What is Java Swing?

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 **GUI**s 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.

Here the following APIs interfaces and classes are available:

The following interfaces and it's descriptions to be used by the Java swing.

|  |  |
| --- | --- |
| **Interfaces** | **Descriptions** |
| **Action** | This interface performed the action with the **ActionListener** where the multiple controls are used for same purposes. |
| **BoundedRangeModel** | This interface defines the data model of components like: sliders and progressBars. |
| **ButtonModel** | It defines the state model for the buttons like: radio buttons, check boxes etc. |
| **CellEditor** | This interface used by the developer for creating the new editor and it has the new components implement interfaces. The **CellEditor** implements the wrapper based approach. |
| **ComboBoxEditor** | In this interface, the editor component used to **JComboBox**components. |
| **ComboBoxModel** | This interface represents the data model in a list model with the selected items. |
| **DesktopManager** | This interface has **JDesktopPane** object. The **JInternalFrame**implements in the JDesktopPane with the help of DesktopManager. |
| **Icon** | This interface used to graphical representation of the components. It has fixed size picture. |
| **JComboBox.KeySelectionManager** | This interface has **KeySelectionManager**and used for the combo box data model. |
| **ListCellRenderer** | This interface used for paint the cell in the list with the help of "rubber stamps" . |
| **ListModel** | This interface used for **JList** components method. It gets the value of each cell of list. |
| **ListSelectionModel** | This interface indicates the components, which are stable or not. |
| **MenuElement** | This interface used where the any components are implements in the menu. |
| **MutableComboBoxModel** | This interface extends from the **ComboBoxModel**. It is a mutable version of ComboBoxModel. |
| **Renderer** | It defines the requirements of an object for displaying the values. |
| **RootPaneContainer** | This interface uses the **RootPane** properties and it has the components like: JFrame, JInternalFrame and JWindow etc. |
| **Scrollable** | This interface provides the scrolling to show the large amount of data with the help of **JScrollPane**. |
| **ScrollPaneConstants** | This interface used for **JScrollPane** components. |
| **SingleSelectionModel** | This interface used to select the one index in a model. |
| **SwingConstants** | You can set the components on the screen to own requirements. |
| **UIDefaults.ActiveValue** | It constructs the DefaultListCellRenderer. |
| **UIDefaults.LazyValue** | This enables one to store an entry in the default table. The entered value is not constructed until first time is a real value is created through it using*LazyValue.createValue()* method. |
| **WindowConstants** | This interface has two methods setDefaultCloseOperation and getDefaultCloseOperation and provides the window close opration. |

The following classes and it's descriptions to  be used by the Java swing.

|  |  |
| --- | --- |
| **Classes** | **Descriptions** |
| **AbstractAction** | This class handles the any types of action and provides JFC Action interface. |
| **AbstractButton** | This class defines the nature of buttons and menu items. |
| **AbstractCellEditor** | It provides a list and contents of the data model. |
| **AbstractListModel** | This class defines the data model which provides the list with its contents. |
| **ActionMap** | This class works with **InputMap** and performs any action when the key is pressed. |
| **BorderFactory** | This class extends from Object and creates the Border instance in the factory. |
| **Box** | It provides the fixed spaces between two components and uses the**BoxLayout**object of the layout manager. |
| **Box.Filler** | This class participates in the Layout and uses the lightweight components. |
| **BoxLayout** | This class uses the arranging the multiple components either horizontally or vertically. The Box container uses this class. |
| **ButtonGroup** | This class used to create the multiple buttons in a **ButtonGroup** object. |
| **CellRandererPane** | This class used to insert the components like: JList, JTable and JTree. |
| **ComponentInputMap** | This class has **ComponentInputMap** constructor and creates the components with the help of InpuMap. |
| **DebugGraphics** | It extends from the **Graphics** and used to debug the graphics |
| **DefaultBoundedRangeModel** | This class provides the implementation of default BoundedRangeModel. |
| **DefaultButtonModel** | This class implements the generic ButtonModel. |
| **DefaultCellEditor** | It implements the TableCellEditor and TreeCellEditor for the table and tree cells. |
| **DefaultComboBoxModel** | It provides the default model for combo boxes. |
| **DefaultDesktopManager** | It implements the **DesktopManager.**The DesktopManager has the JInternalFrame for creating the internal fame in a frame. |
| **DefaultFocusManager** | It provides the implementing the **FocusManager**. |
| **DefaultListCellRanderer** | It implements the default **ListCellRanderer**. |
| **DefaultListCellRanderer.UIResource** | This extends the **DefaultListCellRanderer**and implementing in the**UIResource**. |
| **DefaultListModel** | It extends the **AbstractListModel** and implementing the *java.util.Vector*. |
| **DefaultListSelectionModel** | This class used for select the list in a data model. |
| **DefaultSingleSelectionModel** | This class provides the default **SingleSelectionModel**. |
| **FocusManager** | It handles all focus like: **gainedFocus** and **lostFocus**. |
| **GrayFilter** | It extends the **RGBImageFilter** and used for disabling the image through the button. |
| **ImageIcon** | This class implements the **Icon**and paints the icons from the images. |
| **InputMap** | This class uses the **ActionMap** to performed the action when you press any key of keyboard. It bounds data between the input event and an object. |
| **InputVerifier** | This class helps you when you works with the text fields through the focus. |
| **JApplet** | This class extends the **Applet** and implements the **Accessible** and**RootPaneContainer**. |
| **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. |
| **JCheckBoxMenuItem** | It extends the **JMenuItem**and determines the items which is selected or deselected. |
| **JColorChooser** | It extends the JComponent and implementing the Accessable. Here, you choose and manipulate the colors. |
| **JComboBox** | This class extends the JComboBox. It provides the drop-down list where user select only one item or value at a time. But combo box is a combination of multiple text or buttons etc. |
| **JComponent** | In java swing, All components are used the JComponent except the top-level containers like: JFrame, JDialog etc. |
| **JDesktopPane** | This class extends the JLayeredPane and when you create the object of JInternalFrame to be maintained in the JDesktopPane. The JDesktopPane has DesktopManager. |
| **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. |
| **JEditorPane** | This class extends the JTextComponent. It edits the component by the EditorKit. |
| **JFileChooser** | This class provides the facility to choosing the file. |
| **JFrame** | It extends the Frame and supports the swing components architecture. |
| **JInternalFrame** | This class extends from the **JComponent**and provides the facility to dragging, closing, resizing and menu bar of the internal frame. The JInternalFrame added into the JDesktopPane. |
| **JInternalFrame.JDesktopIcon** | It displays the desktop icon and create the instance of JInternalFrame and iconify. |
| **JLabel** | This class used to show the small text and image. |
| **JLayeredPane** | It has JFC/Swing container that can be used to overlap the components to each other. |
| **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. |
| **JMenuItem** | This class used to create new menu items in the mebus. |
| **JOptionPane** | It used to create some different types of dialog box like: message dialog box, error dialog box etc. |
| **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. |
| **JPopupMenu** | This class used to create a popup menu. It provides small window where the various types of choices are available. |
| **JPopupMenu.Separator** | Here the popup menu and the separator 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. |
| **JRadioButtonMenuItem** | It extends the**JMenuItem**and implements the radio button menu item |
| **JRootPane** | This class provides the component behind the scenes by JFrame, JWindow, JDialog etc. for providing the task-orientation and functionality. |
| **JScrollBar** | This class used to create a scroll bar. It provides the view content area where you show the content to scroll this. |
| **JScrollPane** | It provides the scrollable view components. |
| **JSeparator** | This class use the separator among the components. |
| **JSlider** | This class provides a control to represent a numeric value by dragging the slider. |
| **JSplitPane** | This class used to divides the two components graphically like: top and button, left and right. |
| **JTabbedPane** | This class provides the tab component through which you can switch from one component to another component regarding to the specific tab button by clicking on that. |
| **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. |
| **JTextPane** | This class provides the component like JTexArea for multiple lines text with more capabalities. |
| **JToggleButton** | It implements two state button that means selected or deselected. |
| **JToggleButton.ToggleButtonModel** | It extends the **DefaultButtonModel** and provides the **ToggleButton**model. |
| **JToolBar** | It provides set of command buttons icons that performs the different actions or controls. |
| **JToolBar.Separator** | It provides the tool bar separator. |
| **JToolTip** | It shows the tool tips related to it's components. |
| **JTree** | It shows the data in a hierarchical way. |
| **JTree.DynamicUtilTreeNode** | This extends the **DefaultMutableTreeNode** and create children nodes. |
| **JTree.EmptySelectionModel** | It does not allows the any selection. |
| **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. |
| **KeyStroke** | This class controls the key events on the keyboard for the equivalent device. |
| **LayoutFocusTraversalPolicy** | This class conducts the sorting objects according to their size, type, position or orientation. |
| **LookAndFeel** | It provides the dramatically changes in the component like frame related to the graphics for the application. Through this the application can be done very efficient and easier. |
| **MenuSelectionManager** | It has menu selection hierarchy. |
| **OverlayLayout** | The layout manager arrange the components. |
| **ProgressMonitor** | This class is used to monitoring the progress of any operation to be done. |
| **ProgressMonitorInputStream** | This class creates a progress monitor to monitor the progress of reading input from the input stream. It cleanups all the rights when the stream is closed. |
| **RepaintManager** | This class manage and override the repaint requests. |
| **ScrollPaneLayout** | It implements the **LayoutManager** and manage the components like: scroll bar, row header, column header etc. |
| **ScrollPaneLayout.UIResource** | It extends the **ScrollPaneLayout**and implements the **UIResource**. |
| **SizeRequirements** | It calculates the size and positions of components. |
| **SizeSequence** | It represents the order list of size and it's positions. |
| **SwingUtilities** | This class has utilities methods for swing. |
| **Timer** | Actions perform the predefined rate. |
| **ToolTipManager** | It manages the all tool tips. |
| **UIDefaults** | It extends the **Hashtable** and you set/get the value with the help of UIManager. |
| **UIDefaults.LazyInputMap** | This class creates a Input Map through it's createValue() method. The array of key after binding is passed to the constructor of this. Example of binding of key is array of pressing key information (e.g. ctrl + c or alt + f). |
| **UIDefaults.ProxyLazyValue** | This class is used to create a lazy value which is used to delay loading of the class to create instance for that. |
| **UIManager** | This class has track of the current look and feel details. |
| **UIManager.LookAndFeelInfo** | This is the nested class of **UIManager** class i.e. used for getting information about all the look and feels installed with the software development kit. |
| **ViewportLayout** | It implements the **LayoutManager**and defines the policy for the layout. |

The following Exceptions and it's description to be used by the Java swing.

|  |  |
| --- | --- |
| **Exception** | **Descriptions** |
| **UnsupportedLookAndFeelException** | This exception occurred when the look and feel classes are not supported to user's system. |

Creating a Frame:

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 the setVisible(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() toEXIT\_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.

**Here is the code of the program :**

|  |
| --- |
| **import**javax.swing.\*;  **public class**Swing\_Create\_Frame{   **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);   } } |

Setting an Icon for a Frame

In this section, you will learn how to set an icon for the frame in Java Swing.

This program helps us to set the icon (image) on the title bar of the frame. When you open frame or window the icon situated on the title bar is seen on the taskbar also. For this purposes, various methods as follows has been used:

frame.setIconImage(Toolkit.getDefaultToolkit().getImage("icon\_confused.gif"));  
Above method sets the icon for the frame or window after getting the image using the **Image** class method namedgetImage().

frame.getDefaultToolkit():  
This is the method of the **Toolkit** class which gets the default toolkit.

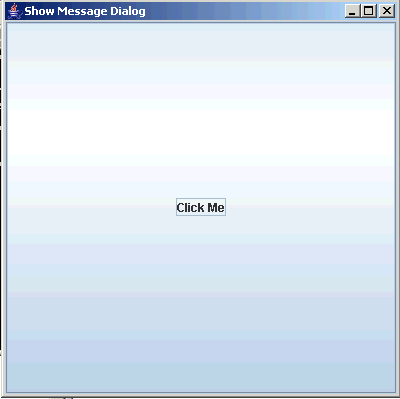
**Here is the code of program:**

|  |
| --- |
| **import**javax.swing.\*; **import**java.awt.\*;  **public class**SettingIconFrame{   **public static void**main(String[] args){   JFrame frame = **new**JFrame("Setting an Icon for a frame");   frame.setIconImage(Toolkit.getDefaultToolkit().getImage("download.jpg"));   frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);   frame.setSize(400,400);   frame.setVisible(**true**);   } } |

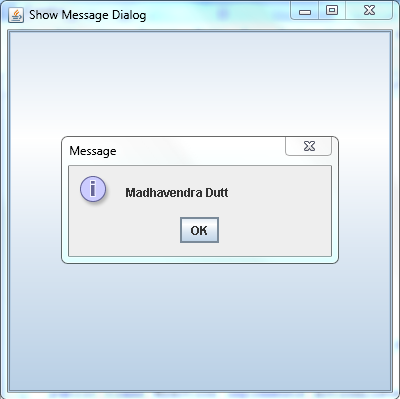
Show Dialog Box in Java - Swing Dialogs

Message dialog box is used to display informative messages to the user. In this section we will use JOptionPane class to display the message Dialog box. Our program display "Click Me" button on the window and when user clicks on it program displays Message box with "OK" button and message "Roseindia.net".

When you run the program following window will be displayed:



When you click on "Click Me" button, following Message is displayed:



**Program description:**

**JOptionPane Class:**

In non-swing application we were using **System.in** class for input or output some text or numeric values but now in the swing application we can use **JOptionPane** to show the output or show the message. This way of inputting or outputting works very efficiently in the Swing Applications. The window for showing message for input or output makes your application very innovative.

**JOptionPane** class is available in the *javax.swing.\*;* package. This class provide various types of message dialog box as follows:

* A simple message dialog box which has only one button i.e. "Ok". This type of message dialog box is used only for showing the appropriate message and user can finish the message dialog box by clicking the "Ok" button.
* A message dialog box which has two or three buttons. You can set several values for viewing several message dialog box as follows:  
  1.)  "Yes" and "No"  
  2.)  "Yes", "No" and "Cancel"  
  3.)  "Ok", and "Cancel"
* A input dialog box which contains two buttons "Ok" and "Cancel".

The JOptionPane class has three methods as follows:

* **showMessageDialog():**First is the **showMessageDialog()** method which is used to display a simple message.
* **showInputDialog():**Second is the **showInputDialog()** method which is used to display a prompt for inputting. This method returns a String value which is entered by you.
* **showConfirmDialog():**And the last or third method is the **showConfirmDialog()** which asks the user for confirmation (Yes/No) by displaying message. This method return a numeric value either 0 or 1. If you click on the "Yes" button then the method returns 1 otherwise 0.

**How program Works:**

This program illustrates you how to show a message dialog box when you click on the button.

showMessageDialog():  
This method is used to show a message dialog box which contains some text messages. This is being used with two arguments in the program where the first argument is the parent object in which the dialog box opens and another is the message which has to be shown.

**Here is the code of the program:**

|  |
| --- |
| **import**javax.swing.\*; **import**java.awt.event.\*;  **public class**ShowDialogBox{   JFrame frame;   **public static void**main(String[] args){   ShowDialogBox db = **new**ShowDialogBox();   }    **public**ShowDialogBox(){   frame = **new**JFrame("Show Message Dialog");   JButton button = **new**JButton("Click Me");   button.addActionListener(**new**MyAction());   frame.add(button);   frame.setSize(400, 400);   frame.setVisible(**true**);   frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);   }    **public class**MyAction **implements**ActionListener{   **public void**actionPerformed(ActionEvent e){   JOptionPane.showMessageDialog(frame,"Madhavendra Dutt");   }   } } |

Setting icon on the button in Java

This section illustrates you how to show the icon on the button in Java Swing.

This program sets the icon on the button in Java Swing.

Code description:

setIcon(Icon):  
Above method sets the specified Icon on the button.

ImagIcon(String image\_name):  
Above code creates an object of image.

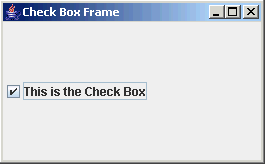
**Here is the code of the program:**

|  |
| --- |
| **import**javax.swing.\*; **import**java.awt.\*;  **public class**IconButton{   **public static void**main(String[] args){   JFrame frame = **new**JFrame("Icon on button");   JButton button = **new**JButton("Click");   Icon imgicon = **new**ImageIcon("download.jpg");   JPanel panel = **new**JPanel();   button.setIcon(imgicon);   panel.add(button);   frame.add(panel, BorderLayout.NORTH);   frame.setSize(400, 400);   frame.setVisible(**true**);   frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);   } } |

Creating Check Box in Java Swing

This section illustrates you how to create a Check Box component in Java Swing.

In this section, you can learn simply creating the Check Box in Java Swing. Check Boxes are created in swing by creating the instance of the **JCheckBox** class using it's constructor which contains the string which has to be shown beside the check box on the frame or window like this:



This is written like:

**JCheckBox** chk = new **JCheckBox**("This is the Check Box");

This component of the *javax.swing.\*;* is added to the frame using the add(component) method of the **JFrame**class.

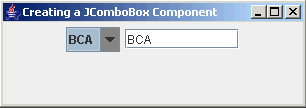
**Here is the code of the program:**

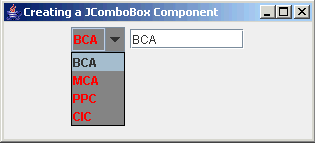
|  |
| --- |
| **import**javax.swing.\*;  **public class**CreateCheckBox{   **public static void**main(String[] args){   JFrame frame = **new**JFrame("Check Box Frame");   JCheckBox chk = **new**JCheckBox("This is the Check Box");   frame.add(chk);   frame.setSize(400, 400);   frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);   frame.setVisible(**true**);   } } |

Create a JComboBox Component in Java

In this section, you will learn about the JComboBox Component of swing in java. The JComboBox is used to display drop-down list. It provides you options to select an item from the item list. You can never select more than one item from a combo box. Combo Box can be editable or only non-editable means only readable.

This program displays a simple combo box on the frame which contains multiple items like: BCA, MCA, PPC etc. The background color of this combo box is gray and foreground color is red. Here, the background color of the combo box and the foreground color is set using the setBackground(Color) and setForeground(Color) method of the **JComboBox** class. When you select the item from the combo box then the selected item is displayed in the text box. Image of the result for the given program is as follows in the pictorial form:



And  


APIs used in the program:

**JComboBox:**This is the class which is used to create a combo box in swing using it's constructor.

itemStateChanged():  
This is the method which receives the ItemEvent generated by the addItemListener() method of the **JComboBox**class. The event is generated when you select item from the combo box.

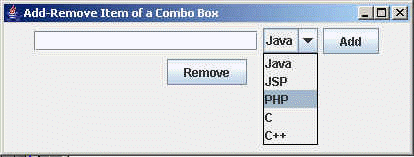
**Here is the code of program:**

|  |
| --- |
| **import**javax.swing.\*; **import**java.awt.\*; **import**java.awt.event.\*;  **public class**ComboBox{   JComboBox combo;   JTextField txt;   **public static void**main(String[] args) {   ComboBox b = **new**ComboBox();   }    **public**ComboBox(){   String course[] = {"BCA","MCA","PPC","CIC"};   JFrame frame = **new**JFrame("Creating a JComboBox Component");   JPanel panel = **new**JPanel();   combo = **new**JComboBox(course);   combo.setBackground(Color.gray);   combo.setForeground(Color.red);   txt = **new**JTextField(10);   panel.add(combo);   panel.add(txt);   frame.add(panel);   combo.addItemListener(**new**ItemListener(){   **public void**itemStateChanged(ItemEvent ie){   String str = (String)combo.getSelectedItem();   txt.setText(str);   }   });   frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);   frame.setSize(400,400);   frame.setVisible(**true**);   } } |

Combo Box operation in Java Swing

In this section, you can learn how to operate the Combo Box component, you will learn how to add items to the combo box, remove items from the combo box.

This program shows a text field, a combo box and two command buttons, first is for the adding items to the combo box and another is for the removing items from the combo. When you click on the add command button then the text of the text box is added to the combo box if the text box is not blank otherwise a message box will display with the message "Please enter text in Text Box". But when you click on the remove button the item at the 0th (zero) position of the combo box will be remove from the combo box if the combo box has one item at least otherwise a message box will display with the message "Item not available.". Following is the image for the result of the given program:



This program has used various java APIs for doing required are explained as follows:

JComboBox combo = new JComboBox(items);  
The above code has been used to create a combo box in this program. The JComboBox instance combo is created  using the constructor of the **JComboBox** class of the *javax.swing.\*;* package. This constructor holds the string array in which items for the combo box are kept.

getItemCount():  
This is the method of the **JComboBox** class which return the number of the items present is the combo box.

getItemAt(index):  
this is the method of the **JComboBox** class which returns the name of the item of the combo box at the specified position. This specification of position the item in the combo box is held by the getItemAt() method as a parameter.

showMessageDialog():  
This the method of the **JOptionPane** class of *javax.swing.\*;* package. This method displays some messages in the special dialog box. This method holds two argument in this program in which first is the parent object name and another is the message text which has to be displayed.

addItem(String):  
This is the method of the **JComboBox** class which adds items to the combo box. This method takes a string argument which is to be used to add to the combo box.

removeItemAt(index):  
This is the method of the **JComboBox** class which remove the item at the specified position of the combo box. This method holds the integer value for the position number of he of the item in combo box to remove it.

**Here is the code of the program:**

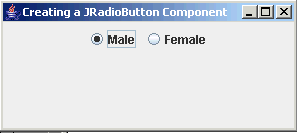
|  |
| --- |
| **import**javax.swing.\*; **import**java.awt.event.\*;  **public class**AddRemoveItemFromCombo{   JComboBox combo;   JTextField txtBox;   **public static void**main(String[] args){   AddRemoveItemFromCombo ar = **new**AddRemoveItemFromCombo();   }    **public**AddRemoveItemFromCombo(){   JFrame frame = **new**JFrame("Add-Remove Item of a Combo Box");   String items[] = {"Java", "JSP", "PHP", "C", "C++"};   combo = **new**JComboBox(items);   JButton button1 = **new**JButton("Add");   txtBox = **new**JTextField(20);   button1.addActionListener(**new**ActionListener(){   **public void**actionPerformed(ActionEvent e){   **if**(!txtBox.getText().equals("")){   **int**a = 0;   **for**(**int**i = 0; i < combo.getItemCount(); i++){   **if**(combo.getItemAt(i).equals(txtBox.getText())){   a = 1;   **break**;   }   }   **if**(a == 1)  JOptionPane.showMessageDialog(null,"Combo has already this item.");   **else**   combo.addItem(txtBox.getText());   }   **else**{   JOptionPane.showMessageDialog(null,"Please enter text in Text Box");   }   }   });   JButton button2 = **new**JButton("Remove");   button2.addActionListener(**new**ActionListener(){   **public void**actionPerformed(ActionEvent e){   **if**(combo.getItemCount() > 0)   combo.removeItemAt(0);   **else**   JOptionPane.showMessageDialog(null,"Item not available");   }   });   JPanel panel = **new**JPanel();   JPanel panel1 = **new**JPanel();   panel.add(txtBox);   panel.add(combo);   panel.add(button1);   panel.add(button2);   frame.add(panel); //  frame.add(panel1);   frame.setSize(400, 400);   frame.setVisible(**true**);   frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);    } } |

Create a JRadioButton Component in Java

In this section, you will learn how to create a radio button in java swing. Radio Button is like check box. Differences between check box and radio button are as follows:

1. Check Boxes are separated from one to another where Radio Buttons are the different-different button like check box from a same ButtonGroup.
2. You can check multiple check boxes at once but this can never done in the case of radio button. You can select only one radio button at once from a group of the radio button.
3. You can check or uncheck the check box but you can on check the radio button by clicking it once.

Here, you will see the JRadioButton component creation procedure in java with the help of this program. This example provides two radio buttons same ButtonGroup. These radio buttons represent the option for choosing male or female. Following is the image for the result of the given program:



The creation of JRadioButton are completed by the following methods:

**ButtonGroup:**This is the class of the *javax.swing.\*;* package, which is used to create a group of radio buttons from which you can select only one option from that group of the radio buttons. This class is used by creating an instance of using its constructor. Radio Buttons are added to the specified group using the add(JRadioButton) method of the **ButtonGroup** class.

**JRadioButton:**This is the class has been used to create a single radio button for the application.

**setSelected():**This method sets the value of the radio button. This method takes a boolean value either *true* or *false.* If you pass *true* value then the radio button will be selected otherwise the radio button is not selected.

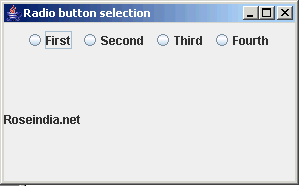
**Here is the code of program:**

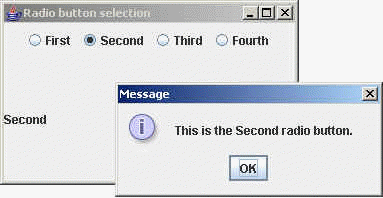
|  |
| --- |
| **import**javax.swing.\*; **import**java.awt.\*;  **public class**CreateRadioButton{   **public static void**main(String[] args) {   CreateRadioButton r = **new**CreateRadioButton();   }    **public**CreateRadioButton(){   JRadioButton Male,Female;   JFrame frame = **new**JFrame("Creating a JRadioButton Component");   JPanel panel = **new**JPanel();   ButtonGroup buttonGroup = **new**ButtonGroup();   Male = **new**JRadioButton("Male");   buttonGroup.add(Male);   panel.add(Male);   Female = **new**JRadioButton("Female");   buttonGroup.add(Female);   panel.add(Female);   Male.setSelected(**true**);   frame.add(panel);   frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);   frame.setSize(400,400);   frame.setVisible(**true**);   } } |

Selecting a Radio Button component in Java

In this section, you will learn how to set the radio buttons in a group so that only one can be selected at a time.

This program shows five radio buttons with labeled by "First", "Second", "Third", "Fourth" and "Fifth". This program also show a label which contains the text "Roseindia.net" but when you click on any radio button from a ButtonGroup the text of the selected radio button is shown on the label and a message box will be shown with message holds the selected radio button label. This is done through the generating event for the different-different radio buttons. Following are the screen shots for the result of the given program:





For this purposes, there are some APIs or methods have been used as follows:

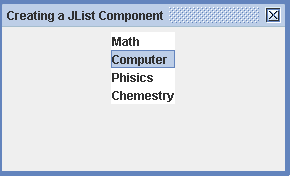
getActionCommand():  
This is the method of the **ActionEvent** class which returns the source title in string of the generated event.

**Here is the code of the program:**

|  |
| --- |
| **import**javax.swing.\*; **import**java.awt.\*; **import**java.awt.event.\*;  **public class**SelectRadioButton{   JLabel label;   **public static void**main(String[] args){   SelectRadioButton sr = **new**SelectRadioButton();   }    **public**SelectRadioButton(){   JFrame frame = **new**JFrame("Radio button selection");   JRadioButton first = **new**JRadioButton("First");   JRadioButton second = **new**JRadioButton("Second");   JRadioButton third = **new**JRadioButton("Third");   JRadioButton fourth = **new**JRadioButton("Fourth");   JRadioButton fifth = **new**JRadioButton("Fifth");   JPanel panel = **new**JPanel();   panel.add(first);   panel.add(second);   panel.add(third);   panel.add(fourth);   panel.add(fifth);   ButtonGroup bg = **new**ButtonGroup();   bg.add(first);   bg.add(second);   bg.add(third);   bg.add(fourth);   bg.add(fifth);   first.addActionListener(**new**MyAction());   second.addActionListener(**new**MyAction());   third.addActionListener(**new**MyAction());   fourth.addActionListener(**new**MyAction());   fifth.addActionListener(**new**MyAction());   label = **new**JLabel("Roseindia.net");   frame.add(panel, BorderLayout.NORTH);   frame.add(label, BorderLayout.CENTER);   frame.setSize(400, 400);   frame.setVisible(**true**);   frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);   }    **public class**MyAction **implements**ActionListener{   **public void**actionPerformed(ActionEvent e){   label.setText(e.getActionCommand());   JOptionPane.showMessageDialog(null,"This is the " + e.getActionCommand() +  " radio button.");   }   } } |

Create a JList Component in Java

In this section, you will learn how to create a JList component of swing. JList is a component of GUI. It provides the multiple items in a list and sometimes it shows the data in multiple columns in a list. Lists are used to select item from the item's group like combo box but the major difference between list and the combo box is that you can select only one item from the combo box since you can select more than one item at once from the list. You can easily add or remove items from or to the JList component of swing. The JList component has been shown in this article below:



This program provides a List. The given List has multiple subjects as items like: Math, Computer, Physics and Chemistry.

**JList:**This is the class which is used to create a list which contains items. This class extends the JComponent class in java swing. There has been using string array values in this program.

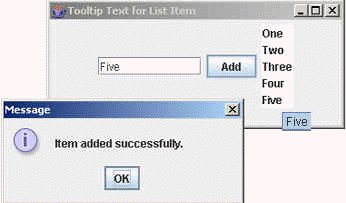
**Here is the code of program:**

|  |
| --- |
| **import**javax.swing.\*;  **public class**CreateJList{   **public static void**main(String[] args) {   String subject[] = {"Math", "Computer", "Phisics", "Chemestry"};   JFrame frame = **new**JFrame("Creating a JList Component");   JPanel panel = **new**JPanel();   frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);   JList list = **new**JList(subject);   frame.setUndecorated(**true**);   frame.getRootPane().setWindowDecorationStyle(JRootPane.PLAIN\_DIALOG);   panel.add(list);   frame.add(panel);   frame.setSize(400,400);   frame.setVisible(**true**);   } } |

Setting Tool Tip Text for items in a JList Component

In this section, you will learn how to set the tool tip text for items present in the JList component of the Java Swing. Tool Tip text is the help text of any component for user. When you rest the mouse cursor on the component then at that point a message which small font and yellow background stay there for few seconds. This text show the information about that component.

This program has used the tool tip text for items present in the JList component in Java Swing. In this program, you can add more and more items. You can enter the item name in the text box and click on the "Add" button. When you move the mouse pointer around the items in the list, it shows the specific item name as a tool tip text like the following image:



Following are some methods and APIs are explained as follows:

**JScrollPane:**This is the class of *javax.swing.\*;* package of Java Swing. This class is used to create scroll bar (Horizontal or Vertical) for any component. This program has used this for creating scroll bar for the text area. It creates scroll bar using it's constructor which holds the component name for which the scroll bar has to be created.

**DefaultListModel:**This is the class of *javax.swing.\*;* package of Java. This class is used to create a list model which is helpful for adding items for the list. This class has used own method to add items in the list.

locationToIndex():  
This is the method of the **MultiListUI** class which is imported from the *javax.swing.plaf.multi.\*;* package of Java. This method locate the item to the index where the mouse pointer points. This method takes a integer value for locating item from the list according to the given point.

getModel():  
This is the method of **JList** class which holds the list of item which are shown in the JList component of Java Swing. It returns the list model.

getElementAt(index):  
This is the method of **DefaultListModel** class which gets the item from the returned list model by getModel() method according to the given integer index no. as parameter.

addElement(String):  
This is the method of **DefaultListModel** class which adds the item into the list.

**Here is the code of the program:**

|  |
| --- |
| **import**javax.swing.\*; **import**java.awt.\*; **import**java.awt.event.\*;  **public class**TooltipTextOfList{   **private**JScrollPane scrollpane = **null**;   JList list;   JTextField txtItem;   DefaultListModel model;   **public static void**main(String[] args){   TooltipTextOfList tt = **new**TooltipTextOfList();   }    **public**TooltipTextOfList(){   JFrame frame = **new**JFrame("Tooltip Text for List Item");   String[] str\_list = {"One", "Two", "Three", "Four"};   model = **new**DefaultListModel();   **for**(**int**i = 0; i < str\_list.length; i++)   model.addElement(str\_list[i]);   list = **new**JList(model){   **public**String getToolTipText(MouseEvent e) {   **int**index = locationToIndex(e.getPoint());   **if**(-1 < index) {   String item = (String)getModel().getElementAt(index);   **return**item;   } **else**{   **return null**;   }   }   };   txtItem = **new**JTextField(10);   JButton button = **new**JButton("Add");   button.addActionListener(**new**MyAction());   JPanel panel = **new**JPanel();   panel.add(txtItem);   panel.add(button);   panel.add(list);   frame.add(panel, BorderLayout.CENTER);   frame.setSize(400, 400);   frame.setVisible(**true**);   frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);   }    **public class**MyAction **extends**MouseAdapter **implements**ActionListener{   **public void**actionPerformed(ActionEvent ae){   String data = txtItem.getText();   **if**(data.equals(""))   JOptionPane.showMessageDialog(null,"Please enter text in the Text Box.");   **else**{   model.addElement(data);   JOptionPane.showMessageDialog(null,"Item added successfully.");   txtItem.setText("");   }   }   } } |

JSlider Component of Java Swing

**[http://www.roseindia.net/images/previous.gif](http://www.roseindia.net/java/example/java/swing/LimitValueSpinner.shtml)**    **[http://www.roseindia.net/images/bt_home.gif](http://www.roseindia.net/java/example/java/swing/index.shtml)**  **[http://www.roseindia.net/images/next.gif](http://www.roseindia.net/java/example/java/swing/SwingProgressBar.shtml)**

In this section, you will learn how to create a JSlider component of Java Swing. A Slider is a Swing tool which you can use for selecting a range. There is minimum chances of being mistake to illegal input values.

In this program, events on the JSlider component have also been shown. If you increase or decrease the slider by selecting then the actual position of the slider will be displayed on a label. The figure for the result of the given program is followed below:

After drag the JSlider Component:  


For these purposes, some methods and APIs have been used to create a JSlider component and performs various tasks related to the slider. Methods and APIs are as follows:

**JSlider:**This is the class which creates the slider for the swing application. For creating the slider this class creates a instance using it's constructor JSlider().

**ChangeListener:**  
This is the interface of which is used to call stateChanged() method which receives the event generated by the slider using addChangeListener() method of the **JSlider** class.

**ChangeEvent:**  
This is the class which handle the event generated by the JSlider component on change the state.

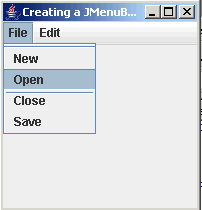
addChangeListener(object):  
This is the method of the **JSlider** class which is used to handle event on change the selected state of the JSlider component.

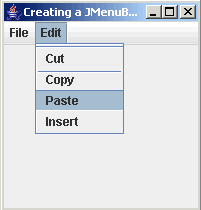
**Here is the code of the program:**

|  |
| --- |
| **import**javax.swing.\*; **import**javax.swing.event.\*; **import**java.awt.\*; **import**java.awt.event.\*;  **public class**CreateSlider{   JSlider slider;   JLabel label;   **public static void**main(String[] args){   CreateSlider cs = **new**CreateSlider();   }    **public**CreateSlider(){   JFrame frame = **new**JFrame("Slider Frame");   slider = **new**JSlider();   slider.setValue(70);   slider.addChangeListener(**new**MyChangeAction());   label = **new**JLabel("Slider Example");   JPanel panel = **new**JPanel();   panel.add(slider);   panel.add(label);   frame.add(panel, BorderLayout.CENTER);   frame.setSize(400, 400);   frame.setVisible(**true**);   frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);   }    **public class**MyChangeAction **implements**ChangeListener{   **public void**stateChanged(ChangeEvent ce){   **int**value = slider.getValue();   String str = Integer.toString(value);   label.setText(str);   }   } } |

Create menus and submenus in Java

In this section, you will learn about creation of menus, submenus and Separators in Java Swing. Menu bar contains a collection of menus. Each menu can have multiple menu items these are called submenu. Similarly, all menus have multiples menu items. The Separator divides the menu items in a separate groups like same types of menu Items are divided into a individual parts. For pictorial representation, the image for the result of the given program is given below:





This program shows how to create menu bar, menus, submenus and Separators. Here, all items shows on a frame with the help of following methods and APIs:

**JMenuBar:**This is the class which constructs a menu bar that contains several menus.

**JMenu(String):**This is the constructor of **JMenu** class. This constructor constructs the new menu. It takes the string type value which is the name label for the menu.

**JMenuItem(String):**This is the constructor of **JMenuItem** class which constructs new menu items for the specific menu. It takes string types value which is the label for the menu item.

**JSeparator():**This is the constructor of **JSeparator**class which adds an extra line between menu items. This line, only separates the menu items.

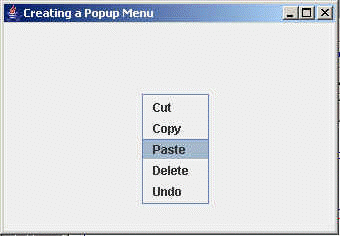
**setJMenuBar():**This method is used to set the menu bar to the specified frame. It takes the object of he **JMenuBar**class**.**

**Here is the code of program:**

|  |
| --- |
| **import**javax.swing.\*;  **public class**SwingMenu{   **public static void**main(String[] args) {   SwingMenu s = **new**SwingMenu();   }    **public**SwingMenu(){   JFrame frame = **new**JFrame("Creating a JMenuBar, JMenu, JMenuItem and seprator Component");   frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);   JMenuBar menubar = **new**JMenuBar();   JMenu filemenu = **new**JMenu("File");   filemenu.add(**new**JSeparator());   JMenu editmenu = **new**JMenu("Edit");   editmenu.add(**new**JSeparator());   JMenuItem fileItem1 = **new**JMenuItem("New");   JMenuItem fileItem2 = **new**JMenuItem("Open");   JMenuItem fileItem3 = **new**JMenuItem("Close");   fileItem3.add(**new**JSeparator());   JMenuItem fileItem4 = **new**JMenuItem("Save");   JMenuItem editItem1 = **new**JMenuItem("Cut");   JMenuItem editItem2 = **new**JMenuItem("Copy");   editItem2.add(**new**JSeparator());   JMenuItem editItem3 = **new**JMenuItem("Paste");   JMenuItem editItem4 = **new**JMenuItem("Insert");   filemenu.add(fileItem1);   filemenu.add(fileItem2);   filemenu.add(fileItem3);   filemenu.add(fileItem4);   editmenu.add(editItem1);   editmenu.add(editItem2);   editmenu.add(editItem3);   editmenu.add(editItem4);   menubar.add(filemenu);   menubar.add(editmenu);   frame.setJMenuBar(menubar);   frame.setSize(400,400);   frame.setVisible(**true**);   } } |

Create a Popup Menu in Java

Here, you will learn how to create a Popup menu in Java. Popup menu is the list of menu which is displayed at that point on the frame where you press the right mouse button. Following figure shows the popup menu when you right click on the frame:



This program illustrates you about the creation of the popup menu. Following methods and APIs have been used in this program for getting popup menu:

**JPopupMenu:**This is the class which constructs the popup menu using it's constructor. This class is helpful to add the object of the **JMenuItem** class which creates a particular menu.

**isPopupTrigger():**This is the method of the **MouseEvent** class of the *java.awt.event.\*;* package. This method returns a boolean type value either *true* or *false*. This method returns *true* if the event is generated when the popup is triggered.

**mouseReleased**This is the method of the **FormView.MouseEventListener** class which is imported from the*javax.swing.text.html.\*;* package. This method receives the generated mouse event when the object is release by clicking the mouse.

**getX():**This is the method of the **MouseEvent** class which is imported from the *java.awt.event.\*;* package. This method returns the integer type value which is the position on the x-axis for the source component where you click the mouse.

**getY():**This is also the method of the the **MouseEvent** class. This method returns the vertical positions of the y-coordinate for the source component where you click the mouse.

**show(me.getComponent(), me.getX(), me.getY()):**This is the method of **JPopupMenu** class which displays the popup menu where you press the right mouse button on the specified location or positions. This positions calculated by the**getX()**and **getY()**.

**getComponent():**This is the method of the **ComponentEvent** class of the *java.awt.event.\*;* package. This method returns the source component of the generated event.

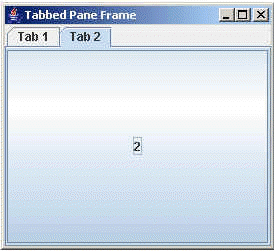
**Here is the code of program:**

|  |
| --- |
| **import**javax.swing.\*; **import**java.awt.event.\*;  **public class**PopUpMenu{   JPopupMenu Pmenu;   JMenuItem menuItem;   **public static void**main(String[] args) {   PopUpMenu p = **new**PopUpMenu();   }    **public**PopUpMenu(){   JFrame frame = **new**JFrame("Creating a Popup Menu");   frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);   Pmenu = **new**JPopupMenu();   menuItem = **new**JMenuItem("Cut");   Pmenu.add(menuItem);   menuItem = **new**JMenuItem("Copy");   Pmenu.add(menuItem);   menuItem = **new**JMenuItem("Paste");   Pmenu.add(menuItem);   menuItem = **new**JMenuItem("Delete");   Pmenu.add(menuItem);   menuItem = **new**JMenuItem("Undo");   Pmenu.add(menuItem);   menuItem.addActionListener(**new**ActionListener(){   **public void**actionPerformed(ActionEvent e){}   });   frame.addMouseListener(**new**MouseAdapter(){   **public void**mouseReleased(MouseEvent Me){   **if**(Me.isPopupTrigger()){   Pmenu.show(Me.getComponent(), Me.getX(), Me.getY());   }   }   });   frame.setSize(400,400);   frame.setVisible(**true**);   } } |

Creating a JTabbedPane Container in Java Swing

In this section, you will learn how to create the JTabbedPane container in Java Swing. The example for illustration is given in which, all the things related to the creation of JTabbedPane container have been illustrated in efficient manner.

This program has used various tools of swing to implement the JTabbed component of Java. The following figure shows the JTabbedPane component of Java Swing:



These are explained as follows:

**JTabbedPane:**This is the class of *javax.swing.\*;* package which creates the JTabbedPane component of Java Swing which contains separate button for the separate tab. You can also container component with the specific tab.

add():  
This is the method of **JTabbedPane** class which is used to add container component to the JTabbedPane component of Java Swing. This method takes different-different types of arguments but in this program this method has taken two argument, first is the title for the tab and another is the component name which is specified for a particular tab to show for that.

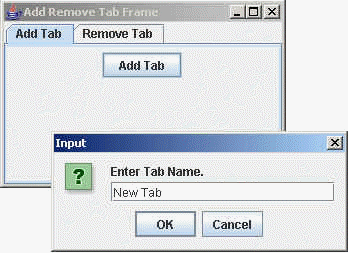
**Here is the code of the program:**

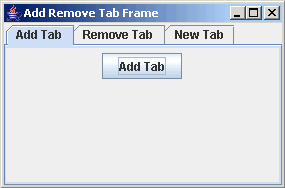
|  |
| --- |
| **import**javax.swing.\*; **import**java.awt.\*;  **public class**CreateTabbedPane{   **public static void**main(String[] args){   JFrame frame = **new**JFrame("Tabbed Pane Frame");   frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);   JTabbedPane tab = **new**JTabbedPane();   frame.add(tab, BorderLayout.CENTER);   JButton button = **new**JButton("1");   tab.add("Tab 1", button);   button = **new**JButton("2");   tab.add("Tab 2", button);   frame.setSize(400,400);   frame.setVisible(**true**);   } } |

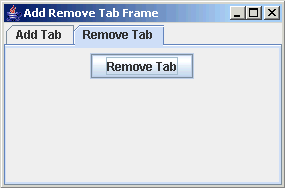
Add and Remove Tab to a JTabbedPane Container

Screen Shot for result of the program:









This section shows you how to add a tab to the JTabbedPane container and how to remove tab from a JTabbedPane container.

In this section, you will learn about adding tab to the JTabbedPane container and removing tab from the JTabbedPane container. This program shows two tabs and two panels with separate command button. The panel regarding to the "Add Tab" tab contains a command button labeled with the text "Add Tab" and another panel regarding to the "Remove Tab" also contains the "Remove Tab". If you click on the "Add Tab" button then a input box is opened and takes new tab name which has to be generated and add a new tab with the given name with a panel which contains a label labeled with the "Your program is working successfully." otherwise if you click on the "Remove Tab" button then the last tab is removed from the JTabbedPane container.

Following some method and APIs are used for adding or removing tabs from the JTabbedPane container:

JTabbedPane.remove(index):  
This is the method of the **JTabbedPane** class. This method is used to remove tab from the JTabbedPane container present at the given position to the method as a parameter.

getTabCount():  
This is the method of the **JTabbedPane** class. This method is used to count the number of tabs present in the JTabbedPane container. This method a integer value.

**Here is the code of the program:**

|  |
| --- |
| **import**javax.swing.\*; **import**java.awt.\*; **import**java.awt.event.\*; **import**java.io.\*;  **public class**AddRemoveTab{   JTabbedPane tab;   **public static void**main(String[] args){   AddRemoveTab ar = **new**AddRemoveTab();   }    **public**AddRemoveTab(){   JFrame frame = **new**JFrame("Add Remove Tab Frame");   tab = **new**JTabbedPane();   frame.add(tab, BorderLayout.CENTER);   JPanel panel = **new**JPanel();   JButton button = **new**JButton("Add Tab");   button.addActionListener(**new**MyAction());   panel.add(button);   tab.add("Add Tab", panel);   JPanel panel1 = **new**JPanel();   JButton button1 = **new**JButton("Remove Tab");   button1.addActionListener(**new**MyAction());   panel1.add(button1);   tab.add("Remove Tab", panel1);   frame.setSize(400, 400);   frame.setVisible(**true**);   frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);   }    **public class**MyAction **implements**ActionListener{   **public void**actionPerformed(ActionEvent e){   String str = e.getActionCommand();   **if**(str.equals("Add Tab")){   String st = JOptionPane.showInputDialog(null, "Enter Tab Name.");   **if**(!st.equals("")){   JPanel panel2 = **new**JPanel();   JLabel label = **new**JLabel("Your program is working successfully.");   panel2.add(label);   tab.add(st, panel2);   }   }   **else if**(str.equals("Remove Tab")){   tab.remove(tab.getTabCount()-1);   }   }   } } |

**pack() method and the setSize() method of the Java Swing.**

pack() vs. setSize() Method in Java

This section provides you an example which illustrates you about the main difference between the pack() method and the setSize() method of the Java Swing.

Here, you can see the given example provides a JFrame with some button component like ***Button 1, Button 2, Button3, Button 4, Button 5*** etc. on it. The frame is seen by using the pack() method. If you do not use the pack method frame is not resized and only the title of the frame will bee seen. To show everything on the frame, frame will be resized by the pack() method or the setSize() method of Java Swing.

The main difference between the pack() method and setSize() method of Java Swing is as follows:

pack() method gives sets the frame size as per need while the setSize() method takes two parameter in which one is for the width of the frame and another one is for the height of the frame to represent it.

**Here is the code of the program**

|  |
| --- |
| **import**java.awt.\*; **import**java.awt.event.\*; **import**javax.swing.\*;  **public class**Pack **extends**JFrame {   **private**Button button1 = **new**Button("Button 1");   **private**Button button2 = **new**Button("Button 2");   **private**Button button3 = **new**Button("Button 3");   **private**Button button4 = **new**Button("Button 4");   **private**Button button5 = **new**Button("Button 5");    **public**Pack() {   **super**("pack() vs. setSize() method Example");   **this**.addWindowListener(**new**WindowAdapter() {   **public void**windowClosing(WindowEvent e) {   System.exit(0);   }   });    Container contentPane = **this**.getContentPane();   contentPane.setLayout(**new**FlowLayout());    contentPane.add(button1);   contentPane.add(button2);   contentPane.add(button3);   contentPane.add(button4);   contentPane.add(button5);    //this.setSize(450, 90);   **this**.pack();   }    **public static void**main(String[] args) {   Pack mainFrame = **new**Pack();   mainFrame.setVisible(**true**);   }   } |

Login Form in Swing

**[http://www.roseindia.net/images/previous.gif](http://www.roseindia.net/java/example/java/swing/color-effect-image.shtml)**    **[http://www.roseindia.net/images/bt_home.gif](http://www.roseindia.net/java/example/java/swing/index.shtml)**  **[http://www.roseindia.net/images/next.gif](http://www.roseindia.net/java/example/java/swing/calculate.shtml)**

**This section illustrates you how to create a Login form.**

To create a Login Form, we have used two class files: **1) NextPage.java  
2) LoginDemo.java**

In the **LoginDemo.java,** we have created two text fields text1 and text2 to set the text for username and password. A button is created to perform an action. The method **text1.getText()** get the text of username and the method  **text2.getText()** get the text of password which the user enters. Then we have create a condition that if the value of text1 and text2 is MADHAV, the user will enter into the next page on clicking the submit button. The**NextPage.java** is created to move the user to the next page. In case if the user enters the invalid username and password , the class **JOptionPane** provides the MessageDialog to shows the error message.

**Here is the code of NextPage.java**

|  |
| --- |
| **import**javax.swing.\*; **import**java.awt.\*;   **class**NextPage **extends**JFrame {   NextPage()  {  setDefaultCloseOperation(javax.swing.   WindowConstants.DISPOSE\_ON\_CLOSE);  setTitle("Welcome");  setSize(400, 200);   }  } |

**Here is the code of LoginDemo.java**

|  |
| --- |
| **import**javax.swing.\*; **import**java.awt.\*; **import**java.awt.event.\*;   **class**Login **extends**JFrame **implements**ActionListener {  JButton SUBMIT;  JPanel panel;  JLabel label1,label2;  **final**JTextField  text1,text2;   Login()   {   label1 = **new**JLabel();   label1.setText("Username:");   text1 = **new**JTextField(15);    label2 = **new**JLabel();   label2.setText("Password:");   text2 = **new**JPasswordField(15);     SUBMIT=**new**JButton("SUBMIT");      panel=**new**JPanel(**new**GridLayout(3,1));   panel.add(label1);   panel.add(text1);   panel.add(label2);   panel.add(text2);   panel.add(SUBMIT);   add(panel,BorderLayout.CENTER);   SUBMIT.addActionListener(**this**);   setTitle("LOGIN FORM");   }  **public void**actionPerformed(ActionEvent ae)   {   String value1=text1.getText();   String value2=text2.getText();   **if**(value1.equals("MADHAV") && value2.equals("MADHAV")) {   NextPage page=**new**NextPage();   page.setVisible(**true**);   JLabel label = **new**JLabel("Welcome: "+value1);   page.getContentPane().add(label);   }   **else**{   System.out.println("enter the valid username and password");   JOptionPane.showMessageDialog(this,"Incorrect login or password",   "Error",JOptionPane.ERROR\_MESSAGE);   } } }  **class**LoginDemo {   **public static void**main(String arg[])   {   **try**   {   Login frame=**new**Login();   frame.setSize(300,100);   frame.setVisible(**true**);   }   **catch**(Exception e)   {JOptionPane.showMessageDialog(null, e.getMessage());}   } } |