EVENT HANDLING

1) Two Event Handling Mechanisms
2) The Delegation Event Model
   a. Events
   b. Event Sources
   c. Event Listeners
3) Event Classes
4) Sources of Events
5) Event Listener Interfaces
6) Adapter Classes

TWO EVENT HANDLING MECHANISMS

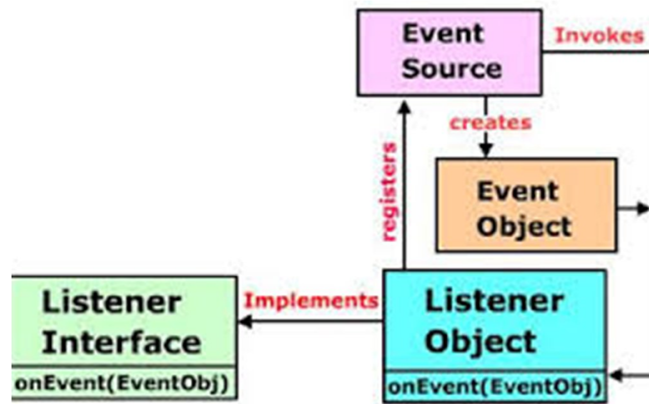
- **Event** is the change in the state of the object or source. **Events** are generated as result of user interaction with the graphical user interface components.
- **For example**, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page are the activities that causes an event to happen.
- Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism have the code which is known as event handler that is executed when an event occurs.
- The Way in which events are handled changed significantly between the original version of java i.e 1.0 and modern versions of java. The Modern approach is called the Delegation Event Model. This model defines the standard mechanism to generate and handle the events.
- The Delegation Event Model has the following key participants namely

  Source- The source is an object on which event occurs. Source is responsible for providing information of the occurred event to its handler. Java provide as with classes for source object.

  Listener - It is also known as event handler. Listener is responsible for generating response to an event. From java implementation point of view the listener is also an object. Listener waits until it receives an event. Once the event is received, the listener process the event then returns.

DELEGATION EVENT MODEL

- The Modern approach to handling events is based on the **delegation event model**, which defines standard and consistent mechanisms to generate and process events.
- Its concept is quite simple:
- A source generates an event and sends it to one or more listeners.
- In this schema, the listener simply waits until it receives an event. Once an event is received, the listener processes the event and then returns.
- The advantage of this design is that the application logic that processes events is cleanly separated from the user interface logic that generates those events.
- A User interface element is able to "delegate" the processing of an event to a separate piece of code.
  In the delegation Event model, listeners must register with a source in order to receive an event notification.
- This provides an important benefit: notifications are sent only to listeners that want or receive them.
  This is a more efficient want to handle events than the design used by the old java 1.0 approach.



- In this model, there is a source, which generates events.
- There is a Listener, which can listen to the happenings of an event and initiate an action.
- A Listener has to register with a source.
- When an event takes place, it is notified to the listeners, which are registered with the source.
- The Listener then initiates an action.

Events

- In the delegation model, an event is an object that describes a state change in a source.
- It can be generated as a consequence of a person interacting with the elements in a graphical user interface.
- Some of the activities that cause events to be generated are pressing a button,entering a character via a Keyboard, selecting an item in a list etc..
- Events may also occur that are not directly caused by interactions with a user interface. For Example: an event may be generated when a timer expires, a counter exceeds a value etc..

Event Sources

- An Event sources is a GUI Object which generates **Events.** Buttons, ListBoxes and Menus etc., are common Event sources in GUI based applications. **(or)**
- The Graphical User Interface Components that generates the Events are called **Event Sources**.
- A Source must register listener in order for the listener to receive notifications about a specific type of event.
- Each type of Event has its own Registration method. Here is the general form
- Public void addTypeListener(TypeListener el)
- Where , type is the name of the event, and el is a reference to the event listener.
- For Example, the method that registers a keyboard event listener is called **addKeyListener().**
- when an event occurs, all registered listeners are notified and receive a copy of the event object.
- A source must also provide a method that allows a listener to unregister an interest in a specific type of event. The general form of such a method is
- public void removeTypeListener(TypeListener el)

Event Listeners

- A Listener is an Object that is notified when on event occurs.
- It has two major requirements.

- First, it must have been registered with one or more sources to receive notifications about specific types of events.
- Second, it must implement methods to receive and process these notifications.

EVENT CLASSES

The Classes that represent events are called Event Classes.

EVENT CLASS	DESCRIPTION
ActionEvent	Generates when a Button is pressed, a List item is double_clicked, or a Menu item is selected.
ItemEvent	Generated when a check box or list item is clicked Also occurs when a choice selection is made or a checkable MenuItem is selected or deselected.
TextEvent	Generated when the value of a TextArea or TextField is changed.
AdujustementEvent	Generated when a ScrollBar is manipulated
ContainerEvent	Generated when a component is added to or removed form a container.
KeyEvent	Generated when inpur is received form the keyboard.
FocusEvent	Generated when a component gains or loses keyboard focus.

SOURCES OF EVENTS

An Event sources is a GUI Object which generates **Events.** Buttons, ListBoxes and Menus etc., are common Event sources in GUI based applications. **(or)**
The Graphical User Interface Components that generates the Events are called **Event Sources**.
Some of the User Interface Components that can generate Events are

EVENT SOURCE	DESCRIPTION
Button	Generates Action Events when the Button is Pressed
CheckBox	Generates Item Events when the checkbox is Selected or Deselected.
Choice	Generates Item Events when the choice is changed.
List	Generates Action Events when an item is DoubleClicked. Generates Item Events when a Item is Selected or Deselected.
MenuItem	Generates Action Events when an Menu item is Selected. Generates Item Events when a Checkable Menu Item is Selected and Deselected.
ScrollBar	Generates Adjustment Events when the scroll bar is manipulated.
TextComponent	Generates text events when the user enters a character.
Window	Generates window Events when a window is activated, closed, deactivated, deiconified, iconified, opened or quit.

A Source must register listener in order for the listener to receive notifications about a specific type of event.
Each type of Event has its own Registration method. Here is the general form

```
Public void addTypeListener(TypeListener el)
```

EVENTLISTENER

When an Event occurs, the event source invokes the appropriate method defined by the listener and provides an event object as its argument.
The below table lists commonly used listener interfaces and provides a brief description of methods that they define.

INTERFACE	DESCRIPTION
ActionListener	Defines one method to receive action events
ItemListener	Defines one method to recognize when the state of an item changes.
TextListener	Defines one method to recognize when a text value changes.
AdjustementListener	Defines one method to receive adjustment event.
ContainerListener	Defines two method to recognize when a component is added to or removed from a container.
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed.
FocusListener	Defines two methods to recognize when a component gains or losses keyboard focus.

EVENT SOURCE	EVENT CLASS	EVENT LISTENER	METHODS IN LISTENER INTERFACE
Button Clicked MenuItem Selected	ActionEvent	ActionListener	void actionPerformed(ActionEvent ae)
Combo box item selected	ActionEvent ItemEvent	Action Listener ItemListener	void actionPerformed(ActionEvent ae) void itemStateChanged(ItemEvent ie)
List Item Selected	ListSelectionEvent	ListSelectionListener	void valueChanged(ListSelectionEvent le)
RadioButton Selected	ActionEvent ItemEvent	Action Listener ItemListener	void actionPerformed(ActionEvent ae) void itemStateChanged(ItemEvent ie)
Check Box Selected	ActionEvent ItemEvent	Action Listener ItemListener	void actionPerformed(ActionEvent ae) void itemStateChanged(ItemEvent ie)
Scroll Bar Repositioned	AdjustmentEvent	AdustmentListener	void adjustmentValueeChanged(AdjustmetEvent ae)
Window Changed	WindowEvent	WindowListener	void windowAcitivated(WindowEvent we) void windowClosed(WindowEvent we) void windowClosing(WindowEvent we) void windowdeactivated(WindowEvent we) void windowDeiconified(WindowEvent we) void windowIconified(WindowEvent we) void windowOpened(WindowEvent we)
Focus Changed	FocusEvent	FocusListener	void focusLost(FocusEvent fe) void focusGain(FocusEvent fe)
Key Pressed	KeyEvent	KeyListener	void keyPressed(KeyEvent ke) void keyReleased(KeyEven tke) void keyTyped(KeyEvent ke)
Mouse clicked	MouseEvent	MouseListener	void mouseClicked(MouseEvent me) void mouseEntered(MouseEvent me) void mouseExited(MouseEvent me)

			void mousePressed(MouseEvent me) void mouseReleased(MouseEvent me)
Mouse moved	MouseEvent	MouseMotionListiner	void mouseDragged(MouseEvent me) void mouseMoved(MouseEvent me)
Text value is changed	TextEvent	TextListener	void textValueChanged(TextEvent te)

Adapter Classes

- Java Provides a special feature, called an adapter class, that can simplify the creation of event handlers in certain situations.
- An Adapter class provides an empty implementation of all methods in an event listener interface.
- Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface.
- Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface.
- Commonly used Listener interfaces implemented by Adapter Classes are

Adapter Class	Listener Interface
ComponentAdapter	ComponentListener
CaontainerAdapter	ContainerListener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MousrListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener

Here's a mouse adapter that beeps when the mouse is clicked

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

public class MouseBeeper extends MouseAdapter  {

  public void mouseClicked(MouseEvent evt) {
    Toolkit.getDefaultToolkit().beep();
  }

}
```

Without extending the MouseAdapter class, I would have had to write the same class like this

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


public class MouseBeeper implements MouseListener  {

  public void mouseClicked(MouseEvent evt) {
```

```
  Toolkit.getDefaultToolkit().beep();
 }

 public void mousePressed(MouseEvent evt) {}
 public void mouseReleased(MouseEvent evt) {}
 public void mouseEntered(MouseEvent evt) {}
 public void mouseExited(MouseEvent evt) {}

}
```

SWING

1) The Origin of Swing
2) Swing is built on the AWT
3) Two Key Features of Swing
4) Swing Components and Containers
5) A Simple Swing Application
6) Event Handling
7) Create a Swing Applet

THE ORIGIN OF SWING

- Swing is a set of classes that provide more powerful and flexible GUI Components than AWT(Abstract Window Toolkit).
- The AWT defines a basic set of controls, windows and dialog boxes that support a usable, but limited graphical interface.
- It was a response to deficiencies present in AWT.
- One reason for limited nature of the AWT is that , it translates its various visual components into their platform-specific equivalents i.e the look and feel of a component is defined by the platform, not by java.
- Because AWT components use native code resourcess, they are referred as **heavyweight.**
- Several problems of AWT are
  - First, because of variations between OS, a component might look, or even act differently on different platform.
  - Second, the look and feel of each component was fixed and could not be easily changed.
  - Third, the use of heavy weight components caused some frustrating restrictions like heavy weight component is always rectangular and opaque.
- To overcome the limitations and restrictions of AWT, a better approach is needed and the solution was Swing, which is introduced in 1997.

SWING IS BUILT ON THE AWT

- Although Swing eliminates a number of limitations inherent in the AWT, swing does not replace it.
- Instead, swing is built on the foundation of AWT. This is the reason why the AWT is still a crucial part of java.
- Swing also uses the same event handling mechanism as the AWT.
- Therefore, a basic understanding of the AWT and of event handling is required to use swing.

TWO KEY FEATURES OF SWING

- Swing was created to address the limitations present in the AWT. It does this through two key features
  1) Lightweight components
  2) Pluggable look and feel.
- Together, they provide elegant, yet easy –to-use solution to the problems of the AWT.

1) **Swing components are LightWeight**
   - Swing components are lightweight, means that they are written entirely in java and do not map directly to platform-specific peers.
   - Because light weight components do not translate into native peers, the look and feel of each component is determined by swing, not by underlying operating system.
   - i.e Each component will work in a consistent manner across all platforms.

2) **Swing supports a pluggable look and feel**
   - Swing supports a pluggable look and feel(PLAF). Since swing follows MVC architecture, it is possible to separate the look and feel of the component from the logic of the component.
   - Separating out the look and feel provides a significant advantage
   - It becomes possible to change the way that a component is rendered without affecting any of its other aspects.
   - In other words, it is possible to "plug in" a new look and feel for any given component without creating any side effects in the code that uses that component.

COMPONENTS AND CONTAINERS

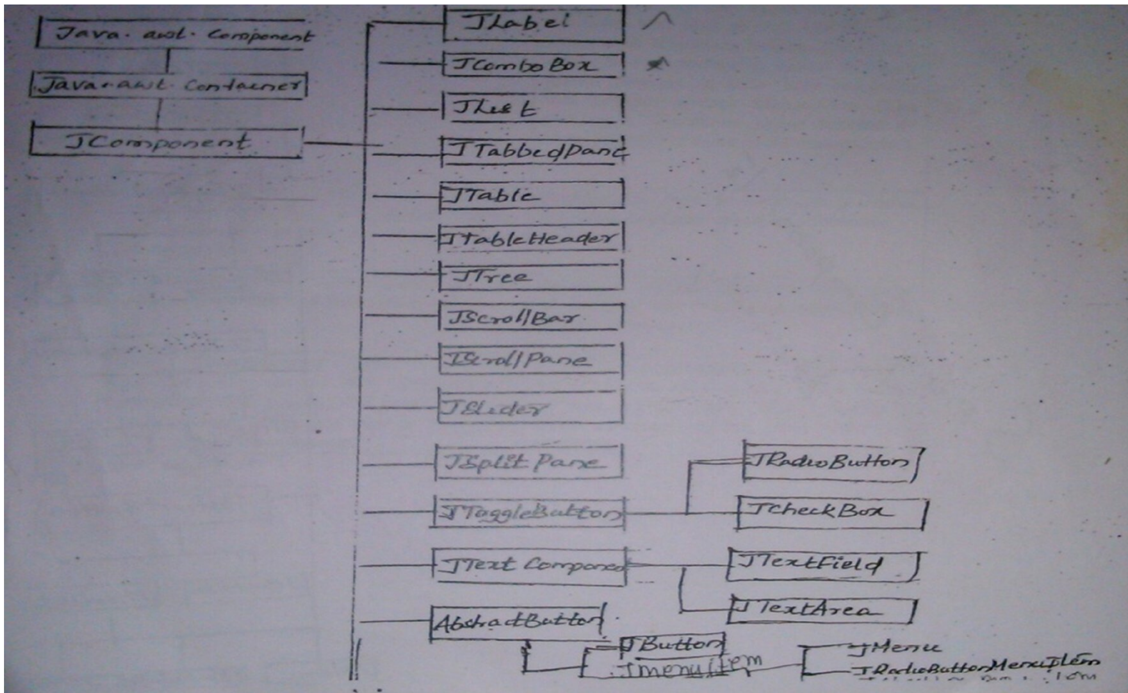
Swing GUI consists of two key items

1. Components
2. Containers

- However, this distinction is mostly conceptual because all containers are also Components
- A Component is an independent visual control, such as a push button or radio button. A Container holds a group of components.
- Thus, a container is a special type of component that is designed to hold other components. Furthermore, in order for a component to be displayed, it must be held within a container.
- Thus, all Swing GUIs will have at least one container.
- Because containers are components, a container can also hold other containers.
- This enables Swing to define what is called a Containment Hierarchy, at the top of which must be a top-level container.

1) **Components**

In general, Swing components are derived from JComponent class. JComponent provides the functionality that is common to all components. All of Swing's components are represented by classes defined within the package javax.swing. Notice that all component classes begin with the letter J.

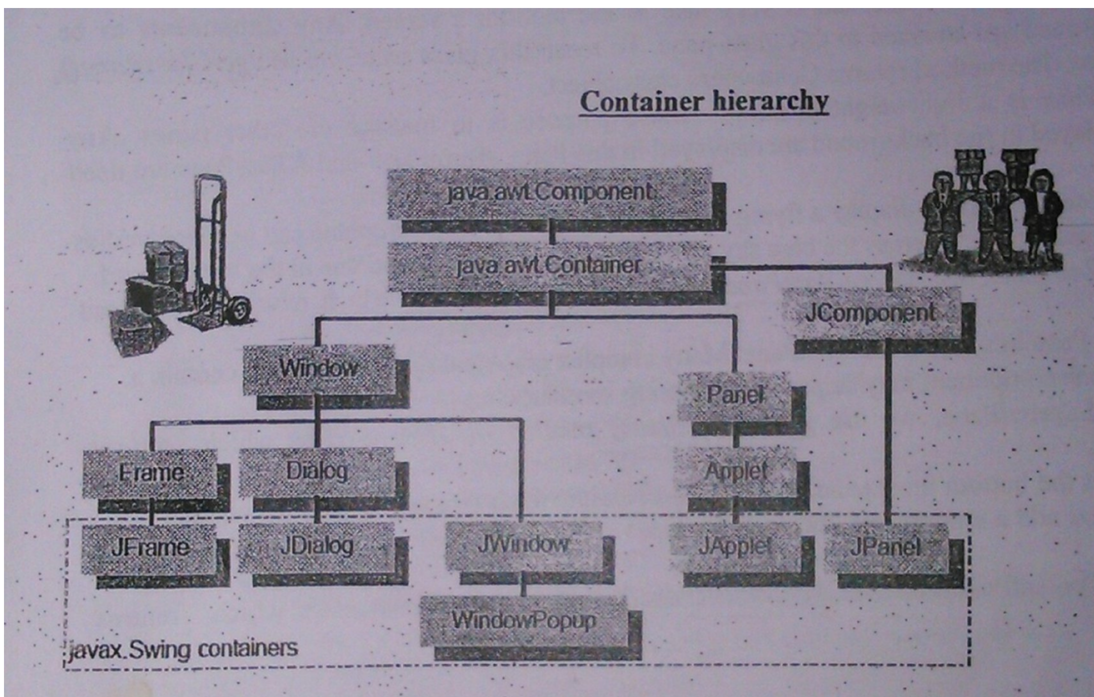


2) <u>Containers</u>

Swing defines two types of Containers.

2.1) Top-level containers (or) Heavy Weight Containers

2.2) Non Top-level Containers (or) Light Weight Containers



1) **Top-level containers (or) Heavy Weight Containers**
   - The First type of containers supported by Swing are heavy Weight containers (or) Top-level containers:
   - They are JFrame, JApplet, JDialog, JWindow.
   - These containers do not inherit JComponent. They do, however, inherit the AWT classes Component and Container. So, we call them as Heavy Weight Containers. This makes the top-level containers a special case in the Swing component library.
   - Top-level containers are defined as those which can be displayed directly on the desktop.
   - The one most commonly used for applications is JFrame. The one used for Applets is JApplet.
2) **Non Top-level Containers (or) LightWeight Containers**
   - The second type of Containers supported by Swing are Light Weight Containers (or) Non top-level Containers.
   - JPanel comes under Light Weight Containers because it inherits from JComponent.
   - Light Weight Components are often used to organize and manage groups of related controls that are contained within an outer container.
1) **Top-level Container Panes**

Each top-level container defines a set of window Panes. A Window Pane represents a free area of a window where some text or component can be displayed. We have 4 types of window panes available in javax.Swing package.

These panes can be imagined like transparent sheets lying one below the other.

- The Four Panes are
  1) JGlassPane
  2) JRoot Pane
  3) JLayeredPane
  4) JContentPane

1) **JGlassPane:** This is the Firtst pane and is very lose to the monitor's screen. Any components to be displayed in the foreground are attached to this glass pane. To reach this glass pane, we use getGlassPane() method of JFrame class. The method returns Component class object.
2) **JRootPane:** JRootPane is a lightweight container whose purpose is to manage the other panes. Any components to be displayed in the background are displayed in this Pane. JRootPane and JGlassPane are used in animation also

   ForEg: Suppose we want to display a flying aero plane in the sky. The aeroplane can be displayed as a.gif of.jpg file in the glass pane whereas the blue sky can be displayed in the JRootPane in the background.

   To reach this Rootpane, we use getRootPane() method of JFrame class which returns Component class object.
3) **JLayeredPane:** This Pane lies below the RootPane. Many complex graphical applications will contain a number of layers(e.g. one component may be partially covering another, etc.)

   To reach this LayeredPane, we use getLayeredPane() method of JFrame class which returns Component class object.
4) **JContentPane:** This is the bottom most pane of all. The pane with which you will add visual components. In other words, when you add a component, such as a button, to a top-level container, you will add it to the content pane.

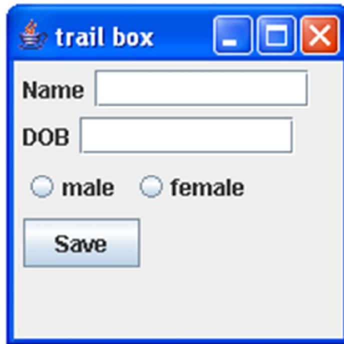
To reach this ContentPane, we use getContentPane() method of JFrame class which returns container class object

ASIMPLE SWING APPLICATION

The best way to understand the structure of a swing program is to work through an example. There are two types of java programs in which swing is typically used. The first is a Desktop application. The second is the applet. A small example to illustrate the create of swing application is

```
import java.awt.*;
import javax.swing.*;
class Demo2 extends JFrame
{
	JLabel i1,i2;
	JTextField t1,t2;
	Container con;
	ButtonGroup rbg;
	Demo2()
	{
		setSize(400,400);
		con=getContentPane();
		con.setLayout(new FlowLayout(FlowLayout.LEFT));
		i1=new JLabel("Name");
		t1=new JTextField(10);
		i2=new JLabel("DOB");
		t2=new JTextField(10);
		rbg=new ButtonGroup();
		JRadioButton rb1=new JRadioButton("male");
		JRadioButton rb2=new JRadioButton("female");
		con.add(i1);
		con.add(t1);
		con.add(i2);
		con.add(t2);
		rbg.add(rb1);
		rbg.add(rb2);
		con.add(rb1);
		con.add(rb2);
		JButton b1=new JButton("Save");
		con.add(b1);
	}
}
class Demo3
{
	public static void main(String args[])
	{
		Demo2 ob=new Demo2();
		ob.setTitle("trail box");
		ob.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		ob.setVisible(true);
	}
}
```

OUTPUT



EVENT HANDLING

- The Event Handling mechanism used by Swing is the same as that used by the AWT. This approach is called the Delegation Event Model.
- In many cases , swing uses the same events as does the AWT, and these events are packaged in java.awt.event. Events specific to swing are stored in javax.swing.event.
- Although events are handled in swing in the same way as they are with the AWT, it is still useful to work through a simple example.
- The following program handles event generated by a swing push button

```
import java.awt.event.*;
import javax.swing.*;
import java.applet.*;
/*
<applet code=Demo1 width=200 height=200>
</applet>
*/
public class Demo1  extends  JApplet implements ActionListener
{
    JButton b1,b2;
    public void init()
    {
            b1 = new JButton("Alpha");
            b2 = new JButton("Beta");
            b1.addActionListener(this);
            b2.addActionListener(this);
            add(b1);
            add(b2);
    }
            public void actionPerformed(ActionEvent ae)
            {
                    if (ae.getSource() == b1)
                            showStatus("Alpha is pressed");
                    else
                            showStatus("Beta is pressed");
            }

    }
```

OUTPUT

CREATE A SWING APPLET

- The second type of program that commonly uses swing is the applet. Swing-based applets are similar to AWT-based applets, but with important difference.
- A swing applet extends JApplet rather than Applet. JApplet is derived from Applet. Thus, JApplet includes all of the functionality found in Applet and adds support for swing.
- JApplet is a top-level container, which means that it is not derived from JComponent. Because JApplet is a top-level container, it includes the various panes and the components are added to JApplets content pane in the same way that components are added to JFrame's contentpane.
- A Simple Applet Program

```
import java.awt.*;
import javax.swing.*;
import java.applet.*;
/*
<applet code=AppletDemo width=300 height=300>
</applet>
*/
public class AppletDemo  extends JApplet
{

      public void paint(Graphics g)
      {
                  g.drawString("Hello , This is a Simple Applet Program", 20,20);
      }

}
```

OUTPUT

EXPLORING SWING

a) **JLabel and ImageIcon**
b) **JTextField**
c) **The swing Buttons**
   I) **Jbutton**
   II) **JRadioButton**
   III) **JToggleButton**
   IV) **JCheckBox**
d) **JTabbedPane**
e) **JScrollPane**
f) **JList**
g) **JComboBox**
h) **JTrees**
i) **JTable**

a) JLabel and ImageIcon

- JLabel is Swing's easiest-to-use component. It is a passive component In that it does not respond to user input. The text can be changed by the application and not by the user. A JLabel can be used to display text and/or an icon.
- The JLabel has the following int type constants that indicate the alignment of the labels content.
- **JLabel.CENTER,JLabel.LEFT, JLabel.RIGHT,JLabel.TOP,JLabel.BOTTOM**
- JLabel defines several constructors. Here are 3 of them

  JLabel(Icon i)// Creates a label using the Icon i
  JLabel(String str)//Creates a Label with the specified String str.
  JLabel(String str, Icon i, int align)// Creates a Label using the icon I, String Str and With the specified Alignment.
- JLabel class has number of methods. Some of them are

  Icon getIcon()//Returns the icon of the Label
  String getText()//Returns the text of the Label
  Void setFont(Font font)//Sets the font for the Label's text
  Void setText(String str)//Sets the specified String str as the Label's content.

Program to Create JLabel on JFrame
//PROGRAM IILUSTRATES THE USE OF JLabel CLASS

```
import java.awt.*;
import javax.swing.*;
class lbl extends JFrame
{
   JLabel I1,I2,I3;
   Container con;
   Icon Img1,Img2,Img3;
   lbl()
   {
          setSize(400,400);
          con=getContentPane();
          con.setLayout(new FlowLayout(FlowLayout.LEFT));
          Img1=new ImageIcon("lion.jpg");
          Img2=new ImageIcon("giraffee.jpg");
```

```
            Img3=new ImageIcon("PBear.jpg");
            l1=new JLabel("LION",Img1,JLabel.LEFT);
            l2=new JLabel("GIRAFFE",Img2,JLabel.LEFT);
            l3=new JLabel("POLAR BEAR",Img3,JLabel.LEFT);
            con.add(l1);
            con.add(l2);
            con.add(l3);
    }
}
class JLabelDemo
{
    public static void main(String args[])
    {
            lbl ob=new lbl();
            ob.setTitle("JLabel in JFrame");
            ob.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            ob.setVisible(true);
    }
}
```

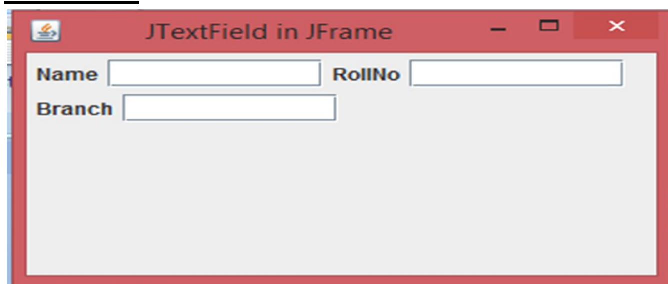
OUTPUT:

b) JTextField

- The JTextField class implements a single-line text-entry area, usually called an edit control. TextField allow the user to enter Strings. JTextField is a subclass of JTextComponent, which is a subclass of JComponent/
- The Alignment of the text is defined by the following int type constants.
- **TextField.LEFT, JTextField.CENTER, JTextField.RIGHT**
- JTextField defines the following constructors.
  JTextField() // Creates a new Text field; the text is set to null and the number of columns Is set to 0.
  JTextField(int columns) //Creates a new empty textfirld with the specified number of columns.
  JTextField(String str) // Creates a new Text field with the specified string as text.
- JTextField class has number of methods. Some of them are
  String getText() // Returns the text contained in this Text Field.
  String getSelectedText() // Returns the selected text contained in this text field.
  Void setEditable(Boolean edit) // Sets the textfield to editable(true) or not editable(false)
- **Program to create JTextField on JFrame**

```
//PROGRAM IILUSTRATES THE USE OF JTextField CLASS
Import java.awt.*;
Import javax.swing.*;
Class Txt extends JFrame
{
   JLabel I1,I2,I3;
   JTextField t1,t2,t3;
   Container con;
   Txt()
   {
           setSize(350,200);
           con=getContentPane();
           con.setLayout(new FlowLayout(FlowLayout.LEFT));
           I1=new JLabel("Name");
           t1=new JTextField(10);
           I2=new JLabel("RollNo");
           t2=new JTextField(10);
           I3=new JLabel("Branch");
           t3=new JTextField(10);
           con.add(I1);
           con.add(t1);
           con.add(I2);
           con.add(t2);
           con.add(I3);
           con.add(t3);
   }
}
class JTextFieldDemo
{
   public static void main(String args[])
   {
           Txt ob=new Txt();
           ob.setTitle("JTextField in JFrame");
           ob.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
           ob.setVisible(true);
   }
}
```

OUTPUT:



c) **The swing Buttons**

I) Jbutton

- The JButton is a concrete subclass of AbstractButton which is a subclass of JComponent. The counterpart of JButton in AWT is Button. Perphaps the most widely used control is the push button.
- A Push button is a component that contains a label and that generates an event when it is pressed.
- **JButton defines four constructors**

  JButton() \\ constructs a button with no label.
  JButton(String Str**)** \\ constructs a button with a specified label.
  JButton(Icon i) \\ constructs a button with the icon I as button
  JButton(String str, icon i) \\ constructs a button with the icon I as button and the string as Label
- **JButton has several methods. Some of them are**

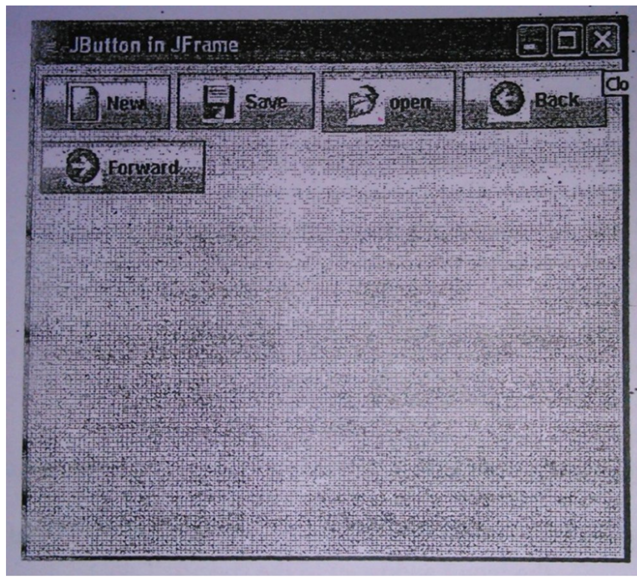
  void setText(String str) \\ sets the buttons label to the specified string
  void getText() \\ Returns the label of the button
  Icon getIcon() \\ Returns the icon of the button
  void setToolTipText(String str) \\ Sets the Tool Tip text to the button
  void setMinemonic(char c) \\ sets Shortcut key to the button
- **Program to Create JButton in JFrame**

```
//PROGRAM IILUSTRATES THE USE OF JButton CLASS
import java.awt.*;
import javax.swing.*;
class Btn extends JFrame
{
    JButton b1,b2,b3,b4,b5;
    Btn()
    {
        setSize(400,350);
        Container con=getContentPane();
        con.setLayout(new FlowLayout(FlowLayout.LEFT));
        Icon Img1=new ImageIcon("new.jpg");
        Icon Img2=new ImageIcon("save.jpg");
        Icon img3=new ImageIcon("open.jpg");
        Icon img4=new ImageIcon("back.jpg");
      Icon Img5=new ImageIcon("forward.jpg");

      b1=new JButton("New",Img1);
      b2=new JButton("Save",Img2);
      b3=new JButton("open",Img3);
      b4=new JButton("Back",Img4);
      b5=new JButton("Forward",Img5);
      b1.setToolTipText("New");
      b2.setForeground(Color.red);
      b3.setMnemonic('c'); // Shortcut key for save Button(ALT+C)
      con.add(b1);
      con.add(b2);
      con.add(b3);
      con.add(b4);
```

```
con.add(b5);
}
}
Class JButtonDemo
{
        public static void main(String args[])
        {
         Btn ob=new Btn();
         ob.setTitle("JButton in JFrame");
         ob.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         ob.setVisible(true);
        }
}
```

OUTPUT:



II) JRadioButton

- Radio buttons are like check boxes. In Radio button, theselection is displayed in a round graphics. Radio buttons are generally used to represent a collection of mutually exclusive options i.e, out of several options,only one will be selected state and all the remaining are in deselected state. The radio buttons are created using JRadioButton class, which is subclass of JToggleButton.
- The JRadioButton must be placed in a Button Group. The Button group is created using the ButtonGroup class which has no argument constructor. After creating JRadioButton, the radio buttons are to be placed to the ButtonGroup using add() method.
- If the JRadio buttons are not grouped using Buttongroup, then each radio buttn will behave exactly like JCheckBox.
- JRadioButton generates ActionEvent, ItemEvent and ChangeEvent
- JRadioButton defines several constructors

JRadioButton() // Creates a radio button without any label; the Radio Buton is set to deselected state

JRadioButton(String str)//Creates a radio button with the string str as label; the ardio button is set to deselected state
JRadioButton(String str, Boolean state)//Creates a radio button with the string str as label; the radio button is set to the specified state
JRadioButton(Icon i)// Creates a radio button using the icon I; the radio button is set to deselected state
JRadioButton(Icon I, Boolean state)//Creates a radio button using the icon i;the radio button is set to deselected state.
JRadioButton(String str,Icon i)// Creates a radio button using the icon I with the string str as label
JRadioButton(String str, Icon I, Boolean state)//Creates a radio button using the Icon I with the string str set as label; the radio button is set to the specified state.

Note: JRadioButton is like a JcheckBox . To use JRadioButton in mutually exclusive selection of One option out of many options, it is to be grouped using Buttongroup. All JradioButtons are to be added to the ButtonGroup. All JRadiobuttons are to be added to the ButtonGroup.

- Program to create JRadioButton in JFrame

```
import java.awt.*;
import javax.swing.*;
class RButton extends JFrame
{
	JRadioButton rb1,rb2,rb3,rb4,mb,fb;
	ButtonGroup rbg;
	JLabel l1,l2;
	RButton()
	{
		setSize(600,300);
		Container c = getContentPane();
		c.setLayout(new FlowLayout());
		l1 = new JLabel("Buttons to be added");
		l2 = new JLabel("Buttons Not added to group");
		mb = new JRadioButton("Male");
		fb = new JRadioButton("Female");
		rb1= new JRadioButton("Times New Roman");
		rb2 = new JRadioButton("Courier");
		rb3 = new JRadioButton("Terminal");
		rb4 = new JRadioButton("Arial");
		rbg = new ButtonGroup();
		c.add(l1);
		c.add(rb1);
		c.add(rb2);
		c.add(rb3);
		c.add(rb4);
		rbg.add(rb1);
		rbg.add(rb2);
		rbg.add(rb3);
		rbg.add(rb4);
		c.add(l2);
		c.add(mb);
		c.add(fb);
	}
}
class JRadioButtonDemo
{
```

```
        public static void main(String args[])
        {
                RButton ob = new RButton();
                ob.setTitle("JRadioButton in Jframe");
                ob.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                ob.setVisible(true);
        }
}
```

OUTPUT

III) JToggleButton

- A useful variation on the push button is called a toggle button. A Toggle button looks like a push button, but it acts differently because it has two states: Pushed and Release.
- i.e, when you press a toggle button a second time, it releases(pops up);
- Therefore, each time a toggle button is pushed, it toggles between its two states.
- Toggle Buttons are objects of the JToggleButton class. JToggleButton implements AbstractButton.
- In addition to creating standard toggle buttons, JToggleButton is a superclass for two other Swing components that also represent tow-state Swing. These are JCheckBox and JradioButton.
- JToggleButton defines the following constructors

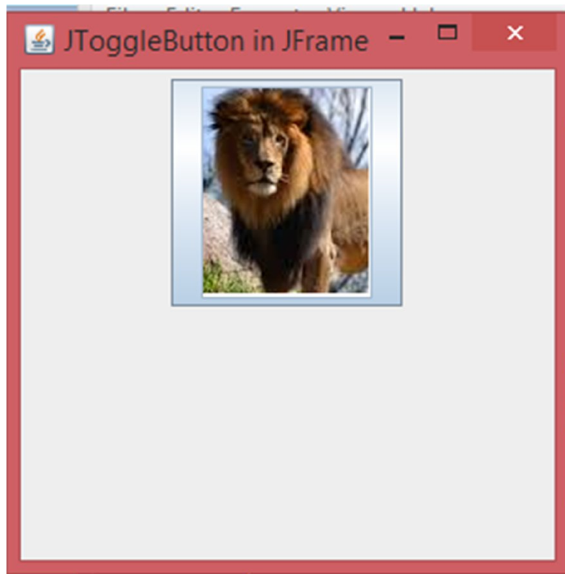
  JToggleButton()\\ Creates a toggle button without a label and is set to deselected state.
  JToggleButton(Icon i) // Creates a ToggleButton using the icon and is set to deselected state.
  JToggleButton(Icon I, Boolean State) //Creates a toggle button using the icon I and is set to Specified state.
  JToggleButton(String str) //Crates a ToggleButton with the specified label and is set to Deselected state.
  JToggleButton(String str,Icon I,Boolean state) \\ Creates a toggle button the specified label Using the icon I and is set to the specified State.
- JToggleButton has so many methods.Some of them are

  Object getItem() \\ Returns the reference of the JtoggleButton]
  Boolean isSelected \\ returns true if button is selected otherwise false.
- **Program to Create JToggleButton in JFrame**

```
// JToggleButton in JFrame
import java.awt.*;
import javax.swing.*;
class TButton extends JFrame
 {
    JToggleButton tb1;
    Icon img1;
```

```
          TButton()
          {
               setSize(300,300);
               Container c=getContentPane();
               c.setLayout(new FlowLayout());
               img1=new ImageIcon("lion.jpg");
               tb1=new JToggleButton(img1);
               c.add(tb1);
                                                                   }
     }
     class JToggleButtonDemo
     {
         public static void main(String args[])
          {
                TButton ob=new TButton();
                 ob.setTitle("JToggleButton in JFrame");
                 ob.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                 ob.setVisible(true);
          }
     }
```

OUTPUT:



IV JCheckBox

The JCheckBox class provides the functionality of a check box. Its immediate superclass is JToggleButton, which provides support for two-state buttons.

JCheckBox defines the following constructors

JCheckBox() //Creates an initially unselected check box button with no text, no icon.
JCheckBox(Icon icon)//Creates an initially unselected check box with an icon.
JCheckBox(String text)//Creates an initially unselected check box with text.

JCheckBox(Icon icon, boolean selected)//Creates a check box with an icon and specifies whether or not it is initially selected.

```
import java.awt.*;
import javax.swing.*;
class Check extends JFrame
{
 JCheckBox r,s,m;
 JLabel l1,l2;
 Check()
 {
		setSize(600,300);
		Container c = getContentPane();
		c.setLayout(new FlowLayout());
		l1 = new JLabel("Hobbies");
		r = new JCheckBox("Reading");
		s = new JCheckBox("Singing");
		m = new JCheckBox("Listening Music");
		c.add(l1);
		c.add(r);
		c.add(s);
		c.add(m);
 }
}
class JCheckBoxDemo
{
 public static void main(String args[])
 {
 Check ob = new Check();
 ob.setTitle("JCheckBox in Jframe");
 ob.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 ob.setVisible(true);
 }
}
```

OUTPUT

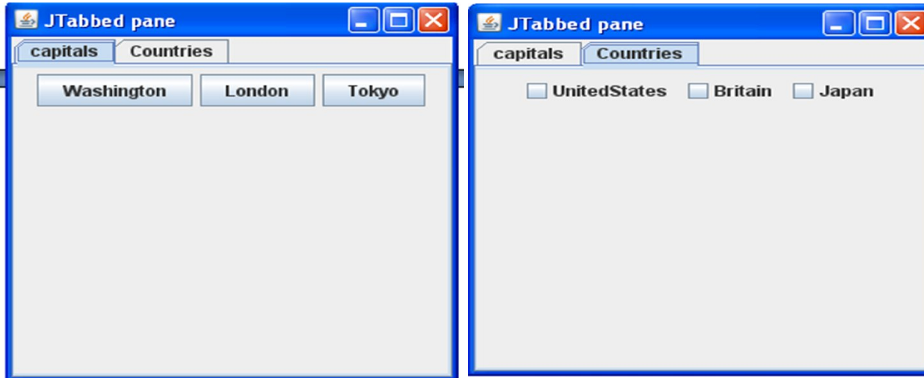
d) JTabbedPane Class

- A pane represents a frame area. A tabbed pane represents a frame with tabs attached to it. JTabbedPane is useful to create a tabbed pane, such that on each tab sheet a group of components can be added.
- JTabbedPane defines 3 constructors. Default constructor creates an empty control with the tabs positioned across the top of the pane.
- Tabs are added by calling addTab() method.

```
import java.awt.*;
import javax.swing.*;
class JTabbedPanelDemo extends JFrame
{
	JTabbedPanelDemo()
	{
		Container c = getContentPane();
		JTabbedPane jtp = new JTabbedPane();
		jtp.addTab("capitals", new CapitalsPanel());
		jtp.addTab("Countries", new CountriesPanel());
		c.add(jtp);
	}
	public static void main(String args[])
	{
		JTabbedPanelDemo demo = new JTabbedPanelDemo();
		demo.setTitle("JTabbed pane");
		demo.setSize(300,400);
		demo.setVisible(true);
		demo.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}
}
class CapitalsPanel extends JPanel
{
	CapitalsPanel()
	{
		JButton b1 = new JButton("Washington");
		JButton b2 = new JButton("London");
		JButton b3 = new JButton("Tokyo");
		add(b1);
		add(b2);
		add(b3);
	}
}

class CountriesPanel extends JPanel
{
	CountriesPanel()
	{
		JCheckBox c1 = new JCheckBox("UnitedStates");
		JCheckBox c2 = new JCheckBox("Britain");
		JCheckBox c3 = new JCheckBox("Japan");
		add(c1);
		add(c2);
		add(c3);
	}
}
```

OUTPUT



f) JList

- JList is a subclass of JComponent. JList creates a graphical display of a list of items and allows the user to select one or more items.
- JList class provides a compact, multiple-choice, scrolling selection list.
- JList provides these constructors

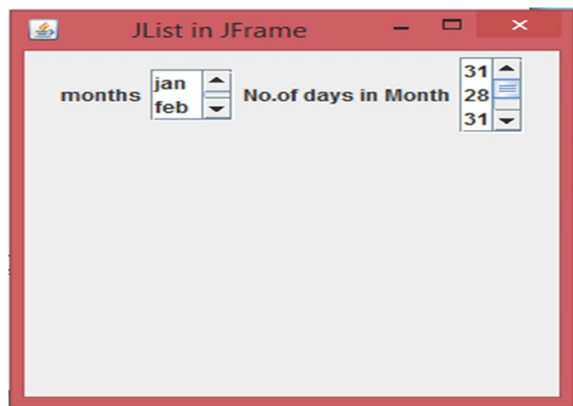
  JList()//Creates a list and only one entry will be visible always
  JList(ListModel lm)//creates a list with items contained in the list model lm; the list model is an instance of DefaultList Model
- A swing list does not have a scrollbar to display the list. Hance, the list is to be placed inside a JScrollPane object.
- A JList is created by passing vector or an array of object as argument. In JList, there is no method for adding or removing items in the list. i.e.,Lists created by JList by passing an array or vector do not have many methods to manipulating the items in the list.
- The DefaultListModel, which is a concrete subclass of AbstractListModel has methods for adding and removing items from a JList.
- A List with DefaultListModel is created in the following steps
  Create an instances of DefaultListModel
  Add the items to this model
  Create the Jlist by passing this model as argument in the constructor

  For eg :
  DefaultListModel lm = new DefaultListModel();
  lm.addElement("January");
  lm.addElement("February");
  JList month = new JList(lm);
- Some of the methods of DefaultListModel are
  void addElement(Object ob)\\ adds the given object at the end of the list
  void clear()\\removes all elements from the list
  int getSize()\\returns the number of items in the list
  void InsertElementAt(Object ob,int index)\\Inserts the given object at the specified index

```
import java.awt.*;
import javax.swing.*;
class Lst extends JFrame
{
	JList lst1,lst2;
	String Months[] = {"jan","feb","mar","apr","may","june","july","aug","sep","oct"};
	Lst()
	{
```

```
        Container c = getContentPane();
        c.setLayout(new FlowLayout());
        lst1 = new JList(Months);
        DefaultListModel lm = new DefaultListModel();
        lm.addElement(31);
        lm.addElement(28);
        lm.addElement(31);
        lm.addElement(30);
        lm.addElement(31);
        lm.addElement(30);
        lm.addElement(31);
        lm.addElement(31);
        lm.addElement(30);
        lm.addElement(31);
        lst2 = new JList(lm);
        lst1.setVisibleRowCount(2);
        lst2.setVisibleRowCount(3);
        JScrollPane Monthssp = new JScrollPane(lst1);
        JScrollPane dayssp = new JScrollPane(lst2);
        JLabel l1 = new JLabel("months");
        JLabel l2 = new JLabel("No.of days in Month");
        c.add(l1);
        c.add(Monthssp);
        c.add(l2);
        c.add(dayssp);
    }
}
class JListDemo
{
    public static void main(String args[])
    {
        Lst ob = new Lst();
        ob.setTitle("JList in JFrame");
        ob.setSize(300,300);
        ob.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ob.setVisible(true);
    }
}
```

OUTPUT



g) JComoBox

- The user can select a single item only. A combo box is a visual Swing graphical component that gives popup list when clicked.
- It is the combination of JList and JTextField.
- In Combo box, only one item is visible at a time. A Popup menu displays the choices a user can select from.
- In JList, the items cannot be edited, but in combo box, the items can be edited by setting the JComboBox editable.
- JCombobox defines the following constructors

JComboBox() \\ Creates empty Combo Box

JComboBox(object[] arr) \\ Creates a combo box taking the items from the specified Object Array

String 1st=("India","America","germany");

Eg: JComboBox box=new JComboBox(1st);

JComboBox(Vector v)\\ Creates a combo box taking the items from the specified vector

- JComboBox has so many methods. Some of them are

Void addItem(Object obj) \\ adds the specified object to the list

Eg:box.addItem("Japan");

Object getSelectItem() \\Returns the currently selected item

Eg: Object obj=box.getSelectedItem();

Int getSelectIndex() \\ Returns the index of item in the list

Eg: int I=box.getSelectedIndex();

Int getItemCount() \\ Returns the number of items in the list

Eg: int I=box.getItemCount();

Boolean isEditable() \\ Returns a Boolean specifying whether the combobox items are Editable or not

Eg: Boolean x=box.isEditable();

Void removeItem(Object ob) \\ Removes the specified item from the list

Eg: box.removeItem("germany");

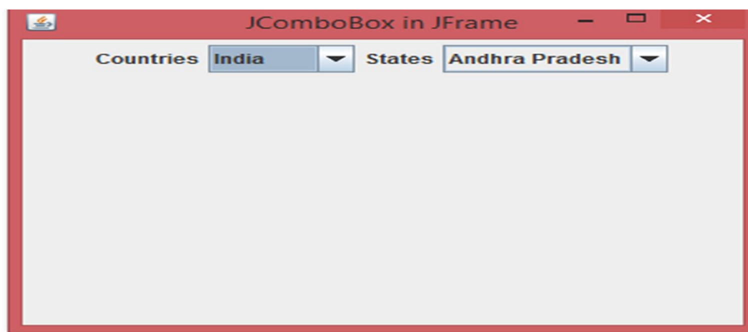
- **Program to Create JComboBox in JFrame**

```
// JCombo Box Demo
    import java.awt.*;
    import javax.swing.*;
    class cmbbox extends JFrame
    {
            JComboBox box1,box2;
            String str[]={"Andhra Pradesh","Tamil Nadu","Karnataka"};
            cmbbox()
             {
                  Container c=getContentPane();
                  c.setLayout(new FlowLayout());
                  box1=new JComboBox();
                  box2=new JComboBox(str);
                  JLabel I1=new JLabel("Countries");
                  JLabel I2=new JLabel("States");
                  box1.addItem("India");
                  box1.addItem("germany");
```

```
                box1.addItem("Japan");
                c.add(l1);
                c.add(box1);
                c.add(l2);
                c.add(box2);

        }
}
class JComboDemo
{
public static void main(String args[])
{
    cmbbox ob=new cmbbox();
    ob.setTitle("JComboBox in JFrame");
  ob.setSize(400,300);
  ob.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  ob.setVisible(true);
}
}
```

OUTPUT:



i)JTable

JTable is a component that displays rows and columns of data. JTable supplies several constructors. The one commonly used is

JTable(Object data[][],Object colHeads[])

```
import java.awt.*;
import javax.swing.*;
class tbl extends JFrame
{

        String colshead[]={"sno","sname"};
    Object[][] data = {{"100","aaa"},{"101","bbb"},{"102","ccc"},{"103","ddd"}                    };
        tbl()
        {
            Container c=getContentPane();
            c.setLayout(new FlowLayout());
             JTable tb = new JTable(data,colshead);
```

```
                JScrollPane sp = new JScrollPane(tb);
            c.add(sp);
    }
}
class JTableDemo
{
public static void main(String args[])
{
    tbl ob=new tbl();
    ob.setTitle("JTable in JFrame");
    ob.setSize(400,300);
    ob.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    ob.setVisible(true);
}
}
```

OUTPUT