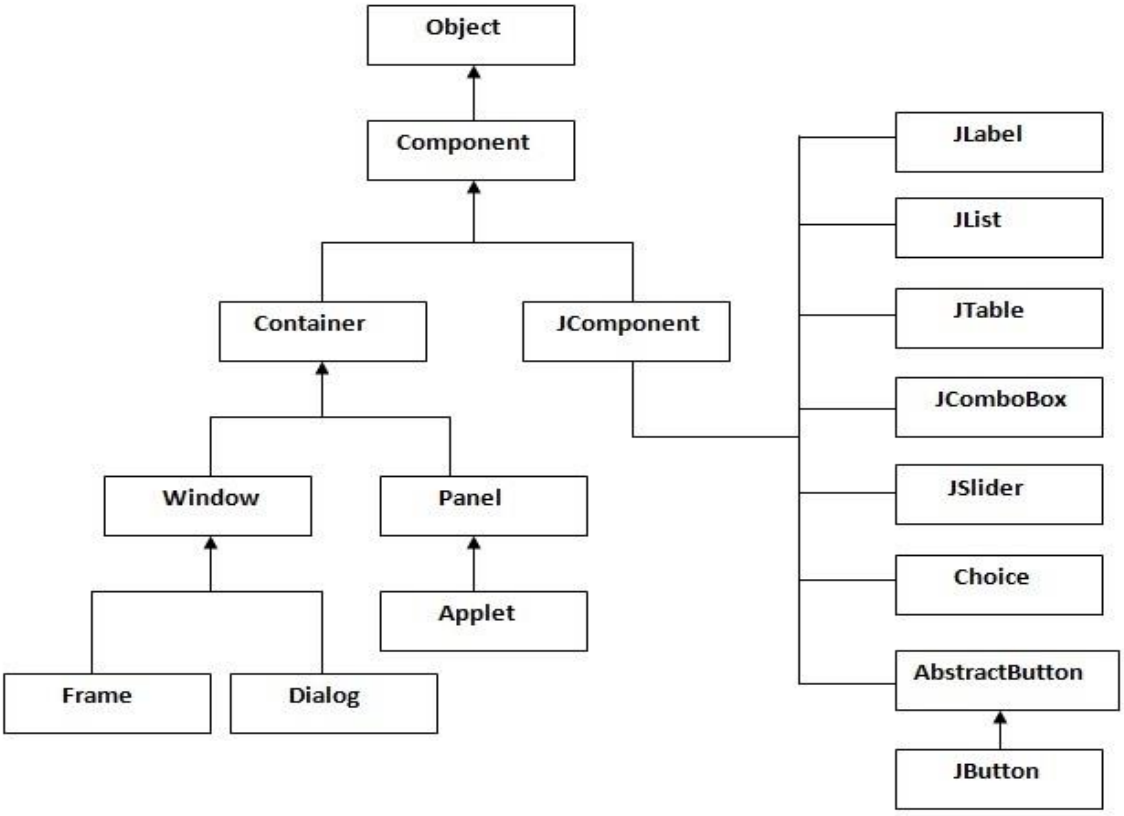
1. **Swing GUI Components: -** The Swing toolkit includes a rich array of components: from basic components, such as buttons and check boxes, to rich and complex components, such as tables and text. Even deceptively simple components, such as text fields, offer sophisticated functionality, such as formatted text input or password field behavior. There are file browsers and dialogs to suit most needs, and if not, customization is possible. If none of Swing's provided components are exactly what you need, you can leverage the basic Swing component functionality to create your own.
2. **Pluggable Look-and-Feel Support: -** A program can specify the look and feel of the platform (Any OS) it is running on, or it can specify to always use the Java look and feel, and without recompiling, it will just work.
3. **Data Transfer: -** Data transfer, via cut, copy, paste, and drag and drop, is essential to almost any application. Support for data transfer is built into Swing and works between Swing components within an application, between Java applications, and between Java and native applications.
4. **Internationalization: -** This feature allows developers to build applications that can interact with users worldwide in their own languages and cultural conventions. Applications can be created that accept input in languages that use thousands of different characters, such as Japanese, Chinese, or Korean.
5. **Swing's layout managers: -** make it easy to honor a particular orientation required by the UI. For example, the UI will appear right to left in a locale where the text flows right to left. This support is automatic: You need only code the UI once and then it will work for left to right and right to left, as well as honor the appropriate size of components that change as you localize the text.
6. **Accessibility API: -** People with disabilities use special software — assistive technologies — that mediate the user experience for them. Such software needs to obtain a wealth of information about the running application in order to represent it in alternate media: for a screen reader to read the screen with synthetic speech or render it via a Braille display, for a screen magnifier to track the caret and keyboard focus, for on-screen keyboards to present dynamic keyboards of the menu choices and toolbar items and dialog controls, and for voice control systems to know what the user can control with his or her voice. The accessibility API enables these assistive technologies to get the information they need, and to programmatically manipulate the elements that make up the graphical user interface.

Characteristics of Swing

1. platform independent
2. customizable
3. extensible

4. configurable
5. lightweight

Hierarchy of swing:



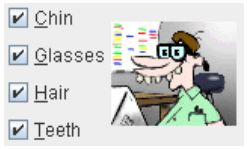
A Visual Guide to Swing Components (Java Look and Feel)

The Java Swing provides the multiple platform independent APIs interfaces for interacting between the users and GUIs components. All Java Swing classes imports from the import javax.swing.*; package. Java provides interactive features for design the GUIs toolkit or components like: labels, buttons, text boxes, checkboxes, combo boxes, panels and sliders etc. All AWT flexible components can be handled by the Java Swing. The Java Swing supports the plugging between the look and feel features. The look and feel that means the dramatically changing in the component like JFrame, JWindow, JDialog etc. for viewing it into the several types of window.

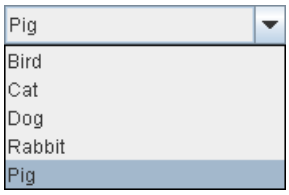
Basic Controls

Simple components that are used primarily to get input from the user; they may also show simple state.

JButton

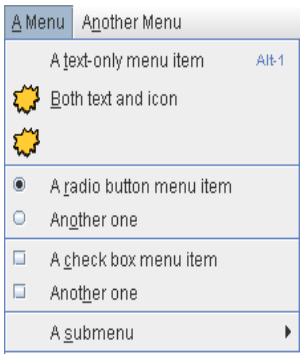


JCheckBox

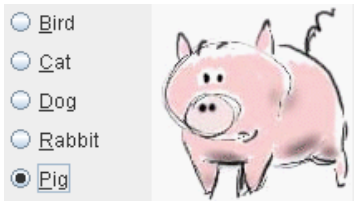


JComboBox

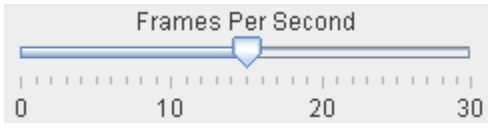
JList



JMenu



JRadioButton



JSlider

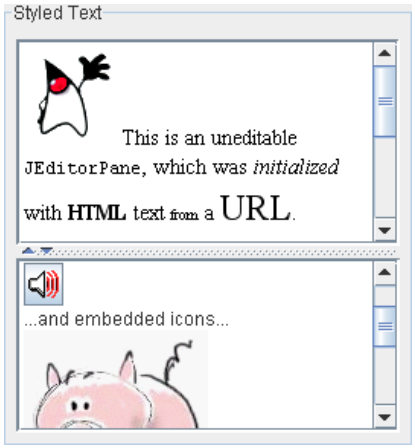
JSpinner

JTextField

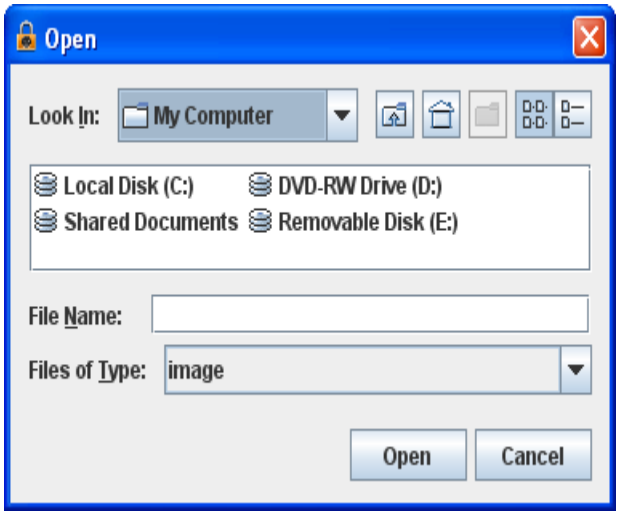
JPasswordField

Interactive Displays of Highly Formatted Information: -

These components display highly formatted information that can be modified by the user.



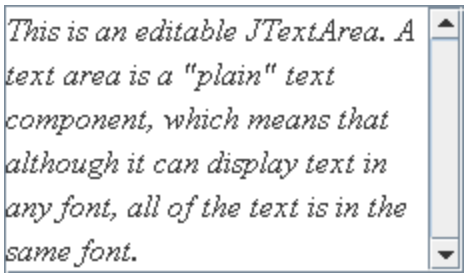
JEditorPane



JFileChooser

Host	User	Password	Last Modified
Biocca Games	Freddy	!#asf6Awwzb	Mar 16, 2006
zabble	ichabod	Tazb!34$fZ	Mar 6, 2006
Sun Developer	fraz@hotmail.co...	AasW541!fbZ	Feb 22, 2006
Heirloom Seeds	shams@gmail....	bkz[ADF78!	Jul 29, 2005
Pacific Zoo Shop	seal@hotmail.c...	vbAf124%z	Feb 22, 2006

JTable

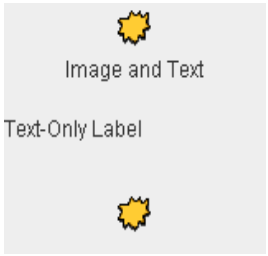


JTextArea

JTree

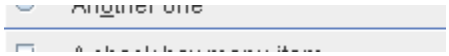
Uneditable Information Displays : -

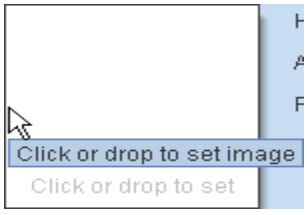
These components exist solely to give the user information.



JLabel

JProgressBar


JSeparator

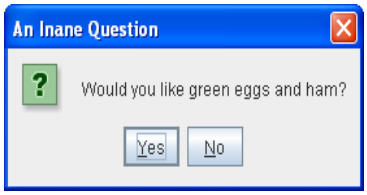


JToolTip

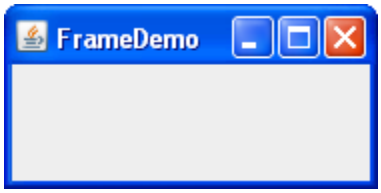
Top-Level Containers: -

At least one of these components must be present in any Swing application.

JApplet



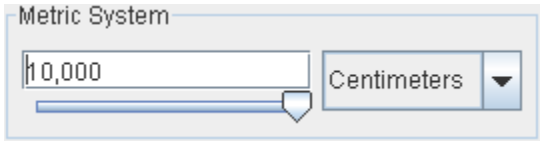
JDialog



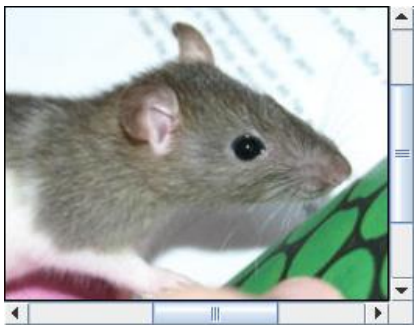
JFrame

General-Purpose Containers

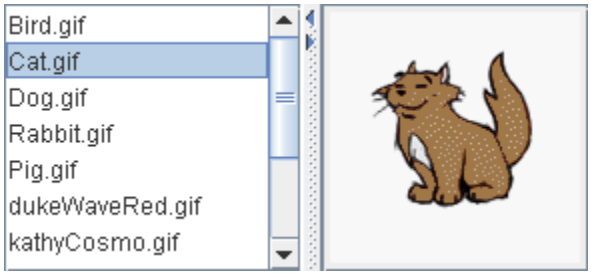
These general-purpose containers are used in most Swing applications.



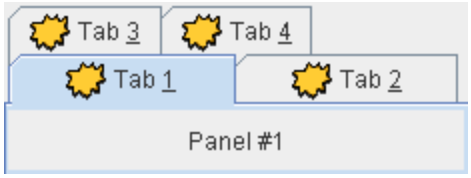
JPanel



JScrollPane



JSplitPane



JTabbedPane


JToolBar

Java Swing Components Swing API (javax.swing.*)

Class	Description
JButton	This class extends the **AbstractButton** and you can create the new button.
JCheckBox	This class extends the **JToggleButton** and implements the check box in which buttons are selected or deselected.
JComboBox	In java swing, All components are used the JComponent except the top-level containers like: JFrame, JDialog etc.
JDialog	It extends the **Dialog**. This class used to create the dialog window and when you want to create the custom dialog window with the help of **JOptionPane** method.
JFileChooser	This class provides the facility to choosing the file.
JLabel	This class used to show the small text and image.
JList	This class used to create a list where you select the one or more than objects.
JMenu	This class used to create a new menu where you add the JMenuItems. When you select the item then shows the popup menu items in the JMenuBar.
JMenuBar	It used to create a new menu bar where the JMenu objects are added.
JPanel	It extends the **JComponent** and used to create a new panel.
JPassworkField	It provides the single line text editing. Here, don't available the original characters but view type indication characters are available.
JProgressBar	It shows the integer types values in percent within a bounded range to

	determine the working process.
JRadioButton	It implements the radio button and shows the state of an item selected or deselected.
JScrollBar	This class used to create a scroll bar. It provides the view content area where you show the content to scroll this.
JSeparator	This class use the separator among the components.
JScrollPane	It provides the scrollable view components.
JSlider	This class provides a control to represent a numeric value by dragging the slider.
JTable	It provides the user interface component and represents the two dimensional data.
JTextArea	It provides the multi line plain text area.
JTextField	It provides the facility to editing the text in a single line.
JToolBar	It provides set of command buttons icons that performs the different actions or controls.
JTree	It shows the data in a hierarchical way.
JViewPort	It gives you about the underlying information.
JWindow	It extends **window** and shows the any location or area on the desktop. It couldn't any title bar and window management buttons.

JFrame:

This program shows you how to create a frame in Java Swing Application. The frame in java works like the main window where your components (controls) are added to develop an application. In the Java Swing, top-level windows are represented by the **JFrame** class. Java supports the look and feel and decoration for the frame.

For creating java standalone application you must provide GUI for a user. The most common way of creating a frame is, using single argument constructor of the **JFrame** class. The argument of the constructor is the title of the window or frame. Other user interface are added by constructing and adding it to the container one by one. The frame initially are not visible and to make it visible thesetVisible(true) function is called passing the boolean value *true*. The close button of the frame by default performs the hide operation for the JFrame. In this example we have changed this behavior to window close operation by setting the setDefaultCloseOperation() to EXIT_ON_CLOSE value.

setSize (400, 400):
Above method sets the size of the frame or window to width (400) and height (400) pixels.

setVisible(true):
Above method makes the window visible.

setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE):
Above code sets the operation of close operation to Exit the application using the System exit method.

Example :

```
import javax.swing.*;

public class JFramedemo{
  public static void main(String[] args){
  JFrame frame = new JFrame("Frame in Java Swing");
  frame.setSize(400, 400);
  frame.setVisible(true);
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
}
```

JLabel:

A display area for a short text string or an image, or both. A label does not react to input events. As a result, it cannot get the keyboard focus. A label can, however, display a keyboard alternative as a convenience for a nearby component that has a keyboard alternative but can't display it.

A JLabel object can display text, an image, or both. You can specify where in the label's display area the label's contents are aligned by setting the vertical and horizontal alignment. 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.

You can also specify the position of the text relative to the image. By default, text is on the trailing edge of the image, with the text and image vertically aligned.

Setting or Getting the Label's Contents	
Method or Constructor	**Purpose**
JLabel(Icon) JLabel(Icon, int) JLabel(String) JLabel(String, Icon, int) JLabel(String, int) JLabel()	Creates a JLabel instance, initializing it to have the specified text/image/alignment. The int argument specifies the horizontal alignment of the label's contents within its drawing area. The horizontal alignment must be one of the following constants defined in the SwingConstantsinterface (which JLabel implements): LEFT, CENTER, RIGHT, LEADING, or TRAILING. For ease of localization, we strongly recommend using LEADING and TRAILING, rather than LEFT and RIGHT.
void setText(String) String getText()	Sets or gets the text displayed by the label. You can use HTML tags to format the text, as described in Using HTML in Swing Components.
void setIcon(Icon) Icon getIcon()	Sets or gets the image displayed by the label.
void setDisabledIcon(Icon) Icon getDisabledIcon()	Sets or gets the image displayed by the label when it is disabled. If you do not specify a disabled image, then the look and feel creates one by manipulating the default image.

JButton:

The JButton class is used to create buttons that have plateform-independent implementation.

Method or Constructor	Purpose
JButton(Action) JButton(String, Icon)	Create a JButton instance, initializing it to have the specified text/image/action.

JButton(String) JButton(Icon) JButton()	
void setAction(Action) Action getAction()	Set or get the button's properties according to values from the Action instance.
void setText(String) String getText()	Set or get the text displayed by the button. You can use HTML formatting, as described in Using HTML in Swing Components.
void setIcon(Icon) Icon getIcon()	Set or get the image displayed by the button when the button isn't selected or pressed.
void setDisabledIcon(Icon) Icon getDisabledIcon()	Set or get the image displayed by the button when it is disabled. If you do not specify a disabled image, then the look and feel creates one by manipulating the default image.
void setPressedIcon(Icon) Icon getPressedIcon()	Set or get the image displayed by the button when it is being pressed.
void setSelectedIcon(Icon) Icon getSelectedIcon() void setDisabledSelectedIcon(Icon) Icon getDisabledSelectedIcon()	Set or get the image displayed by the button when it is selected. If you do not specify a disabled selected image, then the look and feel creates one by manipulating the selected image.

Example:

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

public class ImageButton{
ImageButton(){
JFrame f=new JFrame();


JButton b=new JButton(new ImageIcon("b.jpg"));
b.setBounds(130,100,100, 40);

f.add(b);

f.setSize(300,400);
f.setLayout(null);
f.setVisible(true);

f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      }

public static void main(String[] args) {
```

```
    new ImageButton();
}
}
```

JTextField:

The class JTextField is a component which allows the editing of a single line of text.

Class constructors

S.N.	Constructor & Description
1	**JTextField()** Constructs a new TextField.
2	**JTextField(Document doc, String text, int columns)** Constructs a new JTextField that uses the given text storage model and the given number of columns.
3	**JTextField(int columns)** Constructs a new empty TextField with the specified number of columns.
4	**JTextField(String text)** Constructs a new TextField initialized with the specified text.
5	**JTextField(String text, int columns)** Constructs a new TextField initialized with the specified text and columns.

Class methods

S.N.	Method & Description
1.	**void addActionListener(ActionListener l)** Adds the specified action listener to receive action events from this textfield.
2.	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this JTextField.
3.	**Action getAction()** Returns the currently set Action for this ActionEvent source, or null if no Action is set.
4.	**Action[] getActions()** Fetches the command list for the editor.
5.	**int getColumns()** Returns the number of columns in this TextField.
6.	**protected int getColumnWidth()** Returns the column width.
7.	**int getHorizontalAlignment()** Returns the horizontal alignment of the text.
8.	**void removeActionListener(ActionListener l)** Removes the specified action listener so that it no longer receives action events from this textfield.
9.	**void scrollRectToVisible(Rectangle r)** Scrolls the field left or right.

10	**void setAction(Action a)** Sets the Action for the ActionEvent source.
11	**void setActionCommand(String command)** Sets the command string used for action events.
12	**void setColumns(int columns)** Sets the number of columns in this TextField, and then invalidate the layout.
13	**void setDocument(Document doc)** Associates the editor with a text document.
14	**void setFont(Font f)** Sets the current font.
15	**void setHorizontalAlignment(int alignment)** Sets the horizontal alignment of the text.

Example:

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

public class TFInFrame extends JFrame {
   public TFInFrame(){
      super("JTextField in a JFrame");
      // Set the style of the frame
      add(new JTextField("I'm a JTextField", 25));

      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setResizable(false);
      setLayout(new FlowLayout());
      pack();
      setLocationRelativeTo(null);
      setVisible(true);
   }
   public static void main(String[] args){
      new TFInFrame();
   }
}
```

JComboBox :

The JComboBox class is used to create the combobox (drop-down list). At a time only one item can be selected from the item list.

Commonly used Constructors of JComboBox class:

JComboBox()

JComboBox(Object[] items)

Commonly used methods of JComboBox class:

1) public void addItem(Object anObject): is used to add an item to the item list.

2) public void removeItem(Object anObject): is used to delete an item to the item list.

3) public void removeAllItems(): is used to remove all the items from the list.

4) public void setEditable(boolean b): is used to determine whether the JComboBox is editable.

5) public void addActionListener(ActionListener a): is used to add the ActionListener.

6) public void addItemListener(ItemListener i): is used to add the ItemListener.

Example :

```
import javax.swing.*;
public class Combo {
JFrame f;
Combo(){
   f=new JFrame("Combo ex");

   String country[]={"India","Aus","U.S.A","England","Newzeland"};

   JComboBox cb=new JComboBox(country);
   cb.setBounds(50, 50,90,20);
   f.add(cb);

   f.setLayout(null);
   f.setSize(400,500);
   f.setVisible(true);

}
public static void main(String[] args) {
   new Combo();

}
}
```

JRadioButton *:*

The JRadioButton class is used to create a radio button.

Commonly used Constructors of JRadioButton class:

JRadioButton(): creates an unselected radio button with no text.
JRadioButton(String s): creates an unselected radio button with specified text.
JRadioButton(String s, boolean selected): creates a radio button with the specified text and selected status.

Commonly used Methods of AbstractButton class:

1) public void setText(String s): is used to set specified text on button.
2) public String getText(): is used to return the text of the button.
3) public void setEnabled(boolean b): is used to enable or disable the button.
4) public void setIcon(Icon b): is used to set the specified Icon on the button.
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.
7) public void addActionListener(ActionListener a): is used to add the action listener to this object.

Example:

```
import javax.swing.*;
public class Radio {
JFrame f;

Radio(){
f=new JFrame();

JRadioButton r1=new JRadioButton("A) Male");
JRadioButton r2=new JRadioButton("B) FeMale");
r1.setBounds(50,100,70,30);
r2.setBounds(50,150,70,30);

ButtonGroup bg=new ButtonGroup();
bg.add(r1);bg.add(r2);

f.add(r1);f.add(r2);

f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String[] args) {
   new Radio();
}
}
```

JTextArea :

The JTextArea class is used to create a text area. It is a multiline area that displays the plain text only.

Commonly used Constructors:

•JTextArea(): creates a text area that displays no text initially.
•JTextArea(String s): creates a text area that displays specified text initially.
•JTextArea(int row, int column): creates a text area with the specified number of rows and columns that displays no text initially..
•JTextArea(String s, int row, int column): creates a text area with the specified number of rows and columns that displays specified text.

Commonly used methods of JTextArea class:

1) public void setRows(int rows): is used to set specified number of rows.
2) public void setColumns(int cols):: is used to set specified number of columns.
3) public void setFont(Font f): is used to set the specified font.
4) public void insert(String s, int position): is used to insert the specified text on the specified position.
5) public void append(String s): is used to append the given text to the end of the document.

Example:

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

public class TArea {
   JTextArea area;
   JFrame f;
   TArea(){
   f=new JFrame();

   area=new JTextArea(300,300);
   area.setBounds(10,30,300,300);

   area.setBackground(Color.black);
   area.setForeground(Color.white);

   f.add(area);

   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
}
   public static void main(String[] args) {
      new TArea();
   }
}
```

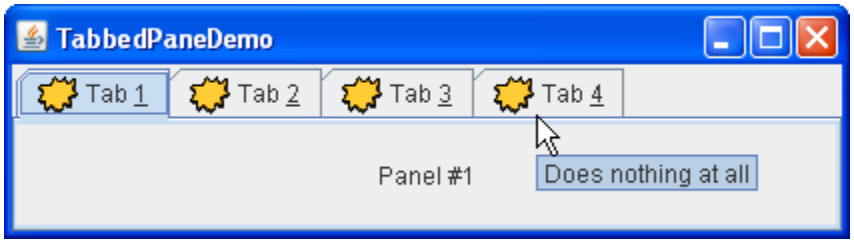
JTabbedPane:

The JTabbedPane class, you can have several components, such as panels, share the same space. The user chooses which component to view by selecting the tab corresponding to the desired component. If you want similar functionality without the tab interface, you can use a card layout instead of a tabbed pane.

To Create Tabbed Panes

To create a tabbed pane, instantiate JTabbedPane, create the components you wish it to display, and then add the components to the tabbed pane using the addTab method.

The following picture introduces an application called TabbedPaneDemo that has a tabbed pane with four tabs.



Creating and Setting Up a Tabbed Pane	
Method or Constructor	**Purpose**
JTabbedPane() JTabbedPane(int) JTabbedPane(int, int)	Creates a tabbed pane. The first optional argument specifies where the tabs should appear. By default, the tabs appear at the top of the tabbed pane. You can specify these positions (defined in the SwingConstants interface, which JTabbedPane implements): TOP, BOTTOM, LEFT, RIGHT. The second optional argument specifies the tab layout policy. You can specify one of these policies (defined in JTabbedPane): WRAP_TAB_LAYOUT orSCROLL_TAB_LAYOUT.
addTab(String, Icon, Component, String) addTab(String, Icon, Component) addTab(String, Component)	Adds a new tab to the tabbed pane. The first argument specifies the text on the tab. The optional icon argument specifies the tab's icon. The component argument specifies the component that the tabbed pane should show when the tab is selected. The fourth argument, if present, specifies the tool tip text for the tab.
Inserting, Removing, Finding, and Selecting Tabs	
Method	**Purpose**
insertTab(String, Icon, Component, String, int)	Inserts a tab at the specified index, where the first tab is at index 0. The arguments are the same as for addTab.
remove(Component)	Removes the tab corresponding to the specified component or index.

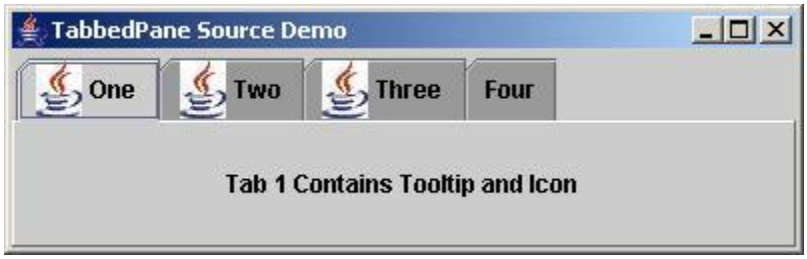
removeTabAt(int)	
removeAll()	Removes all tabs.

Example:

```
import javax.swing.JTabbedPane;
import javax.swing.ImageIcon;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JFrame;
import java.awt.*;
import java.awt.event.*;

public class JTabbedPaneDemo extends JPanel {

	public JTabbedPaneDemo() {
		ImageIcon icon = new ImageIcon("java-swing-tutorial.JPG");
		JTabbedPane jtbExample = new JTabbedPane();
		JPanel jplInnerPanel1 = createInnerPanel("Tab 1 Contains Tooltip and Icon");
		jtbExample.addTab("One", icon, jplInnerPanel1, "Tab 1");
		jtbExample.setSelectedIndex(0);
		JPanel jplInnerPanel2 = createInnerPanel("Tab 2 Contains Icon only");
		jtbExample.addTab("Two", icon, jplInnerPanel2);
		JPanel jplInnerPanel3 = createInnerPanel("Tab 3 Contains Tooltip and Icon");
		jtbExample.addTab("Three", icon, jplInnerPanel3, "Tab 3");
		JPanel jplInnerPanel4 = createInnerPanel("Tab 4 Contains Text only");
		jtbExample.addTab("Four", jplInnerPanel4);
		// Add the tabbed pane to this panel.
		setLayout(new GridLayout(1, 1));
		add(jtbExample);
	}
	protected JPanel createInnerPanel(String text) {
		JPanel jplPanel = new JPanel();
		JLabel jlbDisplay = new JLabel(text);
		jlbDisplay.setHorizontalAlignment(JLabel.CENTER);
		jplPanel.setLayout(new GridLayout(1, 1));
		jplPanel.add(jlbDisplay);
		return jplPanel;
	}
	public static void main(String[] args) {
```

```
                JFrame frame = new JFrame("TabbedPane Source Demo");
                frame.addWindowListener(new WindowAdapter() {

                        public void windowClosing(WindowEvent e) {
                                System.exit(0);
                        }
                });
                frame.getContentPane().add(new JTabbedPaneDemo(),
                                BorderLayout.CENTER);
                frame.setSize(400, 125);
                frame.setVisible(true);
        }
}
```

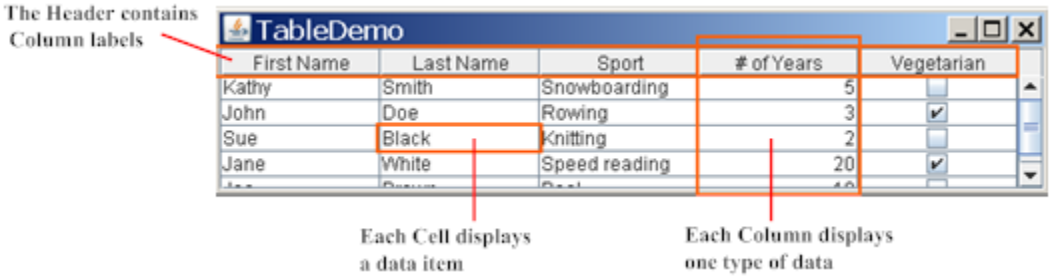
Output



JTable:

JTable is a user-interface component that presents data in a two-dimensional table format.

With the JTable class you can display tables of data, optionally allowing the user to edit the data. JTable does not contain or cache data; it is simply a view of your data. Here is a picture of a typical table displayed within a scroll pane:



The table in SimpleTableDemo.java declares the column names in a String array:

String[] columnNames = {"First Name",

```
                "Last Name",
                "Sport",
                "# of Years",
                "Vegetarian"};
```

Its data is initialized and stored in a two-dimensional Object array:

```
Object[][] data = {
    {"Kathy", "Smith",
     "Snowboarding", new Integer(5), new Boolean(false)},
    {"John", "Doe",
     "Rowing", new Integer(3), new Boolean(true)},
    {"Sue", "Black",
     "Knitting", new Integer(2), new Boolean(false)},
    {"Jane", "White",
     "Speed reading", new Integer(20), new Boolean(true)},
    {"Joe", "Brown",
     "Pool", new Integer(10), new Boolean(false)}
};
```

Then the Table is constructed using these data and columnNames:

```
JTable table = new JTable(data, columnNames);
```

There are two JTable constructors that directly accept data (SimpleTableDemo uses the first):

- JTable(Object[][] rowData, Object[] columnNames)

The advantage of these constructors is that they are easy to use. However, these constructors also have disadvantages:

- They automatically make every cell editable.
- They treat all data types the same (as strings). For example, if a table column has Boolean data, the table can display the data in a check box. However, if you use either of the two JTable constructors listed previously, your Boolean data is displayed as a string. You can see this difference in the Vegetarian column of the previous figure.
- They require that you put all of the table's data in an array or vector, which may not be appropriate for some data. For example, if you are instantiating a set of objects from a database, you might want to query the objects directly for their values, rather than copying all their values into an array or vector.

Example:

```
import javax.swing.*;
public class MyTable {
   JFrame f;
```

```
MyTable(){
   f=new JFrame();

   String data[][]={ {"101","Amit","670000"},
           {"102","Jai","780000"},
                   {"101","Sachin","700000"}};
   String column[]={"ID","NAME","SALARY"};

   JTable jt=new JTable(data,column);
   jt.setBounds(30,40,200,300);

   JScrollPane sp=new JScrollPane(jt);
   f.add(sp);

   f.setSize(300,400);
//  f.setLayout(null);
   f.setVisible(true);
}
public static void main(String[] args) {
   new MyTable();
}
}
```

Menu Bars and Menus

A top-level window can have a menu bar associated with it. A menu bar displays a list of top-level menu choices. Each choice is associated with a drop-down menu. This concept is implemented in the AWT by the following classes: **MenuBar**, **Menu**, and **MenuItem**. In general, a menu bar contains one or more **Menu** objects. Each **Menu** object contains a list of **MenuItem** objects. Each **MenuItem** object represents something that can be selected by the user. Since **Menu** is a subclass of **MenuItem**, a hierarchy of nested submenus can be created. It is also possible to include checkable menu items. These are menu options of type **CheckboxMenuItem** and will have a check mark next to them when they are selected.

To create a menu bar, first create an instance of **MenuBar**. This class defines only the default constructor. Next, create instances of **Menu** that will define the selections displayed on the bar. Following are the constructors for **Menu**:

Menu( ) throws HeadlessException
Menu(String *optionName*) throws HeadlessException
Menu(String *optionName*, boolean *removable*) throws HeadlessException

Here, *optionName* specifies the name of the menu selection. If *removable* is **true**, the menu can be removed and allowed to float free. Otherwise, it will remain attached to the menu bar. (Removable menus are implementation-dependent.) The first form creates an empty menu.

Individual menu items are of type **MenuItem**. It defines these constructors:

MenuItem( ) throws HeadlessException
MenuItem(String *itemName*) throws HeadlessException
MenuItem(String *itemName*, MenuShortcut *keyAccel*) throws HeadlessException

Here, *itemName* is the name shown in the menu, and *keyAccel* is the menu shortcut for this item.

You can disable or enable a menu item by using the **setEnabled( )** method. Its form is shown here:

void setEnabled(boolean *enabledFlag*)

If the argument *enabledFlag* is **true**, the menu item is enabled. If **false**, the menu item is disabled.

You can determine an item's status by calling **isEnabled( )**. This method is shown here:

boolean isEnabled( )

isEnabled( ) returns **true** if the menu item on which it is called is enabled. Otherwise, it returns **false**.

You can change the name of a menu item by calling **setLabel( )**. You can retrieve the current name by using **getLabel( )**. These methods are as follows:

void setLabel(String *newName*)
String getLabel( )

Here, *newName* becomes the new name of the invoking menu item. **getLabel( )** returns the current name.

EXAMPLE

```
import java.awt.*;
import javax.swing.*;
class menudemo extends JFrame
{
MenuBar mbar;
Menu menu1,menu2;
MenuItem m1,m2,m3,m4,m5;
   public menudemo()
   {
        setTitle("MENU APPLICATION"); // Set the title
   setSize(400,400); // Set size to the frame
   setLayout(new FlowLayout()); // Set the layout
   setVisible(true); // Make the frame visible
   setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   setLocationRelativeTo(null);  // Center the frame
   // Create the menu bar
   mbar=new MenuBar();
      // Create the menu
   menu1=new Menu("File");
   menu2=new Menu("Edit");
      // Create MenuItems
   m1=new MenuItem("New");
   m2=new MenuItem("Open");
   m3=new MenuItem("Save");
   m4=new MenuItem("Cut");
   m5=new MenuItem("Paste");
      // Attach menu items to menu
   menu1.add(m1);
   menu1.add(m2);
   menu1.add(m3);
    // Attach menu items to submenu
   menu2.add(m4);
   menu2.add(m5);

   // Attach menu to menu bar
   mbar.add(menu1);
   mbar.add(menu2);

   // Set menu bar to the frame
   setMenuBar(mbar);
   }
   public static void main(String args[])
   {
   new menudemo();
   }   }
```