

Java Applet

Applet is a special type of program that is embedded in the webpage to generate the dynamic content.

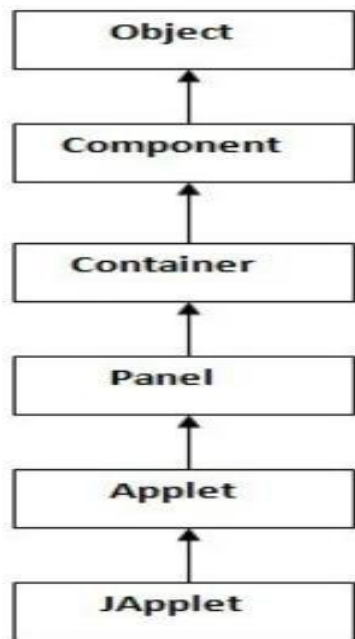
It runs inside the browser and works at client side.

Advantage of Applet

There are many advantages of applet. They are as follows:

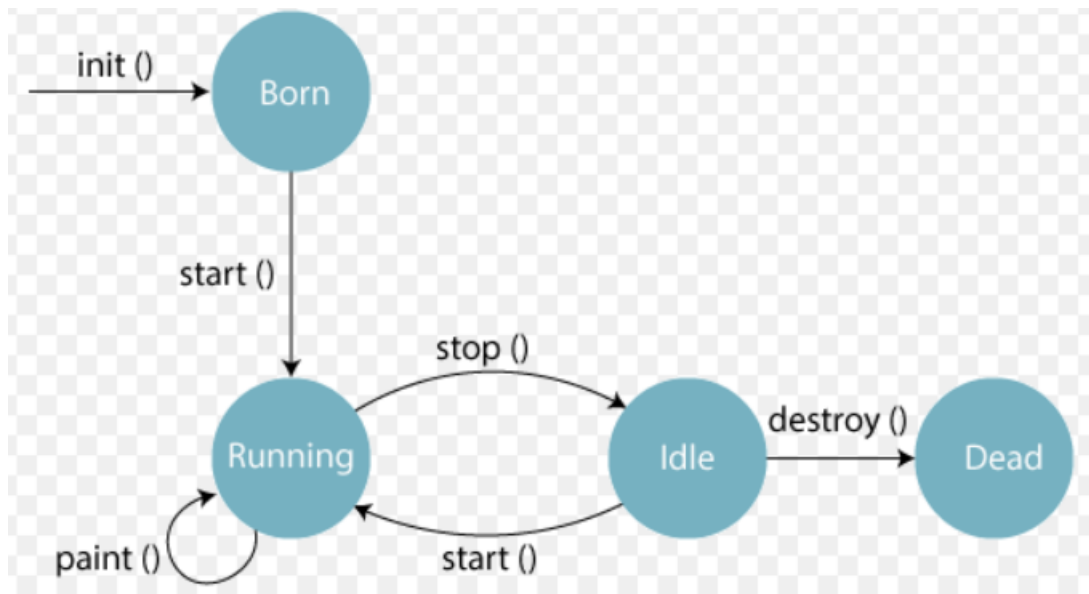
- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many platforms, including Linux, Windows, Mac Os etc.

Hierarchy of Applet



Lifecycle of Java Applet

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.



Methods of Applet Lifecycle

Applet Class provides 4 Life cycle methods and Component Class provides one method

- `public void init():` is used to initialize the Applet. It is invoked only once.
- `public void start():` is invoked after the `init()` method or browser is maximized. It is used to start the Applet.
- `public void stop():` is used to stop the Applet. It is invoked when Applet is stopped or browser is minimized.
- `public void destroy():` is used to destroy the Applet. It is invoked only once.
- `Public void paint(Graphics g):` is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

How to run an Applet?

There are two ways to run an applet

1. By html file.
2. By appletViewer tool (for testing purpose).

Simple example of Applet by html file:

To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

```
//AppletDemo.java
import java.applet.Applet;
import java.awt.Graphics;
public class AppletDemo extends Applet{

    public void paint(Graphics g){
        g.drawString("Hello",150,150);
    }

}
```

myapplet.html

```
<html>
<body>
<applet code="AppletDemo.class" width="300" height="300">
</applet>
</body>
</html>
```

Simple example of Applet by appletviewer tool:

To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: **appletviewer AppletDemo.java**. Now Html file is not required but it is for testing purpose only.

```
//AppletDemo.java
import java.applet.Applet;
import java.awt.Graphics;
public class AppletDemo extends Applet{

    public void paint(Graphics g){
        g.drawString("welcome to applet",150,150);
    }

}
/*
<applet code="AppletDemo.class" width="300" height="300">
</applet>
*/
```

Java Applets vs. Applications

Feature	Application	Applet
main() method	Present	Not present
Execution	Requires JRE	Requires a browser like Chrome, Firefox, IE, Safari, Opera, etc.
Nature	Called as stand-alone application as application can be executed from command prompt	Requires some third party tool help like a browser to execute
Restrictions	Can access any data or software available on the system	cannot access any thing on the system except browser's services
Security	Does not require any security	Requires highest security for the system as they are untrusted

Passing Parameters to applet

Parameters specify extra information that can be passed to an applet from the HTML page.

Parameters are specified using the HTML's param tag.

getParameter() :It is used to retrieve the parameters passed from the HTML page.

The syntax of getParameter() method is as follows:

String getParameter(String param-name)

```
import java.applet.Applet;
import java.awt.Graphics;
public class ParamDemo extends Applet
{
    public void paint(Graphics g)
    {
        String str=getParameter("msg");
        String str1=getParameter("msg1");
        g.drawString(str,50, 50);
        g.drawString(str1,500, 500);
    }
}
```

HTML

```
<html>
<body>
<applet code="ParamDemo.class" width="300" height="300">
<param name="msg" value="Welcome to applet">
<param name="msg1" value="Welcome to event handling">
</applet>
</body>
</html>
```

Creation of JApplet or Swing Applet

As we prefer Swing to AWT. Now we can use JApplet that can have all the controls of swing.

The JApplet class extends the Applet class.

Example of EventHandling in JApplet:

```
import java.applet.*;
import javax.swing.*;
import java.awt.event.*;
public class EventJApplet extends JApplet implements ActionListener{
    JButton b;
    JTextField tf;
    public void init(){

        tf=new JTextField();
        tf.setBounds(30,40,150,20);

        b=new JButton("Click");
        b.setBounds(80,150,70,40);

        add(b);add(tf);
        b.addActionListener(this);

        setLayout(null);
    }

    public void actionPerformed(ActionEvent e){
        tf.setText("Welcome");
    }
}
```

EventJApplet.html

```
<html>
<body>
<applet code="EventJApplet.class" width="300" height="300">
</applet>
</body>
</html>
```

Graphics class

There are numerous methods in the Graphics class to draw different kinds of shapes, such as a round rectangle, an arc, a polygon, etc.

The following table lists a few of those methods.

Return	Description
void	<i>drawLine</i> (int x1, int y1, int x2, int y2) Draws a straight line from point (x1, y1) to point (x2, y2).
void	<i>drawRect</i> (int x, int y, int width, int height) Draws a rectangle whose upper-left corner's coordinate is (x, y). The specified width and height are the width and height of the rectangle, respectively.
void	<i>fillRect</i> (int x, int y, int width, int height) It is the same as <i>drawRect</i> () method with two differences. It fills the area with the current color of the Graphics object. Its width and height are one pixel less than the specified width and height.
void	<i>drawOval</i> (int x, int y, int width, int height) Draws an oval that fits into a rectangle defined with point (x, y) as its upper-left corner and the specified width and height. If you specify the same width and height, it will draw a circle.
void	<i>fillOval</i> (int x, int y, int width, int height) It draws an oval and fills the area with the current color.
void	<i>drawstring</i> (String str, int x, int y) It draws the specified string str. The baseline of the leftmost character is at point (x, y).
void	<i>setColor</i> (Color c): is used to set the graphics current color to the specified color.
void	<i>setFont</i> (Font font): is used to set the graphics current font to the specified font.

```

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

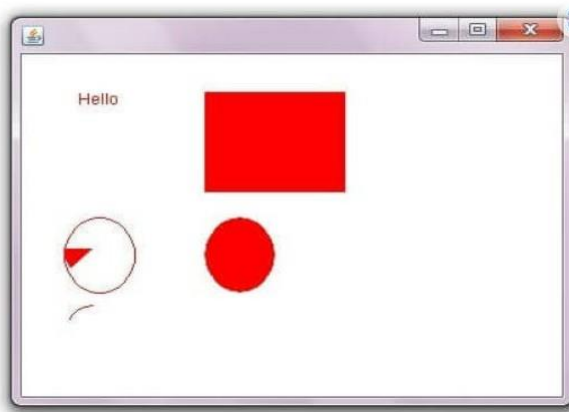
public class DisplayGraphics extends Canvas{

    public void paint(Graphics g) {
        g.drawString("Hello",40,40);
        setBackground(Color.WHITE);
        g.fillRect(130, 30,100, 80);
        g.drawOval(30,130,50, 60);
        setForeground(Color.RED);
        g.fillOval(130,130,50, 60);
        g.drawArc(30, 200, 40,50,90,60);
        g.fillArc(30, 130, 40,50,180,40);
    }

    public static void main(String[] args) {
        DisplayGraphics m=new DisplayGraphics();
        JFrame f=new JFrame();
        f.add(m);
        f.setSize(400,400);
        //f.setLayout(null);
        f.setVisible(true);
    }
}

```

Output:



Introduction to Java Swing

Swing is a Java Foundation Classes [JFC] library and an extension of the AbstractWindow Toolkit [AWT].

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

Features of Swings.

- Swing is an Extension library to the AWT (Abstract Window Toolkit)
- Swing is Entirely written in Java.
- Java Swing Components are Platform-independent
- The SwingComponents are lightweight.
- Swing Supports a Pluggable look and feels And Swing provides morepowerful components
- Swing Follows MVC.

Difference Between AWT and Swing

Java AWT	Java Swing
AWT components are platform-dependent .	Java swing components are platform-independent .
AWT components are heavyweight .	Swing components are lightweight .
AWT doesn't support pluggable look and feel .	Swing supports pluggable look and feel .
AWT provides less components than Swing.	Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
AWT doesn't follows MVC (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing follows MVC .

MVC Architecture

The MVC architecture includes the following 3 components:

- Controller
- Model
- View

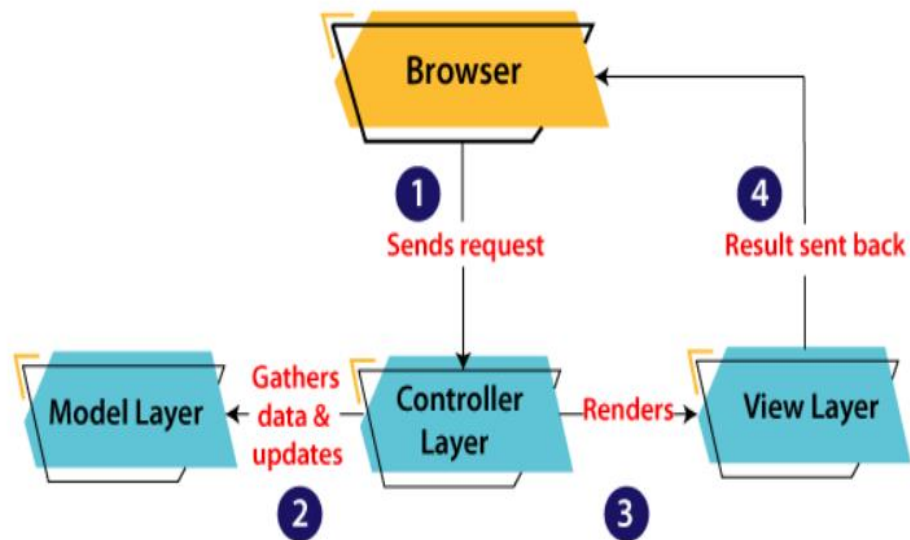


Fig: Architecture of MVC

Model: It represents the data and the business logic of the application. The model is responsible for maintaining the state and handling the core functionality. It does not depend on the view or the controller.

View: It represents the presentation layer, which is the user interface. The view is responsible for displaying the data to the user and sending user actions to the controller. It listens to changes in the model to update the UI accordingly.

Controller: It acts as an intermediary between the model and the view. It handles user input and updates the model accordingly. The controller also updates the view when the model changes.

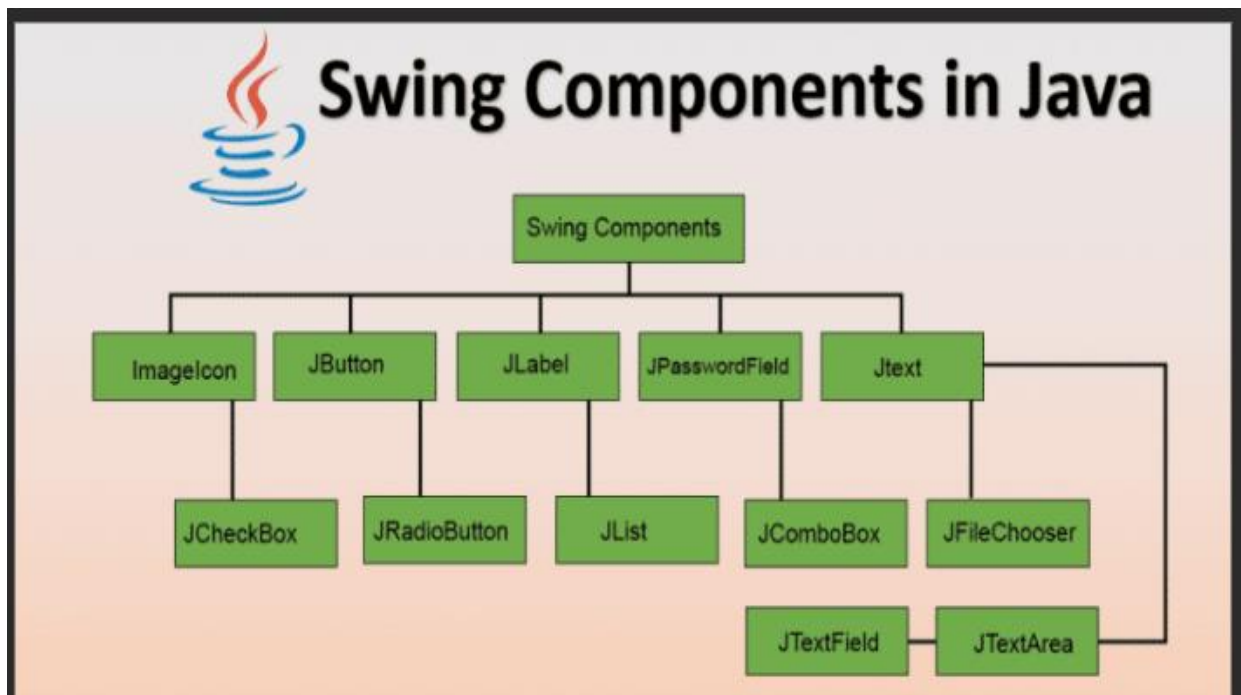
SWING Containers

Following is the list of commonly used containers while designed GUI using SWING.

Sr.No.	Container & Description
--------	-------------------------

- | | |
|---|---|
| | Panel |
| 1 | JPanel is the simplest container. It provides space in which any other component can be placed, including other panels. |
| | Frame |
| 2 | A JFrame is a top-level window with a title and a border. |
| | Window |
| 3 | A JWindow object is a top-level window with no borders |

The hierarchy of java swing Components is given below



JLabel

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.

Constructors

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.

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:



JTextField

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

Constructors

JTextField(): Creates a new TextField

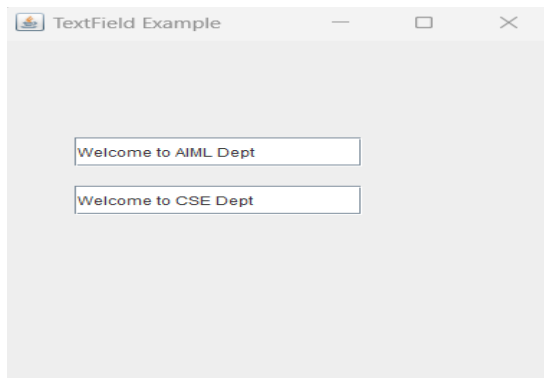
JTextField(String text): Creates a new TextField initialized with the specified text.

JTextField(String text, int columns): Creates a new TextField initialized with the specified text and columns.

Example:

```
import javax.swing.*;
class TextFieldExample
{
    public static void main(String args[])
    {
        JFrame f= new JFrame("TextField Example");
        JTextField t1,t2;
        t1=new JTextField("Welcome to AIML Dept");
        t1.setBounds(50,100, 200,30);
        t2=new JTextField("Welcome to CSE Dept");
        t2.setBounds(50,150, 200,30);
        f.add(t1); f.add(t2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

```
}
```



JTextArea

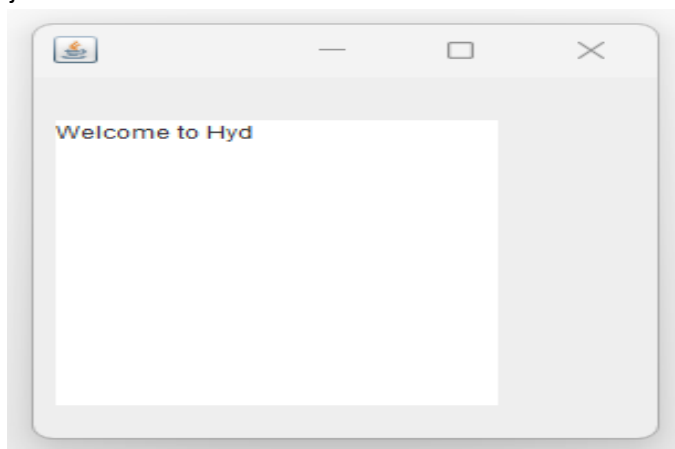
The object of a JTextArea class is a multi-line region that displays text.

It allows the editing of multiple line text.

It inherits JTextComponent class.

Example

```
import javax.swing.*.*;
public class TextAreaExample
{
    public static void main(String args[])
    {
        JFrame f= new JFrame();
        JTextArea area=new JTextArea("Welcome to Hyd");
        area.setBounds(10,30, 200,200);
        f.add(area);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```



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.

Constructors

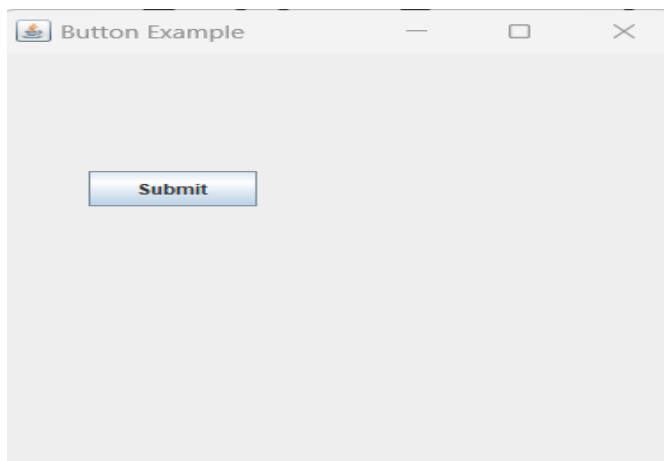
JButton() It creates a button with no text and icon.

JButton(String s) It creates a button with the specified text.

Example:

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

Output:



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.

Constructors

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.

Example

```
import javax.swing.*;

public class RadioButtonExample {
    RadioButtonExample() {
        JFrame f = new JFrame();
        JRadioButton r1 = new JRadioButton("Yes");
        JRadioButton r2 = new JRadioButton("No");
        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();
    }
}
```



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".

Constructors

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

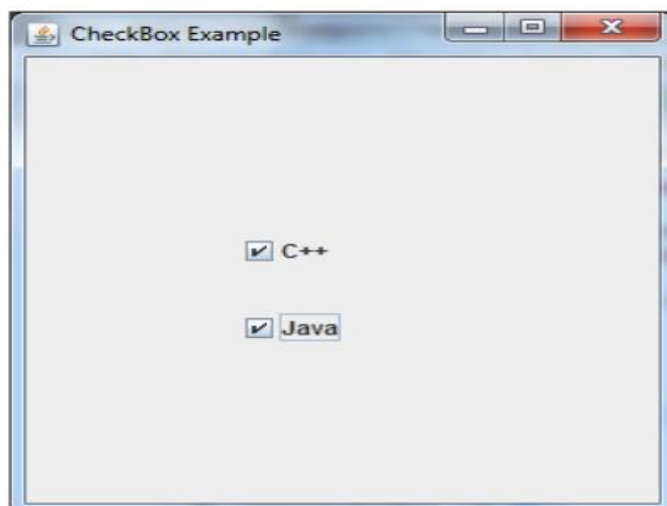
JCheckBox(String s): Creates an initially unselected check box with text.

JCheckBox(String text, boolean selected): Creates a check box with text and specifies Whether or not it is initially selected.

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:



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.

Constructors

JComboBox(): Creates a JComboBox with a default data model.

JComboBox(Object[] items) Creates a JComboBox that contains the elements in the specified array.

Example

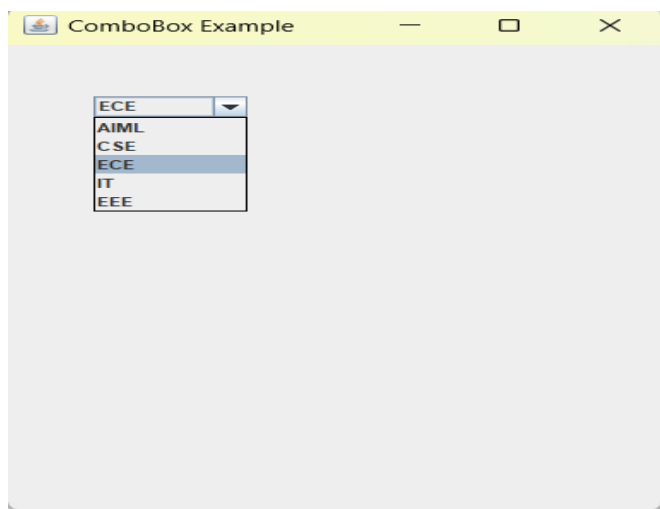
```
import javax.swing.*;

public class ComboBoxExample {
    JFrame f;

    ComboBoxExample(){
        f=new JFrame("ComboBox Example");
        String branch[]={"AIML","CSE","ECE","IT","EEE"};
        JComboBox cb=new JComboBox(branch);
        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();
    }
}
```

Output:



JList

The object of JList class represents a list of text items.

The list of text items can be set up so that the user can choose either one item or multiple items.

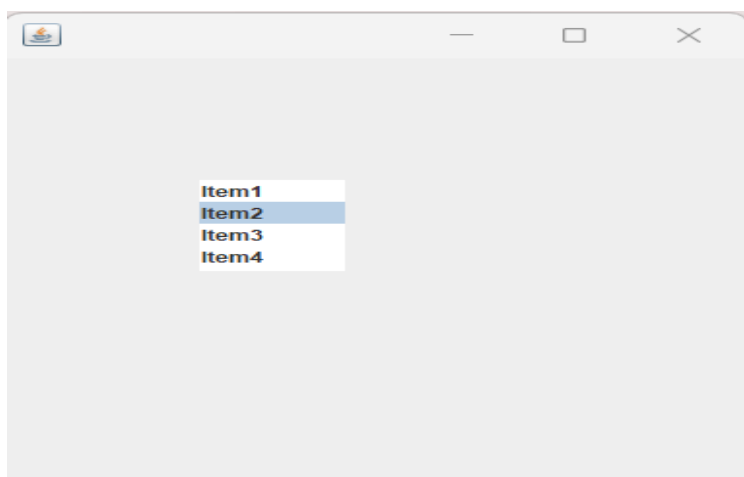
Constructors

JList(): Creates a JList with an empty, read-only, model.

JList(ary[] listData):Creates a JList that displays the elements in the specified array.

```
import javax.swing.*.*;
public class ListExample
{
    ListExample(){
        JFrame f= new JFrame();
        DefaultListModel<String> l1 = new DefaultListModel<>();
        l1.addElement("Item1");
        l1.addElement("Item2");
        l1.addElement("Item3");
        l1.addElement("Item4");
        JList<String> list = new JList<>(l1);
        list.setBounds(100,100, 75,75);
        f.add(list);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new ListExample();
    }
}
```

Output:



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

JScrollPane(): It creates a scroll pane.

JScrollPane(int, int): The two int parameters, when present, set the vertical and horizontal scroll bar policies (respectively).

Example

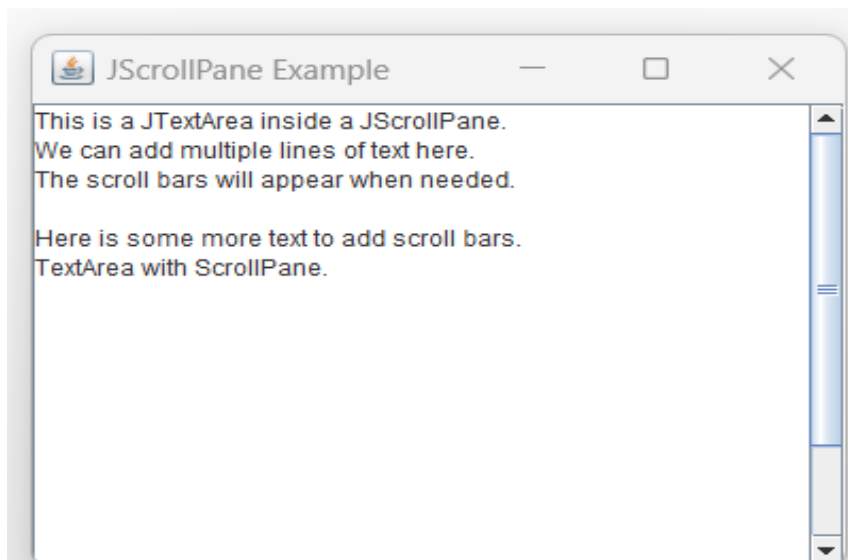
```
import javax.swing.*;
import java.awt.*;

public class JScrollPaneExample {

    public static void main(String[] args) {

        JFrame frame = new JFrame("JScrollPane Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 300);
        JTextArea textArea = new JTextArea(20, 30);
        textArea.setText("This is a JTextArea inside a JScrollPane.\n" +
            "We can add multiple lines of text here.\n" +
            "The scroll bars will appear when needed.\n\n" +
            "Here is some more text to add scroll bars.\n"+
            "TextArea with Scrollbar.");
        // Create a JScrollPane and add the JTextArea to it
        JScrollPane scrollPane = new JScrollPane(textArea);
        // Add the JScrollPane to the frame
        frame.add(scrollPane, BorderLayout.CENTER);
        // Set the frame visible
        frame.setVisible(true);
    }
}
```

Output:



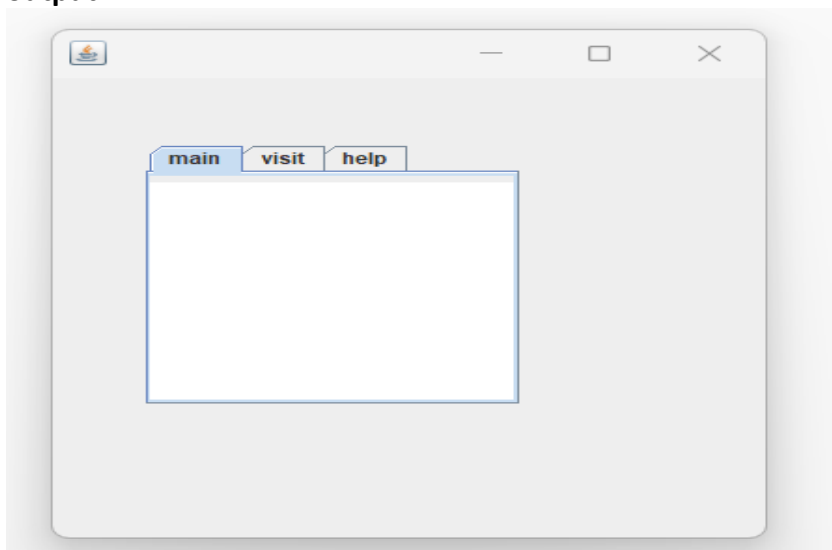
JTabbedPane

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.

```
import javax.swing.*.*;
public class TabbedPaneExample {
    TabbedPaneExample(){
        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);
    }
    public static void main(String[] args) {
        new TabbedPaneExample();
    }
}
```

output:



JTree

JTree class is used to display the tree structured data or hierarchical data.

JTree is a complex component.

It has a 'root node' at the top most which is a parent for all nodes in the tree.

It inherits JComponent class.

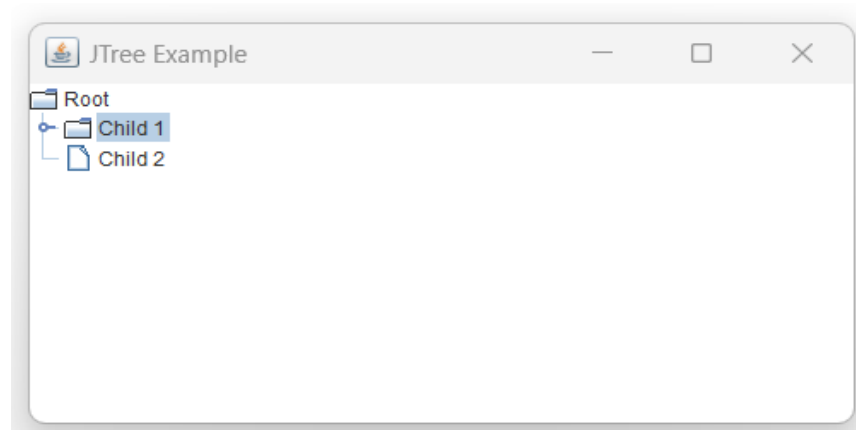
Constructors

JTree(): Creates a JTree with a sample model.

JTree(TreeNode root): Creates a JTree with the specified TreeNode as its root, which displays the root node.

Example:

```
import javax.swing.*.*;
import javax.swing.tree.DefaultMutableTreeNode;
public class JTreeExample {
    public static void main(String[] args) {
        // Create the frame
        JFrame frame = new JFrame("JTree Example");
        // Create the root node
        DefaultMutableTreeNode root = new DefaultMutableTreeNode("Root");
        // Create child nodes
        DefaultMutableTreeNode child1 = new DefaultMutableTreeNode("Child 1");
        DefaultMutableTreeNode child2 = new DefaultMutableTreeNode("Child 2");
        // Add child nodes to the root node
        root.add(child1);
        root.add(child2);
        // Create sub-child nodes
        DefaultMutableTreeNode subChild1 = new DefaultMutableTreeNode("Sub Child 1");
        DefaultMutableTreeNode subChild2 = new DefaultMutableTreeNode("Sub Child 2");
        // Add sub-child nodes to child1
        child1.add(subChild1);
        child1.add(subChild2);
        JTree jt=new JTree(root);
        frame.add(jt);
        frame.setSize(200,200);
        frame.setVisible(true);
    }
}
```



JTable

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

Constructors

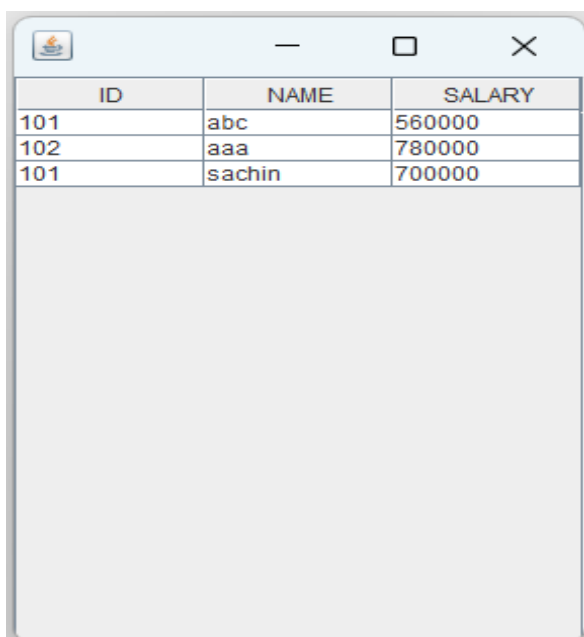
JTable():Creates a table with empty cells.

JTable(Object[][] rows, Object[] columns): Creates a table with the specified data.

Example

```
public class TableExample {  
    public static void main(String[] args) {  
        JFrame f = new JFrame();  
        String data[][] = { { "101", "abc", "560000" },  
                             { "102", "aaa", "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);  
    }  
}
```

output:



ID	NAME	SALARY
101	abc	560000
102	aaa	780000
101	sachin	700000