Introduction to Java Swing

Swing is a Java Foundation Classes [JFC] library and an extension of the Abstract Window Toolkit [AWT]. Swing offers much-improved functionality over AWT, new components, expanded components features, excellent event handling with drag and drop support.

Swing API is a set of extensible GUI Components to ease the developer's life to create JAVA based Front End/GUI Applications. It is build on top of AWT API and acts as a replacement of AWT API, since it has almost every control corresponding to AWT controls. Swing component follows a Model-View-Controller architecture to fulfill the following criterias.

- A single API is to be sufficient to support multiple look and feel.
- API is to be model driven so that the highest level API is not required to have data.
- API is to use the Java Bean model so that Builder Tools and IDE can provide better services to the developers for use.

Introduction of Java Swing

Swing has about four times the number of User Interface [UI] components as AWT and is part of the standard Java distribution. By today's application GUI requirements, AWT is a limited implementation, not quite capable of providing the components required for developing complex GUI's required in modern commercial applications. The AWT component set has quite a few bugs and really does take up a lot of system resources when compared to equivalent Swing resources. Netscape introduced its Internet Foundation Classes [IFC] library for use with Java. Its Classes became very popular with programmers creating GUI's for commercial applications.

- Swing is a Set Of API (API- Set Of Classes and Interfaces)
- Swing is Provided to Design a Graphical User Interfaces
- Swing is an Extension library to the AWT (Abstract Window Toolkit)
- Includes New and improved Components that have been enhancing the looks and Functionality of GUI's

- Swing can be used to build(Develop) The Standalone swing GUI Apps Also as Servlets And Applets
- It Employs model/view design architecture
- Swing is more portable and more flexible than AWT, The Swing is built on top of the AWT
- Swing is Entirely written in Java
- Java Swing Components are Platform-independent And The Swing Components are lightweight
- Swing Supports Pluggable look and feels And Swing provides more powerful components
- such as tables, lists, Scrollpanes, Colourchooser, tabbedpane, etc
- Further Swing Follows MVC

Many programmers think that JFC and Swing is one and the same thing, but that is not so.

JFC contains Swing [A UI component package] and quite a number of other items:

- Cut and paste: Clipboard support
- Accessibility features: Aimed at developing GUI's for users with disabilities
- The Desktop Colors Features Has been Firstly introduced in Java 1.1
- The Java 2D: it has Improved colors, images, and also texts support
-

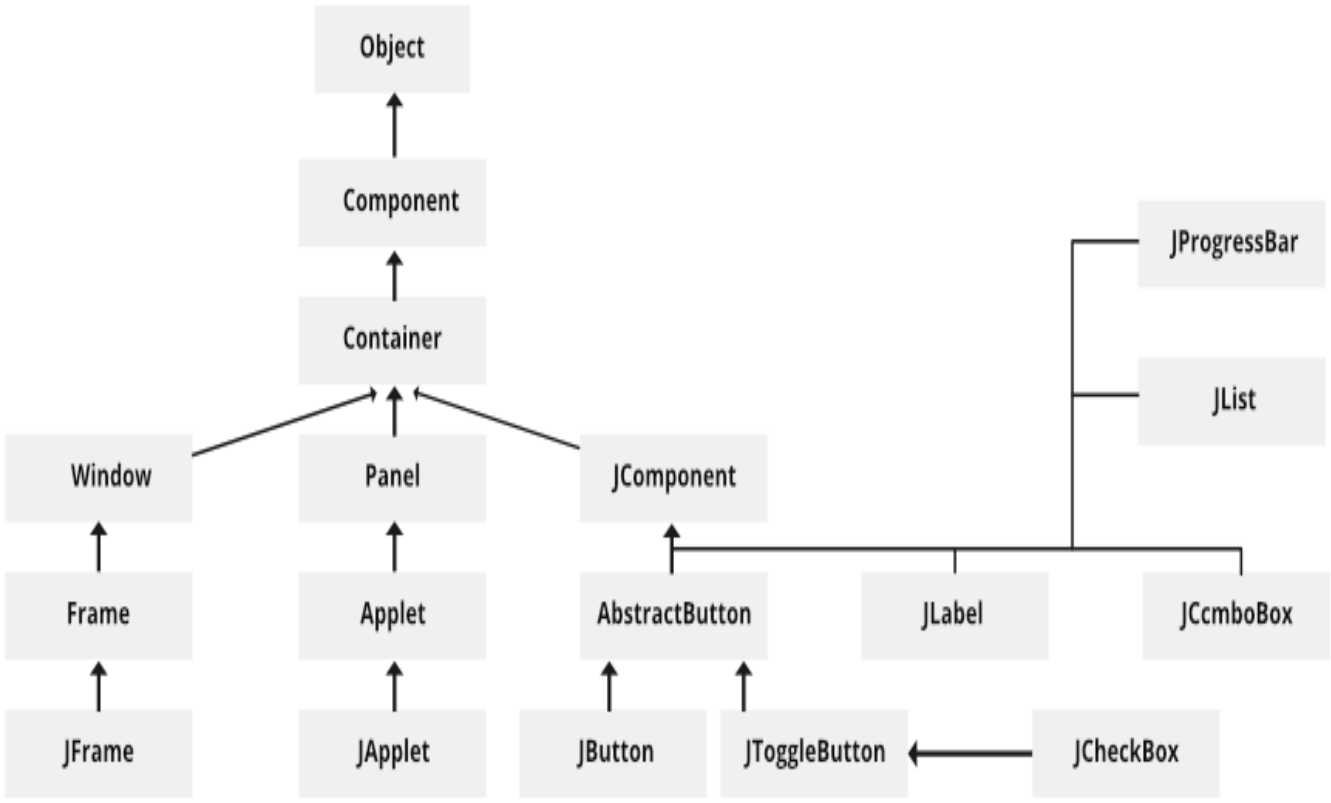
Features Of Swing Class

- Pluggable look and feel
- Uses MVC architecture
- Lightweight Components
- Platform Independent
- Advance features such as JTable, JTabbedPane, JScollPane etc
- Java is a platform-independent language and runs on any client machine, the GUI look and feel, owned and delivered by a platform specific O/S, simply does not affect an application's GUI constructed using Swing components
- **Lightweight Components:** Starting with the JDK 1.1, its AWT supported lightweight component development. For a component to qualify as lightweight, it must not depend on any non-Java [O/s

based) system classes. Swing components have their own view supported by Java's look and feel classes

- **Pluggable Look and Feel:** This feature enables the user to switch the look and feel of Swing components without restarting an application. The Swing library supports components look and feel that remains the same across all platforms wherever the program runs. The Swing library provides an API that gives real flexibility in determining the look and feel of the GUI of an application
- **Rich Controls** − Swing provides a rich set of advanced controls like Tree, TabbedPane, slider, colorpicker, and table controls.
- **Highly Customizable** − Swing controls can be customized in a very easy way as visual apperance is independent of internal representation.

Swing Classes Hierarchy



The MVC Connection

- In general, a visual component is a composite of **three distinct aspects:**
  1. The way that the component looks when rendered on the screen
  2. The way such that the component reacts to the user
  3. The state information associated With the component
- Over the years, one component architecture has proven itself to be exceptionally effective:- **Model-View-Controller** or **MVC** for short.
- In MVC terminology, the **model** corresponds to the state information associated with the Component
- The **view** determines how the component is displayed on the screen, including any aspects of the view that are affected by the current state of the model.
- The **controller** determines how the component reacts to the user

The simplest Swing components have capabilities far beyond AWT components as follows:

- Swing buttons and the labels can be displaying images instead of or in addition to text
- The borders around most Swing components can be changed easily. For example: it is easy to put a 1 pixel border around the outside of a Swing label
- Swing components do not have to be rectangular. Buttons, for example, can be round
- Now The Latest Assertive technologies such as screen readers can easily getting the information from Swing components. For example: A screen reader tool can easily capture the text that is displayed on a Swing button or label

Components of Swing Class the task's percentage

Class	Description
Component	A Component is the Abstract base class for about the non menu user-interface controls of SWING. Components are represents an object

Class	Description
	with graphical representation
Container	A Container is a component that can container SWING Components
JComponent	A JComponent is a base class for all swing UI Components In order to use a swing component that inherits from JComponent, component must be in a containment hierarchy whose root is a top-level Swing container
JLabel	A JLabel is an object component for placing text in a container
JButton	This class creates a labeled button
JColorChooser	A JColorChooser provides a pane of controls designed to allow the user to manipulate and select a color
JCheckBox	A JCheckBox is a graphical(GUI) component that can be in either an on-(true) or off-(false) state
JRadioButton	The JRadioButton class is a graphical(GUI) component that can be in either an on-(true) or off-(false) state. in the group
JList	A JList component represents the user with the scrolling list of text items
JComboBox	A JComboBox component is Presents the User with a show up Menu of

Class	Description
	choices
JTextField	A JTextField object is a text component that will allow for the editing of a single line of text
JPasswordField	A JPasswordField object it is a text component specialized for password entry
JTextArea	A JTextArea object s a text component that allows for the editing of multiple lines of text
ImageIcon	A ImageIcon control is an implementation of the Icon interface that paints Icons from Images
JScrollbar	A JScrollbar control represents a scroll bar component in order to enable users to Select from range values
JOptionPane	JOptionPane provides set of standard dialog boxes that prompt users for a value or Something
JFileChooser	A JFileChooser it Controls represents a dialog window from which the user can select a file.
JProgressBar	As the task progresses towards completion, the progress bar displays the tasks percentage on its completion

Class	Description
JSlider	A JSlider this class is lets the user graphically(GUI) select by using a value by sliding a knob within a bounded interval.
JSpinner	A JSpinner this class is a single line input where the field that lets the user select by using a number or an object value from an ordered sequence

AWT V/S SWING

S.NO	AWT	Swing
1.	Java AWT is an API to develop GUI applications in Java	Swing is a part of Java Foundation Classes and is used to create various applications.
2.	The components of Java AWT are heavy weighted.	The components of Java Swing are light weighted.
3.	Java AWT has comparatively less functionality as compared to Swing.	Java Swing has more functionality as compared to AWT.
4.	The execution time of AWT is more than Swing.	The execution time of Swing is less than AWT.
5.	The components of Java AWT are platform dependent.	The components of Java Swing are platform independent.

S.NO	AWT	Swing
6.	MVC pattern is not supported by AWT.	MVC pattern is supported by Swing.
7.	AWT provides comparatively less powerful components.	Swing provides more powerful components.

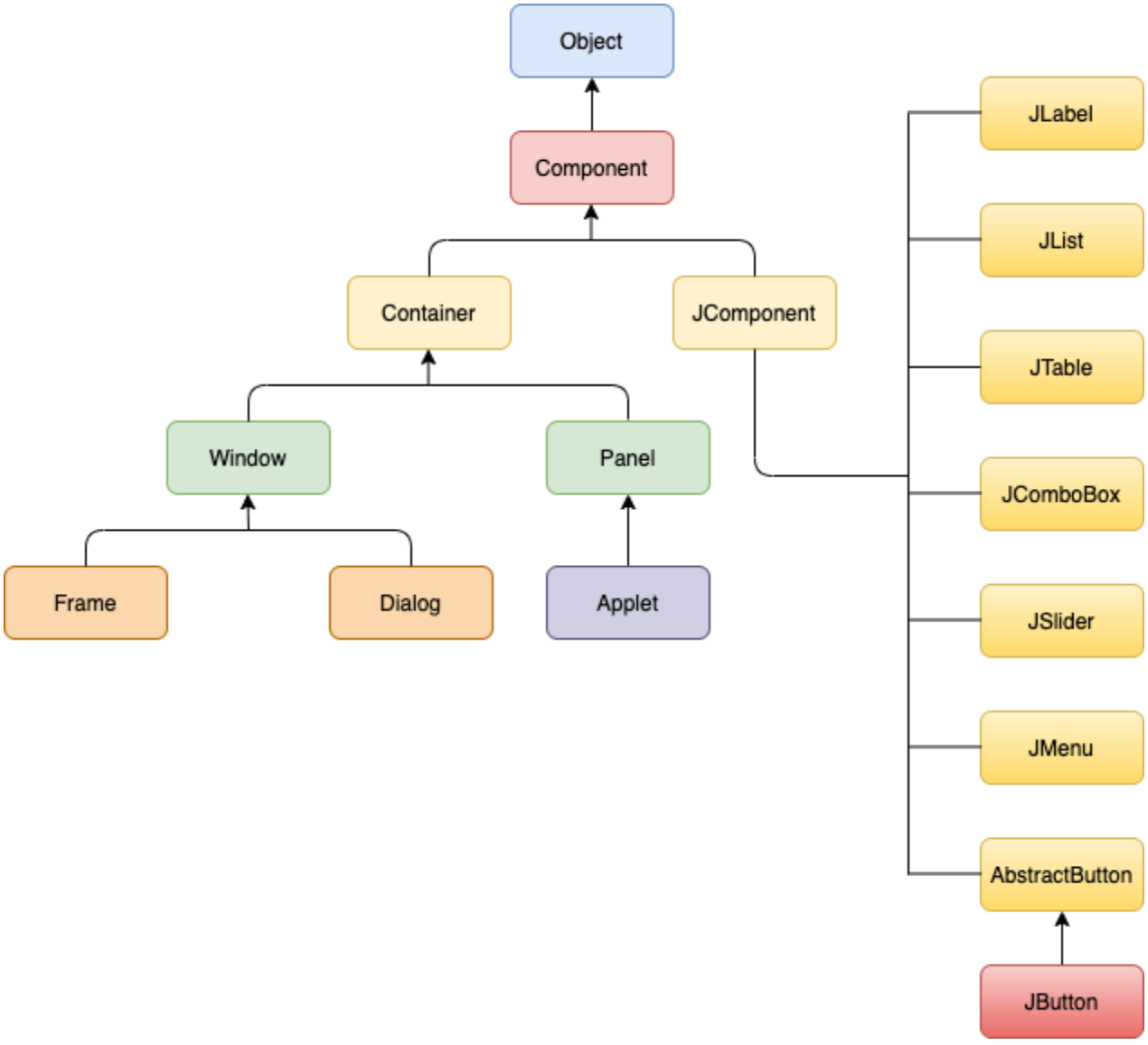
MORE DIFFERENT

Context	AWT	Swing
API Package	The AWT Component classes are provided by the java.awt package.	The Swing component classes are provided by the javax.swing package.
Operating System	The Components used in AWT are mainly dependent on the operating system.	The Components used in Swing are not dependent on the operating system. It is completely scripted in Java.
Weightiness	The AWT is heavyweight since it uses the resources of the operating system.	The Swing is mostly lightweight since it doesn't need any Operating system object for processing. The Swing

		Components are built on the top of AWT.
Appearance	The Appearance of AWT Components is mainly not configurable. It generally depends on the operating system's look and feels.	The Swing Components are configurable and mainly support pluggable look and feel.
Number of Components	The Java AWT provides a smaller number of components in comparison to Swing.	Java Swing provides a greater number of components than AWT, such as list, scroll panes, tables, color choosers, etc.
Full-Form	Java AWT stands for Abstract Window Toolkit.	Java Swing is mainly referred to as Java Foundation Classes (JFC).
Peers	Java AWT has 21 peers. There is one peer for each control and one peer for the dialogue. Peers are provided by the operating system in the form of widgets themselves.	Java Swing has only one peer in the form of OS's window object, which provides the drawing surface used to draw the Swing's widgets (label, button, entry fields, etc.) developed directly by Java Swing Package.
Functionality and Implementation	Java AWT many features that are completely developed by the developer. It serves as a thin layer of development on the top of the OS.	Swing components provide the higher-level inbuilt functions for the developer that facilitates the coder to write less code.
Memory	Java AWT needs a higher amount of memory for the execution.	Java Swing needs less memory space as compared to Java AWT.
Speed	Java AWT is slower than swing in terms of performance.	Java Swing is faster than the AWT.

Java Swing Hierarchy

Java defines the class hierarchy for all the Swing Components, which is shown in the following image.



JApplet class in Applet

As we prefer Swing to AWT. Now we can use JApplet that can have all the controls o
JApplet class extends the Applet class.

Example of EventHandling in JApplet:

```
1. import java.applet.*;
2. import javax.swing.*;
3. import java.awt.event.*;
4. public class EventJApplet extends JApplet implements ActionListener{
5. JButton b;
6. JTextField tf;
7. public void init(){
8.
9. tf=new JTextField();
10.tf.setBounds(30,40,150,20);
11.
12.b=new JButton("Click");
13.b.setBounds(80,150,70,40);
14.
15.add(b);add(tf);
16.b.addActionListener(this);
17.
18.setLayout(null);
19.}
20.
21.public void actionPerformed(ActionEvent e){
22.tf.setText("Welcome");
23.}
24.}
```

In the above example, we have created all the controls in init() method because it is invoke

myapplet.html

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

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.

Nested Class

Modifier and Type	Class	Description
protected class	JFrame.AccessibleJFrame	This class implements accessibility support for the JFrame class.

Fields

Modifier and Type	Field	Description
protected AccessibleContext	accessibleContext	The accessible context property.
static int	EXIT_ON_CLOSE	The exit application default window close operation.
protected JRootPane	rootPane	The JRootPane instance that manages the contentPane and optional menuBar for this frame, as well as the glassPane.
protected boolean	rootPaneCheckingEnabled	If true then calls to add and setLayout will be forwarded to the contentPane.

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

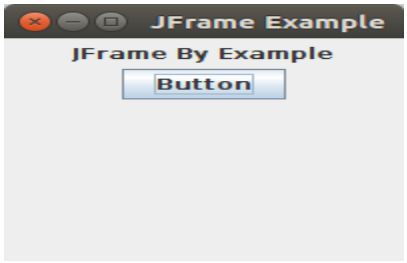
Modifier and Type	Method	Description
protected void	addImpl(Component comp, Object constraints, int index)	Adds the specified child Component.
protected JRootPane	createRootPane()	Called by the constructor methods to create the default rootPane.
protected void	frameInit()	Called by the constructors to init the JFrame properly.
void	setContentPane(Containe 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

```
1. import java.awt.FlowLayout;
2. import javax.swing.JButton;
3. import javax.swing.JFrame;
4. import javax.swing.JLabel;
5. import javax.swing.JPanel;
6. public class JFrameExample {
7.     public static void main(String s[]) {
8.         JFrame frame = new JFrame("JFrame Example");
9.         JPanel panel = new JPanel();
10.        panel.setLayout(new FlowLayout());
11.        JLabel label = new JLabel("JFrame By Example");
12.        JButton button = new JButton();
13.        button.setText("Button");
14.        panel.add(label);
15.        panel.add(button);
16.        frame.add(panel);
17.        frame.setSize(200, 300);
18.        frame.setLocationRelativeTo(null);
19.        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20.        frame.setVisible(true);
21.    }
22. }
```

Output



Java JPanel

The JPanel is a simplest container class. It provides space in which an application can attach any other component. It inherits the JComponents class.

It doesn't have title bar.

JPanel class declaration

```
1. public class JPanel extends JComponent implements Accessible
```

Commonly used Constructors:

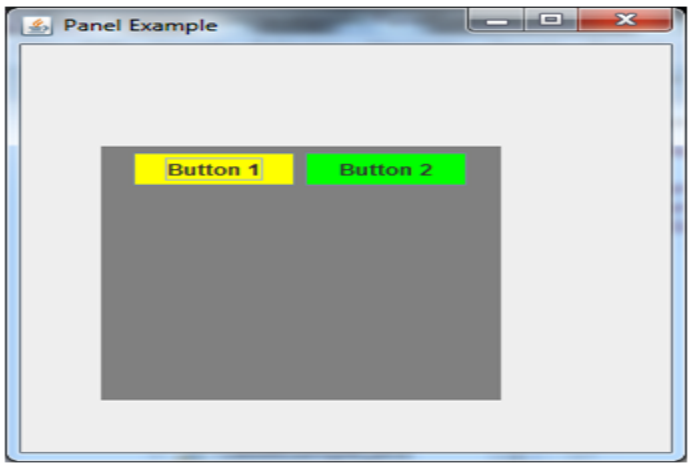
Constructor	Description
JPanel()	It is used to create a new JPanel with a double buffer and a flow layout.
JPanel(boolean isDoubleBuffered)	It is used to create a new JPanel with FlowLayout and the specified buffering strategy.
JPanel(LayoutManager layout)	It is used to create a new JPanel with the specified layout manager.

Java JPanel Example

```
1. import java.awt.*;
2. import javax.swing.*;
3. public class PanelExample {
4.     PanelExample()
5.       {
6.       JFrame f= new JFrame("Panel Example");
```

```
7.          JPanel panel=new JPanel();
8.          panel.setBounds(40,80,200,200);
9.          panel.setBackground(Color.gray);
10.         JButton b1=new JButton("Button 1");
11.         b1.setBounds(50,100,80,30);
12.         b1.setBackground(Color.yellow);
13.         JButton b2=new JButton("Button 2");
14.         b2.setBounds(100,100,80,30);
15.         b2.setBackground(Color.green);
16.         panel.add(b1); panel.add(b2);
17.         f.add(panel);
18.                 f.setSize(400,400);
19.                 f.setLayout(null);
20.                 f.setVisible(true);
21.         }
22.         public static void main(String args[])
23.         {
24.         new PanelExample();
25.         }
26.    }
```

Output:



BorderLayout (LayoutManagers)

Java LayoutManagers

The LayoutManagers are used to arrange components in a particular manner. The **Java LayoutManagers** facilitates us to control the positioning and size of the components in GUI forms. LayoutManager is an interface that is implemented by all the classes of layout managers. There are the following classes that represent the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout
9. javax.swing.SpringLayout etc.

Java FlowLayout

The Java FlowLayout class is used to arrange the components in a line, one after another (in a flow). It is the default layout of the applet or panel.

Fields of FlowLayout class

1. **public static final int LEFT**
2. **public static final int RIGHT**
3. **public static final int CENTER**
4. **public static final int LEADING**
5. **public static final int TRAILING**

Constructors of FlowLayout class

1. **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
2. **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
3. **FlowLayout(int align, int hgap, int vgap):** creates a flow layout with the given alignment and the given horizontal and vertical gap.

Example of FlowLayout class: Using FlowLayout() constructor

FileName: FlowLayoutExample.java

```
1. // import statements
2. import java.awt.*;
3. import javax.swing.*;
4.
5. public class FlowLayoutExample
6. {
7.
8. JFrame frameObj;
9.
10.// constructor
11.FlowLayoutExample()
12.{
13.    // creating a frame object
14.    frameObj = new JFrame();
15.
16.     // creating the buttons
17.    JButton b1 = new JButton("1");
18.    JButton b2 = new JButton("2");
```

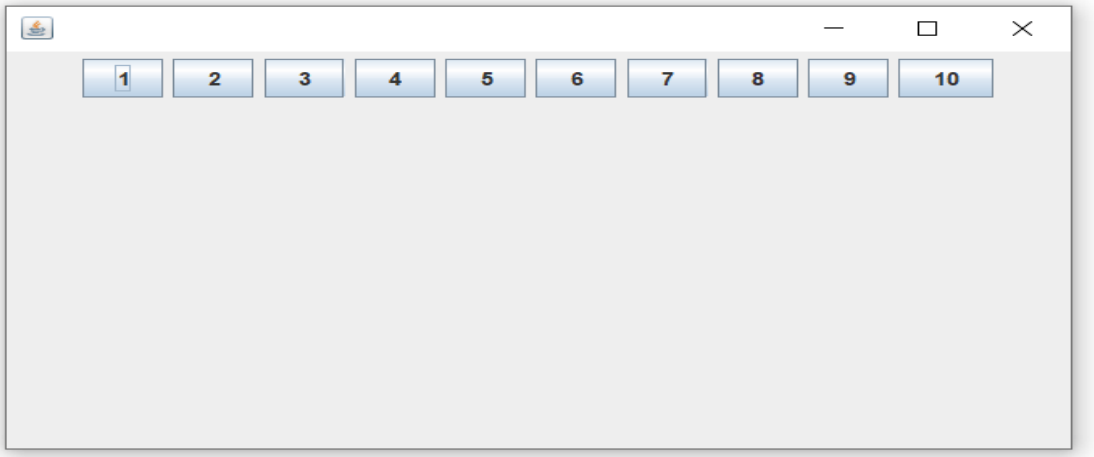
```
19.    JButton b3 = new JButton("3");
20.    JButton b4 = new JButton("4");
21.    JButton b5 = new JButton("5");
22.    JButton b6 = new JButton("6");
23.    JButton b7 = new JButton("7");
24.    JButton b8 = new JButton("8");
25.    JButton b9 = new JButton("9");
26.    JButton b10 = new JButton("10");
27.
28.
29.    // adding the buttons to frame
30.    frameObj.add(b1); frameObj.add(b2); frameObj.add(b3); frameObj.add(b4);

31.    frameObj.add(b5); frameObj.add(b6);  frameObj.add(b7);  frameObj.add(b8);

32.    frameObj.add(b9);  frameObj.add(b10);
33.
34.    // parameter less constructor is used
35.    // therefore, alignment is center
36.    // horizontal as well as the vertical gap is 5 units.
37.    frameObj.setLayout(new FlowLayout());
38.
39.    frameObj.setSize(300, 300);
40.    frameObj.setVisible(true);
41.}
42.
43.// main method
44.public static void main(String argvs[])
45.{
46.    new FlowLayoutExample();
```

47.}
48.}

Output:



Java SpringLayout

A **SpringLayout** *arranges the children of its associated container according to a set of constraints*. Constraints are nothing but horizontal and vertical distance between two-component edges. Every constraint is represented by a SpringLayout.Constraint object.

Each child of a SpringLayout container, as well as the container itself, has exactly one set of constraints associated with them.

Each edge position is dependent on the position of the other edge. If a constraint is added to create a new edge, than the previous binding is discarded. SpringLayout doesn't automatically set the location of the components it manages.

Constructor

SpringLayout(): The default constructor of the class is used to instantiate the SpringLayout class.

Nested Classes

Modifier and Type	Class	Description
static class	SpringLayout.Constraints	It is a Constraints object helps to govern component's size and position change in a container that is controlled by SpringLayout

SpringLayout Fields

Modifier and Type	Field	Description
static String	BASELINE	It specifies the baseline of a component.
static String	EAST	It specifies the right edge of a component's bounding rectangle.
static String	HEIGHT	It specifies the height of a component's bounding rectangle.
static String	HORIZONTAL_CENTER	It specifies the horizontal center of a component's bounding rectangle.
static String	NORTH	It specifies the top edge of a component's bounding rectangle.

static String	SOUTH	It specifies the bottom edge of a component's bounding rectangle.
static String	VERTICAL_CENTER	It specifies the vertical center of a component's bounding rectangle.
static String	WEST	It specifies the left edge of a component's bounding rectangle.
static String	WIDTH	It specifies the width of a component's bounding rectangle.

SpringLayout Methods

Modifier and Type	Method	Description
void	addLayoutComponent(Component component, Object constraints)	If constraints is an instance of SpringLayout. Constraints, associates the constraints with the specified component.
void	addLayoutComponent(String name, Component c)	Has no effect, since this layout manager does not use a per-component string.
Spring	getConstraint(String edgeName, Component c)	It returns the spring controlling the distance between the specified edge of the

		component and the top or left edge of its parent.
SpringLayout.Constraints	getConstraints(Component c)	It returns the constraints for the specified component.
float	getLayoutAlignmentX(Container p)	It returns 0.5f (centered).
float	getLayoutAlignmentY(Container p)	It returns 0.5f (centered).
void	invalidateLayout(Container p)	It Invalidates the layout, indicating that if the layout manager has cached information it should be discarded.
void	layoutContainer(Container parent)	It lays out the specified container.
Dimension	maximumLayoutSize(Container parent)	It is used to calculates the maximum size dimensions for the specified container, given the components it contains.
Dimension	minimumLayoutSize(Container parent)	It is used to calculates the minimum size dimensions for the specified container, given

		the components it contains.
Dimension	preferredLayoutSize(Container parent)	It is used to calculates the preferred size dimensions for the specified container, given the components it contains.

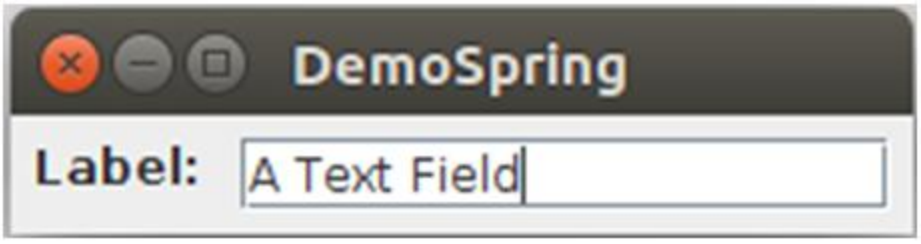
Example: 1

FileName: MySpringDemo.java

```
1. import java.awt.Container;
2. import javax.swing.JFrame;
3. import javax.swing.JLabel;
4. import javax.swing.JTextField;
5. import javax.swing.SpringLayout;
6. public class MySpringDemo {
7.     private static void createAndShowGUI() {
8.         JFrame frame = new JFrame("MySpringDemp");
9.         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10.
11.         Container contentPane = frame.getContentPane();
12.         SpringLayout layout = new SpringLayout();
13.         contentPane.setLayout(layout);
14.
15.         JLabel label = new JLabel("Label: ");
16.         JTextField textField = new JTextField("My Text Field", 15);
17.         contentPane.add(label);
18.         contentPane.add(textField);
19.
```

```
20.             layout.putConstraint(SpringLayout.WEST, label,6,SpringLayout.WEST, c
    ontentPane);
21.             layout.putConstraint(SpringLayout.NORTH, label,6,SpringLayout.NORT
    H, contentPane);
22.             layout.putConstraint(SpringLayout.WEST, textField,6,SpringLayout.EAST
    , label);
23.             layout.putConstraint(SpringLayout.NORTH, textField,6,SpringLayout.N
    ORTH, contentPane);
24.             layout.putConstraint(SpringLayout.EAST, contentPane,6,SpringLayout.E
    AST, textField);
25.             layout.putConstraint(SpringLayout.SOUTH, contentPane,6,SpringLayout
    .SOUTH, textField);
26.
27.             frame.pack();
28.             frame.setVisible(true);
29.         }
30.         public static void main(String[] args) {
31.             javax.swing.SwingUtilities.invokeLater(new Runnable() {
32.                 public void run() {
33.                     createAndShowGUI();
34.                 }
35.             });
36.         }
37.     }
```

Output:



Example: 2

The following program does the arrangement of the components in a JFrame. We have created one class named "MySpringDemo1 class" and have also created 4 JButton components called "btn1", "btn2", "btn3", "btn4", "btn5" and then adding them to the JFrame using the method add(). Using the method setvisible(), we have set the visibility of the frame. The setLayout() method sets the layout.

FileName: MySpringDemo1.java

```
1. // Importing the different Packages.
2. import javax.swing.*;
3. import java.awt.*;
4. // the MySpringDemo1 class
5. public class MySpringDemo1
6. {
7. // main method
8. public static void main(String argvs[])
9. {
10.// the main window
11.// Method for setting the default look and feel
12.// decorated status of the JFrame.
13.JFrame.setDefaultLookAndFeelDecorated(true);
14.// Creating an object of the "JFrame" class
15.JFrame f = new JFrame("Spring Layout Example");
16.// Function to set the default
17.// close operation status of JFrame
18.f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19.// method to determine the
20.// size status of the JFrame
21.f.setSize(310, 210);
```

```
22.// to get the content pane
23.Container cntt = f.getContentPane();
24.// Creating Object of "SpringLayout" class
25.SpringLayout sprLayout = new SpringLayout();
26.// for setting the layout class
27.f.setLayout(sprLayout);
28.// Initializing the object
29.// "btn1" of the JButton class.
30.Component btn1 = new JButton("C++");
31.// Initializing the object
32.// "btn2" of the JButton class.
33.Component btn2 = new JButton("Python");
34.// Initializing the object
35.// "btn3" the JButton class.
36.Component btn3 = new JButton("JAVA");
37.// Initializing the object
38.// "btn4" of the JButton class.
39.Component btn4 = new JButton("NETWORKING");
40.// Adding the JButton "btn1" on the frame f
41.f.add(btn1);
42.// Adding the JButton "btn2" on the frame f
43.f.add(btn2);
44.// Adding the JButton "btn3" on the frame f
45.f.add(btn3);
46.// Adding the JButton "btn4" on the frame f
47.f.add(btn4);
48.// It is used for inserting the
49.// layout constraint in the JFrame by using
50.// the springLayout class on the btn1 JButton
51.sprLayout.putConstraint(SpringLayout.WEST, btn1,
```

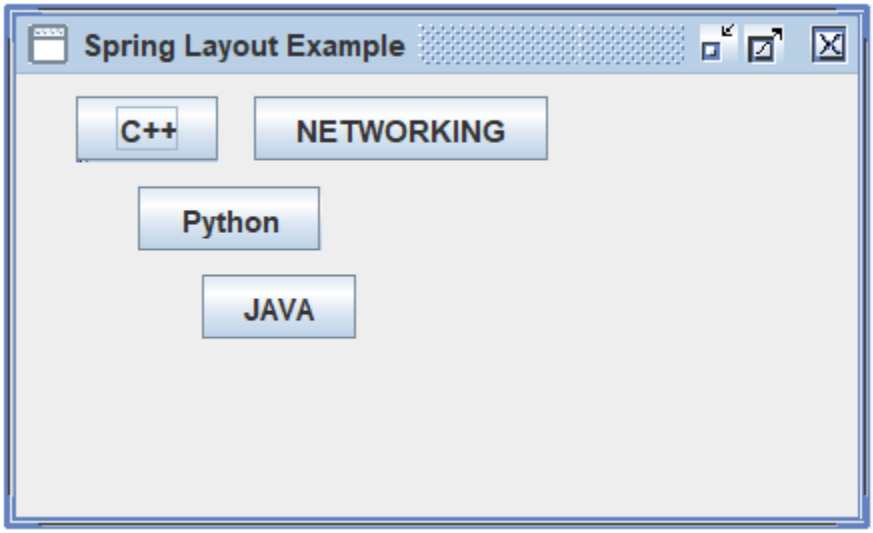
```
24, SpringLayout.WEST, cntt);

sprLayout.putConstraint(SpringLayout.NORTH, btn1,
    9, SpringLayout.NORTH, cntt);
// It is used for inserting the
// layout constraint in the JFrame using
// the springLayout class on the btn2 JButton
sprLayout.putConstraint(SpringLayout.WEST, btn2,
49, SpringLayout.WEST, cntt);
sprLayout.putConstraint(SpringLayout.NORTH, btn2,
        10, SpringLayout.SOUTH, btn1);
// It is used for inserting the
// layout constraint in the JFrame using
// springLayout class on the btn3 JButton
sprLayout.putConstraint(SpringLayout.WEST, btn3,
74, SpringLayout.WEST, cntt);

sprLayout.putConstraint(SpringLayout.NORTH, btn3,
        9, SpringLayout.SOUTH, btn2);
// It is used for inserting the
// layout constraint in the JFrame using
// sprLayout class on the btn4 JButton
sprLayout.putConstraint(SpringLayout.WEST, btn4,
        14, SpringLayout.EAST, btn1);
sprLayout.putConstraint(SpringLayout.NORTH, btn4,
    9, SpringLayout.NORTH, cntt);
// method for setting the
// visibility status of the JFrame
f.setVisible(true);
}
```

82.}

Output:



Java BoxLayout

The **Java BoxLayout class** is used to arrange the components either vertically or horizontally. For this purpose, the BoxLayout class provides four constants. They are as follows:

Note: The BoxLayout class is found in javax.swing package.

Fields of BoxLayout Class

1. **public static final int X_AXIS:** Alignment of the components are horizontal from left to right.
2. **public static final int Y_AXIS:** Alignment of the components are vertical from top to bottom.
3. **public static final int LINE_AXIS:** Alignment of the components is similar to the way words are aligned in a line, which is based on the ComponentOrientation property of the container. If the

ComponentOrientation property of the container is horizontal, then the components are aligned horizontally; otherwise, the components are aligned vertically. For horizontal orientations, we have two cases: left to right, and right to left. If the value ComponentOrientation property of the container is from left to right, then the components are rendered from left to right, and for right to left, the rendering of components is also from right to left. In the case of vertical orientations, the components are always rendered from top to bottom.

4. **public static final int PAGE_AXIS:** Alignment of the components is similar to the way text lines are put on a page, which is based on the ComponentOrientation property of the container. If the ComponentOrientation property of the container is horizontal, then components are aligned vertically; otherwise, the components are aligned horizontally. For horizontal orientations, we have two cases: left to right, and right to left. If the value ComponentOrientation property of the container is also from left to right, then the components are rendered from left to right, and for right to left, the rendering of components is from right to left. In the case of vertical orientations, the components are always rendered from top to bottom.

Constructor of BoxLayout class

1. **BoxLayout(Container c, int axis):** creates a box layout that arranges the components with the given axis.

Example of BoxLayout class with Y-AXIS:

FileName: BoxLayoutExample1.java

```
1.  import java.awt.*;
2.  import javax.swing.*;
3.
4.  public class BoxLayoutExample1 extends Frame {
5.   Button buttons[];
6.
7.   public BoxLayoutExample1 () {
8.     buttons = new Button [5];
9.
10.   for (int i = 0;i<5;i++) {
11.      buttons[i] = new Button ("Button " + (i + 1));
12.      // adding the buttons so that it can be displayed
13.      add (buttons[i]);
14.    }
15.  // the buttons will be placed horizontally
16.setLayout (new BoxLayout (this, BoxLayout.Y_AXIS));
17.setSize(400,400);
18.setVisible(true);
19.}
20.  // main method
21.public static void main(String args[]){
22.BoxLayoutExample1 b=new BoxLayoutExample1();
23.}
24.}
```

output:

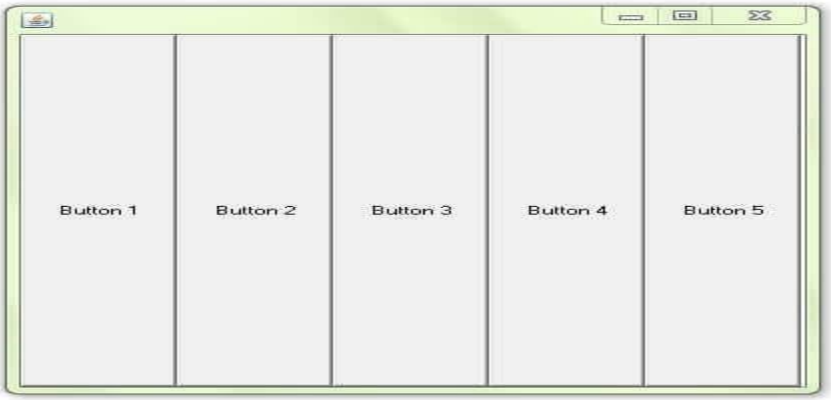
Example of BoxLayout class with X-AXIS

FileName: BoxLayoutExample2.java

```
1. import java.awt.*;
2. import javax.swing.*;
3.
4. public class BoxLayoutExample2 extends Frame {
5.  Button buttons[];
6.
7.  public BoxLayoutExample2() {
8.    buttons = new Button [5];
9.
10.  for (int i = 0;i<5;i++) {
11.     buttons[i] = new Button ("Button " + (i + 1));
12.     // adding the buttons so that it can be displayed
13.     add (buttons[i]);
14.   }
15. // the buttons in the output will be aligned vertically
16.setLayout (new BoxLayout(this, BoxLayout.X_AXIS));
17.setSize(400,400);
```

```
18.setVisible(true);
19.}
20. // main method
21.public static void main(String args[]){
22.BoxLayoutExample2 b=new BoxLayoutExample2();
23.}
24.}
```

Output:



Example of BoxLayout Class with LINE_AXIS

The following example shows the effect of setting the value of ComponentOrientation property of the container to RIGHT_TO_LEFT. If we do not set the value of ComponentOrientation property, then the components would be laid out from left to right. Comment line 11, and see it yourself.

FileName: BoxLayoutExample3.java

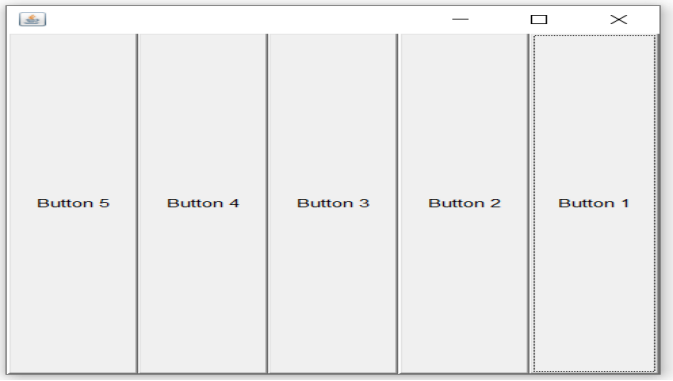
```
1. // import statements
2. import java.awt.*;
3. import javax.swing.*;
4.
5. public class BoxLayoutExample3 extends Frame
```

```
6.  {
7.  Button buttons[];
8.
9.  // constructor of the class
10.public BoxLayoutExample3()
11.{
12.buttons = new Button[5];
13.setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);  // line 11

14.
15.for (int i = 0; i < 5; i++)
16.{
17.  buttons[i] = new Button ("Button " + (i + 1));
18.
19.  // adding the buttons so that it can be displayed
20.  add (buttons[i]);
21.}
22.
23.// the ComponentOrientation is set to RIGHT_TO_LEFT. Therefore,
24.// the added buttons will be rendered from right to left
25.setLayout (new BoxLayout(this, BoxLayout.LINE_AXIS));
26.setSize(400, 400);
27.setVisible(true);
28.}
29.
30.// main method
31.public static void main(String argvs[])
32.{
33.// creating an object of the class BoxLayoutExample3
34.BoxLayoutExample3 obj = new BoxLayoutExample3();
```

35.}
36.}

Output:



Example of BoxLayout Class with PAGE_AXIS

The following example shows how to use PAGE_AXIS.

FileName: BoxLayoutExample4.java

```
1. // import statements
2. import java.awt.*;
3. import javax.swing.*;
4.
5. public class BoxLayoutExample4 extends Frame
6. {
7. Button buttons[];
8. // constructor of the class
9. public BoxLayoutExample4()
10.{
11.JFrame f = new JFrame();
12.JPanel pnl = new JPanel();
13.buttons = new Button[5];
14.GridBagConstraints constrntObj = new GridBagConstraints();
```

```
15.
16.constrntObj.fill = GridBagConstraints.VERTICAL;
17.for (int i = 0; i < 5; i++)
18.{
19.  buttons[i] = new Button ("Button " + (i + 1));
20.
21.  // adding the buttons so that it can be displayed
22.  add(buttons[i]);
23.}
24.
25.// the components will be displayed just like the line is present on a page
26.setLayout (new BoxLayout(this, BoxLayout.PAGE_AXIS));
27.setSize(400, 400);
28.setVisible(true);
29.}
30.
31.// main method
32.public static void main(String argvs[])
33.{
34.// creating an object of the class BoxLayoutExample4
35.BoxLayoutExample4 obj = new BoxLayoutExample4();
36.}
```

Output:

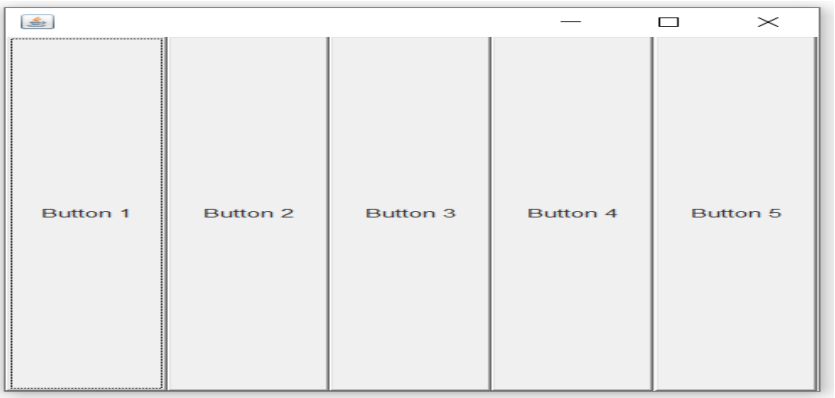
The above output shows that the buttons are aligned horizontally. Now, we will display the buttons vertically using the PAGE_AXIS.

FileName: BoxLayoutExample5.java

```
1. // import statementss
2. import java.awt.*;
3. import javax.swing.*;
4.
5. public class BoxLayoutExample5 extends Frame
6. {
7. Button buttons[];
8.
9. // constructor of the class
10.public BoxLayoutExample5()
11.{
12.JFrame f = new JFrame();
13.buttons = new Button[5];
14.
15.// Creating a Box whose alignment is horizontal
16.Box horizontalBox = Box.createHorizontalBox();
17.
18.// ContentPane returns a container
19.Container contentPane = f.getContentPane();
20.
21.for (int i = 0; i < 5; i++)
22.{
23.  buttons[i] = new Button ("Button " + (i + 1));
24.
25.  // adding the buttons to the box so that it can be displayed
```

```
26.  horizontalBox.add(buttons[i]);
27.}
28.
29.// adding the box and the borderlayout to the content pane
30.contentPane.add(horizontalBox, BorderLayout.NORTH);
31.
32.// now, the rendered components are displayed vertically.
33.// it is because the box is aligned horizontally
34.f.setLayout (new BoxLayout(contentPane, BoxLayout.PAGE_AXIS));
35.
36.f.setSize(400, 400);
37.f.setVisible(true);
38.}
39.
40.// main method
41.public static void main(String argvs[])
42.{
43.// creating an object of the class BoxLayoutExample5
44.BoxLayoutExample5 obj = new BoxLayoutExample5();
45.}
46.}
```

Output:



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, Acces sible

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()	t 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

```
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.     f.setLayout(null);
15.     f.setVisible(true);
16.     }
17.     }
```

Output:

Java JLabel Example with ActionListener

```
1.  import javax.swing.*;
2.  import java.awt.*;
3.  import java.awt.event.*;
4.  public class LabelExample extends Frame implements ActionListener{
5.      JTextField tf; JLabel l; JButton b;
6.      LabelExample(){
7.          tf=new JTextField();
8.          tf.setBounds(50,50, 150,20);
```

```
9.          l=new JLabel();
10.         l.setBounds(50,100, 250,20);
11.         b=new JButton("Find IP");
12.         b.setBounds(50,150,95,30);
13.         b.addActionListener(this);
14.         add(b);add(tf);add(l);
15.         setSize(400,400);
16.         setLayout(null);
17.         setVisible(true);
18.     }
19.     public void actionPerformed(ActionEvent e) {
20.         try{
21.         String host=tf.getText();
22.         String ip=java.net.InetAddress.getByName(host).getHostAddress();
23.         l.setText("IP of "+host+" is: "+ip);
24.         }catch(Exception ex){System.out.println(ex);}
25.     }
26.     public static void main(String[] args) {
27.         new LabelExample();
28.     } }
```

Output:

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:

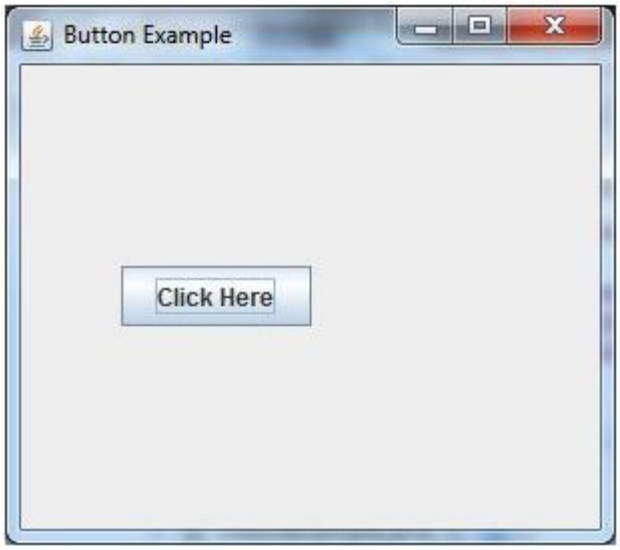
Methods	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text of the button.

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 action listener to this object.

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:

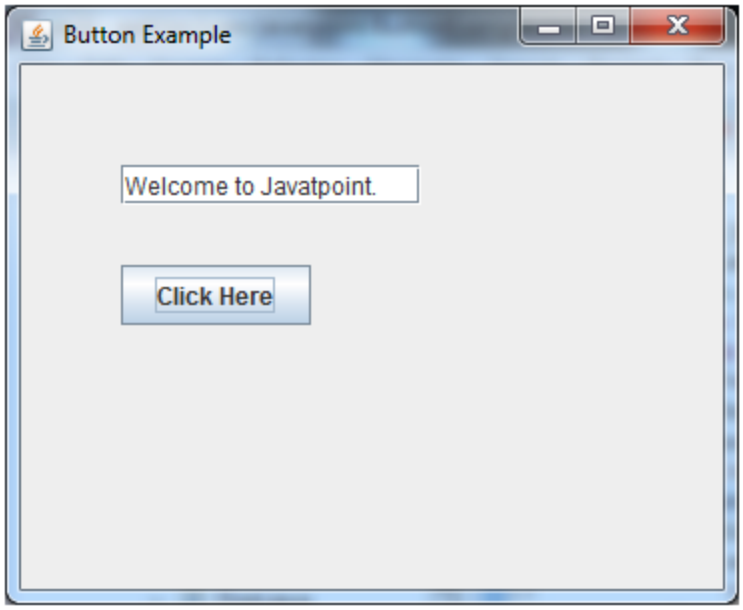


Java JButton Example with ActionListener

```
1. import java.awt.event.*;
2. import javax.swing.*;
3. public class ButtonExample {
4. public static void main(String[] args) {
5.     JFrame f=new JFrame("Button Example");
6.     final JTextField tf=new JTextField();
7.     tf.setBounds(50,50, 150,20);
8.     JButton b=new JButton("Click Here");
9.     b.setBounds(50,100,95,30);
10.    b.addActionListener(new ActionListener(){
11.public void actionPerformed(ActionEvent e){
12.            tf.setText("Welcome to Javatpoint.");
13.        }
14.    });
15.    f.add(b);f.add(tf);
16.    f.setSize(400,400);
```

```
17.     f.setLayout(null);
18.     f.setVisible(true);
19.}
20.}
```

Output:



Example of displaying image on the button:

```
1. import javax.swing.*;
2. public class ButtonExample{
3. ButtonExample(){
4. JFrame f=new JFrame("Button Example");
5. JButton b=new JButton(new ImageIcon("D:\\icon.png"));
6. b.setBounds(100,100,100, 40);
7. f.add(b);
8. f.setSize(300,400);
9. f.setLayout(null);
10.f.setVisible(true);
```

```
11.f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12.    }
13.public static void main(String[] args) {
14.    new ButtonExample();
15.}
16.}
```

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** SwingConstant
   s

Commonly used Constructors:

Constructor	Description
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.
JTextField(int columns)	Creates a new empty TextField with the specified number of columns.

Commonly used Methods:

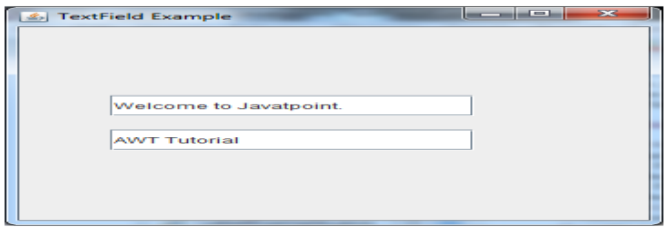
Methods	Description
void addActionListener(ActionListener l)	It is used to add the specified action listener to receive action events from this textfield.
Action getAction()	It returns the currently set Action for this ActionEvent source, or null if no Action is set.
void setFont(Font f)	It is used to set the current font.

void removeActionListener(ActionListener l)	It is used to remove the specified action listener so that it no longer receives action events from this textfield.

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:

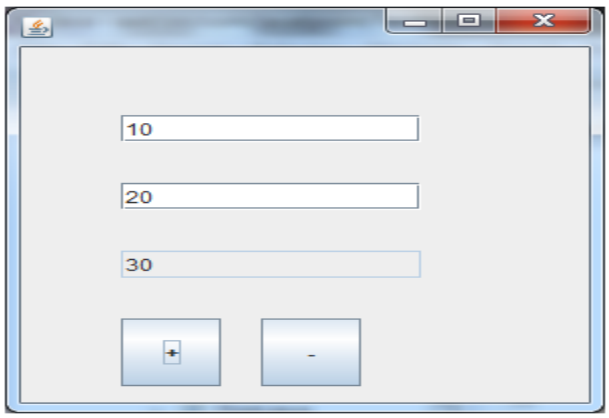


Java JTextField Example with ActionListener

```
1. import javax.swing.*;
2. import java.awt.event.*;
3. public class TextFieldExample implements ActionListener{
4.     JTextField tf1,tf2,tf3;
5.     JButton b1,b2;
6.     TextFieldExample(){
7.         JFrame f= new JFrame();
8.         tf1=new JTextField();
9.         tf1.setBounds(50,50,150,20);
10.        tf2=new JTextField();
11.        tf2.setBounds(50,100,150,20);
12.        tf3=new JTextField();
13.        tf3.setBounds(50,150,150,20);
14.        tf3.setEditable(false);
15.        b1=new JButton("+");
16.        b1.setBounds(50,200,50,50);
17.        b2=new JButton("-");
18.        b2.setBounds(120,200,50,50);
19.        b1.addActionListener(this);
20.        b2.addActionListener(this);
21.        f.add(tf1);f.add(tf2);f.add(tf3);f.add(b1);f.add(b2);
22.        f.setSize(300,300);
23.        f.setLayout(null);
24.        f.setVisible(true);
25.    }
26.    public void actionPerformed(ActionEvent e) {
27.        String s1=tf1.getText();
28.        String s2=tf2.getText();
29.        int a=Integer.parseInt(s1);
```

```
30.         int b=Integer.parseInt(s2);
31.         int c=0;
32.         if(e.getSource()==b1){
33.             c=a+b;
34.         }else if(e.getSource()==b2){
35.             c=a-b;
36.         }
37.         String result=String.valueOf(c);
38.         tf3.setText(result);
39.     }
40.public static void main(String[] args) {
41.     new TextFieldExample();
42.} }
```

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.

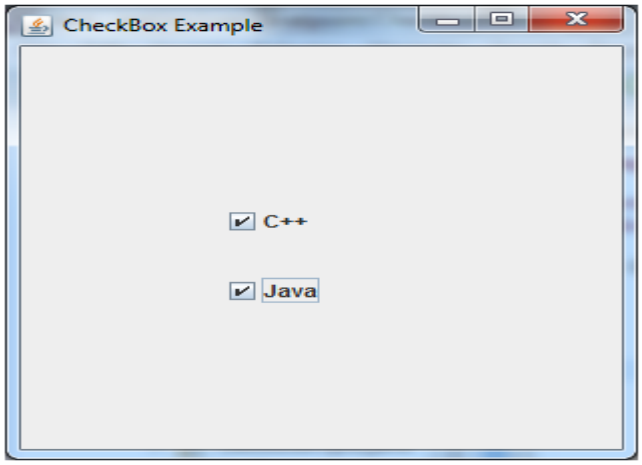
Commonly used Methods:

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.

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:

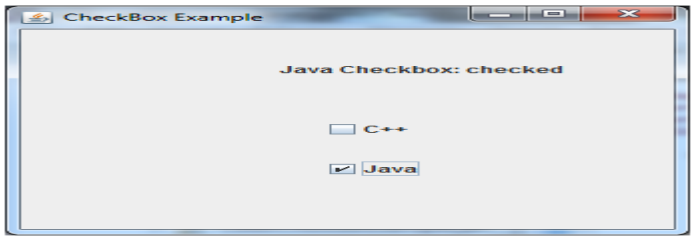


Java JCheckBox Example with ItemListener

```
1. import javax.swing.*;
2. import java.awt.event.*;
3. public class CheckBoxExample
4. {
5.     CheckBoxExample(){
6.       JFrame f= new JFrame("CheckBox Example");
7.       final JLabel label = new JLabel();
8.       label.setHorizontalAlignment(JLabel.CENTER);
9.       label.setSize(400,100);
10.      JCheckBox checkbox1 = new JCheckBox("C++");
11.      checkbox1.setBounds(150,100, 50,50);
12.      JCheckBox checkbox2 = new JCheckBox("Java");
13.      checkbox2.setBounds(150,150, 50,50);
14.      f.add(checkbox1); f.add(checkbox2); f.add(label);
15.      checkbox1.addItemListener(new ItemListener() {
16.          public void itemStateChanged(ItemEvent e) {
17.            label.setText("C++ Checkbox: "
18.            + (e.getStateChange()==1?"checked":"unchecked"));
19.          }
20.       });
21.      checkbox2.addItemListener(new ItemListener() {
22.          public void itemStateChanged(ItemEvent e) {
23.            label.setText("Java Checkbox: "
24.            + (e.getStateChange()==1?"checked":"unchecked"));
25.          }
26.       });
27.      f.setSize(400,400);
28.      f.setLayout(null);
```

```
29.         f.setVisible(true);
30.      }
31.public static void main(String args[])
32.{
33.     new CheckBoxExample();
34.}
35.}
```

Output:



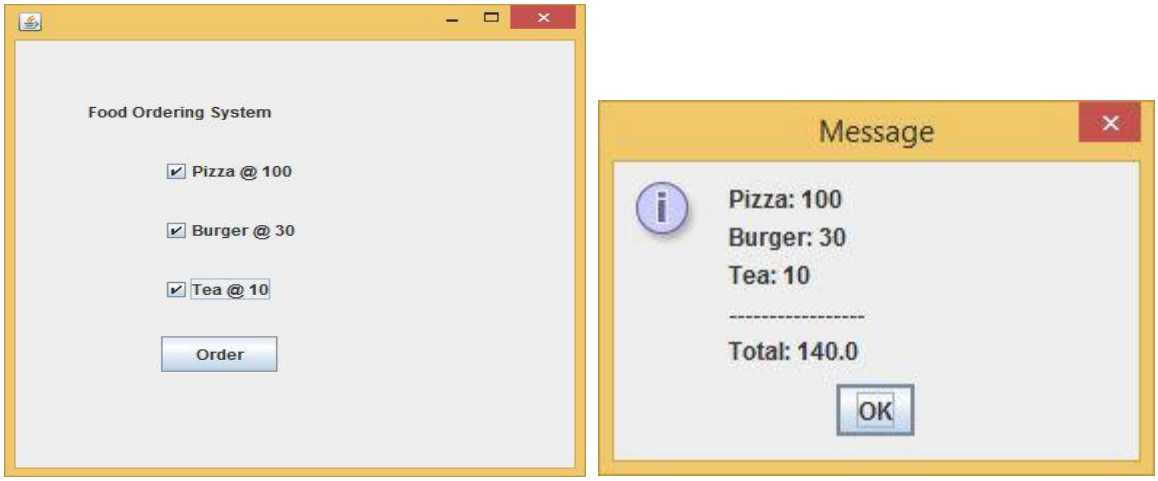
Java JCheckBox Example: Food Order

```
1. import javax.swing.*;
2. import java.awt.event.*;
3. public class CheckBoxExample extends JFrame implements ActionListener{
4.     JLabel l;
5.     JCheckBox cb1,cb2,cb3;
6.     JButton b;
7.     CheckBoxExample(){
8.         l=new JLabel("Food Ordering System");
9.         l.setBounds(50,50,300,20);
10.        cb1=new JCheckBox("Pizza @ 100");
11.        cb1.setBounds(100,100,150,20);
12.        cb2=new JCheckBox("Burger @ 30");
13.        cb2.setBounds(100,150,150,20);
14.        cb3=new JCheckBox("Tea @ 10");
```

```
15.     cb3.setBounds(100,200,150,20);
16.     b=new JButton("Order");
17.     b.setBounds(100,250,80,30);
18.     b.addActionListener(this);
19.     add(l);add(cb1);add(cb2);add(cb3);add(b);
20.     setSize(400,400);
21.     setLayout(null);
22.     setVisible(true);
23.     setDefaultCloseOperation(EXIT_ON_CLOSE);
24. }
25. public void actionPerformed(ActionEvent e){
26.     float amount=0;
27.     String msg="";
28.     if(cb1.isSelected()){
29.         amount+=100;
30.         msg="Pizza: 100\n";
31.     }
32.     if(cb2.isSelected()){
33.         amount+=30;
34.         msg+="Burger: 30\n";
35.     }
36.     if(cb3.isSelected()){
37.         amount+=10;
38.         msg+="Tea: 10\n";
39.     }
40.     msg+="-----------------\n";
41.     JOptionPane.showMessageDialog(this,msg+"Total: "+amount);
42. }
43. public static void main(String[] args) {
44.     new CheckBoxExample();
```

```
45.    }
46.}
```

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.

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.

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

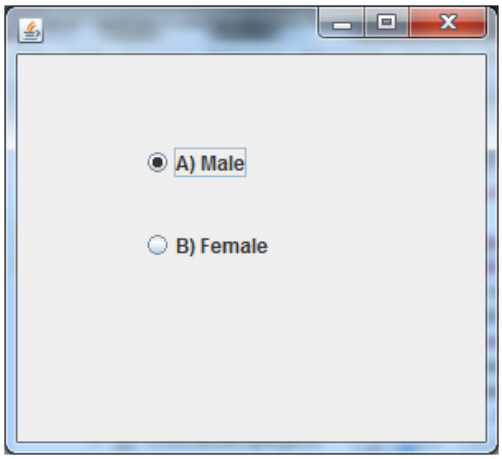
Commonly used Methods:

Java JRadioButton Example

```
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 JRadioButton Example with ActionListener

```
1.  import javax.swing.*;
```

```
2. import java.awt.event.*;
3. class RadioButtonExample extends JFrame implements ActionListener{
4. JRadioButton rb1,rb2;
5. JButton b;
6. RadioButtonExample(){
7. rb1=new JRadioButton("Male");
8. rb1.setBounds(100,50,100,30);
9. rb2=new JRadioButton("Female");
10.rb2.setBounds(100,100,100,30);
11.ButtonGroup bg=new ButtonGroup();
12.bg.add(rb1);bg.add(rb2);
13.b=new JButton("click");
14.b.setBounds(100,150,80,30);
15.b.addActionListener(this);
16.add(rb1);add(rb2);add(b);
17.setSize(300,300);
18.setLayout(null);
19.setVisible(true);
20.}
21.public void actionPerformed(ActionEvent e){
22.if(rb1.isSelected()){
23.JOptionPane.showMessageDialog(this,"You are Male.");
24.}
25.if(rb2.isSelected()){
26.JOptionPane.showMessageDialog(this,"You are Female.");
27.}
28.}
29.public static void main(String args[]){
30.new RadioButtonExample();
31.}}
```

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 array.
JComboBox(Vector<?> items)	Creates a JComboBox that contains the elements in the specified Vector.

Commonly used Methods:

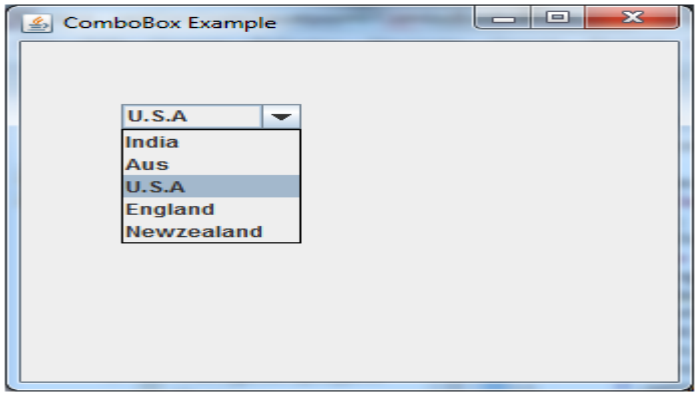
Methods	Description
void addItem(Object anObject)	It is used to add an item to the item list.
void removeItem(Object anObject)	It is used to delete an item to the item list.
void removeAllItems()	It is used to remove all the items from the list.
void setEditable(boolean b)	It is used to determine whether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

Java JComboBox Example

```
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:

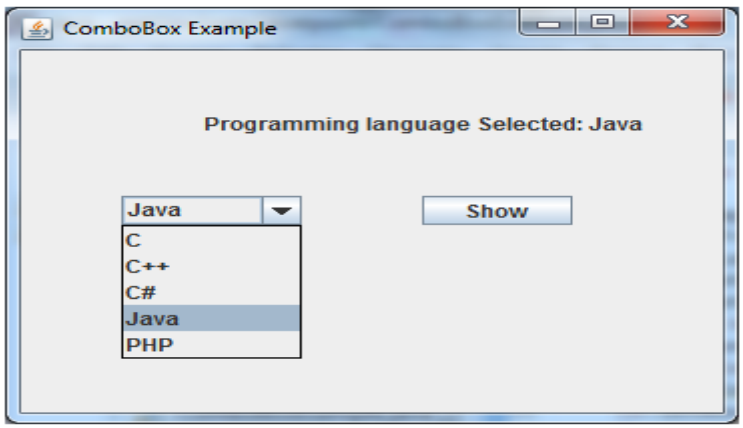


Java JComboBox Example with ActionListener

```
1.  import javax.swing.*;
2.  import java.awt.event.*;
3.  public class ComboBoxExample {
4.  JFrame f;
5.  ComboBoxExample(){
6.      f=new JFrame("ComboBox Example");
7.      final JLabel label = new JLabel();
8.      label.setHorizontalAlignment(JLabel.CENTER);
9.      label.setSize(400,100);
10.     JButton b=new JButton("Show");
```

```
11.    b.setBounds(200,100,75,20);
12.    String languages[]={"C","C++","C#","Java","PHP"};
13.    final JComboBox cb=new JComboBox(languages);
14.    cb.setBounds(50, 100,90,20);
15.    f.add(cb); f.add(label); f.add(b);
16.    f.setLayout(null);
17.    f.setSize(350,350);
18.    f.setVisible(true);
19.    b.addActionListener(new ActionListener() {
20.        public void actionPerformed(ActionEvent e) {
21.String data = "Programming language Selected: "
22.   + cb.getItemAt(cb.getSelectedIndex());
23.label.setText(data);
24.}
25.});
26.}
27.public static void main(String[] args) {
28.    new ComboBoxExample();
29.}
30.}
```

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.

JList class declaration

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

1. **public class** JList **extends** JComponent **implements** Scrollable, Accessible

Commonly used Constructors:

Constructor	Description
JList()	Creates a JList with an empty, read-only, model.
JList(ary[] listData)	Creates a JList that displays the elements in the specified array.
JList(ListModel<ary> dataModel)	Creates a JList that displays elements from the specified, non-null, model.

Commonly used Methods:

Methods	Description
Void addListSelectionListener(ListSelection Listener listener)	It is used to add a listener to the list, to be notified each time a change to the selection occurs.

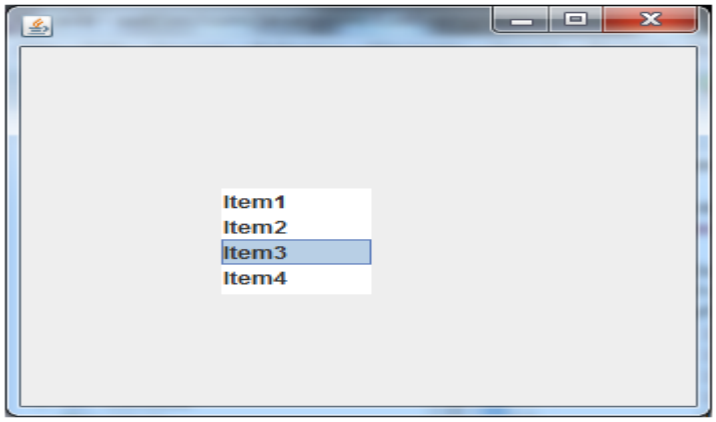
int getSelectedIndex()	It is used to return the smallest selected cell index.
ListModel getModel()	It is used to return the data model that holds a list of items displayed by the JList component.
void setListData(Object[] listData)	It is used to create a read-only ListModel from an array of objects.

Java JList Example

```
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:



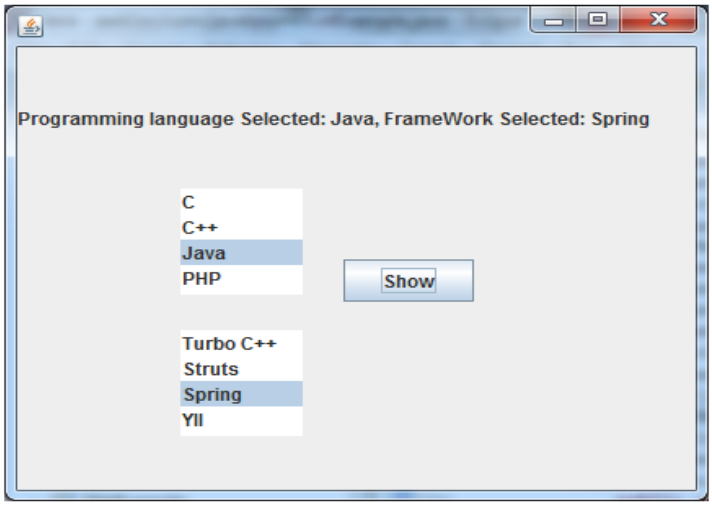
Java JList Example with ActionListener

```
1.  import javax.swing.*;
2.  import java.awt.event.*;
3.  public class ListExample
4.  {
5.      ListExample(){
6.          JFrame f= new JFrame();
7.          final JLabel label = new JLabel();
8.          label.setSize(500,100);
9.          JButton b=new JButton("Show");
10.         b.setBounds(200,150,80,30);
11.         final DefaultListModel<String> l1 = new DefaultListModel<>();
12.           l1.addElement("C");
13.           l1.addElement("C++");
14.           l1.addElement("Java");
15.           l1.addElement("PHP");
16.           final JList<String> list1 = new JList<>(l1);
```

```
17.         list1.setBounds(100,100, 75,75);
18.         DefaultListModel<String> l2 = new DefaultListModel<>();
19.         l2.addElement("Turbo C++");
20.         l2.addElement("Struts");
21.         l2.addElement("Spring");
22.         l2.addElement("YII");
23.         final JList<String> list2 = new JList<>(l2);
24.         list2.setBounds(100,200, 75,75);
25.         f.add(list1); f.add(list2); f.add(b); f.add(label);
26.         f.setSize(450,450);
27.         f.setLayout(null);
28.         f.setVisible(true);
29.         b.addActionListener(new ActionListener() {
30.             public void actionPerformed(ActionEvent e) {
31.                 String data = "";
32.                 if (list1.getSelectedIndex() != -1) {
33.                     data = "Programming language Selected: " + list1.getSelectedVal
    ue();
34.                     label.setText(data);
35.                 }
36.                 if(list2.getSelectedIndex() != -1){
37.                     data += ", FrameWork Selected: ";
38.                     for(Object frame :list2.getSelectedValues()){
39.                         data += frame + " ";
40.                     }
41.                 }
42.                 label.setText(data);
43.             }
44.         });
45.     }
```

```
46.public static void main(String args[])
47.    {
48.   new ListExample();
49.    }}
```

Output:



Java JMenuBar, JMenu and JMenuItem

The JMenuBar class is used to display menubar on the window or frame. It may have several menus.

The object of JMenu class is a pull down menu component which is displayed from the menu bar. It inherits the JMenuItem class.

The object of JMenuItem class adds a simple labeled menu item. The items used in a menu must belong to the JMenuItem or any of its subclass.

JMenuBar class declaration

1. **public class** JMenuBar **extends** JComponent **implements** MenuElement, Acce ssible

JMenu class declaration

```
1. public class JMenu extends JMenuItem implements MenuElement, Accessibl
   e
```

JMenuItem class declaration

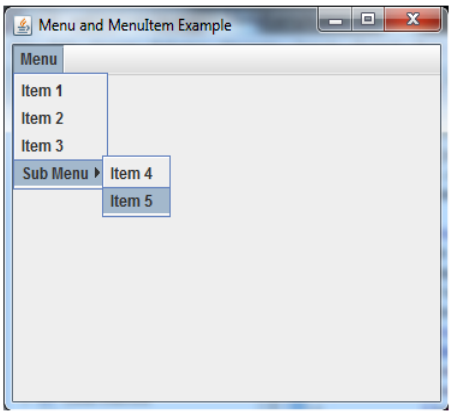
```
1. public class JMenuItem extends AbstractButton implements Accessible, Men
   uElement
```

Java JMenuItem and JMenu Example

```
1. import javax.swing.*;
2. class MenuExample
3. {
4.         JMenu menu, submenu;
5.         JMenuItem i1, i2, i3, i4, i5;
6.         MenuExample(){
7.         JFrame f= new JFrame("Menu and MenuItem Example");
8.         JMenuBar mb=new JMenuBar();
9.         menu=new JMenu("Menu");
10.        submenu=new JMenu("Sub Menu");
11.        i1=new JMenuItem("Item 1");
12.        i2=new JMenuItem("Item 2");
13.        i3=new JMenuItem("Item 3");
14.        i4=new JMenuItem("Item 4");
15.        i5=new JMenuItem("Item 5");
16.        menu.add(i1); menu.add(i2); menu.add(i3);
17.        submenu.add(i4); submenu.add(i5);
18.        menu.add(submenu);
19.        mb.add(menu);
20.        f.setJMenuBar(mb);
21.        f.setSize(400,400);
22.        f.setLayout(null);
```

```
23.         f.setVisible(true);
24.}
25.public static void main(String args[])
26.{
27.new MenuExample();
28.}}
```

Output:



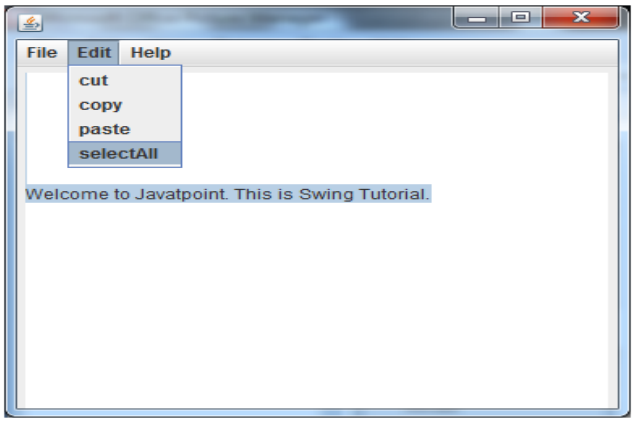
Example of creating Edit menu for Notepad:

```
1. import javax.swing.*;
2. import java.awt.event.*;
3. public class MenuExample implements ActionListener{
4. JFrame f;
5. JMenuBar mb;
6. JMenu file,edit,help;
7. JMenuItem cut,copy,paste,selectAll;
8. JTextArea ta;
9. MenuExample(){
10.f=new JFrame();
11.cut=new JMenuItem("cut");
12.copy=new JMenuItem("copy");
```

```
13.paste=new JMenuItem("paste");
14.selectAll=new JMenuItem("selectAll");
15.cut.addActionListener(this);
16.copy.addActionListener(this);
17.paste.addActionListener(this);
18.selectAll.addActionListener(this);
19.mb=new JMenuBar();
20.file=new JMenu("File");
21.edit=new JMenu("Edit");
22.help=new JMenu("Help");
23.edit.add(cut);edit.add(copy);edit.add(paste);edit.add(selectAll);
24.mb.add(file);mb.add(edit);mb.add(help);
25.ta=new JTextArea();
26.ta.setBounds(5,5,360,320);
27.f.add(mb);f.add(ta);
28.f.setJMenuBar(mb);
29.f.setLayout(null);
30.f.setSize(400,400);
31.f.setVisible(true);
32.}
33.public void actionPerformed(ActionEvent e) {
34.if(e.getSource()==cut)
35.ta.cut();
36.if(e.getSource()==paste)
37.ta.paste();
38.if(e.getSource()==copy)
39.ta.copy();
40.if(e.getSource()==selectAll)
41.ta.selectAll();
42.}
```

```
43.public static void main(String[] args) {
44.    new MenuExample();
45.}
46.}
```

Output:



Java JDialog

The JDialog control represents a top level window with a border and a title used to take some form of input from the user. It inherits the Dialog class.

Unlike JFrame, it doesn't have maximize and minimize buttons.

JDialog class declaration

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

1. **public class** JDialog **extends** Dialog **implements** WindowConstants, Accessibl e, RootPaneContainer

Commonly used Constructors:

Constructor	Description
JDialog()	It is used to create a modeless dialog without a title and without a specified Frame owner.
JDialog(Frame owner)	It is used to create a modeless dialog with specified Frame as its owner and an empty title.
JDialog(Frame owner, String title, boolean modal)	It is used to create a dialog with the specified title, owner Frame and modality.

Java JDialog Example

```
1. import javax.swing.*;
2. import java.awt.*;
3. import java.awt.event.*;
4. public class DialogExample {
5.     private static JDialog d;
6.     DialogExample() {
7.         JFrame f= new JFrame();
8.         d = new JDialog(f , "Dialog Example", true);
9.         d.setLayout( new FlowLayout() );
10.        JButton b = new JButton ("OK");
11.        b.addActionListener ( new ActionListener()
12.        {
13.            public void actionPerformed( ActionEvent e )
14.            {
15.                DialogExample.d.setVisible(false);
```

```
16.             }
17.         });
18.         d.add( new JLabel ("Click button to continue."));
19.         d.add(b);
20.         d.setSize(300,300);
21.         d.setVisible(true);
22.     }
23.     public static void main(String args[])
24.     {
25.         new DialogExample();
26.     }
27.}
```

Output: