**Swing Component**

Swing Framework contains a large set of components which provide rich functionalities and allow high level of customization. All these components are lightweight components. They all are derived from **JComponent** class. It supports the pluggable look and feel.

**JButton**

JButton class provides the functioning of push button. JButton allows an icon, string or both associated with a button.

**JButton** class provides functionality of a button. JButton class has three constuctors,

**JButton**(Icon *ic*)

**JButton**(String *str*)

**JButton**(String *str*, Icon *ic*)

It allows a button to be created using icon, a string or both. JButton supports **ActionEvent**. When a button is pressed an **ActionEvent** is generated.

**Example using JButton**

import javax.swing.\*;

import java.awt.event.\*;

import java.awt.\*;

public class testswing **extends** JFrame

{

testswing()

{

JButton bt1 = new JButton("Yes"); //Creating a Yes Button.

JButton bt2 = new JButton("No"); //Creating a No Button.

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE) //setting close operation.

setLayout(new FlowLayout()); //setting layout using FlowLayout object

setSize(400, 400); //setting size of Jframe

add(bt1); //adding Yes button to frame.

add(bt2); //adding No button to frame.

setVisible(true);

}

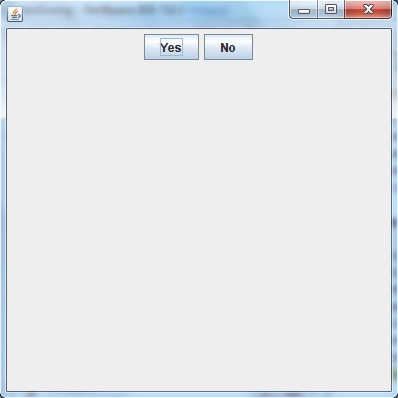
public static void main(String[] args)

{

new testswing();

}

}



**JTextField**

JTextFields allow editing of a single line of text.

**JTextField** is used for taking input of single line of text. It is most widely used text component. It has three constructors,

**JTextField**(int *cols*)

**JTextField**(String *str*, int *cols*)

**JTextField**(String *str*)

*cols* represent the number of columns in text field.

**Example using JTextField**

import javax.swing.\*;

import java.awt.event.\*;

import java.awt.\*;

public class MyTextField **extends** JFrame

{

public MyTextField()

{

**JTextField jtf = new JTextField(20)**; //creating JTextField.

add(jtf); //adding JTextField to frame.

setLayout(new FlowLayout());

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setSize(400, 400);

setVisible(true);

}

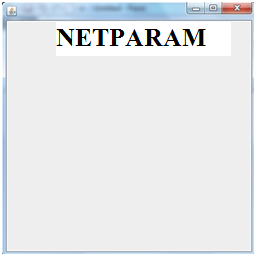
public static void main(String[] args)

{

new MyTextField();

}

}



**JCheckBox**

**JCheckBox** class is used to create checkboxes in frame. Following is constructor for JCheckBox,

**JCheckBox**(String *str*)

**Example using JCheckBox**

import javax.swing.\*;

import java.awt.event.\*;

import java.awt.\*;

public class Test **extends** JFrame

{

public Test()

{

**JCheckBox jcb = new JCheckBox("yes")**; //creating JCheckBox.

add(jcb); //adding JCheckBox to frame.

**jcb = new JCheckBox("no");**  //creating JCheckBox.

add(jcb); //adding JCheckBox to frame.

**jcb = new JCheckBox("maybe");** //creating JCheckBox.

add(jcb); //adding JCheckBox to frame.

setLayout(new FlowLayout());

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setSize(400, 400);

setVisible(true);

}

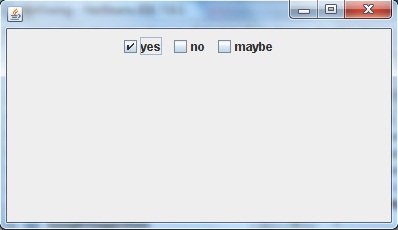
public static void main(String[] args)

{

new Test();

}

}



**JRadioButton**

Radio button is a group of related button in which only one can be selected. JRadioButton class is used to create a radio button in Frames. Following is the constructor for JRadioButton,

**JRadioButton**(String *str*)

**Example using JRadioButton**

import javax.swing.\*;

import java.awt.event.\*;

import java.awt.\*;

public class Test **extends** JFrame

{

public Test()

{

**JRadioButton jcb = new JRadioButton("A")**; //creating JRadioButton.

add(jcb); //adding JRadioButton to frame.

**jcb = new JRadioButton("B")**; //creating JRadioButton.

add(jcb); //adding JRadioButton to frame.

**jcb = new JRadioButton("C")**; //creating JRadioButton.

add(jcb); //adding JRadioButton to frame.

**jcb = new JRadioButton("none")**;

add(jcb);

setLayout(new FlowLayout());

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setSize(400, 400);

setVisible(true);

}

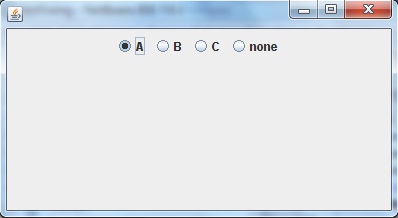
public static void main(String[] args)

{

new Test();

}

}



**JComboBox**

Combo box is a combination of text fields and drop-down list.**JComboBox** component is used to create a combo box in Swing. Following is the constructor for JComboBox,

**JComboBox**(String *arr[]*)

**Example using JComboBox**

import javax.swing.\*;

import java.awt.event.\*;

import java.awt.\*;

public class Test **extends** JFrame

{

String **name[]** = {"Abhi","Adam","Alex","Ashkay"}; //list of name.

public Test()

{

**JComboBox jc = new JComboBox(name)**; //initialzing combo box with list of name.

add(jc); //adding JComboBox to frame.

setLayout(new FlowLayout());

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setSize(400, 400);

setVisible(true);

}

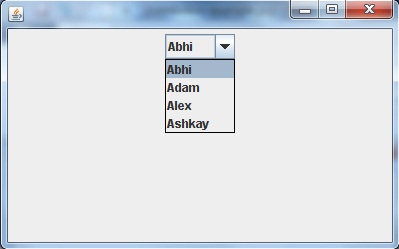
public static void main(String[] args)

{

new Test();

}

}



## JScrollPane

###### **JScrollPane scrolls to view the contents of the component added to it. Scrollbars are used to scroll the contents horizontally and vertically. From JDK 1.4, new methods for mouse wheel movements are added like processMouseWheelEvent(MouseWheelEvent) etc. The mouse wheel can be used to enlarge a photo by rotating the wheel in the display area.**

**import javax.swing.\*;**

**import java.awt.event.\*;**

**import java.awt.\*;**

**public class ScrollPaneDemo extends JApplet**

**{**

**public void init()**

**{**

**Container cp = getContentPane();**

**// cp.setLayout( new BorderLayout()) ; not needed as container default is BorderLayout**

**JPanel jp = new JPanel( ) ;**

**jp.setLayout( new GridLayout( 20, 20 ) ) ;**

**for(int i = 0 ; i < 20 ; i++)**

**for( int j = 0 ; j < 20 ; j + + )**

**jp.add(new JButton("Button " + j));**

**int v = ScrollPaneConstants.VERTICAL\_SCROLLBAR\_AS\_NEEDED ;**

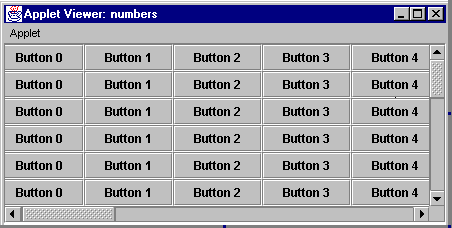
**int h = ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_AS\_NEEDED ;**

**JScrollPane js = new JScrollPane( jp, v, h ) ;**

**cp.add(js, BorderLayout.CENTER) ;**

**}**

**}**

****

# Java JTable

**The JTable class is used to display data in tabular form. It is composed of rows and columns.**

## JTable class declaration

**Let's see the declaration for javax.swing.JTable class.**

### Commonly used Constructors:

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| **JTable()** | **Creates a table with empty cells.** |
| **JTable(Object[][] rows, Object[] columns)** | **Creates a table with the specified data.** |

## Java JTable Example

**import javax.swing.\*;**

**public class TableExample {**

**JFrame f;**

**TableExample(){**

**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.setVisible(true);**

**}**

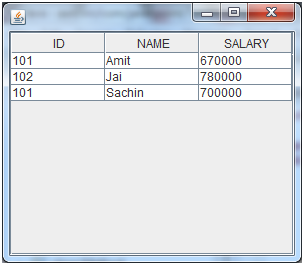
**public static void main(String[] args) {**

**new TableExample();**

**}**

**}**

**Output:**

****

## Java JTable Example with ListSelectionListener

**import javax.swing.\*;**

**import javax.swing.event.\*;**

**public class TableExample {**

**public static void main(String[] a) {**

**JFrame f = new JFrame("Table Example");**

**String data[][]={ {"101","Amit","670000"},**

**{"102","Jai","780000"},**

**{"101","Sachin","700000"}};**

**String column[]={"ID","NAME","SALARY"};**

**final JTable jt=new JTable(data,column);**

**jt.setCellSelectionEnabled(true);**

**ListSelectionModel select= jt.getSelectionModel();**

**select.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);**

**select.addListSelectionListener(new ListSelectionListener() {**

**public void valueChanged(ListSelectionEvent e) {**

**String Data = null;**

**int[] row = jt.getSelectedRows();**

**int[] columns = jt.getSelectedColumns();**

**for (int i = 0; i < row.length; i++) {**

**for (int j = 0; j < columns.length; j++) {**

**Data = (String) jt.getValueAt(row[i], columns[j]);**

**} }**

**System.out.println("Table element selected is: " + Data);**

**}**

**});**

**JScrollPane sp=new JScrollPane(jt);**

**f.add(sp);**

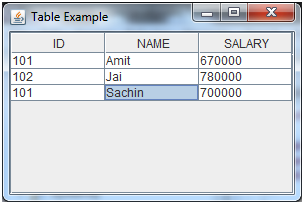
**f.setSize(300, 200);**

**f.setVisible(true);**

**}**

**}**

**Output:**

****

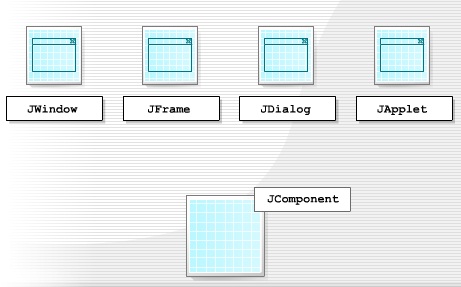
## Pane

In Java, a component is the basic user interface object and is found in all Java applications. Components include lists, buttons, panels, and windows.   
  
To use components, you need to place them in a container.

A container is a component that holds and manages other components. Containers display components using a layout manager.

Swing components inherit from the javax.Swing.JComponent class, which is the root of the Swing component hierarchy. JComponent, in turn, inherits from the Container class in the Abstract Windowing Toolkit (AWT). So Swing is based on classes inherited from AWT.

Swing provides the following useful top-level containers, all of which inherit from JComponent:

[](http://1.bp.blogspot.com/--zXC96ryIOA/TzTOKFVFytI/AAAAAAAAAGI/qisB4_FqkRE/s1600/Swing+components1.6.jpg)

JWindow

JWindow is a top-level window that doesn't have any trimmings and can be displayed anywhere on a desktop. JWindow is a heavyweight component. You usually use JWindow to create pop-up windows and "splash" screens. JWindow extends AWT's Window class.

JFrame

JFrame is a top-level window that can contain borders and menu bars. JFrame is a subclass of JWindow and is thus a heavyweight component. You place a JFrame on a JWindow. JFrame extends AWT's Frame class.

JDialog

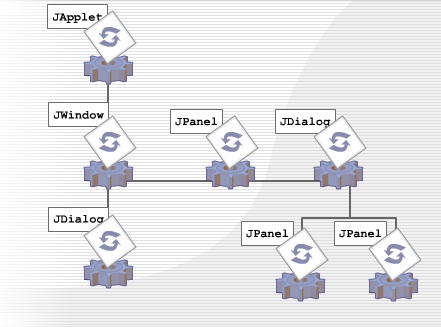
JDialog is a lightweight component that you use to create dialog windows. You can place dialog windows on a JFrame or JApplet. JDialog extends AWT's Dialog class.

JApplet

JApplet is a container that provides the basis for applets that run within web browsers. JApplet is a lightweight component that can contain other graphical user interface (GUI) components. JApplet extends AWT's Applet class.

All Swing components - including the JApplet and JDialog containers - need to be contained at some level inside a JWindow or JFrame.

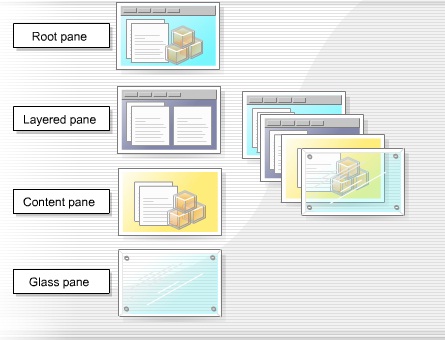
Each top-level container depends on another intermediate container called the root, which provides a number of components to each.

[](http://4.bp.blogspot.com/-dRDplceQMw8/TzTO9vTtF9I/AAAAAAAAAGQ/aY1g5wwzj04/s1600/root+components1.7.jpg)

JApplet is the root container for Swing applets and JFrame is the root container for a standalone GUI application.

Once you've created a root container, you can add components and other containers to it.

Each top-level container consists of the following panes:

[](http://1.bp.blogspot.com/-6cpCqUZjLMM/TzTPflZKc8I/AAAAAAAAAGY/Dly8bpsEL7o/s1600/top+level+container1.8.jpg)

**Root pane**

The root pane is an intermediate container that manages the layered pane, content pane, and glass pane. It can also manage an optional menu bar. You use a root pane to paint over multiple components or to catch input events.

**Layered pane**

The layered pane contains the content pane and the optional menu bar. It can also contain other components, which it arranges so that they overlap each other. This enables you to add pop-up menus to applications. The layered pane provides six functional layers in which you place the components you add to it. You use each of these functional layers for a specific function.

**Content pane**

The content pane holds all the visible components of the root pane, except the menu bar. It covers the visible section of the JFrame or JWindow and you use it to add components to the display area. Java automatically creates a content pane when you create a JFrame or JWindow but you can create your own content pane, which has to be opaque.

**Glass pane**

The glass pane is invisible by default but you can make it visible. When it is visible, it covers the components of the content pane, blocks all input events from reading these components, and can paint over an existing area containing one or more components.

One of the enhancements to JTabbedPane is the use of a component to represent the tab in a JTabbedPane. This new feature offers a convenient way to show several items in a small amount of space. It does this by dividing the information across separate tabs so that a user can select

* one tab to list a particular set of components
* a different tab to list a different set of components

By adding a **Close** button, you can enable the removal of the current tab from JTabbedPane.