SANJIVANI K. B. P. POLYTECHNIC, KOPARGAON

With NBA ACCREDIATED programs , Approved by AICTE, New Delhi, Recognized by Govt. of Maharashtra, Affiliated to Maharashtra State Board of Technical Education, Mumbai, ISO 9001:2015 Certified Institute

Department:- Computer Technology   Class:- TYCM A and B

Name of Subject:- AJP   MSBTE Subject Code:- 22517

Unit -2 Swings

2.1 Swing Introduction

1. Swing is a java foundation classes library and it is an extension to do Abstract Window Toolkit (AWT).

2. Swing is a part of **JFC (Java Foundation Classes)** that is used to create GUI application. It is built on the top of AWT and entirely written in java.

3. The Java Foundation Classes (JFC) are nothing but class libraries specially designed for building GUI's. Foundation classes simply the designing process and reduce the time taken for coding.

4. **Swing is a set of classes this provides more powerful and flexible components than are possible with the AWT.**

5. Swing offers much improved functionality over AWT, new components, expanded components features and excellent event handling with drag and drop facility.

6. Swing was developed to provide a more sophisticated set of GUI components than the earlier AWT(Abstract Window Toolkit)

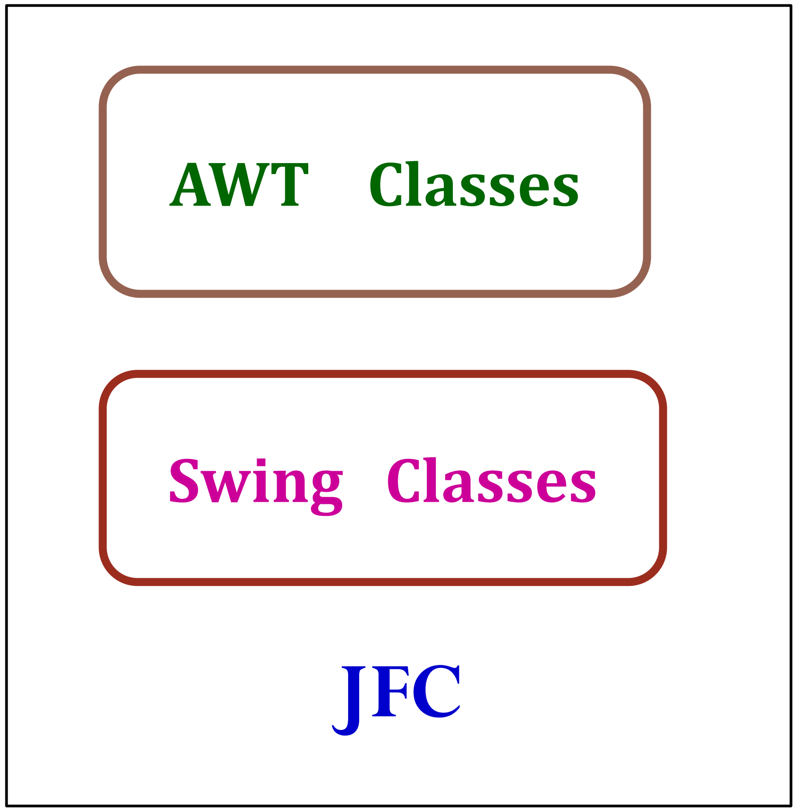
7. In addition to the familiar components, such as buttons, check boxes, and labels,

   Swing provides several exciting , advanced components such as tabbed panes, scroll panes, trees, tables and so on.

8. Even familiar components such as buttons have more capabilities in Swing.

   For example, a button may have both an image and a text string associated with it. Also, the image can be changed as the state of the button changes.

JFC contains two major technologies , one of which is Swing.



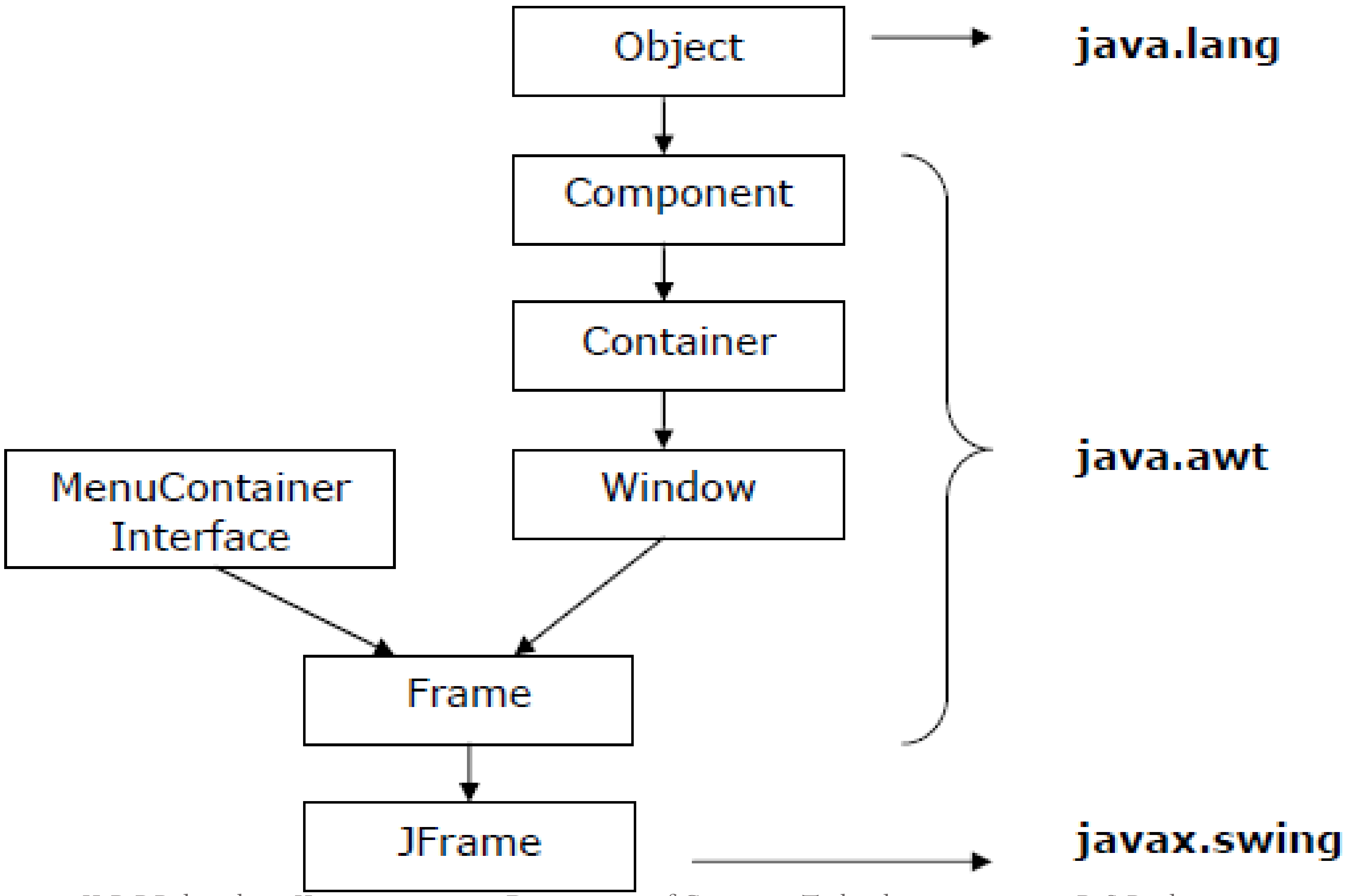
8. Swing components are not implemented by platform-specific code. Instead, they are written entirely in Java and, therefore, are platform-independent.

9. The term lightweight is used to describe such elements. It does not depend on any non-java system classes.

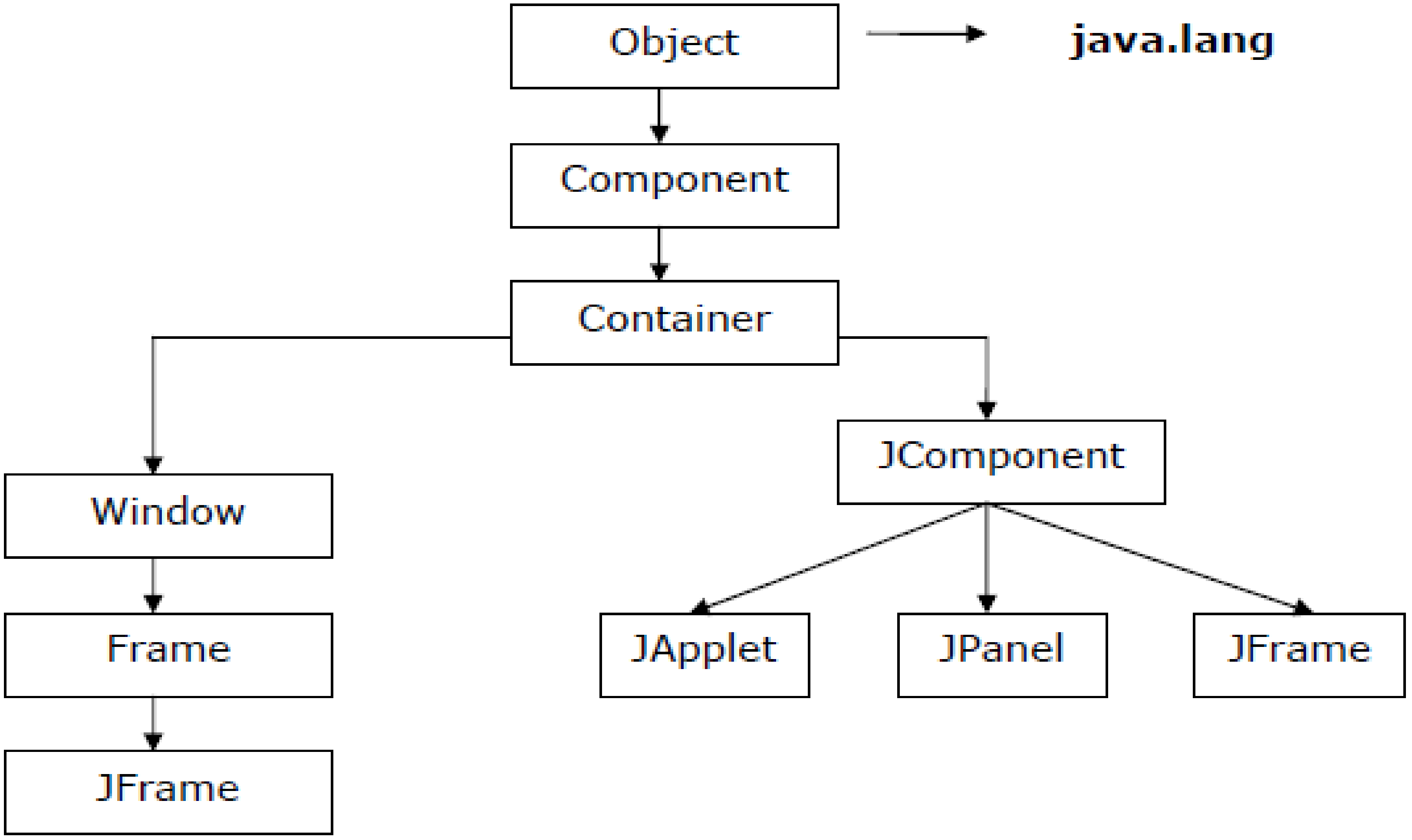
10. Swing components have their own view supported by java's look and feel classes.

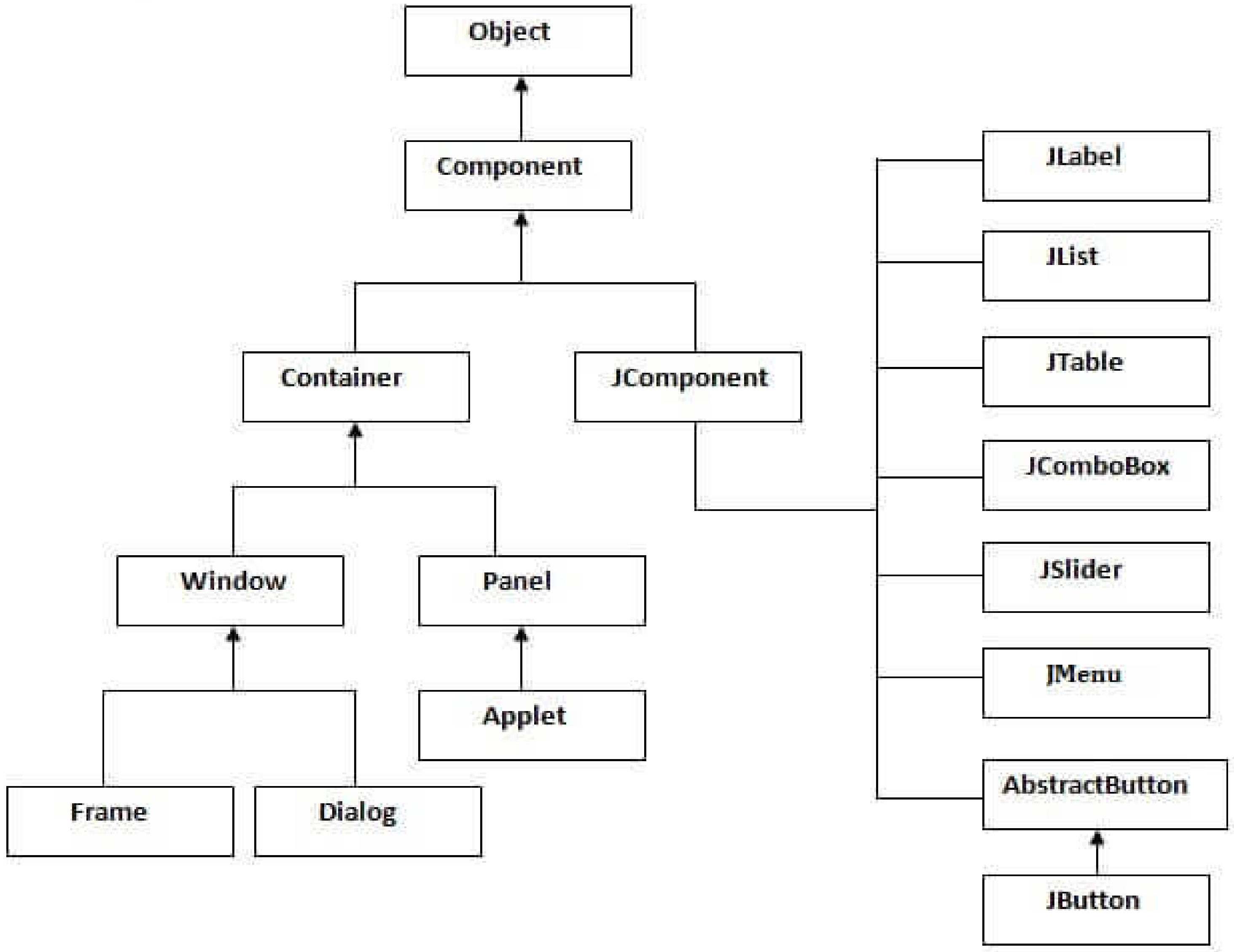
11. Swing components have pluggable look and feel so that with a single set of components you can achieve the same look and feel on any operating system platform.

Steps in Forming an Organization



The Swing family Tree




Object
Component
Container
JComponent
Window
Panel
Applet
Frame
Dialog
JLabel
JList
JTable
JComboBox
JSlider
JMenu
AbstractButton
JButton

Overview of the Swing Class Hierarchy:-

```
java.awt.Component
  └─java.awt.Container
      ├─java.awt.Window
      │   ├─java.awt.Frame
      │   │   └─javax.swing.JFrame
      │   └─java.awt.Dialog
      │       └─javax.swing.JDialog
      │
      └─javax.swing.JComponent
          ├──javax.swing.JPanel,JLabel, JMenuBar etc ...
          └──javax.swing.AbstractButton
               └──javax.swing.JButton, JMenuItem,etc ...
```

Unit -2 Swings

2.1 Swing Features
Difference between AWT and Swing

Swing Features :-

- **Light Weight:** Swing component are independent of native Operating System's API as Swing API controls are rendered mostly using pure JAVA code instead of underlying operating system calls.

- **Rich Control:** Swing provides a rich set of advanced controls like Tree, TabbedPane, Slider, Table Controls and so on.

- **Borders:** We can draw borders in many different styles around components using the **setborder( )** method.

- **Easy mouseless operation :** It is easy to connect keystrokes to components.

- **ToolTips:** We can use the setToolTipText method of JComponent to give components a tooltip, one of those small windows that appear when the mouse hovers over a component and gives explanatory text.

Swing Features :-

- **Easy Scrolling**
  We can connect scrolling to various components-something that was impossible in AWT.

- **Pluggable look and feel**
  We can set the appearance of applets and applications to one of three standard looks. Windows, Motif (Unix) or Metal (Standard swing look).

- ***New Layout Managers***
  Swing introduces the BoxLayout and OverlayLayout layout managers.

Difference between Swing and AWT

S.N	Swing	AWT
1	Swing is also called as **JFC**'s. (Java Foundation classes)	AWT stands for **Abstract windowing toolkit**.
2	Swing components require **javax.swing** package.	AWT components require **java.awt** package
3	It uses **JApplet** class.	It uses **Applet** class.
4	Swings are called **Light weight Component** because swing components sits on the top of AWT components and do the work.	AWT components are called **Heavy weight Component**.
5	Swings components are made in purely java and they are **platform independent.**	AWT components are **platform dependent.**

S.N	Swing	AWT
6	It does not have add( ) method to add control.	It has add( ) method to add control.
7	Swing button class is named **JButton**. You can identify Swing components because their names start with **J**	The AWT button class, for example, is named **Button**
8	Swing supports Pluggable Look and Feel.	AWT does not support pluggable Look and Feel
9	Complex component.	Primitive component.
10	Swing components don't have to be rectangular. Buttons, for example, can be round.	This feature is not supported in AWT.
11	It has Bigger collection of classes and interfaces than AWT.	It has huge collection of classes and interfaces

SN	Swing	AWT
11	You can easily change the behavior or appearance of a Swing component by either invoking methods on it or creating a subclass of it.	This feature is not supported in AWT.
12	Swing buttons and labels can display images instead of, or in addition to, text.	This feature is not supported in AWT.
13	You can easily add or change the borders drawn around most Swing components. For example, it's easy to put a box around the outside of a container or label.	This feature is not supported in AWT.
14	It has more powerful components like Tables, Lists, Scroll panes, color chooser, tabbed pane,etc.	This feature is not supported in AWT.
15	Swing follows MVC. Where Model(M) represents data, View(V) represents presentation and Controller© acts as an interface between model and view.	AWT does not follow MVC (Model-View-Controller).

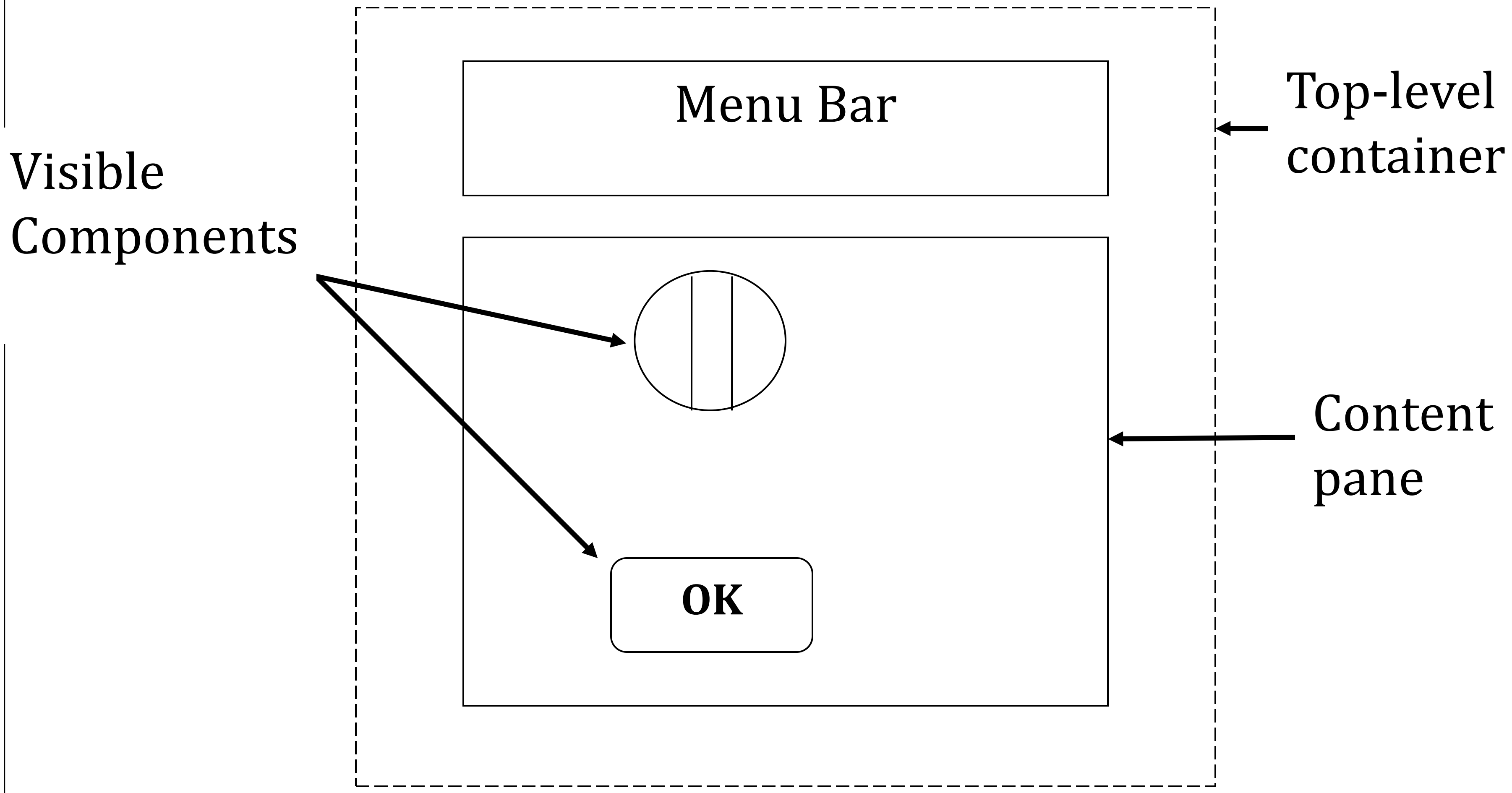
Unit -2 Swings

2.1 Swing Basics

The Swing component classes that are shown below:

Class	Description
AbstractButton	Abstract super-class for Swing buttons.
ButtonGroup	Encapsulates a mutually exclusive set of buttons.
ImageIcon	Encapsulates an icon.
JApplet	The Swing version of Applet.
JButton	The Swing push button class.
JCheckBox	The Swing check box class.
JComboBox	Encapsulates a combo box (a combination of a drop-down list and text field).
JLabel	The Swing version of a label.
JRadioButton	The Swing version of a radio button.
JScrollPane	Encapsulates a scrollable window.
JTabbedPane	Encapsulates a tabbed window.
JTable	Encapsulates a table-based control.
JTextField	The Swing version of a text field.
JTree	Encapsulates a tree-based control.

Working with Swing:-



1. While designing any GUI, we need to have a main window on which we can place or position the different visual components.

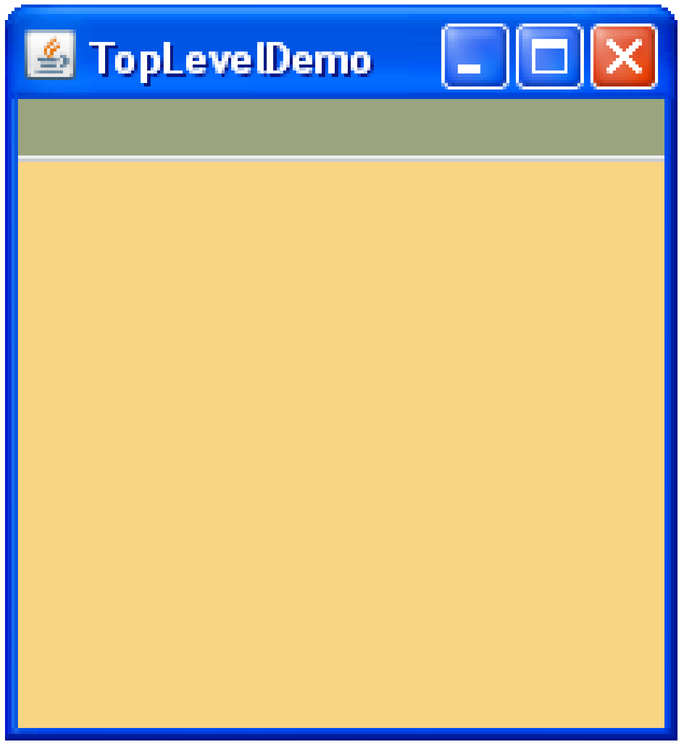
2. In swing, the main window, also called a Top-level container is the root of a hierarchy, which contains all the other swing components that appear inside the window.

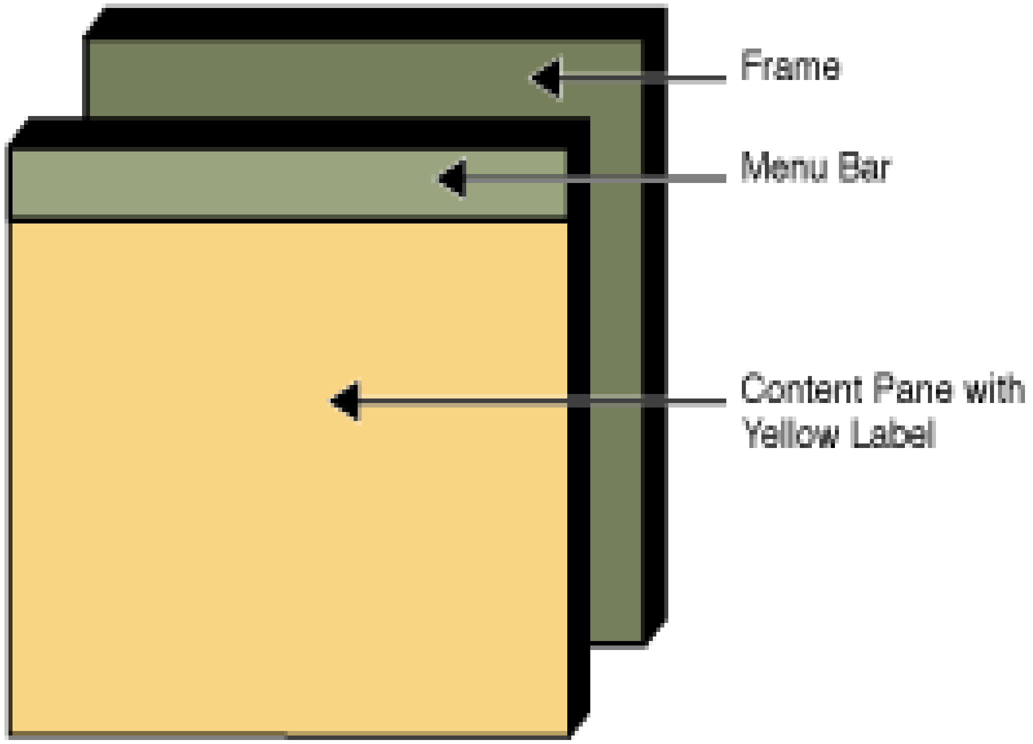
3. All swing applications have at least one Top level container.

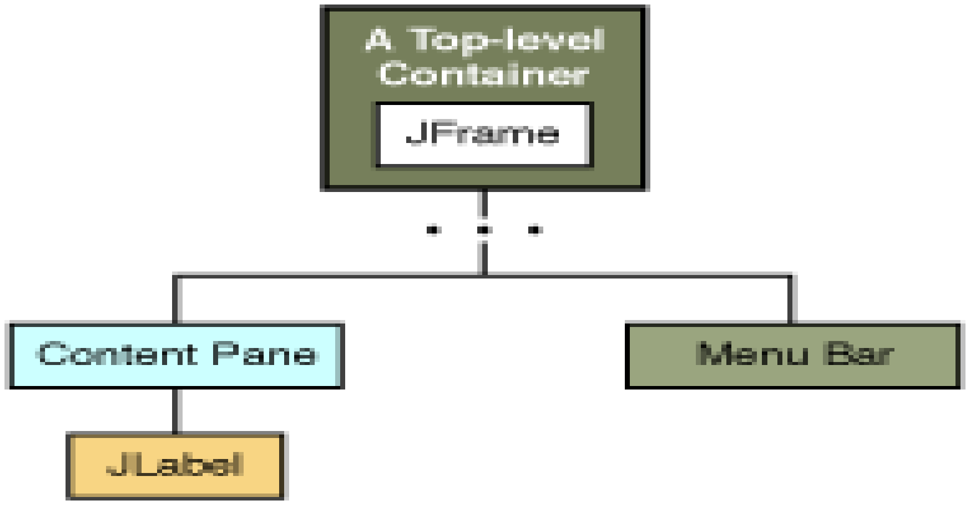
4. Every top level container has one intermediate container called **content pane.**

5. This content pane contains all the visible components in the GUI window.

6. The content pane is the base pane upon which all other components or container objects are placed. One exception to this rule is, if there is a menu bar in the top level container it will be placed outside the content pane.


TopLevelDemo


Frame
Menu Bar
Content Pane with Yellow Label


A Top-level Container
JFrame
Content Pane
Menu Bar
JLabel

1. All swing components names start with **J** . For instance the swing button class is named as JButton commonly used top-level containers are:
2. Just like AWT application, a swing application requires a top-level container. There are 3 top-level containers in swing:

1) JFrame: Used for the application's main window (with an icon, a title, minimize/maximize/close buttons, an optional menu-bar, and a content-pane).

2) JDialog: Used for secondary pop-up window (with a title, a close button and a content-pane).

3) JApplet: Used for the applet's display area(content-pane) inside a browser's window.

4) Similarly to AWT, there are secondary containers (such as Jpanel) which can be used to group and layout relevant components. Panels are an example of intermediate containers.

1. JApplet

1. Fundamental to Swing is the **JApplet class**, which extends Applet.

2. Applets that use Swing must be subclasses of JApplet. JApplet is rich with functionality that is not found in Applet.

3. Whenever to add a component to it, the component is added to the content pane.

4. JApplet supports various "panes," such as the content pane, the glass pane, and the root pane.

5. Components are added to the content pane of swing applets, not directly for the applet.

1. JApplet

6. The layout manager is set for the content pane of a swing applet, not directly for the applet.

7. The default layout manager for the content pane of a swing based applet is Border-Layout.

8. The content pane can be obtained via the method shown here:

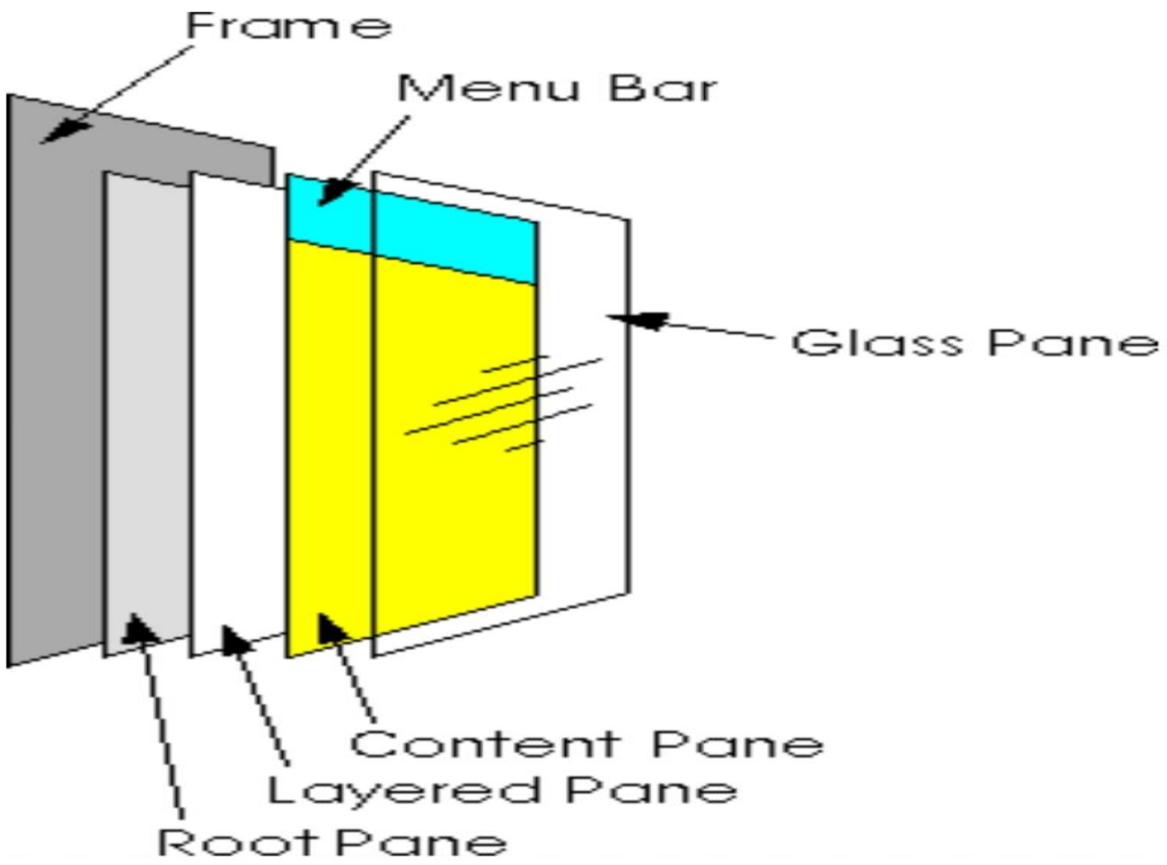
Container getContentPane( )

9. The add( ) method of Container can be used to add a component to a content pane. Its form is shown here:

void add(comp)

Here, comp is the component to be added to the content pane.

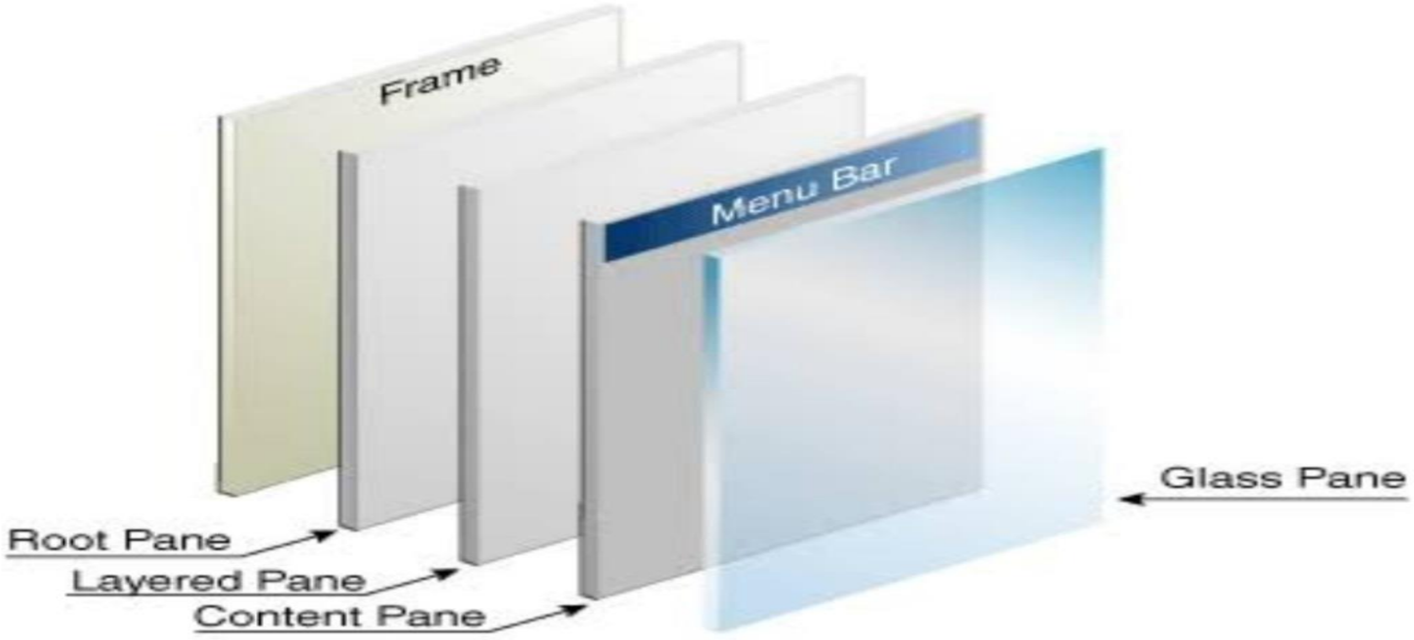
Different Panes in JApplet



1. Root Pane -

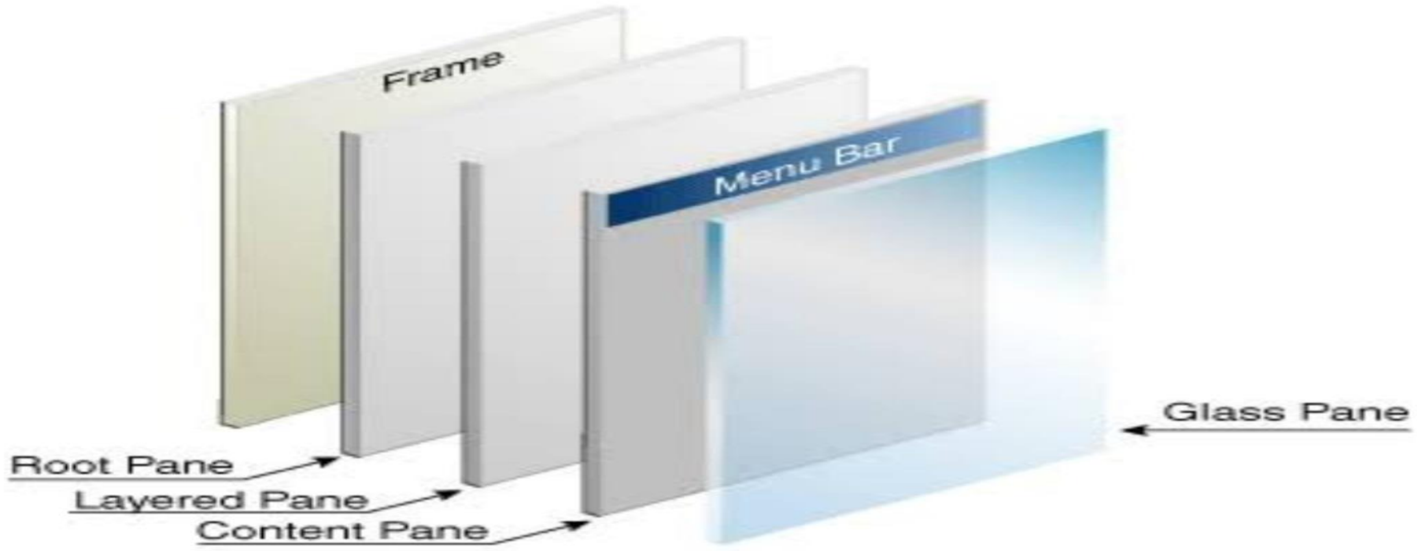
1. Swing containers like JApplet, JFrame, JWindow and JDialog delegate their duties to the root pane represented by the class JRootPane.

2. The root pane is made up of a content pane, layered pane, glass pane and an optional menu bar, other GUI components must be added to one of these roots pane members, rather than to the root pane itself.



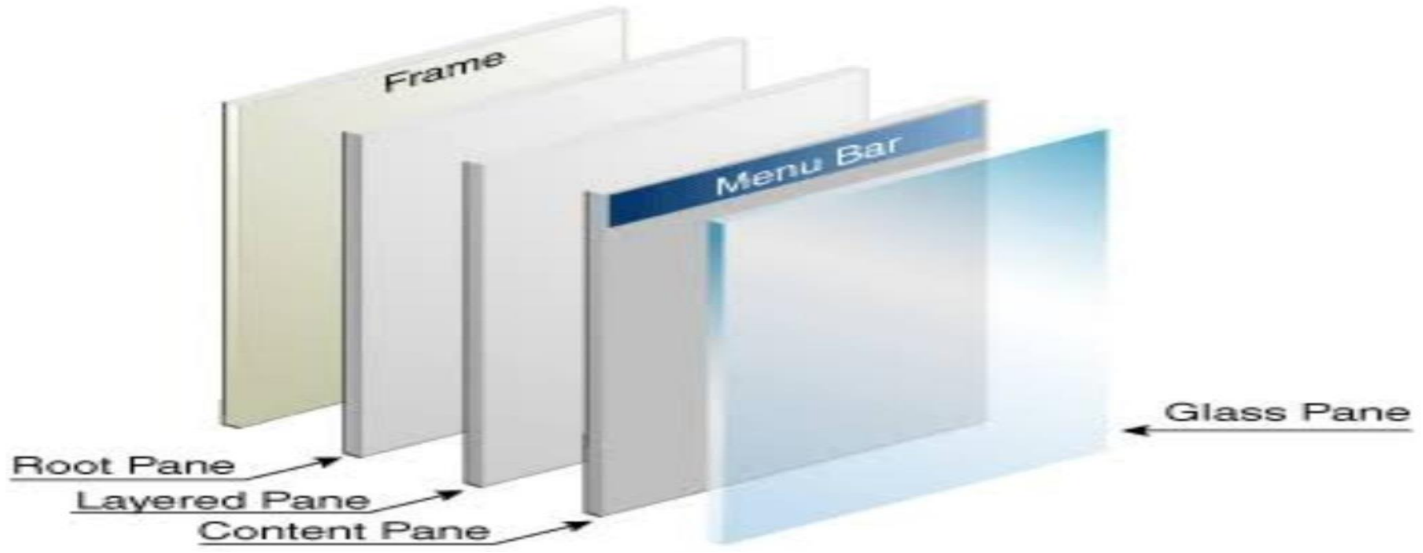
2.Content Pane -

1. It is a pane which is used to hold the components. All the visual components like Buttons, Labels are added to content pane.
2. Fig shows the position of the content pane in the specially layered pane. The menu bar and the content pane sits on the Frame-Content-Layer.
3. By default, it is a instance of Jpanel class and uses border layout.
4. It is accessed via getContentPane( ) method of JApplet and JFrame classes.



3. Glass Pane -

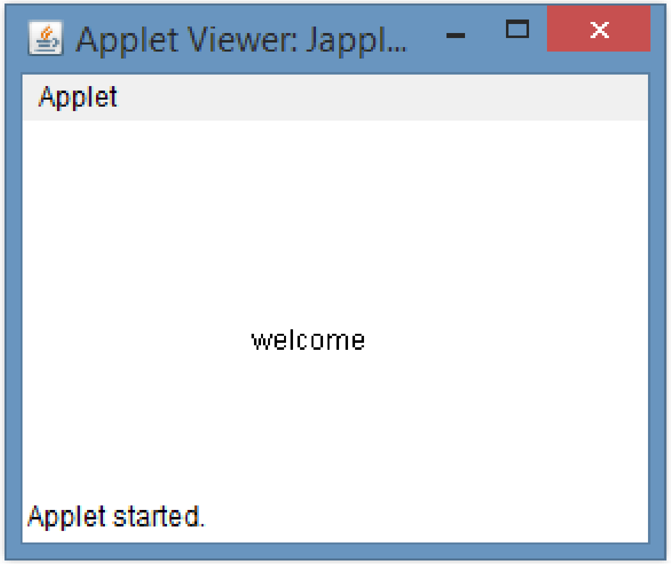
1. It is a member of the multi-layered, root pane. It is used to draw over an area that already contains some components.
2. A glass pane is a top-level pane which covers all other panes. By default, it is a transparent instance of JPanel class.
3. It is used to handle the mouse events affecting the entire container.



JApplet

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

/* <applet code = "JappletEx.class"  width="300"  height="300"> </applet> */

public class JappletEx extends JApplet
 {
  public void paint(Graphics g)
  {
        g.drawString("welcome ", 100, 100);
    }
 }
```

Output -

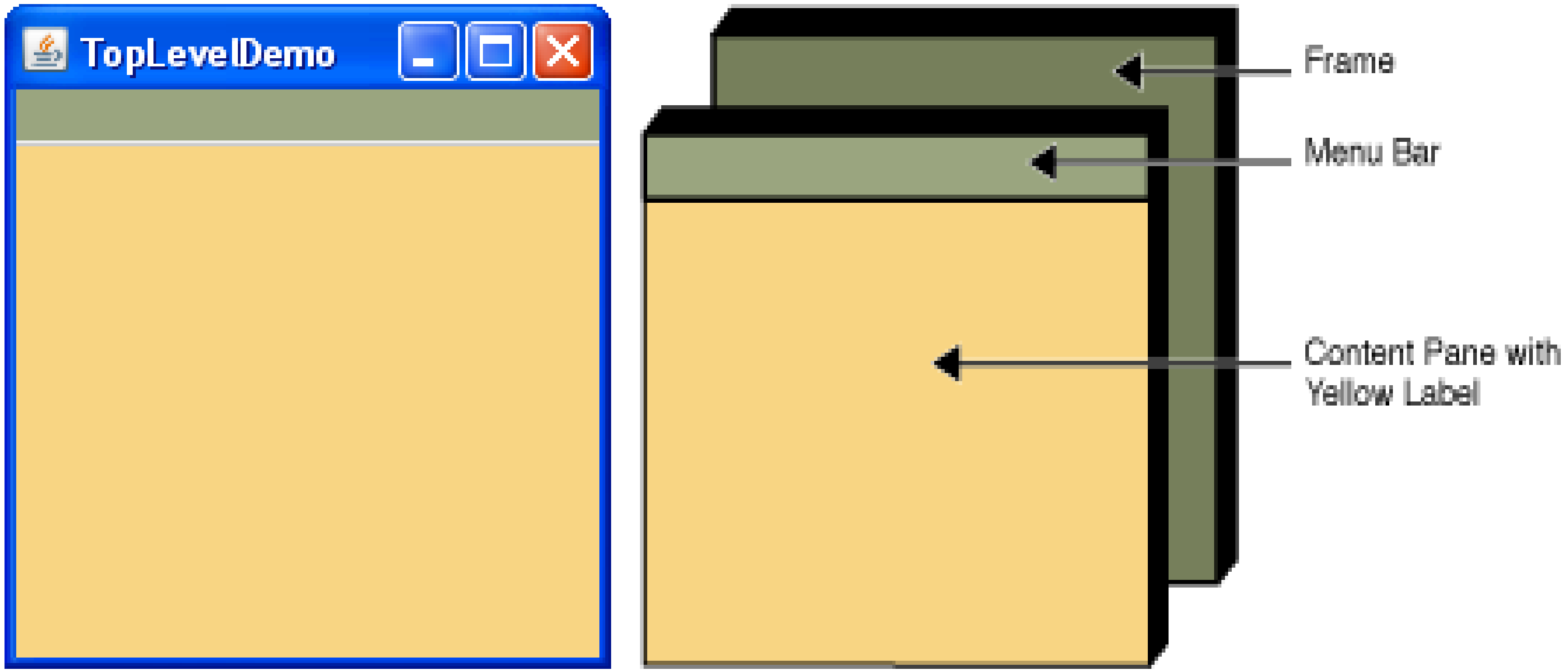


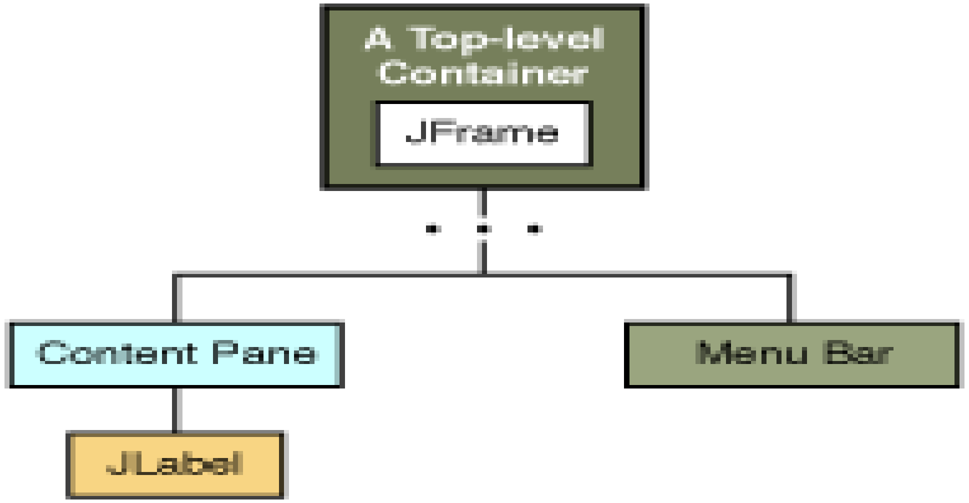
```
welcome
```

```
Applet started.
```

Unit -2 Swings

2.1 Swing JFrame


TopLevelDemo
Frame
Menu Bar
Content Pane with Yellow Label


A Top-level Container
JFrame
Content Pane
Menu Bar
JLabel

Commonly use Methods of Component class

S.N	Method	Description
1	Public void add(Component c)	Add a component on another component.
2	Public void setSize(int width, int height)	Sets size of the component.
3	Public void setLayout(LayoutManager m)	Sets the layout manager for the component.
4	Public void setVisible(boolean b)	Sets the visibility of the component. It is by default false.

1. JFrame

1. The frame is a top-level container or window, on which other Swing components are placed.

Sr. No	constructors	Description
1	JFrame( )	Create a new frame without title.
2	JFrame(String title)	Create a new frame with title.

Sr	Methods	Description
1	Container getContentPane( )	Returns a content pane for the JFrame.
2	void setLayout(LayoutManager)	Sets the LayoutMaanger for the JFrame.
3	void setJMenubar (JMenubar)	Sets the JMenubar for the JFrame.

1. JFrame

1. There are 2 ways to create a JFrame window:

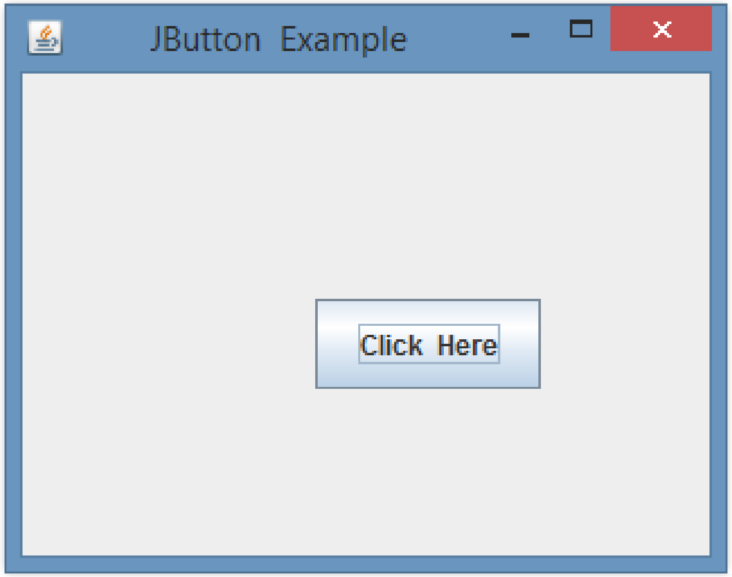
1) By instantiating JFrame class.

2) By extending JFrame class.

Creating JFrame window by instantiating JFrame class.

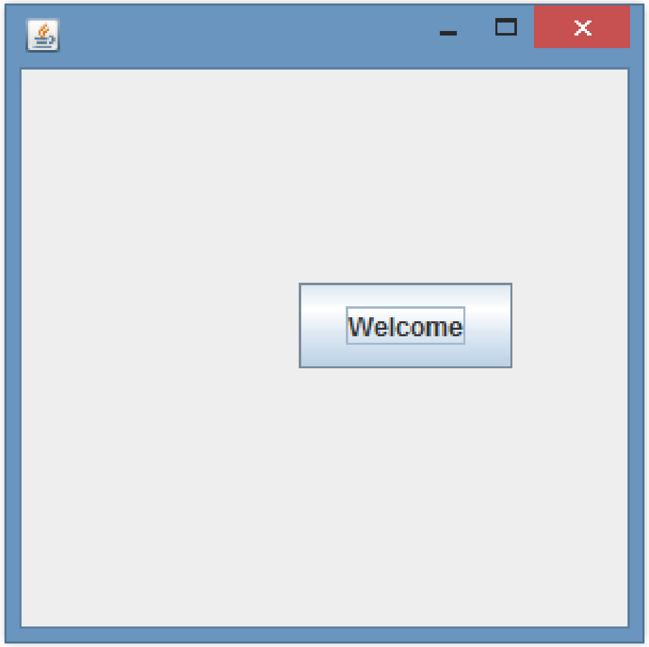
```
import java.awt.*;
import javax.swing.*;

public class JButtonExample
 {
   public static void main(String args[ ])
   {
    JFrame f = new JFrame(“JButton Example");
    JButton  b = new JButton ("Click Here");
    b.setBounds(130, 100, 100, 40);
   f.add(b);
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
   }
}
```

Output -



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

public  class  SwingEx
 {
  SwingEx( )
   {
    JFrame  f  =  new  JFrame( );
     JButton   b  =  new JButton ("Welcome");
     b.setBounds(130, 100,100,40);
    f.add(b);
    f.setSize(300, 300);
    f.setLayout(null);
    f.setVisible(true);
    }
  public  static  void   main(String  args[  ])
   {
    SwingEx  t = new   SwingEx( );
   }
}
```

Output -



Unit -2 Swings

JLabel

2. JLabel

1. One of the simplest Swing component is JLabel.
2. This object is a component for placing text in a container.
3. This class is used to create single line read only text which describe the other component.
4. Labels can display Text as well as Images.
5. Constructors:-

JLabel( )	Create a empty label having no text and icon.
JLabel (String text)	Create a label having some text.
JLabel (ImageIcon i)	Create a label having icon image.
JLabel (String text, ImageIcon i)	Create a label having String text and icon image.

2. JLabel

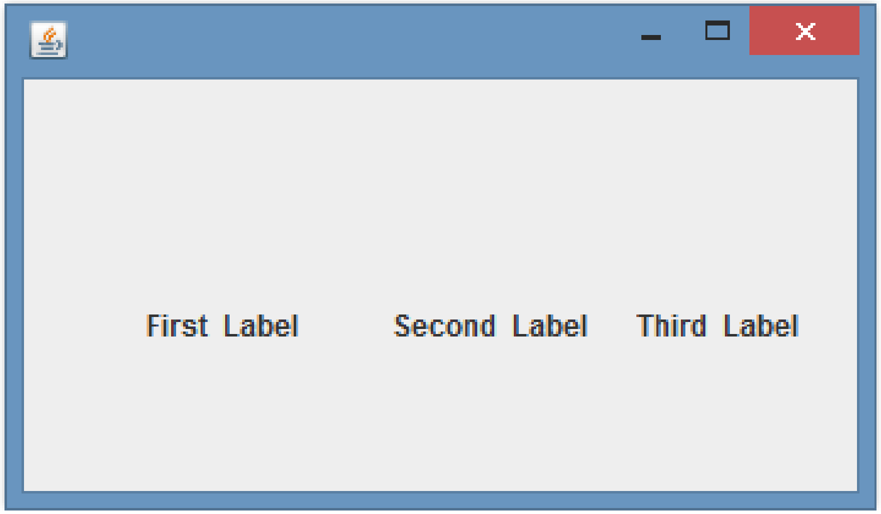
Methods:-

String getText()	Returns text associated with the Label.
void setText(String text)	Used to set the text to the Label
void setIcon(Icon i)	Sets the Icon to the icon.
void setHorizontalAlignment (int a)	This method sets the alignment of the text.

2. JLabel

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

public  class  JLabelExample2 extends  JFrame
 {
  public JLabelExample2( )
   {
    JLabel   one  =  new  JLabel ("First  Label");
    JLabel   two  =  new  JLabel("Second  Label");
   JLabel   three  =  new  JLabel("Third  Label");
    one.setBounds(50, 50, 100, 100);
    two.setBounds(150, 50, 100, 100);
    three.setBounds(250, 50, 100, 100);
    add(one);
   add(two);
   add(three);
    setSize(300, 300);
   setLayout(null);
    setVisible(true);
}
public  static  void   main(String  args[  ])
 {
  JLabelExample2    obj =  new  JLabelExample2( );
   }
}
```

Output -



Unit -2 Swings

ImageIcon

3. ImageIcon

1. Many Swing components, such as Labels, Buttons and Tabbed panes, can be decorated with an icon – a fixed-sized picture.
2. Icons are encapsulated by the ImageIcon class, which implements the icon interface which paints an icon from a GIF, JPEG or PNG image.
3. Constructors:-

ImageIcon(String filename)	Uses the image in the file named filename
ImageIcon(URL url)	uses the image in the resource identified by url.

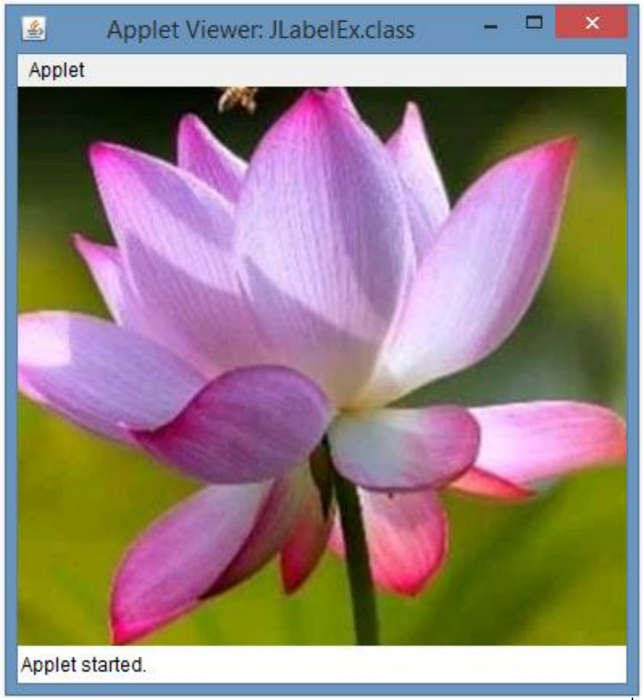
4.Methods :-

int getIconHeight ( )	Returns the Height of the icon in pixels.
int getIconWidth ( )	Returns the Width of the icon in pixels.
int paintIcon (Component comp, Graphics g, int x, int y)	Paints the icon at position x, y on the graphics context g. Additional information about the paint operation can be provided in component.

3. ImageIcon

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

/*<applet  code="JLabelEx.class"  width=300  height=300> </applet> */
public class JLabelEx extends JApplet
{
 public void init( )
 {
  Container  cPane = getContentPane( );
  ImageIcon  i1= new  ImageIcon("flower.jpg");
  JLabel   l1 =  new    JLabel("flower", i1, JLabel.CENTER);
  cPane.add(l1);
  }
}
```

Output -



Unit -2 Swings

JTextField and JTextArea

4. JTextField

1. The Swing text field is encapsulated by the JTextComponent class, which extends JComponent. It provides functionality that is common to Swing text components.
2. The JTextField component allows us to enter or edit a single line of text.

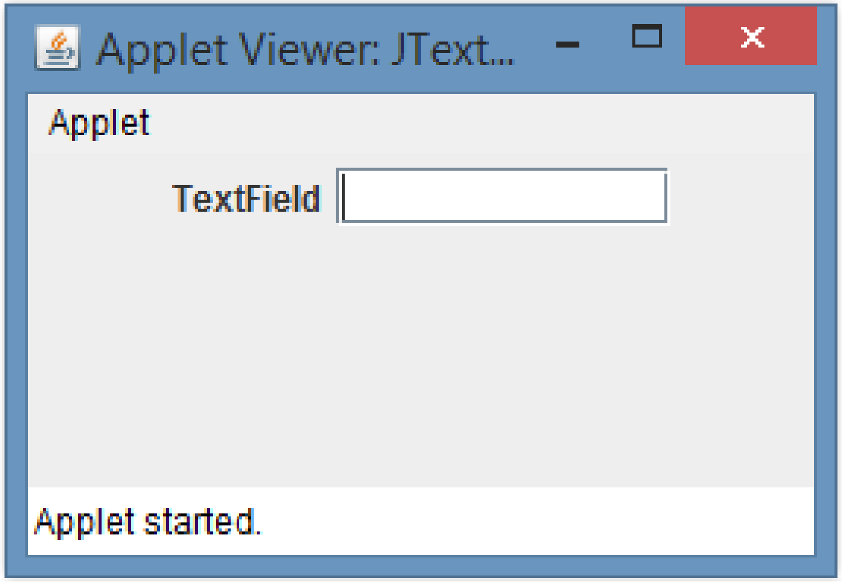
3. Constructors:-

JTextField( )	Create a empty TextField .
JTextField(int cols)	Constructs a new empty TextField with the specified number of columns.
JTextField(String text)	Constructs a new TextField initialized with the specified text.
JTextField(String text, int cols)	Here, text is the string to be presented, and cols is the number of columns in the text field.

4. JTextField

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

/*<applet  code= "JTextFieldEx.class"  width=300 height=300> </applet> */
public  class  JTextFieldEx  extends   JApplet
{
 JLabel   l1;
 JTextField   t1;
 public void init( )
 {
 Container    cPane   =   getContentPane( );
 cPane.setLayout(new FlowLayout( ));
  l1 =  new   JLabel("TextField");
 cPane.add(l1);
 t1   =   new    JTextField(10);
 cPane.add(t1);
 }
}
```

Output -



```
Applet started.
```

5. JTextArea

1. JTextArea is used to accept multiple lines of text from the user.
2. It allows the editing of multiple line text.
3. JTextArea class itself does not handle scrolling but implements the Scrollable interface which allows it to be placed inside a JScrollPane and thus implement scrolling.
4. **Constructors:-**

JTextArea( )	Create a empty TextArea.
JTextArea (int rows, int cols)	**rows** is the number of rows(height) and **cols** is the number of columns(width).
JTextArea (String text)	**text** is the string to be presented
JTextArea (String text, int rows, cols)	Here, **text** is the string to be presented, and **rows** is the number of rows and **cols** is the number of columns in the text Area.

5. JTextArea

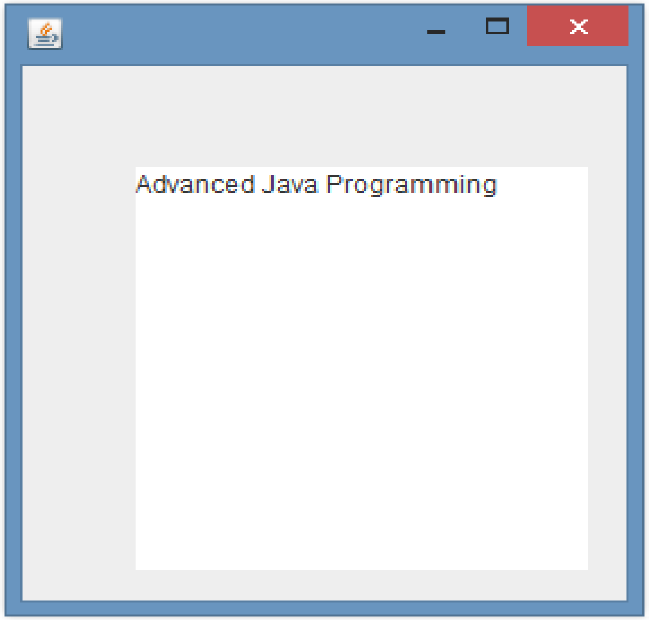
1. Methods – It supports all methods of JTextFields.

Sr. No	Methods	Description
1	void setColumns(int columns)	Sets the number of columns for this text area.
2	void setRows(int rows)	Sets the number of rows for this text area.
3	int getColumn( )	Returns the number of Columns in the text area.
4	int getRows( )	Returns the number of Rows in the text area.
5	void append(String text)	Appends the given text to the text area's current text.
6	void insert(String text , int pos)	Inserts the specified text at the specified position in this text area.

JTextArea

```
import  java.awt.*;
import javax.swing.*;
public  class  JTextAreaExample
 {
   public  static  void   main(String  args[  ])
  {
    JFrame  f  =  new  JFrame( );
    JTextArea  t1  =  new JTextArea("Advanced Java Programming");
    t1.setBounds(50, 50,200, 200);
    f.add(t1);
      f.setSize(400, 400);
   f.setLayout(null);
   f.setVisible(true);
   }
}
```

Output -



```
Advanced Java Programming
```

Unit -2 Swings

JButton

5. JButton

1. This class is used to create a push button.
2. The application result in some action when the button is pushed.
3. This class is subclass of Abstract Button class.

4. Constructors:-

JButton( )	Create a empty button with no text and icon.
JButton(String text)	Create a button with the specified text.
JButton(Icon i)	Create a button with the specified icon image.
JButton(String text, Icon i)	Create a button with the specified text and icon image.

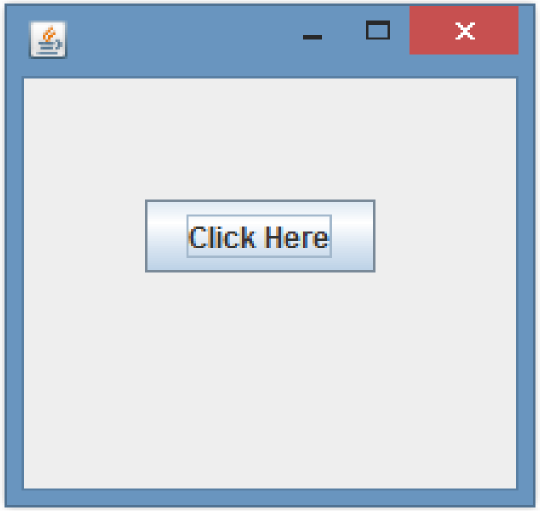
Jbutton- commonly used methods of AbstractButton class

Sr. No	Methods	Description
1	String getLabel( )	Get the Label of the Button.
2	void setLabel(String buttonlabel)	Set or change the Label of the Button.
3	void addActionListener (ActionListener l)	Adds the specified action listener to receive action events from this button.
4	void removeActionListener (ActionListener l)	Removes the specified action listener so that it no longer receives action events from this button.
5	void setIcon (Icon b)	Used to set the specified Icon on the button.

5. JButton

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

public  class  JButtonEx
 {
   public  static  void   main(String  args[  ])
  {
    JFrame  f  =  new  JFrame( );
    JButton b  =  new JButton("Click Here");
    b.setBounds(50, 50,95, 30);
    f.add(b);

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

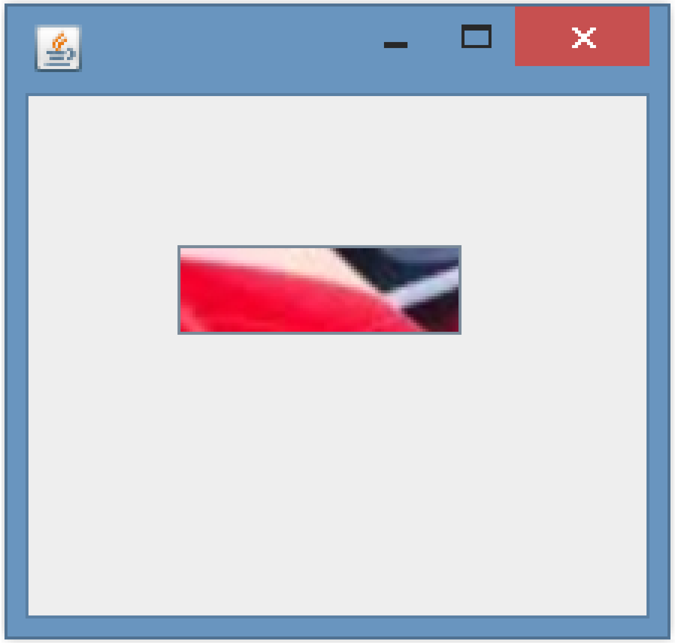
Output -



5. JButton

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

public  class  JButtonEx
 {
   public  static  void   main(String  args[  ])
  {
    JFrame  f  =  new  JFrame( );
   ImageIcon  m=new ImageIcon("car.jpg");
    JButton b  =  new JButton(m);
    b.setBounds(50, 50,95, 30);
    f.add(b);

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

Output -



Unit -2 Swings

JCheckbox

6. JCheckBox

1. A checkbox is a control that may be turned ON or OFF by the user to indicate some option.
2. It is used for selection of option.
3. It is used to turn an option ON (true) or OFF (false).
4. It consists of a small box that can either contains a check mark or not.
5. Clicking on a Checkbox changes its state from “On” to “Off” or from “Off” to “On”.

6. JCheckBox

1. **Constructors:-**
   1. JCheckBox( )
   2. JCheckBox(Icon i)
   3. JCheckBox(Icon i, boolean state)
   4. JCheckBox(String text)
   5. JCheckBox(String text, boolean state)
   6. JCheckBox(String text, Icon i)
   7. JCheckBox(String text, Icon i, boolean state)

1. Here, ***i*** is the icon for the button. The text is specified by *text*. If *state* is true, the check box is initially selected. Otherwise, it is not. The state of the check box can be changed via the following method:

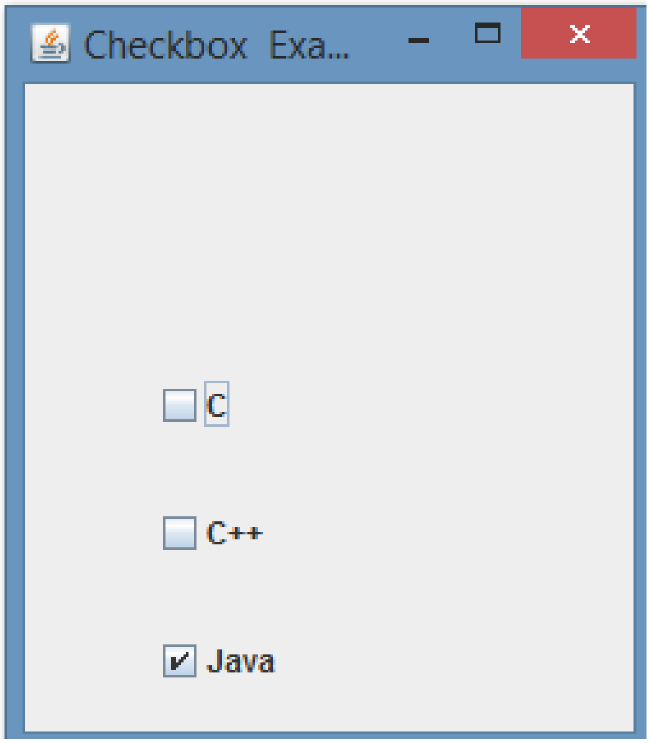
   void setSelected(boolean state)

   Here, state is true if the check box should be checked.

6. JCheckBox

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

public  class  JCheckboxEx
 {
   public  static  void   main(String  args[  ])
  {
    JFrame  f  =  new  JFrame("Checkbox  Example");
    JCheckBox   c1  =  new  JCheckBox ("C");
    c1.setBounds(50, 100, 50,50);
    JCheckBox   c2  =  new  JCheckBox ("C++");
    c2.setBounds(50, 150, 50,50);
    JCheckBox   c3  =  new  JCheckBox ("Java", true);
    c3.setBounds(50, 200, 70,50);
    f.add(c1);
   f.add(c2);
   f.add(c3);
   f.setSize(300, 300);
   f.setLayout(null);
   f.setVisible(true);
   }
}
```

Output -



Unit -2 Swings

JRadioButton

7. JRadioButton

1. This class is used to create radio button with a text or icon.
2. It is used to choose one option from multiple options.
3. It is widely used in exam systems or Quiz.
4. It should be added in ButtonGroup to select one radio button only.
5. Radio buttons must be configured into a group.
6. Only one of the button in that group can be selected at any time.

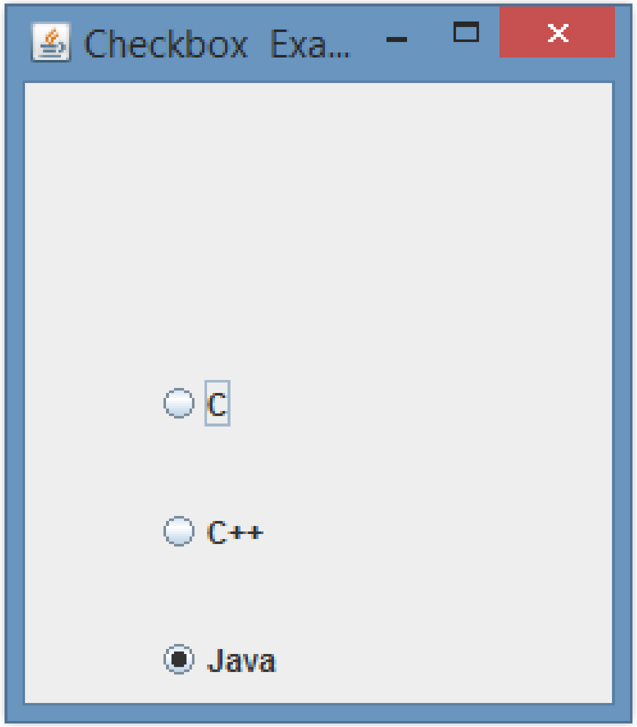
7. JRadioButton

1. **Constructors:-**
   1. JRadioButton( )
   2. JRadioButton(Icon i)
   3. JRadioButton(Icon i, boolean state)
   4. JRadioButton(String text)
   5. JRadioButton(String text, boolean state)
   6. JRadioButton(String text, Icon i)
   7. JRadioButton(String text, Icon i, boolean state)

7.JRadioButton

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

public  class  JRadioButtonEx
 {
   public  static  void   main(String  args[  ])
  {
    JFrame  f  =  new  JFrame("Checkbox  Example");
    JRadioButton  c1  =  new  JRadioButton ("C");
    c1.setBounds(50, 100, 50,50);
    JRadioButton   c2  =  new  JRadioButton ("C++");
    c2.setBounds(50, 150, 50,50);
    JRadioButton   c3  =  new  JRadioButton ("Java", true);
    c3.setBounds(50, 200, 70,50);
    f.add(c1);
   f.add(c2);
   f.add(c3);
   f.setSize(300, 300);
   f.setLayout(null);
   f.setVisible(true);
   }
}
```

Output -

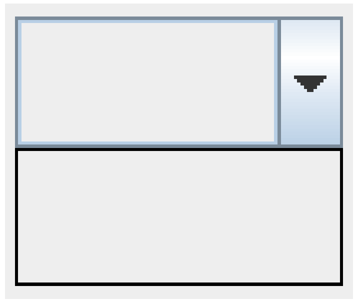


Unit -2 Swings

JComboBox

8. JComboBox

1. Swing provides a combo box (a combination of a text field and a dropdown list) through the JComboBox class.



2. A combo box normally displays one entry. However, it can also display a drop -down list that allows a user to select a different entry.

3. We can also type our selection into the text field.

8. JComboBox

1. **Constructors:-**
   1. JComboBox( )
   2. JComboBox(Vector v)
   3. JComboBox(Objects obj[ ])

2. Here, *v* is a vector that initializes the combo box and *obj* is the array of objects. Items are added to the list of choices via the addItem( ) method, whose signature is shown here:

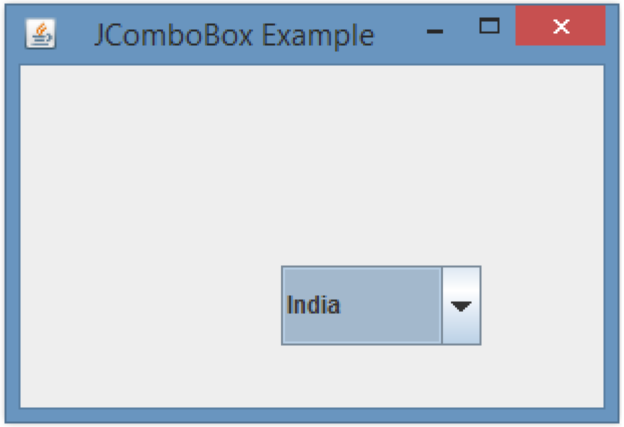
   void addItem(Object obj)

   Here, *obj* is the object to be added to the combo box.

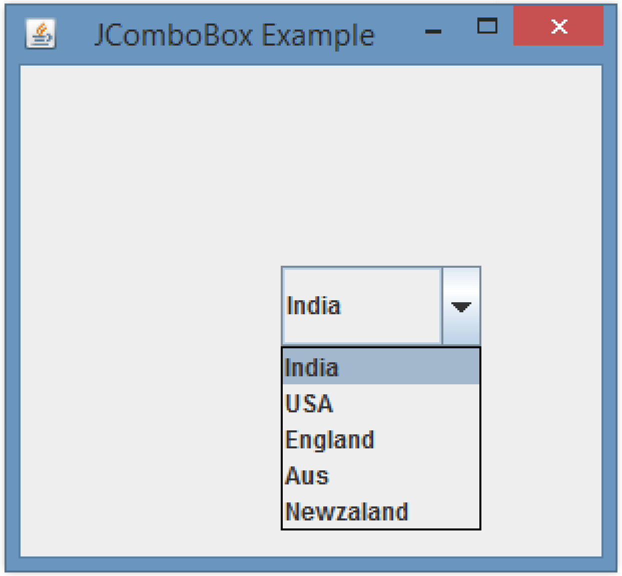
8.JComboBox

```
import java.awt.*;
import javax.swing.*;
public class JComboBoxEx
 {
   public static void main(String args[ ])
   {
    JFrame f = new JFrame("JComboBox Example");
    String country[ ] ={"India", "USA","England","Aus","Newzaland"};
    JComboBox cb = new JComboBox (country);
    cb.setBounds(130, 100, 100, 40);
   f.add(cb);
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
   }
}
```

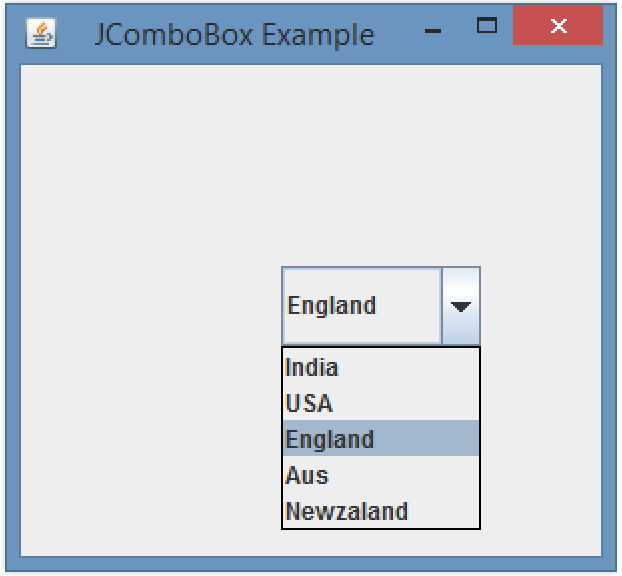
Output -



```
JComboBox Example
India
```



```
JComboBox Example
India
India
USA
England
Aus
Newzaland
```



```
JComboBox Example
England
India
USA
England
Aus
Newzaland
```

Unit -2 Swings

2.4 Advanced Swing Components-JTabbedPane

9.JTabbed Pane

1. A tabbed pane is a component that appears as a group of folders in a file cabinet. Each folder has a title.

2. When a user selects a folder, its contents become visible. Only one of the folders may be selected at a time.

3. Tabbed panes are commonly used for setting configuration options. Tabbed panes are encapsulated by the JTabbedPane class, which extends JComponent.

4. Constructors:-
   1. JTabbedPane( )
   2. JTabbedPane(int tabPlacement)
   3. JTabbedPane(int tabPlacement, int tabLayoutPolicy)

9.JTabbed Panes

1. The first form creates an empty TabbedPane with a default tab placement of JTabbedPane.TOP.
2. Second form creates an empty TabbedPane with the specified tab placement of any of the following:

   JTabbedPane.TOP
   JTabbedPane.BOTTOM
   JTabbedPane.LEFT
   JTabbedPane.RIGHT

3. The third form of constructor creates an empty Tabbed Pane with the specified tab placement and tab layout policy. Tab placements are listed above. Tab layout policy may be either of the following:

   JTabbedPane.WRAP_TAB_LAYOUT
   JTabbedPane.SCROLL_TAB_LAYOUT

1. Tabs are defined via the following method:

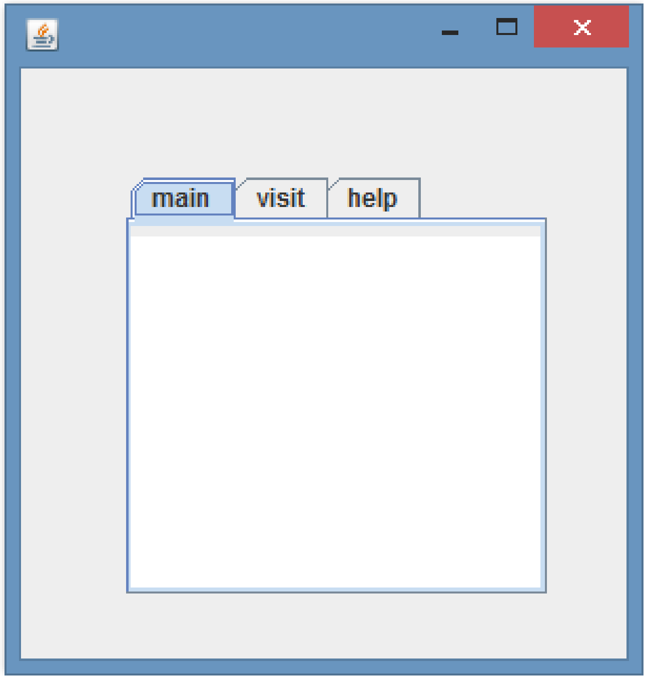
void addTab(String str, Component comp)

2. This is used to add the component in a tabbed pan.
3. Here, str is the title for the tab, and comp is the component that should be added to the tab.
4. Typically, a JPanel or a subclass of it is added.
5. The general procedure to use a tabbed pane in an applet is outlined here:

    1. **Create a JTabbedPane object.**
    2. **Call addTab( ) to add a tab to the pane. (The arguments to this method define the title of the tab and the component it contains.)**
    3. **Repeat step 2 for each tab.**
    4. **Add the tabbed pane to the content pane of the applet.**

```
import java.awt.*;
import javax.swing.*;
public class JTabbedPaneEx
 {
   public static void main(String args[ ])
   {
    JFrame f = new JFrame( );
    JTextArea ta=new JTextArea(200,200);
   JPanel p1 = new JPanel( );
    p1.add(ta);
    JPanel p2 = new JPanel( );
    JPanel p3 = new JPanel( );
    JTabbedPane tp =new JTabbedPane( );
     tp.setBounds(50, 50, 200, 200);
     tp.add("main",p1);
     tp.add("visit",p2);
     tp.add("help",p3);
    f.add(tp);
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
   }
}
```

Output -

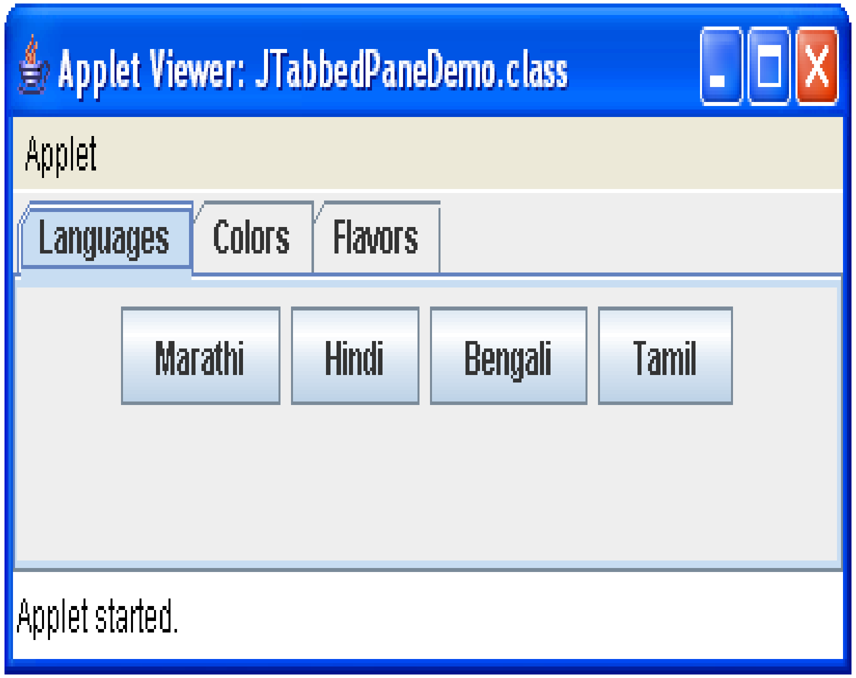


Example :-2

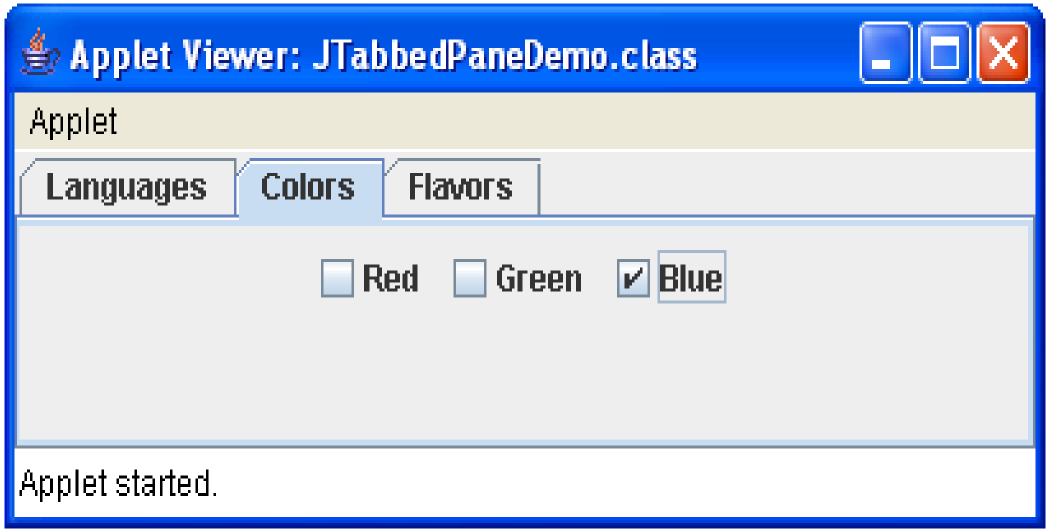
```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

/*<applet code="JTabbedPaneDemo.class" width=400 height=100> </applet>*/
public  class   JTabbedPaneDemo   extends   JApplet
{
public  void   init()
{
JTabbedPane   jtp  =  new  JTabbedPane();

jtp.addTab("Languages", new LangPanel());
jtp.addTab("Colors", new ColorsPanel());
jtp.addTab("Flavors", new FlavorsPanel());
getContentPane().add(jtp);
}
}
```

Example :-2

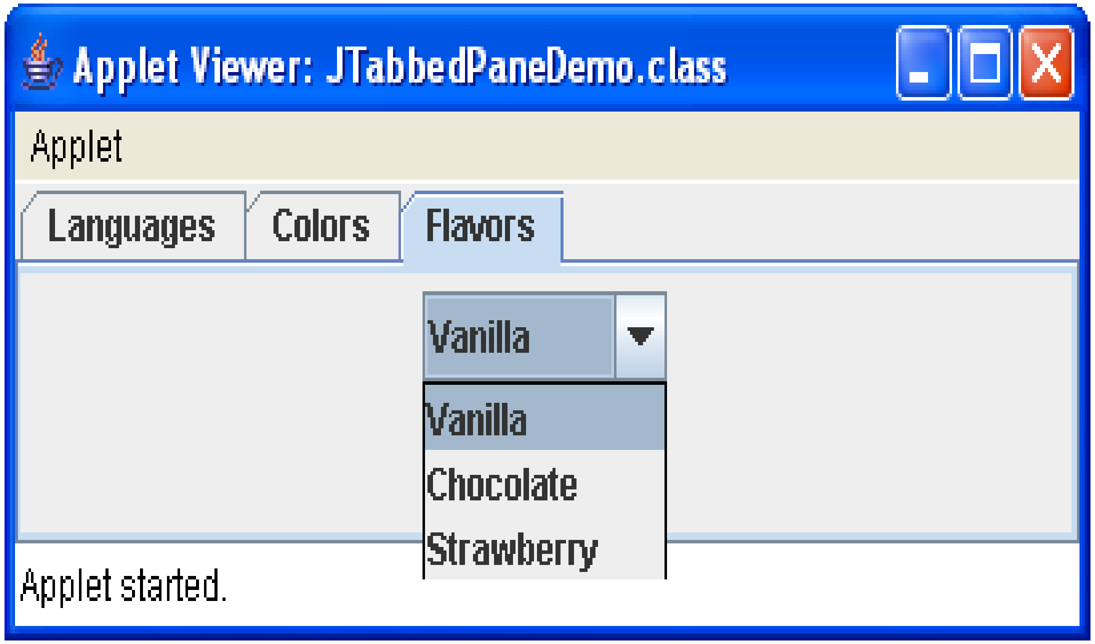
```
class  LangPanel  extends  JPanel
{
public  LangPanel()
{
JButton  b1  =  new JButton("Marathi");
add(b1);
JButton  b2  =  new JButton("Hindi");
add(b2);
Jbutton  b3  =  new JButton("Bengali");
add(b3);
Jbutton  b4  =  new JButton("Tamil");
add(b4);
}
 }
```



```
class  ColorsPanel  extends  Jpanel
 {
public  ColorsPanel()
{
JCheckBox cb1 = new JCheckBox("Red");
add(cb1);
JCheckBox cb2 = new JCheckBox("Green");
add(cb2);
JCheckBox cb3 = new JCheckBox("Blue");
add(cb3);
  }
  }
```



```
class  FlavorsPanel  extends  JPanel
{
public  FlavorsPanel()
{
JComboBox  jcb  =  new  JComboBox();
jcb.addItem("Vanilla");
jcb.addItem("Chocolate");
jcb.addItem("Strawberry");
add(jcb);
}
}
```



Unit -2 Swings

JScrollPane

10.JScrollPane

1. A scroll pane is a component that presents a rectangular area in which a component may be viewed.
2. It is used to make scrollable view of a component. When screen size is limited, we use a scroll pane to display a large component or a component whose size can change dynamically.

3. Horizontal and/or vertical scroll bars may be provided if necessary.

4. Scroll panes are implemented in Swing by the JScrollPane class, which extends JComponent.

5. Constructors :
   1) JScrollPane( )
   2) JScrollPane(Component comp)
   3) JScrollPane(int vsb, int hsb)
   4) JScrollPane(Component comp, int vsb, int hsb)

10. JScrollPane

1. Here, comp is the component to be added to the scroll pane. vsb and hsb are int constants that define when vertical and horizontal scroll bars for this scroll pane are shown.
2. These constants are defined by the Scroll Pane Constants interface.
3. Some examples of these constants are described as follows:

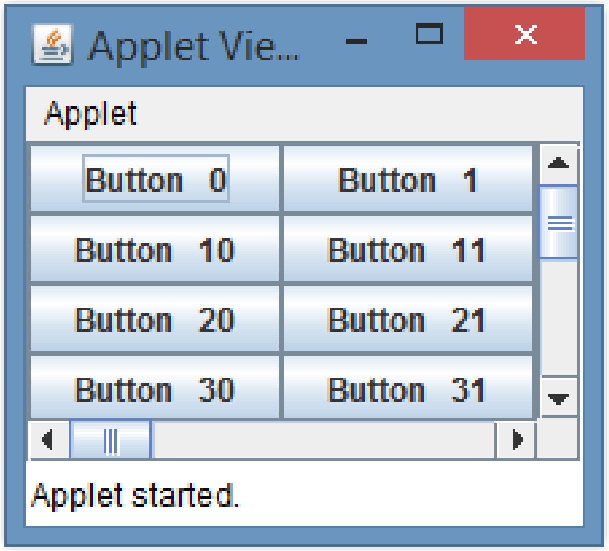
HORIZONTAL_SCROLLBAR_ALWAYS	Always provide Horizontal Scroll bar.
HORIZONTAL_SCROLLBAR_AS_NEEDED	provide Horizontal Scroll bar, if needed.
VERTICAL_SCROLLBAR_ALWAYS	Always provide Vertical Scroll bar.
VERTICAL_SCROLLBAR_AS_NEEDED	provide Vertical Scroll bar, if needed.

1. Here are the steps that you should follow to use a scroll pane in an applet:

1. **Create a JComponent object.**
2. **Create a JScrollPane object. (The arguments to the constructor specify the component and the policies for vertical and horizontal scroll bars.)**
3. **Add the scroll pane to the content pane of the applet.**

```
import java.awt.*;
import javax.swing.*;
/*<applet  code="JScrollPaneEx1.class"  width=300  height=250> </applet> */
public class JScrollPaneEx1 extends Japplet
{
 public void init( )
{
    setLayout(new BorderLayout ( ));
   JPanel jp=new JPanel( );
   jp.setLayout(new GridLayout(10,10));
   int b=0;
  for(int i=0;i<10;i++)  {
        for(int j=0;j<10;j++)   {
              jp.add(new JButton("Button   " +b));
        ++b;  }   }

 int vsb =ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
 int hsb =ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;
 JScrollPane jsp = new JScrollPane(jp, vsb, hsb);
 add(jsp, BorderLayout.CENTER);
 }
}
```

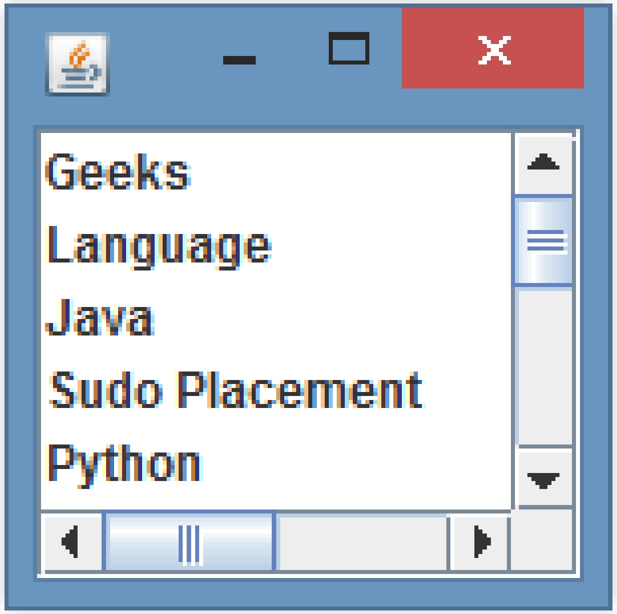
Output -



Applet started.

```
import  java.awt.*;
import javax.swing.*;
public  class  JScrollPaneEx
 {
   public  static  void   main(String  args[  ])
   {
    JFrame  f  =  new  JFrame( );
    String c[ ] = {"Geeks", "Language", "Java", "Sudo Placement", "Python",
                  "CS Subject", "Operating System", "Object Oriented Programming Language",
                  "Data Structure", "Algorithm","Advanced Java Programming Language",
                  "PHP language", "JAVASCRIPT",
                  "C Sharp"};

    JList  list=new JList(c);
    JScrollPane sp=new JScrollPane(list);
    f.setLayout(new BorderLayout( ));
    f.add(sp);
    f.setSize(150,150);
    f.setVisible(true);
   }
}
```

Output -



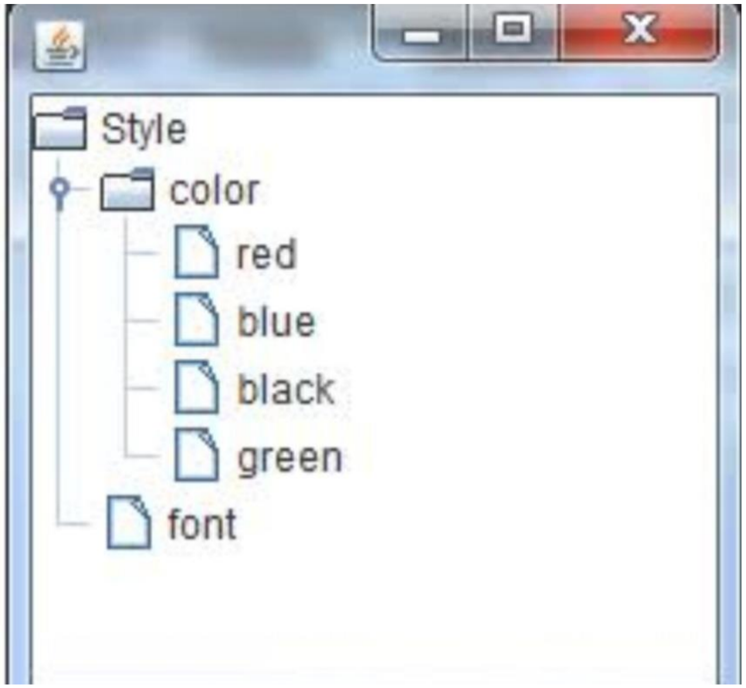
```
Geeks
Language
Java
Sudo Placement
Python
```

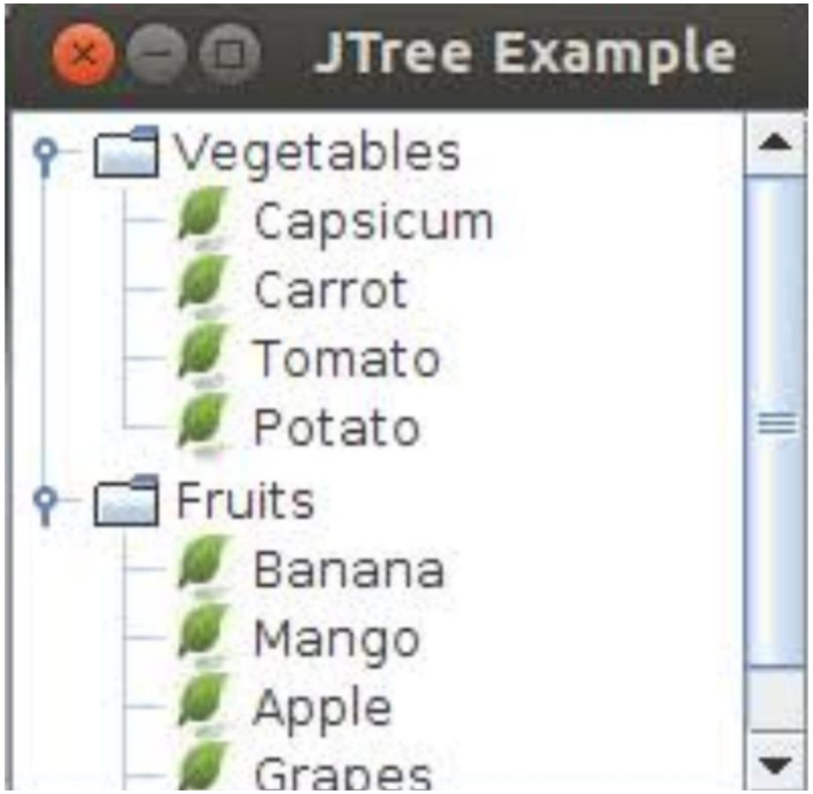
Unit -2 Swings

JTree

11. JTree

1. The JTree class is used to display the tree structured data or hierarchical data.
2. JTree is a complex component.
3. It has a “root node” at the top most which is a parent for all nodes in the tree.
4. It inherits JComponent class.





11. JTree

4. Constructors:-

JTree( )	Creates a JTree with a sample model.
JTree(Object [ ] ob)	Creates a JTree with every element of the specified array as the child of a new root node.
JTree(TreeNode root)	Creates a JTree with the specified TreeNode as its root, which displays the root node.

11. JTree

TreePath getPath( )	It returns a TreePath object that describes the path to the changed node. The TreePath class encapsulates information about a path to a particular node in a tree.

DefaultMutableTreeNode Class :

1. It is used to represent a node in a tree. Using DefaultMutableTreeNode, you can create nodes for the root and for all of the data you want to represent in the tree.

DefaultMutableTreeNode ( )	Creates a node with no associated user object.
DefaultMutableTreeNode (Object ob)	Creates a node to which you can attach children.

1. To create a hierarchy of tree nodes, the add( ) method of DefaultMutableTreeNode can be used:

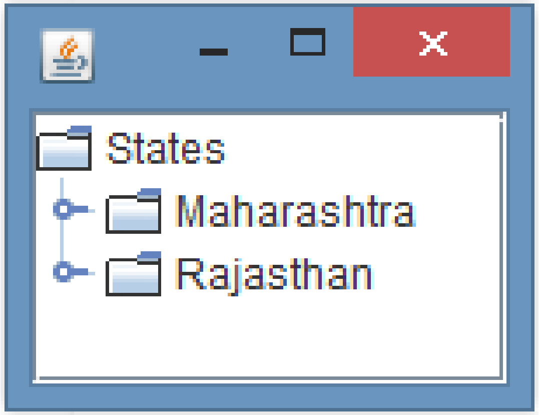
   void add(MutableTreeNode child)

2. Here, child is a mutable tree node that is to be added as a child to the current node.
3. JTree does not provide any scrolling capabilities of its own. Instead, a JTree is typically placed within a JScrollPane. This way, a large tree can be scrolled through a smaller viewport. Here are the steps to follow to use a tree::

   1. **Create an instance of JTree.**
   2. **Create a JScrollPane and specify the tree as the object to be scrolled.**
   3. **Add the tree to the scroll pane.**
   4. **Add the scroll pane to the content pane.**

11.JTree – Program-1

Output -

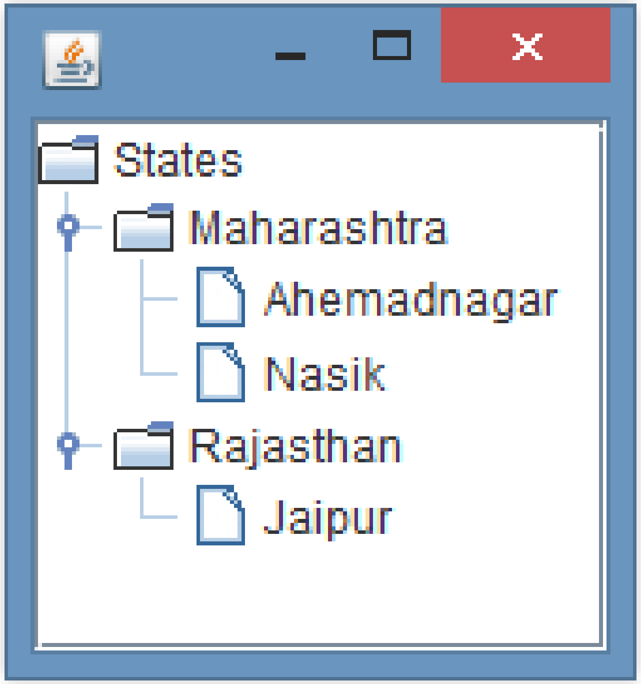

States
Maharashtra
Rajasthan

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

public  class  JTreeEx1
{
  public  static  void   main(String  args[  ])
   {
    JFrame  f  =  new  JFrame( );

     DefaultMutableTreeNode root = new  DefaultMutableTreeNode("States");

     DefaultMutableTreeNode parent1 = new  DefaultMutableTreeNode("Maharashtra");
     DefaultMutableTreeNode child = new  DefaultMutableTreeNode("Ahemadnagar");
     DefaultMutableTreeNode child1 = new  DefaultMutableTreeNode("Nasik");
     DefaultMutableTreeNode parent2 = new  DefaultMutableTreeNode("Rajasthan");
     DefaultMutableTreeNode  child2 = new  DefaultMutableTreeNode("Jaipur");
```

11.JTree – Program-1

Output -

```
 parent1.add(child);
 parent1.add(child1);
 parent2.add(child2);

root.add(parent1);
root.add(parent2);

JTree  jt =new JTree(root);
JScrollPane jsp =new JScrollPane(jt);
f.add(jsp);
f.setSize(200,200);
f.setVisible(true);
}
}
```

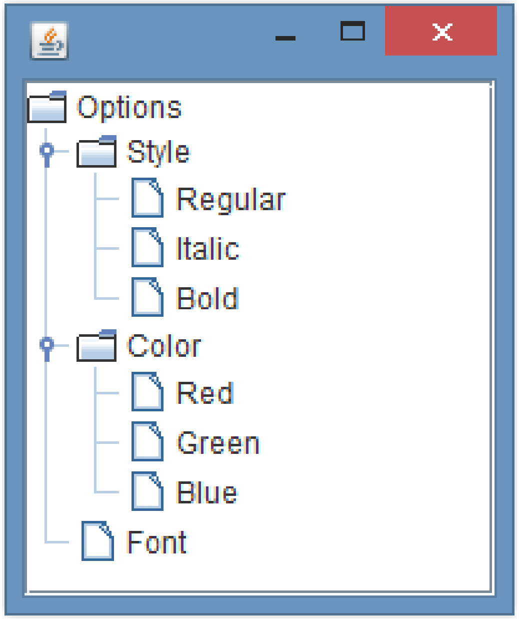


```
States
  Maharashtra
    Ahemadnagar
    Nasik
  Rajasthan
    Jaipur
```

11.JTree – Program-2

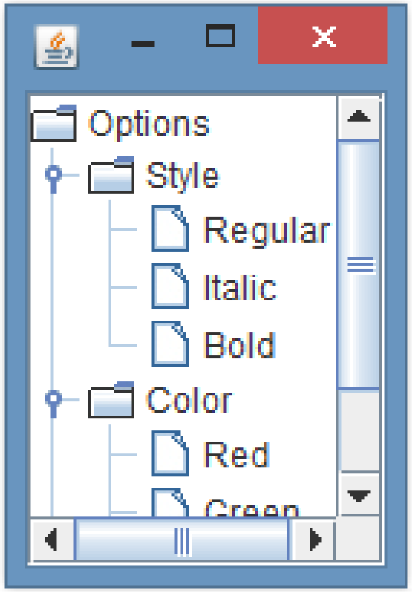
```
import  java.awt.*;
import javax.swing.*;
import  javax.swing.tree.DefaultMutableTreeNode;

public  class  JTreeEx
 {
   public  static  void   main(String  args[  ])
   {
    JFrame  f  =  new  JFrame( );
    //create top node of tree
    DefaultMutableTreeNode  top = new  DefaultMutableTreeNode("Options");

    DefaultMutableTreeNode  style = new  DefaultMutableTreeNode("Style");
    top.add(style);
    DefaultMutableTreeNode  color = new  DefaultMutableTreeNode("Color");
    top.add(color);
    DefaultMutableTreeNode  font = new  DefaultMutableTreeNode("Font");
    top.add(font);
```

Output -



11.JTree – Program-2

```
//create subtree  of  Style
   DefaultMutableTreeNode  s1 = new  DefaultMutableTreeNode("Regular");
   DefaultMutableTreeNode  s2 = new  DefaultMutableTreeNode("Italic");
   DefaultMutableTreeNode  s3 = new  DefaultMutableTreeNode("Bold");
   style.add(s1);   style.add(s2);   style.add(s3);

//create subtree  of  Color
   DefaultMutableTreeNode  c1 = new  DefaultMutableTreeNode("Red");
   DefaultMutableTreeNode  c2 = new  DefaultMutableTreeNode("Green");
   DefaultMutableTreeNode  c3 = new  DefaultMutableTreeNode("Blue");
   color.add(c1);   color.add(c2);   color.add(c3);

   JTree  jt =new JTree(top);
  JScrollPane jsp =new JScrollPane(jt);
   f.add(jsp);
  f.setSize(100,100);
  f.setVisible(true);
  }
}
```



Unit -2 Swings

JTable

12. JTable

1. The JTable class is used to display data in tabular form, means rows and columns.
2. JTable class use to create tables for a GUI based application.
3. You can drag the cursor on column boundaries to resize columns. You can also drag a column to a new position.
4. Depending on its configuration, it is also possible to select a row, column, or cell within the table, and to change the data within a cell.
5. At its core, **JTable is conceptually simple. It is a component that consists of one or more** columns of information. At the top of each column is a heading.
6. In addition to describing the data in a column, the heading also provides the mechanism by which the user can change the size of a column or change the location of a column within the table.
7. **JTable does** not provide any scrolling capabilities of its own. Instead, you will normally wrap a **JTable** inside a **JScrollPane.**

12. JTable

1. **Constructors:-**

JTable( )	Creates a Table with empty cells.
JTree(Object rows [ ] [ ], Object columns[ ])	Creates a table with the specified data. Here, data is a two-dimensional array of the information to be presented, and colHeads is a one-dimensional array with the column headings.

2. Here are the steps required to set up a simple JTable that can be used to display data:

1. Create an instance of JTable.

2. Create a JScrollPane object, specifying the table as the object to scroll.

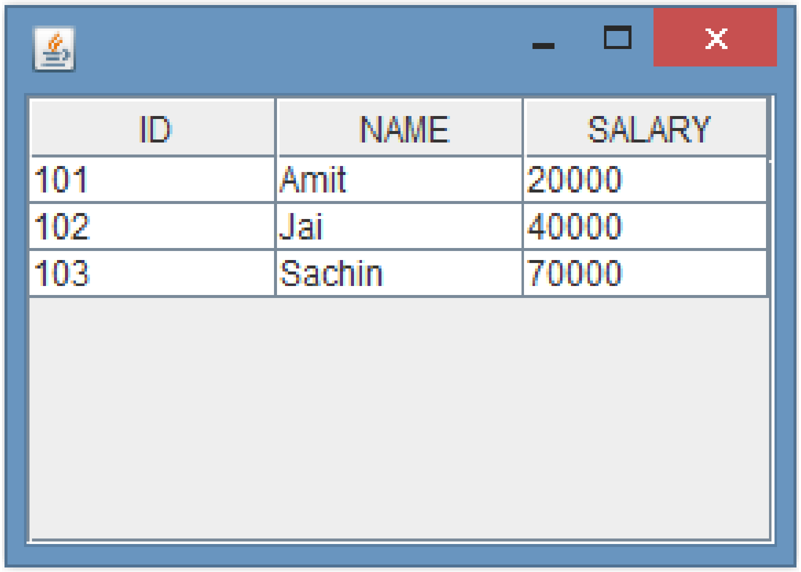
3. Add the table to the scroll pane.

4. Add the scroll pane to the content pane.

JTable

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

public  class  JTableEx
 {
   public  static  void   main(String  args[  ])
   {
    JFrame  f  =  new  JFrame( );
    String  data[ ][ ] ={{ "101","Amit","20000"},
                         { "102","Jai","40000"},
                         { "103","Sachin","70000"}};
   String column[ ]={"ID","NAME","SALARY"};
   JTable jt = new JTable(data,column);
   jt.setBounds(30,40,200,300);
  JScrollPane sp= new JScrollPane(jt);
   f.add(sp);
   f.setSize(300,400);
   f.setVisible(true);
   }
}
```

Output -



ID	NAME	SALARY
101	Amit	20000
102	Jai	40000
103	Sachin	70000

Unit -2 Swings

JProgressBar

14.JProgressBar

1. The JProgressBar class is used to display the progress of the task. It inherits Jcomponent class.
2. JProgressBar visually displays the progress of some specified task.
3. It shows the percentage of completion of specified task.
4. The progress bar fills up as the task reaches it completion.

5. Constructors:-

JProgressBar ( )	It is used to create a horizontal progress bar but no string text.
JProgressBar(int min, int max)	It is used to create a horizontal progress bar with the specified minimum and maximum value.
JProgressBar(int orient)	It is used to create a progress bar with the specified orientation, it can be either Vertical or Horizontal by using SwingConstants.VERTICAL and SwingConstants.HORIZONTAL constants.

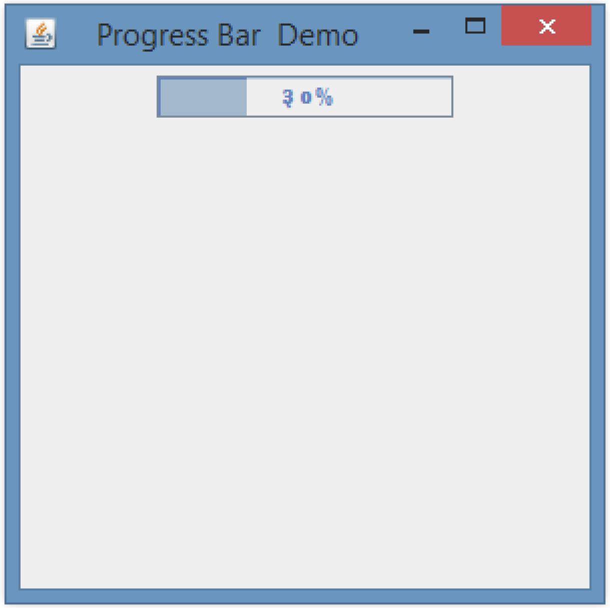
JProgressBar(int orient, int min, int max)	It is used to create a progress bar with the specified orientation, minimum and maximum value.

void setStringPainted (boolean b)	It is used to display whether string should be displayed.
void setString(String s)	It is used to set value to the progress string.
void setOrientation (int orientation)	It is used to set the orientation, it may be either vertical or horizontal by using SwingConstants.VERTICAL and SwingConstants.HORIZONTAL constants.
void setValue(int value)	It is used to set the current value on the progress bar.

JProgreeBar - Program-1

```
import javax.swing.*;

public  class  JProgressBarEx extends  JFrame
 {
    static JProgressBar pb;
   static JPanel  p;
  public  static  void   main(String  args[  ])
  {
    JFrame  f =new JFrame("Progress Bar  Demo");
     p =new JPanel( );
     pb= new  JProgressBar( );
    pb.setValue(0);
    pb.setStringPainted(true);
    p.add(pb);
    f.add(p);
    f. setSize(300, 300);
    f.setVisible(true);
    fill( );
   }
```

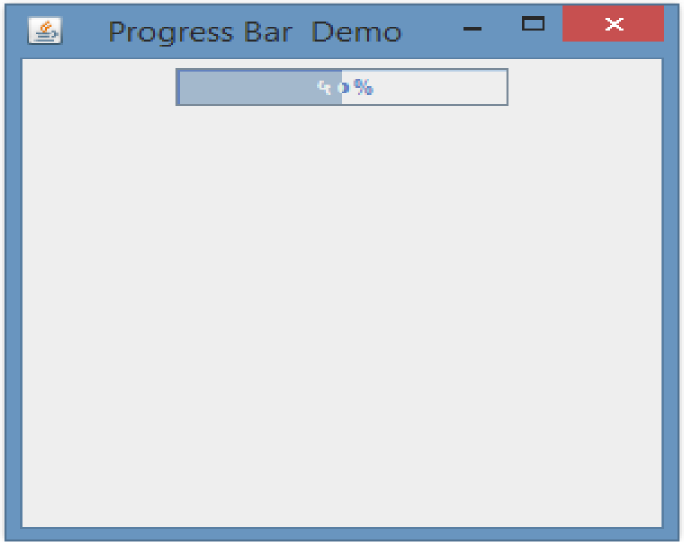
Output -



JProgressBar

```
public  static void fill( )
   {
    int i=0;
    try
     {
      while(i<=100)
        {
          pb.setValue(i+10);
          Thread.sleep(2000);
           i=i+20;
         }
     }
    catch(Exception  e)
       {  }
  }
}
```

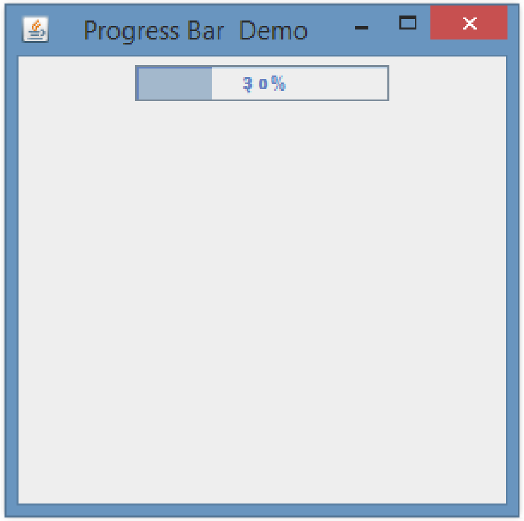
Output -

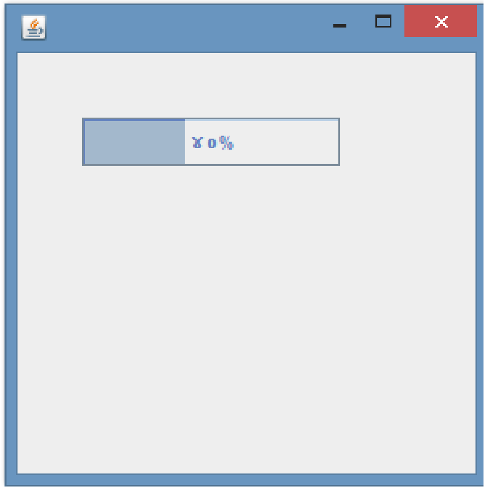


JProgreeBar - Program-2

```
import javax.swing.*;

public  class  JProgressBarEx1 extends  JFrame
 {
   JProgressBar  pb;
   int i=0,num=0;
    JProgressBarEx1( )
  {
    pb= new  JProgressBar(0,2000);
    pb.setBounds(40,40,160,30);
    pb.setValue(0);
    pb.setStringPainted(true);
    add(pb);
    setSize(300, 300);
    setLayout(null);
  }
```

Output -

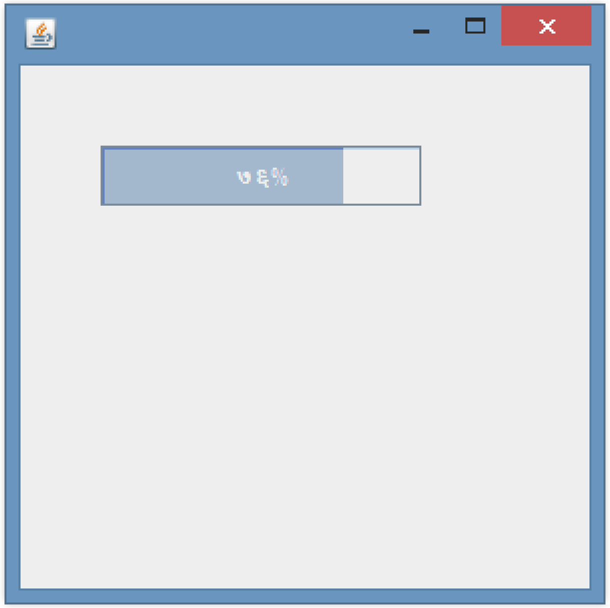




JProgressBar

```
public  void iterate( )
   {
     while(i<=2000)
      {
          pb.setValue(i);
          i=i+20;
       try
        {
          Thread.sleep(150);
         }
      catch(Exception  e)
       {  }
   }
}
 public  static  void   main(String  args[  ])
  {
        JProgressBarEx1 m =new JProgressBarEx1( );
        m.setVisible(true);
        m.iterate( );
  }
}
```

Output -



Unit -2 Swings

JPasswordField

15. JPasswordField

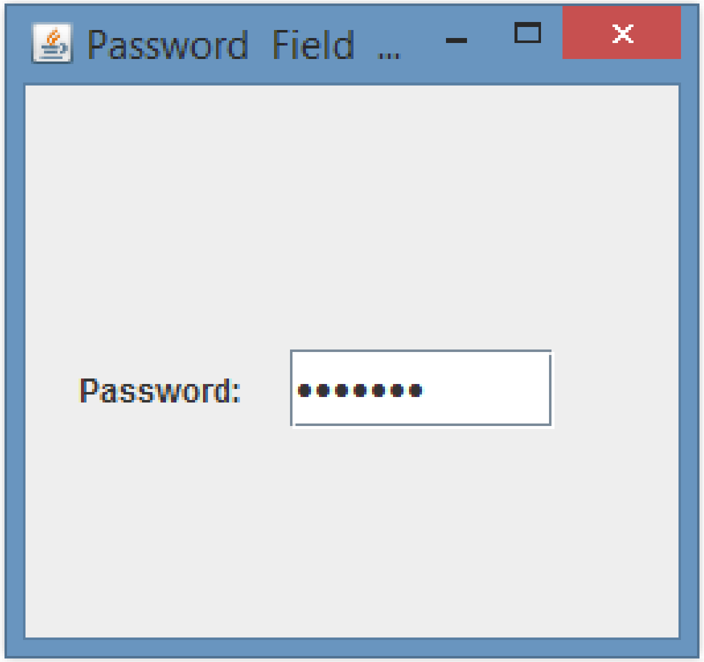
1. The object of a JPasswordField class is a text component specialized for password entry.
2. It allows the editing of a single line of text.
3. It inherits JTextTield class.

JPasswordField ( )	Constructs a new JPassWordField , eith a default document, null starting text string and 0 column width.
JPasswordField (int columns)	Constructs a new empty JPassWordField with the specified number of columns.
JPasswordField (String text)	Constructs a new JPassWordField initialized with the specified text.
JPasswordField (String text, int columns)	Constructs a new JPassWordField initialized with the specified text and columns.

JPasswordField

```
import javax.swing.*;

public class JPasswordEx extends JFrame
{
  public static void main(String args[ ])
  {
    JFrame f =new JFrame("Password Field Example");
    JLabel obj =new JLabel ("Password:");
    obj.setBounds(20,100,80,30);
    JPasswordField pf= new JPasswordField( );
    pf.setBounds(100,100,100,30);
    f.add(obj);
   f.add(pf);
   f. setSize(300, 300);
   f.setLayout(null);
   f.setVisible(true);
  }
 }
```

Output -



Unit -2 Swings

ToolTip

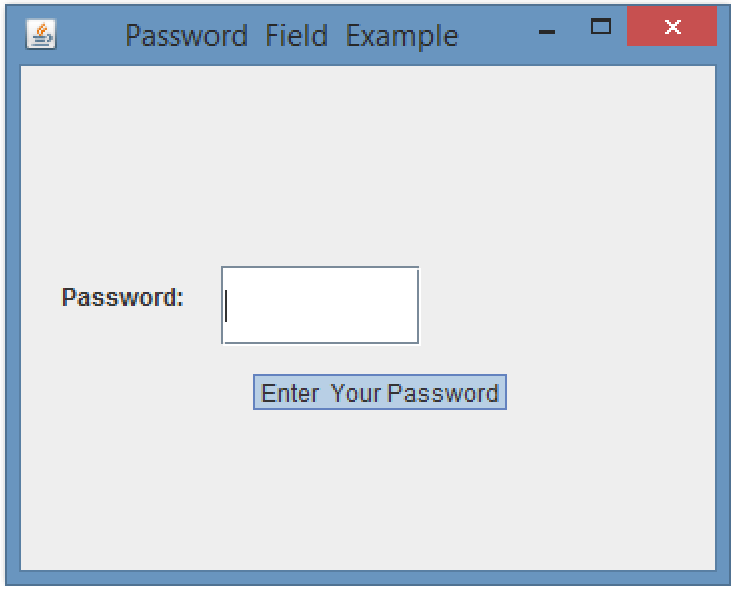
13.ToolTip

1. A ToolTip is a textual pop-up that momentarily appears when the mouse cursor rests inside the components painting region.
2. We can add tooltip text to almost all the components of Java Swing by using the method setToolTipText(String s).
3. This method sets the tooltip of the components to this specified strings.
4. When the cursor enters the boundary of that components a popup appears and text is displayed.
5. A ToolTip is a small pop-up window designed to contain informative text about a component when the mouse moves over it.
6. In short, ToolTip is a string has short description and they are used to explain the functionality of the component.

JToolTip

```
import javax.swing.*;

public  class  JToolTipEx extends  JFrame
 {
  public  static  void   main(String  args[  ])
   {
     JFrame  f =new JFrame("Password  Field  Example");
     JLabel  obj =new JLabel ("Password:");
     obj.setBounds(20,100,80,30);
     JPasswordField  pf= new JPasswordField( );
     pf.setBounds(100, 100, 100, 40);
     pf.setToolTipText("Enter  Your Password");
     f.add(obj);
    f.add(pf);
    f. setSize(300, 300);
    f.setLayout(null);
    f.setVisible(true);
   }
 }
```

Output -


Password Field Example
Password:
Enter Your Password