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

A Java application that is integrated into a webpage is called an applet. It functions as a front-end and is run within the web computer. It makes a page more interactive and dynamic by operating inside the web browser. Applets are hosted on web servers and inserted into HTML pages via the OBJECT or APPLET tags.

It can be compared to a tiny application that runs on the address bar. In addition to updating content in real-time and responding to human input, it may also play basic puzzles or graphics.

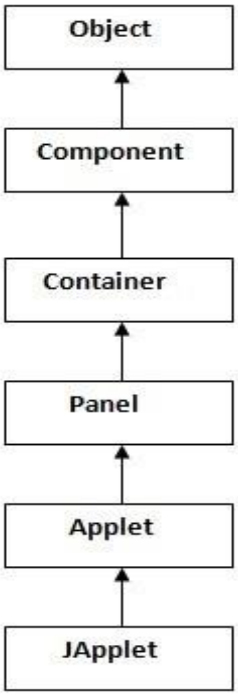
There are many advantages of applet. They are as follows:

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

Drawback of Applet

- Plugin is required at client browser to execute applet.

Hierarchy of Applet



As displayed in the above diagram, Applet class extends Panel. Panel class extends Cont which is the subclass of Component.

Lifecycle of Java Applet

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

Lifecycle methods for Applet:

The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.

java.applet.Applet class

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

1. **public void init():** is used to initialized the Applet. It is invoked only once.
2. **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. **public void destroy():** is used to destroy the Applet. It is invoked only once.

java.awt.Component class

The Component class provides 1 life cycle method of applet.

1. **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.

```
1. //First.java
2. import java.applet.Applet;
3. import java.awt.Graphics;
4. public class First extends Applet{
5.
6. public void paint(Graphics g){
7. g.drawString("welcome",150,150);
8. }
9.
10.}
```

Note: class must be public because its object is created by Java Plugin software that resides on the browser.

myapplet.html

```
<html>
<body>
<applet code="First.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 First.java. Now Html file is not required but it is for testing purpose only.

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

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

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

To execute the applet by appletviewer tool, write in command prompt:

```
c:\>javac First.java
c:\>appletviewer First.java
```

Displaying Graphics in Applet

java.awt.Graphics class provides many methods for graphics programming.

Commonly used methods of Graphics class:

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.

4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

Example of Graphics in applet:**import** java.applet.Applet;

```
1. import java.awt.*;
2.   public class GraphicsDemo extends Applet{
3.   public void paint(Graphics g){
4. g.setColor(Color.red);
5. g.drawString("Welcome",50, 50);
6. g.drawLine(20,30,20,300);
7. g.drawRect(70,100,30,30);
8. g.fillRect(170,100,30,30);
9. g.drawOval(70,200,30,30);
10.  g.setColor(Color.pink);
11.g.fillOval(170,200,30,30);
12.g.drawArc(90,150,30,30,30,270);
13.g.fillArc(270,150,30,30,0,180);
14.  }
15.}
```

myapplet.html

```
1. <html>
2. <body>
3. <applet code="GraphicsDemo.class" width="300" height="300">
4. </applet>
5. </body>
6. </html>
```

Delegation Event Model in Java

The Event model is based on two things, i.e., Event Source and Event Listeners.

- Event Source means any object which creates the message or event.
- Event Listeners are the object which receives the message or event.

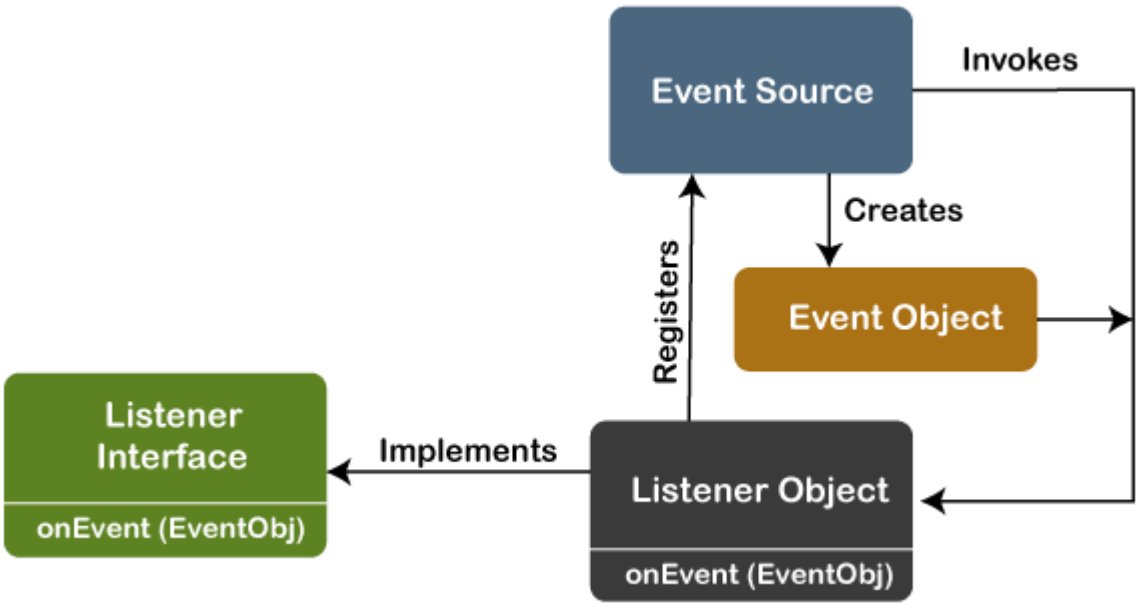
Now, coming to the Delegation Event Model in Java, the Delegation event model is based upon Event Source, Event Listeners, and Event Objects.

- Event Source is the class used to broadcast the events.
- Event Listeners are the classes that receive notifications of events.
- Event Object is the class object which describes the event.

Let's discuss how to implement the Delegation Event Model in Java.

The Delegation Event model is defined to handle events in GUI programming languages. The GUI stands for Graphical User Interface, where a user graphically/visually interacts with the system.

The GUI programming is inherently event-driven; whenever a user initiates an activity such as a mouse activity, clicks, scrolling, etc., each is known as an event that is mapped to a code to respond to functionality to the user. This is known as event handling.



In this model, a source generates an event and forwards it to one or more listeners. The listener waits until it receives an event. Once it receives the event, it is processed by the listener and returns it. The UI elements are able to delegate the processing of an event to a separate function.

The key advantage of the Delegation Event Model is that the application logic is completely separated from the interface logic.

Basically, an Event Model is based on the following three components:

- Events
- Events Sources
- Events Listeners

1. Events

An event refers to an action that occurs within a graphical user interface (GUI), such as a button click, a key press, or a mouse movement. When an event is triggered, it is typically handled by an event listener or handler, which is registered to the graphical component that generates the event.

2. Event Sources

An event source is an object that generates an event. It is typically a graphical user interface (GUI) component, such as a button, a text field, or a menu item, but it can also be other types of objects that generate events, such as timers or sockets.

When an event is triggered on an event source, the event is encapsulated in an event object and passed to all the registered event listeners or handlers. The event source is responsible for notifying the event listeners or handlers of the event and providing them with the information they need to handle the event.

3. Event Listeners

An event listener is an object that is registered to an event source and is responsible for handling events that are generated by that source. When an event occurs on the event source, the event listener is notified and performs the necessary actions to respond to the event.

An event listener in Java is typically implemented as a class that implements a specific event listener interface, such as ActionListener, MouseListener, or KeyListener. Each interface specifies a set of methods that the event listener must implement to handle the corresponding type of event. For example, the ActionListener interface requires the implementation of a single method called actionPerformed, which is called when an action event occurs on the event source.

Types of Events

The events are categories into the following two categories:

The Foreground Events:

The foreground events are those events that require direct interaction of the user. These types of events are generated as a result of user interaction with the GUI component. For example, clicking on a button, mouse movement, pressing a keyboard key, selecting an option from the list, etc.

The Background Events :

The Background events are those events that result from the interaction of the end-user. For example, an Operating system interrupts system failure (Hardware or Software).

To handle these events, we need an event handling mechanism that provides control over the events and responses.

Java Event classes

The following are some commonly used event classes in Java:-

Event Class	Listener Interface	Description
ActionEvent	ActionListener	Represents an action, such as a button click, triggered by a GUI component.
MouseEvent	MouseListener	Represents mouse events like clicks, enters, exits, and button presses on a GUI component.
KeyEvent	KeyListener	Represents keyboard events, such as key presses and releases, from a GUI component.
WindowEvent	WindowListener	Represents window-related events, like opening, closing, or resizing a GUI window.
FocusEvent	FocusListener	Represents focus-related events, including gaining and losing focus on a GUI component.

java MouseListener Interface

The Java MouseListener is notified whenever you change the state of mouse. It is notified against MouseEvent. The MouseListener interface is found in java.awt.event package. It has five methods.

Methods of MouseListener interface

The signature of 5 methods found in MouseListener interface are given below:

```
1. public abstract void mouseClicked(MouseEvent e);
2. public abstract void mouseEntered(MouseEvent e);
3. public abstract void mouseExited(MouseEvent e);
4. public abstract void mousePressed(MouseEvent e);
5. public abstract void mouseReleased(MouseEvent e);
```

Java MouseListener Example

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

```
3. public class MouseListenerExample extends Frame implements MouseListener
   {
4.     Label l;
5.     MouseListenerExample(){
6.         addMouseListener(this);
7.
8.         l=new Label();
9.         l.setBounds(20,50,100,20);
10.        add(l);
11.        setSize(300,300);
12.        setLayout(null);
13.        setVisible(true);
14.    }
15.    public void mouseClicked(MouseEvent e) {
16.        l.setText("Mouse Clicked");
17.    }
18.    public void mouseEntered(MouseEvent e) {
19.        l.setText("Mouse Entered");
20.    }
21.    public void mouseExited(MouseEvent e) {
22.        l.setText("Mouse Exited");
23.    }
24.    public void mousePressed(MouseEvent e) {
25.        l.setText("Mouse Pressed");
26.    }
27.     public void mouseReleased(MouseEvent e) {
28.        l.setText("Mouse Released");
29.    }
30.public static void main(String[] args) {
31.    new MouseListenerExample();
32.}
33.}
```

Output:

Java KeyListener Interface

The **Java KeyListener is notified whenever you change the state of key.** It is notified against KeyEvent. The KeyListener interface is found in java.awt.event package, and it has three methods.

Interface declaration

Following is the declaration for **java.awt.event.KeyListener** interface:

```
1. public interface KeyListener extends EventListener
```

Methods of KeyListener interface

The signature of 3 methods found in KeyListener interface are given below:

Sr. no.	Method name	Description
1.	public abstract void keyPressed (KeyEvent e);	It is invoked when a key has been pressed.
2.	public abstract void keyReleased (KeyEvent e);	It is invoked when a key has been released.
3.	public abstract void keyTyped (KeyEvent e);	It is invoked when a key has been typed.

Methods inherited

This interface inherits methods from the following interface:

- java.awt.EventListener

Java KeyListener Example

In the following example, we are implementing the methods of the KeyListener interface.

KeyListenerExample.java

```
1. // importing awt libraries
2. import java.awt.*;
3. import java.awt.event.*;
4. // class which inherits Frame class and implements KeyListener interface
5. public class KeyListenerExample extends Frame implements KeyListener {
6. // creating object of Label class   and TextArea class
7.  Label l;
8.     TextArea area;
9. // class constructor
10.    KeyListenerExample() {
```

```
11.         // creating the label
12.       l = new Label();
13.// setting the location of the label in frame
14.       l.setBounds (20, 50, 100, 20);
15.// creating the text area
16.       area = new TextArea();
17.// setting the location of text area
18.       area.setBounds (20, 80, 300, 300);
19.// adding the KeyListener to the text area
20.       area.addKeyListener(this);
21.// adding the label and text area to the frame
22.       add(l);
23.add(area);
24.// setting the size, layout and visibility of frame
25.       setSize (400, 400);
26.       setLayout (null);
27.       setVisible (true);
28.   }
29.// overriding the keyPressed() method of KeyListener interface where we set th
    e text of the label when key is pressed
30.   public void keyPressed (KeyEvent e) {
31.       l.setText ("Key Pressed");
32.   }
33.// overriding the keyReleased() method of KeyListener interface where we set t
    he text of the label when key is released
34.   public void keyReleased (KeyEvent e) {
35.       l.setText ("Key Released");
36.   }
37.// overriding the keyTyped() method of KeyListener interface where we set the
     text of the label when a key is typed
38.   public void keyTyped (KeyEvent e) {
39.       l.setText ("Key Typed");
40.   }
41.  // main method
42.   public static void main(String[] args) {
43.       new KeyListenerExample();
44.   }
45.}
```

Output:

Java Swing

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.

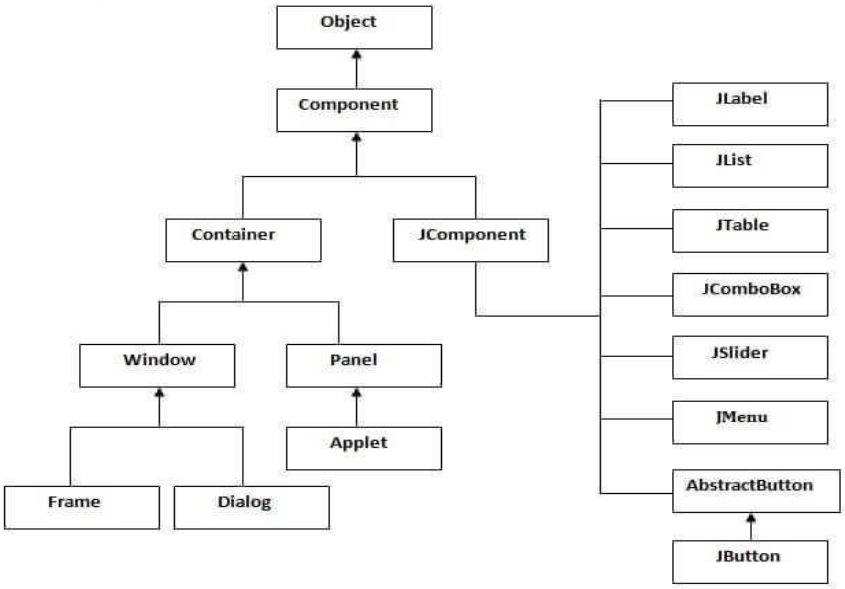
The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

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 **platform-dependent**.	Java swing components are **platform-independent**.
2)	AWT components are **heavyweight**.	Swing components are **lightweight**.
3)	AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	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**.

The hierarchy of java swing API is given below.



Components of Java Swing

Java Swing components

JLabel: JLabel is a component that displays text or an image. It is commonly used to provide information or to label other components.

Java JLabel Example

```
1. import javax.swing.*;
2. class LabelExample
3. {
4. public static void main(String args[])
5.    {
6.    JFrame f= new JFrame("Label Example");
7.    JLabel l1,l2;
8.    l1=new JLabel("First Label.");
9.    l1.setBounds(50,50, 100,30);
10.   l2=new JLabel("Second Label.");
11.   l2.setBounds(50,100, 100,30);
12.   f.add(l1); f.add(l2);
13.   f.setSize(300,300);
14.
15.  f.setLayout(null);
16.   f.setVisible(true);
17.   }
18.   }
```

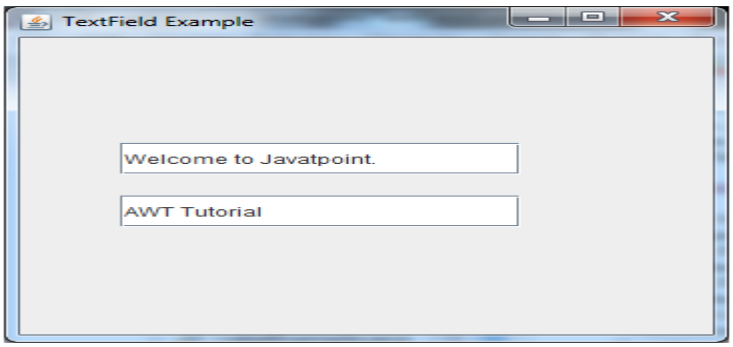
Output:

JTextField: JTextField is a component that allows the user to input text. It is commonly used to get input from the user, such as a name or an address.

Java JTextField Example

```
1. import javax.swing.*;
2. class TextFieldExample
3. {
4. public static void main(String args[])
5.    {
6.    JFrame f= new JFrame("TextField Example");
7.    JTextField t1,t2;
8.    t1=new JTextField("Welcome to Javatpoint.");
9.    t1.setBounds(50,100, 200,30);
10.   t2=new JTextField("AWT Tutorial");
11.   t2.setBounds(50,150, 200,30);
12.   f.add(t1); f.add(t2);
13.   f.setSize(400,400);
14.   f.setLayout(null);
15.   f.setVisible(true);
16.   }
17.   }
```

Output:

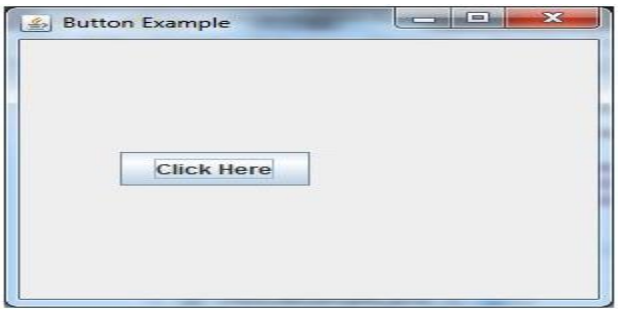


JButton: JButton is a component that represents a clickable button. It is commonly used to trigger actions in a GUI application.

Java JButton Example

```
1. import javax.swing.*;
2. public class ButtonExample {
3. public static void main(String[] args) {
4.     JFrame f=new JFrame("Button Example");
5.     JButton b=new JButton("Click Here");
6.     b.setBounds(50,100,95,30);
7.     f.add(b);
8.     f.setSize(400,400);
9.     f.setLayout(null);
10.    f.setVisible(true);
11.}
12.}
```

Output:



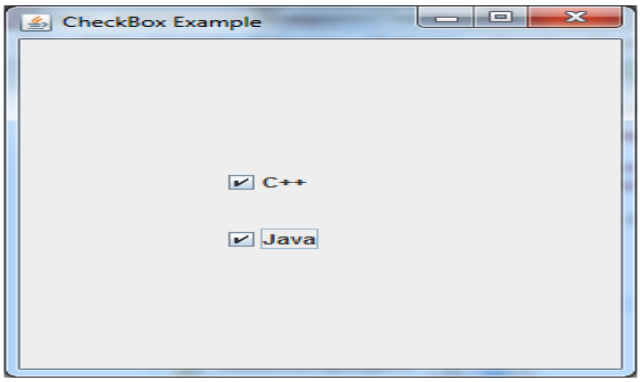
JCheckBox: JCheckBox is a component that represents a checkbox. It is commonly used to get a binary input from the user, such as whether or not to enable a feature

java JCheckBox Example

```
1. import javax.swing.*;
2. public class CheckBoxExample
3. {
4.     CheckBoxExample(){
5.       JFrame f= new JFrame("CheckBox Example");
6.       JCheckBox checkBox1 = new JCheckBox("C++");
7.       checkBox1.setBounds(100,100, 50,50);
8.       JCheckBox checkBox2 = new JCheckBox("Java", true);
9.       checkBox2.setBounds(100,150, 50,50);
10.      f.add(checkBox1);
11.      f.add(checkBox2);
12.      f.setSize(400,400);
13.      f.setLayout(null);
14.      f.setVisible(true);
15.    }
16.public static void main(String args[])
```

```
17.  {
18.  new CheckBoxExample();
19.  }}
```

Output:

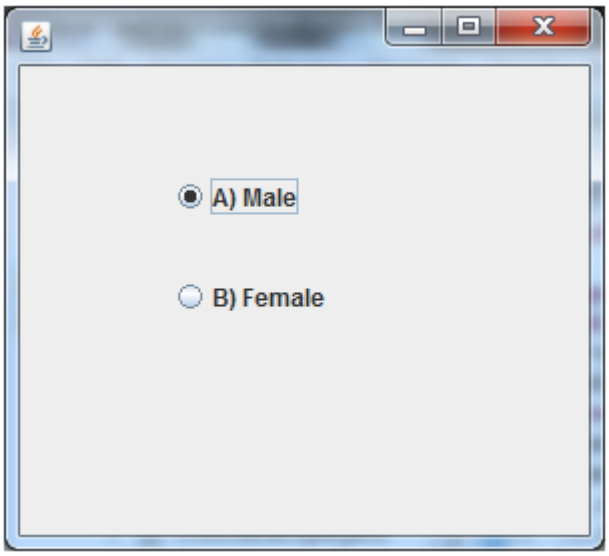


java JRadioButton

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.

```
1.  import javax.swing.*;
2.  public class RadioButtonExample {
3.  JFrame f;
4.  RadioButtonExample(){
5.  f=new JFrame();
6.  JRadioButton r1=new JRadioButton("A) Male");
7.  JRadioButton r2=new JRadioButton("B) Female");
8.  r1.setBounds(75,50,100,30);
9.  r2.setBounds(75,100,100,30);
10.ButtonGroup bg=new ButtonGroup();
11.bg.add(r1);bg.add(r2);
12.f.add(r1);f.add(r2);
13.f.setSize(300,300);
14.f.setLayout(null);
15.f.setVisible(true);
16.}
17.public static void main(String[] args) {
18.  new RadioButtonExample();
19.}
20.}
```

Output:

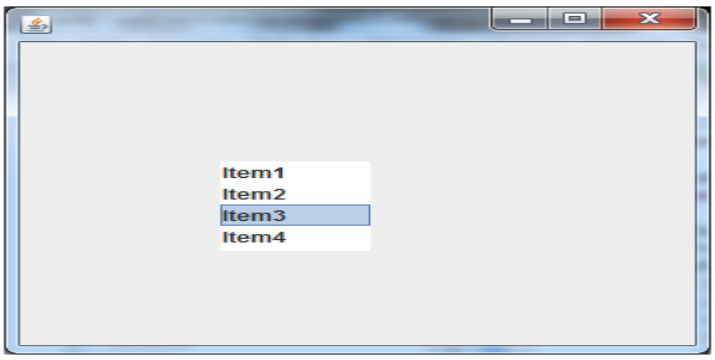


Java JList

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. It inherits JComponent class.

```
1. import javax.swing.*;
2. public class ListExample
3. {
4.     ListExample(){
5.       JFrame f= new JFrame();
6.       DefaultListModel<String> l1 = new DefaultListModel<>();
7.        l1.addElement("Item1");
8.        l1.addElement("Item2");
9.        l1.addElement("Item3");
10.       l1.addElement("Item4");
11.       JList<String> list = new JList<>(l1);
12.       list.setBounds(100,100, 75,75);
13.       f.add(list);
14.       f.setSize(400,400);
15.       f.setLayout(null);
16.       f.setVisible(true);
17.   }
18.public static void main(String args[])
19.  {
20.  new ListExample();
21.  }}
```

Output:

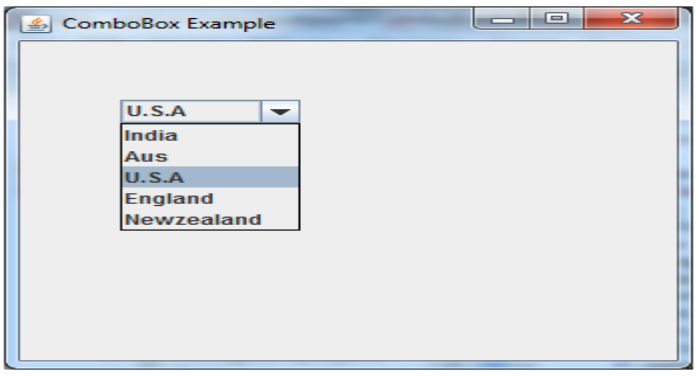


Java JComboBox

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.

```
1. import javax.swing.*;
2. public class ComboBoxExample {
3. JFrame f;
4. ComboBoxExample(){
5.     f=new JFrame("ComboBox Example");
6.     String country[]={"India","Aus","U.S.A","England","Newzealand"};
7.     JComboBox cb=new JComboBox(country);
8.     cb.setBounds(50, 50,90,20);
9.     f.add(cb);
10.    f.setLayout(null);
11.    f.setSize(400,500);
12.    f.setVisible(true);
13.}
14.public static void main(String[] args) {
15.    new ComboBoxExample();
16.}
17.}
```

Output:

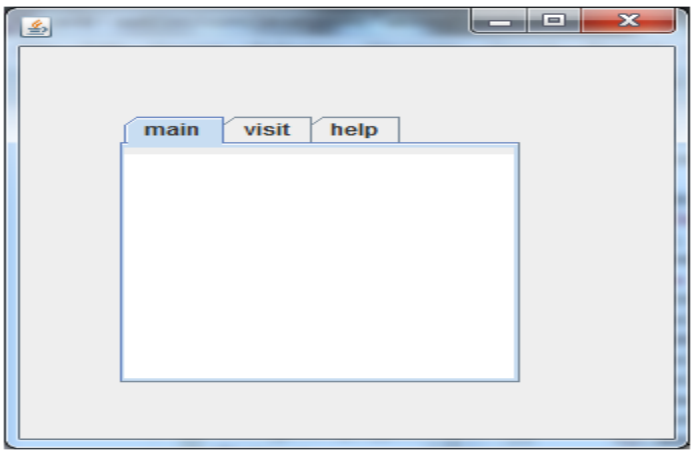


Java JTabbedPane

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.

```
1. import javax.swing.*;
2. public class TabbedPaneExample {
3. JFrame f;
4. TabbedPaneExample(){
5.     f=new JFrame();
6.     JTextArea ta=new JTextArea(200,200);
7.     JPanel p1=new JPanel();
8.     p1.add(ta);
9.     JPanel p2=new JPanel();
10.    JPanel p3=new JPanel();
11.    JTabbedPane tp=new JTabbedPane();
12.    tp.setBounds(50,50,200,200);
13.    tp.add("main",p1);
14.    tp.add("visit",p2);
15.    tp.add("help",p3);
16.    f.add(tp);
17.    f.setSize(400,400);
18.    f.setLayout(null);
19.    f.setVisible(true);
20.}
21.public static void main(String[] args) {
22.    new TabbedPaneExample();
23.}}
```

Output:



jImageIcon

The class **ImageIcon** is an implementation of the Icon interface that paints Icons from Images.

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

public class ImageIconExample {
```

```
    public static void main(String[] args) {
        // Create a JFrame
        JFrame frame = new JFrame("ImageIcon Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Load the image
        ImageIcon icon = new ImageIcon("example_image.png");


        // Create a JLabel with the loaded ImageIcon
        JLabel label = new JLabel(icon);

        // Add the label to the frame
        frame.getContentPane().add(label, BorderLayout.CENTER);
              // Set frame size and make it visible
        frame.setSize(300, 300);
        frame.setVisible(true);
    }
}
```