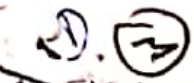


AppletsApplet basics :-

Applets are the small programs that are designed for transmission over the Internet and run within a browser. Applets offer a secure way to dynamically download and execute programs over the Web.

example : program which displays the string "hello java".

```
import java.awt.*;
import java.applet.*;

public class one extends Applet
{
    public void paint (Graphics g)
    {
        g.drawString ("hello java", 10, 10);
    }
}
```

The first import statement imports the Abstract Window Toolkit (AWT) classes. Applets interact with the user through the AWT. The next import statement imports the java.applet package. This package contains the class 'Applet'. Every applet that we create must be a subclass of 'Applet'. The class 'one' must be declared as 'public' because it will be accessed by external code. paint() method is used to display the output of applet. It has one parameter of type 'Graphics'. It describes the graphics environment in which the applet is running.

'drawString()' method is a member of 'Graphics' class. This method displays a string at specified X,Y location. Its syntax is -

drawString (String str, int x, int y)

\* applets does not have a main() method.

After writing the program, do the compilation. Then

2 ways for executing a applet.

1. write short HTML text file that contains a tag that loads the applet. using the applet tag like -

```
<applet code = "one" width = 100 height = 100 >
</applet >
```

the width & height indicates size of applet.

The save these applet tag as filename.html, then we execute the program with the command as -

```
appletviewer filename.html
```

2. simply write the two lines of applet tag at the beginning of a program within the comments. i.e

```
import java.awt.* ;
" java.applet.* ;

/*
  <applet code = "one" width = 100 height = 100>
  </applet >
*/

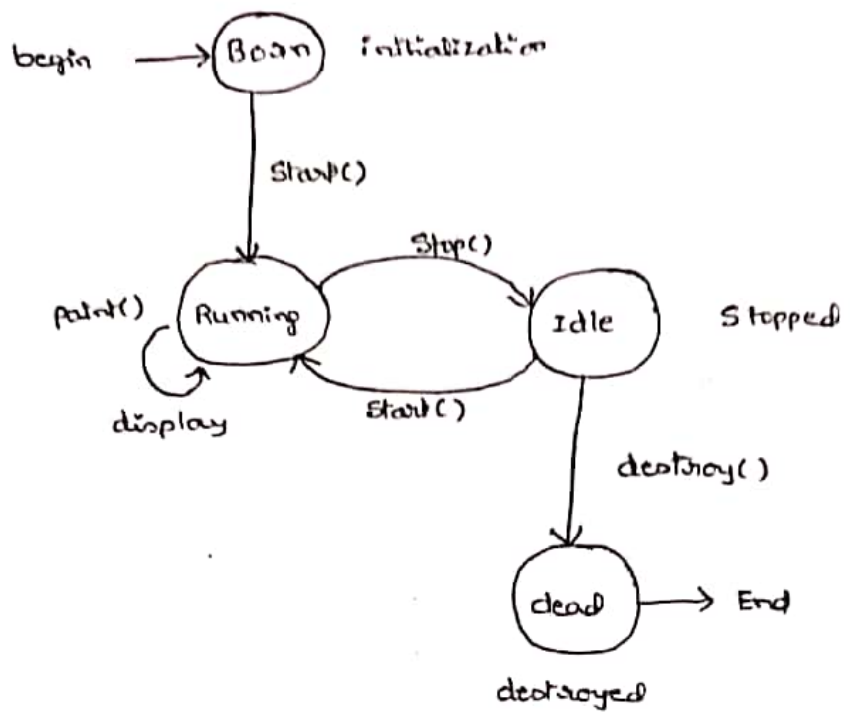
public class one extends Applet
{
  ≡
}
```

Then we execute this program as -

```
appletviewer one.java.
```

complete applet skeleton :-

Applet life cycle :-



There are 4 states in applet life cycle -

- Born is initialization state
- running state
- idle "
- dead "

1. Born state - whenever applet is created it is said to be in born state. Here we do initialization for the applet. This can be done by `init()` method. Syntax is -

```

public void init()
{
    ≡
}
  
```

2. running state - when the applet is started its execution by using `start()` method, then the state is running state. Syntax is -

```

public void start()
{
    ≡
}
  
```

3. idle state - when applet is running & it is stopped by stop() method. Then applet is in idle state. Syntax is -

```
public void Stop()  
{  
    =  
}
```

4. dead state - An applet is said to be dead when it is removed from memory. This is done by destroy() method. Destroying state occurs only once in the applet life cycle. Syntax is -

```
public void destroy()  
{  
    =  
}
```

The Skeleton of applet is shown below -

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

```
"    java. applet.*;
```

```
/*
```

```
<applet code = "one" width = 100 height = 100>
```

```
</applet>
```

```
*/
```

```
public class one extends Applet
```

```
{
```

```
    public void init()
```

```
    {
```

```
        // initialization
```

```
    }
```

```
    public void Start()
```

```
    {
```

```
        // start & resume execution
```

```
    }
```

```
public void destroy()stop()  
{  
    // perform shutdown suspends execution  
}  
  
public void destroy()  
{  
    // perform shutdown  
}  
  
public void paint ( Graphics g )  
{  
    // display contents of window  
}  
}
```

### Applet Initialization and Termination :-

when an applet begins, the following methods are called in this sequence :

1. init()
2. start()
3. paint()

when an applet is terminated, the following methods are called in this sequence :

1. stop()
2. destroy()

The init() method is the first method to be called. Here initialization can be done. The start() method is called after init(). The paint() is called to output an applet.

## Passing parameters to applets :-

We can give user defined parameters to an applet using `<param>` tag. Each `<param>` tag has a 'name' attribute & 'value'. eg: name is color, then value is red.

Inside the applet code, the applet can refer to that parameter by name to find its value. For example, we can change the color of the text displayed to red by an applet using a `<param>` tag as follows:

```
< applet ---- >  
  < param name = "color" value = "red" >  
</applet >
```

There are 2 steps to pass parameters -

1. include `<param>` tag in the HTML document
2. provide code in the applet to pass these parameters.

parameters are passed to an applet when it is loaded. we can define the `init()` method in the applet to get hold of parameters defined in the `<param>` tag. This is done using the `getParameter()` method, it takes 'name' of the parameter & returns a string containing the value of that parameter.

program:-

```
import java.awt.*;
" java.applet.*;

/*
<applet code = "one" width = 100 height = 100>
<param name = "string" value = "Applet">
</applet>
*/
public class one extends Applet
{
    String str;
    public void init()
    {
        str = getParameter("string");
        if (str == null)
            str = "Java";
        str = "Hello" + str;
    }
    public void paint (Graphics g)
    {
        g.drawString (str, 10, 100);
    }
}
```

O/p

hello Applet

if we remove `<param>` tag,

O/p

hello Java



## Java Swings

**Java Swing** is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing

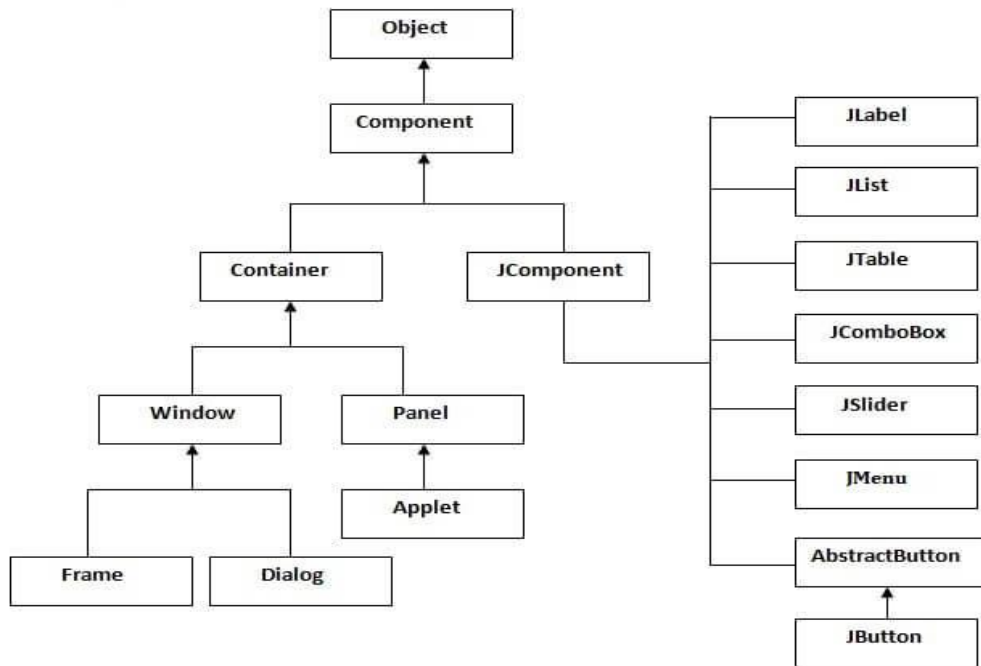
There are many differences between java awt and swing that are given below.

No.	Java AWT	Java Swing
1)	AWT components are <b>platform-dependent</b> .	Java swing components are <b>platform-independent</b> .
2)	AWT components are <b>heavyweight</b> .	Swing components are <b>lightweight</b> .
3)	AWT <b>doesn't support pluggable look and feel</b> .	Swing <b>supports pluggable look and feel</b> .
4)	AWT provides <b>less components</b> than Swing.	Swing provides <b>more powerful components</b> such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT <b>doesn't follow MVC</b> (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing <b>follows MVC</b> .



## Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.



The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default

	false.
--	--------

## Java Swing Examples

There are two ways to create a frame:

- By creating the object of Frame class (association)
- By extending Frame class (inheritance)

We can write the code of swing inside the main(), constructor or any other method.

### Java JFrame

The javax.swing.JFrame class is a type of container which inherits the java.awt.Frame class. JFrame works like the main window where components like labels, buttons, textfields are added to create a GUI. Unlike Frame, JFrame has the option to hide or close the window with the help of setDefaultCloseOperation(int) method.

Constructors

Constructor	Description
JFrame()	It constructs a new frame that is initially invisible.
JFrame(GraphicsConfiguration gc)	It creates a Frame in the specified GraphicsConfiguration of a screen device and a blank title.
JFrame(String title)	It creates a new, initially invisible Frame with the specified title.
JFrame(String title, GraphicsConfiguration gc)	It creates a JFrame with the specified title and the specified GraphicsConfiguration of a screen device.

Useful Methods

Modifier Type	and	Method	Description
protected void		addImpl(Component comp, Object constraints, int index)	Adds the specified child Component.
protected JRootPane		createRootPane()	Called by the constructor methods to create the default rootPane.
protected void		frameInit()	Called by the constructors to init the JFrame properly.
Void		setContentPane(Container contentPane)	It sets the contentPane property
static void		setDefaultLookAndFeelDecorated(boolean defaultLookAndFeelDecorated)	Provides a hint as to whether or not newly created JFrames should have their Window decorations (such as borders, widgets to close the window, title...) provided by the current look and feel.
Void		setIconImage(Image image)	It sets the image to be displayed as the icon for this window.
Void		setJMenuBar(JMenuBar menubar)	It sets the menubar for this frame.

Void	setLayeredPane(JLayeredPane layeredPane)	It sets the layeredPane property.
JRootPane	getRootPane()	It returns the rootPane object for this frame.
TransferHandler	getTransferHandler()	It gets the transferHandler property.

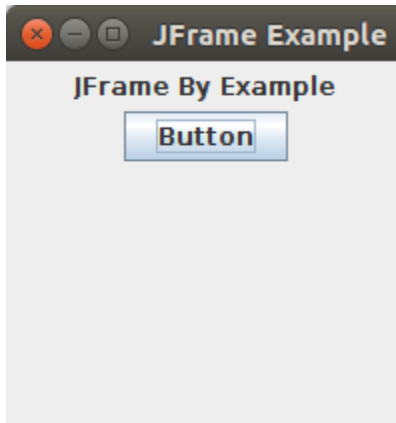
### JFrame Example

```

import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
public class JFrameExample
{
    public static void main(String s[])
    {
        JFrame frame = new JFrame("JFrame Example");
        JPanel panel = new JPanel();
        panel.setLayout(new FlowLayout());
        JLabel label = new JLabel("JFrame By Example");
        JButton button = new JButton();
        button.setText("Button");
        panel.add(label);
        panel.add(button);
        frame.add(panel);
        frame.setSize(200, 300);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}

```

### Output



## **Java JComponent**

The JComponent class is the base class of all Swing components except top-level containers. Swing components whose names begin with "J" are descendants of the JComponent class. For example, JButton, JScrollPane, JPanel, JTable etc. But, JFrame and JDialog don't inherit JComponent class because they are the child of top-level containers.

The JComponent class extends the Container class which itself extends Component. The Container class has support for adding components to the container.

### Constructor

Constructor	Description
JComponent()	Default JComponent constructor.

### Useful Methods

Modifier Type	and Method	Description
void	setActionMap(ActionMap am)	It sets the ActionMap to am.
void	setBackground(Color bg)	It sets the background color of this component.

void	setFont(Font font)	It sets the font for this component.
void	setMaximumSize(Dimension maximumSize)	It sets the maximum size of this component to a constant value.
void	setMinimumSize(Dimension minimumSize)	It sets the minimum size of this component to a constant value.
protected void	setUI(ComponentUI newUI)	It sets the look and feel delegate for this component.
void	setVisible(boolean aFlag)	It makes the component visible or invisible.
void	setForeground(Color fg)	It sets the foreground color of this component.
String	getToolTipText(MouseEvent event)	It returns the string to be used as the tooltip for event.
Container	getTopLevelAncestor()	It returns the top-level ancestor of this component (either the containing Window or Applet), or null if this component has not been added to any container.
TransferHandler	getTransferHandler()	It gets the transferHandler property.

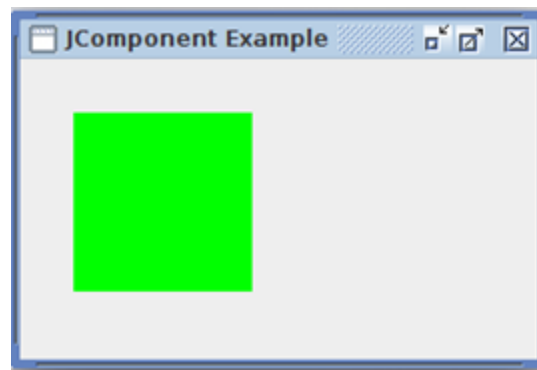
### Java JComponent Example

```
import java.awt.Color;
import java.awt.Graphics;
```

```

import javax.swing.JComponent;
import javax.swing.JFrame;
class MyJComponent extends JComponent {
    public void paint(Graphics g) {
        g.setColor(Color.green);
        g.fillRect(30, 30, 100, 100);
    }
}
public class JComponentExample {
    public static void main(String[] arguments) {
        MyJComponent com = new MyJComponent();
        // create a basic JFrame
        JFrame.setDefaultLookAndFeelDecorated(true);
        JFrame frame = new JFrame("JComponent Example");
        frame.setSize(300,200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // add the JComponent to main frame
        frame.add(com);
        frame.setVisible(true);
    }
}

```



## **Java JButton**

The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

JButton class declaration

Let's see the declaration for javax.swing.JButton class.



1. **public class JButton extends** AbstractButton **implements** Accessible

Commonly used Constructors:

Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

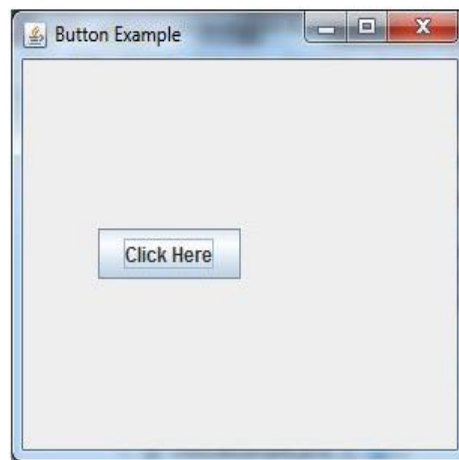
Commonly used Methods of AbstractButton class:

Methods	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the <u>action listener</u> to this object.

```

import javax.swing.*;
public class ButtonExample
{
public static void main(String[] args)
{
    JFrame f=new JFrame("Button Example");
    JButton b=new JButton("Click Here");
    b.setBounds(50,100,95,30);
    f.add(b);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}

```



## **Java JLabel**

The object of JLabel class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly. It inherits JComponent class.

JLabel class declaration

Let's see the declaration for javax.swing.JLabel class.

1. **public class JLabel extends JComponent implements** SwingConstants, Accessible

### Commonly used Constructors:

Constructor	Description
JLabel()	Creates a JLabel instance with no image and with an empty string for the title.
JLabel(String s)	Creates a JLabel instance with the specified text.
JLabel(Icon i)	Creates a JLabel instance with the specified image.
JLabel(String s, Icon i, int horizontalAlignment)	Creates a JLabel instance with the specified text, image, and horizontal alignment.

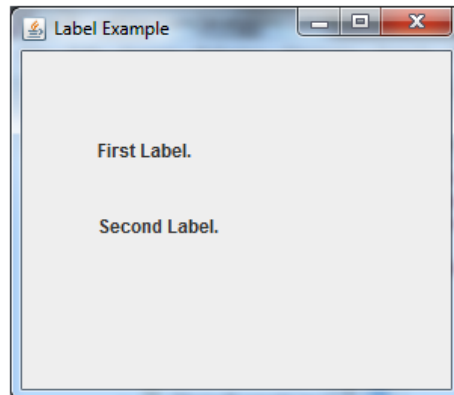
### Commonly used Methods:

Methods	Description
String getText()	It returns the text string that a label displays.
void setText(String text)	It defines the single line of text this component will display.
void setHorizontalAlignment(int alignment)	It sets the alignment of the label's contents along the X axis.
Icon getIcon()	It returns the graphic image that the label displays.
int getHorizontalAlignment()	It returns the alignment of the label's contents along the X axis.

Java JLabel Example

```
import javax.swing.*;
class LabelExample
{
public static void main(String args[])
{
    JFrame f= new JFrame("Label Example");
    JLabel l1,l2;
    l1=new JLabel("First Label.");
    l1.setBounds(50,50, 100,30);
    l2=new JLabel("Second Label.");
    l2.setBounds(50,100, 100,30);
    f.add(l1); f.add(l2);
    f.setSize(300,300);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

Output:



## **Java JTextField**

The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class.

JTextField class declaration

Let's see the declaration for javax.swing.JTextField class.

1. **public class** JTextField **extends** JTextComponent **implements** SwingConstants

### Commonly used Constructors:

Constructor	Description
<code>TextField()</code>	Creates a new TextField
<code>TextField(String text)</code>	Creates a new TextField initialized with the specified text.
<code>TextField(String text, int columns)</code>	Creates a new TextField initialized with the specified text and columns.
<code>TextField(int columns)</code>	Creates a new empty TextField with the specified number of columns.

### Commonly used Methods:

Methods	Description
<code>void addActionListener(ActionListener l)</code>	It is used to add the specified action listener to receive action events from this textfield.
<code>Action getAction()</code>	It returns the currently set Action for this ActionEvent source, or null if no Action is set.
<code>void setFont(Font f)</code>	It is used to set the current font.
<code>void removeActionListener(ActionListener l)</code>	It is used to remove the specified action listener so that it no longer receives action events from this textfield.

## Java JTextField Example

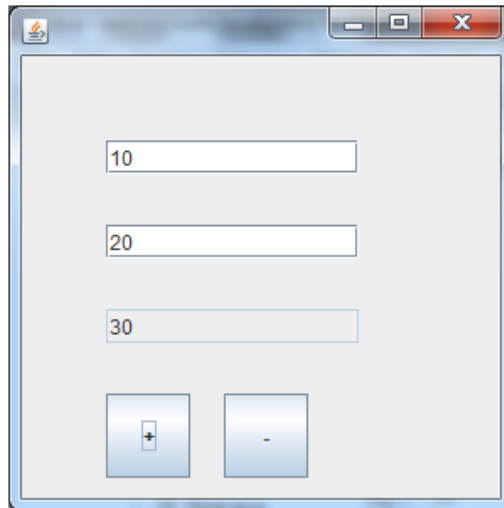
```
import javax.swing.*;
import java.awt.event.*;

public class TextFieldExample implements ActionListener{
    JTextField tf1,tf2,tf3;
    JButton b1,b2;
    TextFieldExample(){
        JFrame f= new JFrame();
        tf1=new JTextField();
        tf1.setBounds(50,50,150,20);
        tf2=new JTextField();
        tf2.setBounds(50,100,150,20);
        tf3=new JTextField();
        tf3.setBounds(50,150,150,20);
        tf3.setEditable(false);
        b1=new JButton("+");
        b1.setBounds(50,200,50,50);
        b2=new JButton("-");
        b2.setBounds(120,200,50,50);
        b1.addActionListener(this);
        b2.addActionListener(this);
        f.add(tf1);f.add(tf2);f.add(tf3);f.add(b1);f.add(b2);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
    public void actionPerformed(ActionEvent e)
    {
        String s1=tf1.getText();
        String s2=tf2.getText();
        int a=Integer.parseInt(s1);
        int b=Integer.parseInt(s2);
        int c=0;
        if(e.getSource()==b1){
            c=a+b;
        }else if(e.getSource()==b2){
            c=a-b;
        }
        String result=String.valueOf(c);
```

```
        tf3.setText(result);
    }

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

Output:



## **Java JCheckBox**

The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on ".It inherits JToggleButton class.

JCheckBox class declaration

Let's see the declaration for javax.swing.JCheckBox class.

1. **public class JCheckBox extends JToggleButton implements Accessible**



### Commonly used Constructors:

Constructor	Description
JJCheckBox()	Creates an initially unselected check box button with no text, no icon.
JChechBox(String s)	Creates an initially unselected check box with text.
JCheckBox(String text, boolean selected)	Creates a check box with text and specifies whether or not it is initially selected.
JCheckBox(Action a)	Creates a check box where properties are taken from the Action supplied.

### Commonly used Methods:

Methods	Description
AccessibleContext getAccessibleContext()	It is used to get the AccessibleContext associated with this JCheckBox.
protected String paramString()	It returns a string representation of this JCheckBox.

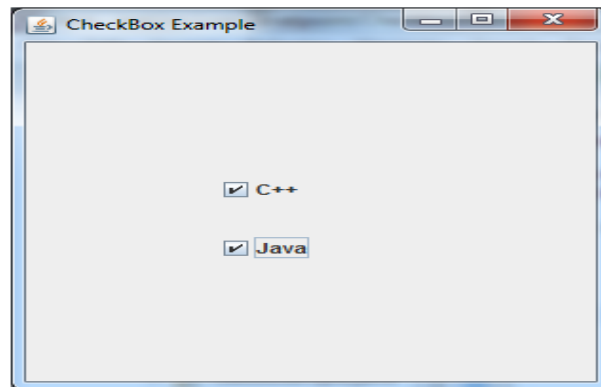
### Example:

```
import javax.swing.*;
public class CheckBoxExample
{
    CheckBoxExample(){
        JFrame f= new JFrame("CheckBox Example");
        JCheckBox checkBox1 = new JCheckBox("C++");
        checkBox1.setBounds(100,100, 50,50);
        JCheckBox checkBox2 = new JCheckBox("Java", true);
        checkBox2.setBounds(100,150, 50,50);
        f.add(checkBox1);
        f.add(checkBox2);
    }
}
```

```

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

Output:



### **Java JRadioButton:**

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.

It should be added in ButtonGroup to select one radio button only.

JRadioButton class declaration

Let's see the declaration for javax.swing.JRadioButton class.

1. **public class JRadioButton extends JToggleButton implements Accessible**

Commonly used Constructors:

Constructor	Description
JRadioButton()	Creates an unselected radio button with no text.

JRadioButton(String s)	Creates an unselected radio button with specified text.
JRadioButton(String s, boolean selected)	Creates a radio button with the specified text and selected status.

Commonly used Methods:

Methods	Description
void setText(String s)	It is used to set specified text on button.
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

### Java JRadioButton Example

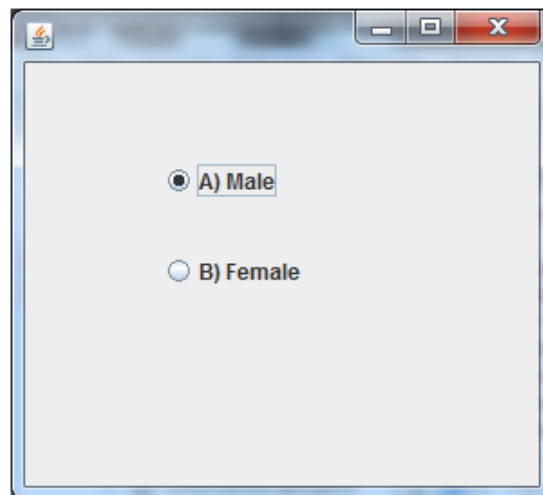
```
import javax.swing.*;
public class RadioButtonExample {
    JFrame f;
    RadioButtonExample(){
        f=new JFrame();
        JRadioButton r1=new JRadioButton("A) Male");
        JRadioButton r2=new JRadioButton("B) Female");
```

```

r1.setBounds(75,50,100,30);
r2.setBounds(75,100,100,30);
ButtonGroup bg=new ButtonGroup();
bg.add(r1);bg.add(r2);
f.add(r1);f.add(r2);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String[] args) {
    new RadioButtonExample();
}
}

```

Output:



## **Java JComboBox**

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits JComponent class.

JComboBox class declaration

Let's see the declaration for javax.swing.JComboBox class.

1. **public class** JComboBox **extends** JComponent **implements** ItemSelectable, ListDataListener, ActionListener, Accessible

### Commonly used Constructors:

Constructor	Description
JComboBox()	Creates a JComboBox with a default data model.
JComboBox(Object[] items)	Creates a JComboBox that contains the elements in the specified <u>array</u> .
JComboBox(Vector<?> items)	Creates a JComboBox that contains the elements in the specified <u>Vector</u> .

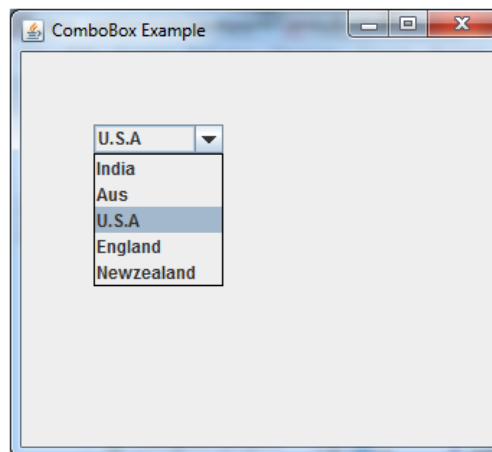
### Commonly used Methods:

Methods	Description
void addItem(Object anObject)	It is used to add an item to the item list.
void removeItem(Object anObject)	It is used to delete an item to the item list.
void removeAllItems()	It is used to remove all the items from the list.
void setEditable(boolean b)	It is used to determine whether the JComboBox is editable.
void addActionListener(ActionListener a)	It is used to add the <u>ActionListener</u> .
void addItemListener(ItemListener i)	It is used to add the <u>ItemListener</u> .

```

import javax.swing.*;
public class ComboBoxExample {
    JFrame f;
    ComboBoxExample(){
        f=new JFrame("ComboBox Example");
        String country[]={"India","Aus","U.S.A","England","Newzealand"};
        JComboBox cb=new JComboBox(country);
        cb.setBounds(50, 50,90,20);
        f.add(cb);
        f.setLayout(null);
        f.setSize(400,500);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new ComboBoxExample();
    }
}

```



## **Java JTabbedPane**

The JTabbedPane class is used to switch between a group of components by clicking on a tab with a given title or icon. It inherits JComponent class.

JTabbedPane class declaration

Let's see the declaration for javax.swing.JTabbedPane class.

1. **public class** JTabbedPane **extends** JComponent **implements** Serializable, Accessible, SwingConstants

Commonly used Constructors:

Constructor	Description
JTabbedPane()	Creates an empty TabbedPane with a default tab placement of JTabbedPane.Top.
JTabbedPane(int tabPlacement)	Creates an empty TabbedPane with a specified tab placement.
JTabbedPane(int tabPlacement, int tabLayoutPolicy)	Creates an empty TabbedPane with a specified tab placement and tab layout policy.

### Java JTabbedPane Example

```
import javax.swing.*;
public class TabbedPaneExample {
    JFrame f;
    TabbedPaneExample(){
        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);
    }
}
```

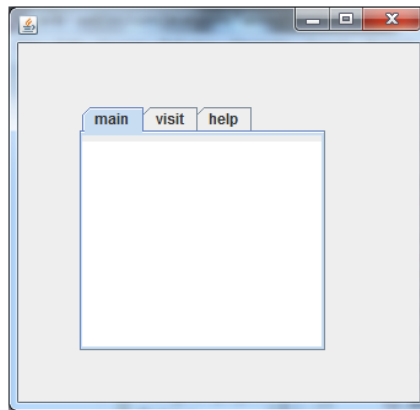


```

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

```

Output:



## **Java JScrollPane**

A JScrollPane 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.

Constructors

Constructor	Purpose
JScrollPane()	It creates a scroll pane. The Component parameter, when present, sets the scroll pane's client. The two int parameters, when present, set the vertical and horizontal scroll bar policies (respectively).
JScrollPane(Component)	
JScrollPane(int, int)	
JScrollPane(Component, int, int)	

## Useful Methods

Modifier	Method	Description
void	setColumnHeaderView(Component)	It sets the column header for the scroll pane.
void	setRowHeaderView(Component)	It sets the row header for the scroll pane.
void	setCorner(String, Component)	It sets or gets the specified corner. The int parameter specifies which corner and must be one of the following constants defined in <code>ScrollPaneConstants</code> : UPPER_LEFT_CORNER, UPPER_RIGHT_CORNER, LOWER_LEFT_CORNER, LOWER_RIGHT_CORNER, LOWER_LEADING_CORNER, LOWER_TRAILING_CORNER, UPPER_LEADING_CORNER, UPPER_TRAILING_CORNER.
Component	getCorner(String)	
void	setViewportView(Component)	Set the scroll pane's client.

### JScrollPane Example

```
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
public class JScrollPaneExample
{
    private static final long serialVersionUID = 1L;
    private static void createAndShowGUI()
{
```

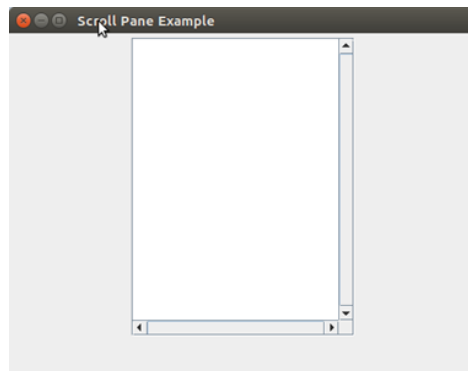
```

// Create and set up the window.
final JFrame frame = new JFrame("Scroll Pane Example");
    // Display the window.
    frame.setSize(500, 500);
    frame.setVisible(true);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    // set flow layout for the frame
    frame.getContentPane().setLayout(new FlowLayout());
    JTextArea textArea = new JTextArea(20, 20);
    JScrollPane scrollableTextArea = new JScrollPane(textArea);
    scrollableTextArea.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBA
R_ALWAYS);
    scrollableTextArea.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWA
YS); frame.getContentPane().add(scrollableTextArea);
}

public static void main(String[] args)
{
    javax.swing.SwingUtilities.invokeLater(new Runnable()
    {
        public void run() {
            createAndShowGUI();
        }
    });
}
}

```

Output:



## **Java JTable**

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

JTable class declaration

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

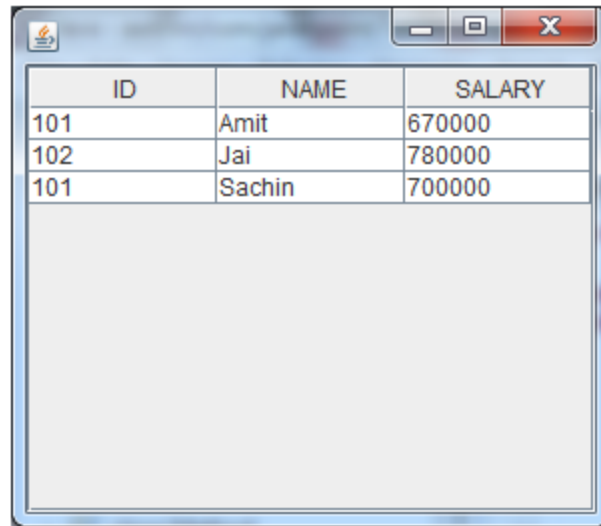
Commonly used Constructors:

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

Java JTable Example

```
import javax.swing.*;
public class TableExample {
    JFrame f;
    TableExample(){
        f=new JFrame();
        String data[][]={ {"101","Amit","670000"},
                           {"102","Jai","780000"},
                           {"101","Sachin","700000"} };
        String column[]={ "ID","NAME","SALARY"};
        JTable jt=new JTable(data,column);
        jt.setBounds(30,40,200,300);
        JScrollPane sp=new JScrollPane(jt);
        f.add(sp);
        f.setSize(300,400);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new TableExample();
    } }
```

Output:



The image shows a Java Swing window with a title bar containing a small icon on the left and standard minimize, maximize, and close buttons on the right. The window contains a table with three columns: ID, NAME, and SALARY. The table has three data rows. Below the table is a large, empty rectangular area.

ID	NAME	SALARY
101	Amit	670000
102	Jai	780000
101	Sachin	700000