# Swing Components

**Swing Components: Introduction, Swing Components hierarchy, Layout Managers, JFrame, JPanel, JButton, JTextField, JPasswordField, JTextArea, JList, JCombobox, JCheckBox, JRadioButton, JScrollPane, JSplitPane, JTabbedPane, JOptionPane, JProgressBar.**

## Java Swing

- ✓ It 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

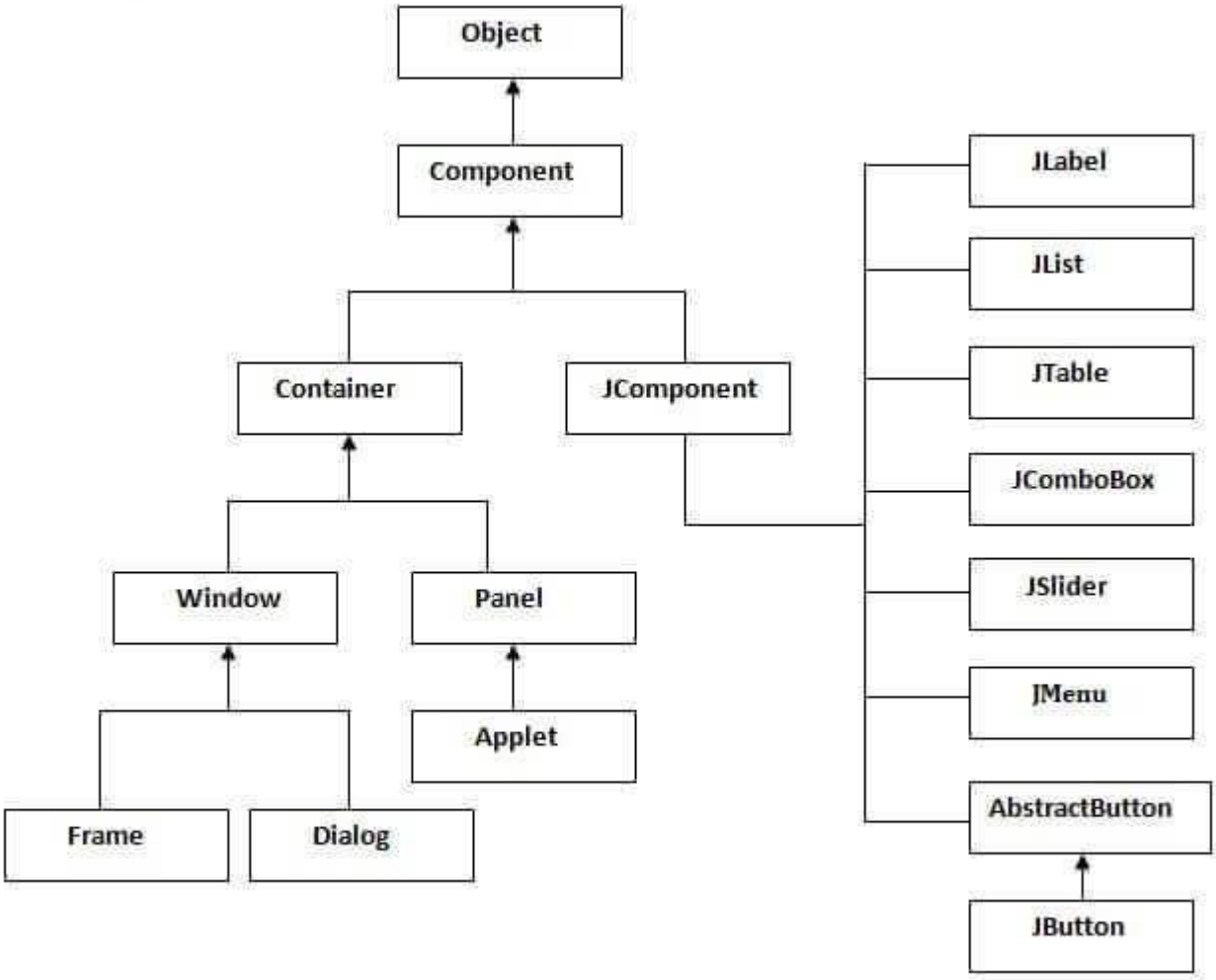
There are many differences between java awt and swing that are given below.

| 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**. |

## Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.




## Commonly used Methods of Component class

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

| Method | Description |
|---|---|

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

**There are two ways to create a frame:** ○ By creating the object of Frame class (association)

- By extending Frame class (inheritance)

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

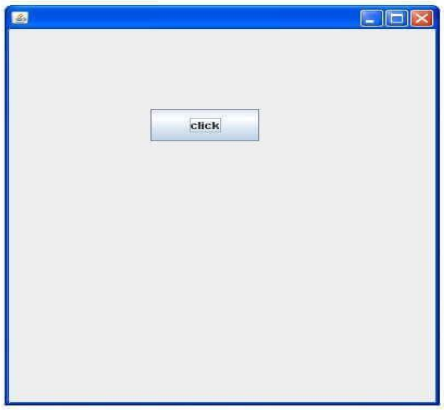
### Simple Java Swing Example

**Let's** see a simple swing example where we are creating one button and adding it on the JFrame object inside the main() method.

```
import javax.swing.*;   public class
FirstSwingExample {   public static
void main(String[] args) {
JFrame f=new JFrame();//creating instance of JFrame

JButton b=new JButton("click");//creating instance of JButton   b.setBounds(130,100,100,
40);//x axis, y axis, width, height

f.add(b);//adding button in JFrame

f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
}
}
```

**You can write the code in constructor**

```
import javax.swing.*;

public class Simple2 {

Simple2()
{
JFrame f=new JFrame();//creating instance of JFrame   JButton
b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);//x axis, y axis, width, height
f.add(b);//adding button in JFrame
f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
}
public static void main(String[] args) {
Simple2 k=new Simple2();
}
}
```




### Simple example of Swing by inheritance

We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

*File: Simple2.java*

```
import javax.swing.*;
public class Simple2 extends JFrame{//inheriting JFrame
Simple2(){
JButton b=new JButton("click");//create button
b.setBounds(130,100,100, 40);

add(b);//adding button on frame
setSize(400,500);
setLayout(null);
setVisible(true);
}
public static void main(String[] args) {
JFrame f;
Simple2 obj=new Simple2();
}}
```

## BorderLayout (LayoutManagers)

### Java LayoutManagers

The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents 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 BorderLayout

The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

1. **public static final int NORTH**
2. **public static final int SOUTH**
3. **public static final int EAST**
4. **public static final int WEST**
5. **public static final int CENTER**

### Constructors of BorderLayout class:

- **BorderLayout():** creates a border layout but with no gaps between the components.
- **JBorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

### Example of BorderLayout class:

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

public class Border {
JFrame f;   Border(){
   f=new JFrame();

   JButton b1=new JButton("NORTH");;
JButton b2=new JButton("SOUTH");;
   JButton b3=new JButton("EAST");;
   JButton b4=new JButton("WEST");;
   JButton b5=new JButton("CENTER");;

   f.add(b1,BorderLayout.NORTH);
   f.add(b2,BorderLayout.SOUTH);
   f.add(b3,BorderLayout.EAST);
   f.add(b4,BorderLayout.WEST);
```

```
    f.add(b5,BorderLayout.CENTER);

    f.setSize(300,300);
    f.setVisible(true);    }
public static void main(String[] args) {
new Border();
}
}
```


## Java GridLayout

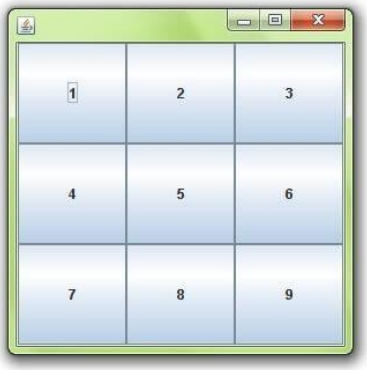
The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.

### Constructors of GridLayout class

1. **GridLayout():** creates a grid layout with one column per component in a row.
2. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.
3. **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps.

### Example of GridLayout class

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

public class MyGridLayout{
JFrame f;   MyGridLayout(){
   f=new JFrame();

   JButton b1=new JButton("1");
   JButton b2=new JButton("2");
   JButton b3=new JButton("3");
   JButton b4=new JButton("4");
   JButton b5=new JButton("5");
      JButton b6=new JButton("6");
      JButton b7=new JButton("7");
      JButton b8=new JButton("8");
      JButton b9=new JButton("9");

   f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
   f.add(b6);f.add(b7);f.add(b8);f.add(b9);

   f.setLayout(new GridLayout(3,3));
   //setting grid layout of 3 rows and 3 columns

   f.setSize(300,300);
   f.setVisible(true);   }
public static void main(String[] args) {
new MyGridLayout();
}
}
```




## Java FlowLayout

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

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

public class MyFlowLayout{
JFrame f;   MyFlowLayout(){
   f=new JFrame();

   JButton b1=new JButton("1");
   JButton b2=new JButton("2");
   JButton b3=new JButton("3");
   JButton b4=new JButton("4");
   JButton b5=new JButton("5");

   f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);

   f.setLayout(new FlowLayout(FlowLayout.RIGHT));
//setting flow layout of right alignment

   f.setSize(300,300);
   f.setVisible(true);   }
public static void main(String[] args) {
new MyFlowLayout();
}
}
```


## Java BoxLayout

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

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

### Fields of BoxLayout class

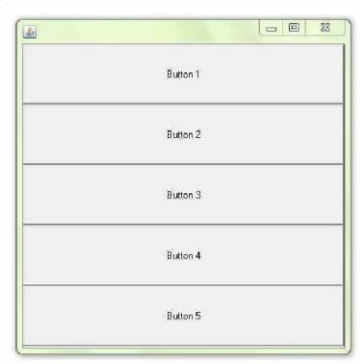
1. **public static final int X_AXIS**
2. **public static final int Y_AXIS**
3. **public static final int LINE_AXIS**
4. **public static final int PAGE_AXIS**

### 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:

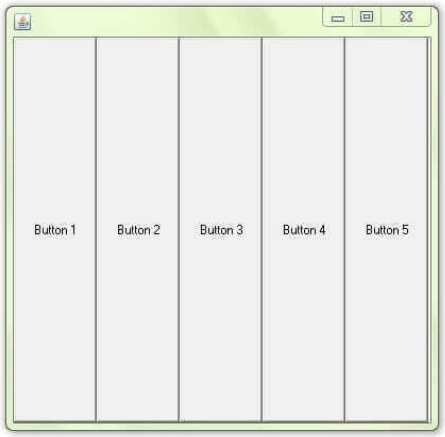
```
import java.awt.*;
import javax.swing.*;

public class BoxLayoutExample1 extends Frame {
Button buttons[];

 public BoxLayoutExample1 () {
   buttons = new Button [5];

   for (int i = 0;i<5;i++) {        buttons[i] = new
Button ("Button " + (i + 1));        add
(buttons[i]);
    }

setLayout (new BoxLayout (this, BoxLayout.Y_AXIS));
setSize(400,400);   setVisible(true);
}

public static void main(String args[]){
BoxLayoutExample1 b=new BoxLayoutExample1();
}
}
```




**Example of BoxLayout class with X-AXIS**

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

public class BoxLayoutExample2 extends Frame {
 Button buttons[];

 public BoxLayoutExample2() {
   buttons = new Button [5];

   for (int i = 0;i<5;i++) {
     buttons[i] = new Button ("Button " + (i + 1));
     add (buttons[i]);
```

```
   }

setLayout (new BoxLayout(this, BoxLayout.X_AXIS));
setSize(400,400);   setVisible(true);
}

public static void main(String args[]){
BoxLayoutExample2 b=new BoxLayoutExample2();
}
}
```




### Java CardLayout

The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

### Constructors of CardLayout class

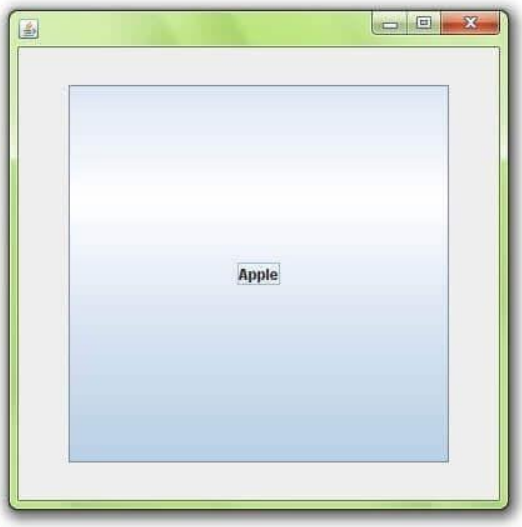
1. **CardLayout():** creates a card layout with zero horizontal and vertical gap.
2. **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.

### Commonly used methods of CardLayout class

- **public void next(Container parent):** is used to flip to the next card of the given container.
- **public void previous(Container parent):** is used to flip to the previous card of the given container.
- **public void first(Container parent):** is used to flip to the first card of the given container.
- **public void last(Container parent):** is used to flip to the last card of the given container.
- **public void show(Container parent, String name):** is used to flip to the specified card with the given name.

### Example of CardLayout class

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

import javax.swing.*;

public class CardLayoutExample extends JFrame implements ActionListener{
CardLayout card;
JButton b1,b2,b3;
Container c;
    CardLayoutExample(){

        c=getContentPane();
        card=new CardLayout(40,30);
//create CardLayout object with 40 hor space and 30 ver space
c.setLayout(card);

        b1=new JButton("Apple");
b2=new JButton("Boy");          b3=new
JButton("Cat");
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);

        c.add("a",b1);c.add("b",b2);c.add("c",b3);

    }
    public void actionPerformed(ActionEvent e) {
card.next(c);
    }

    public static void main(String[] args) {
        CardLayoutExample cl=new CardLayoutExample();
cl.setSize(400,400);
        cl.setVisible(true);
        cl.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```




**For all layout managers tutorial**
https://www.javatpoint.com/java-layout-manager

## 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:

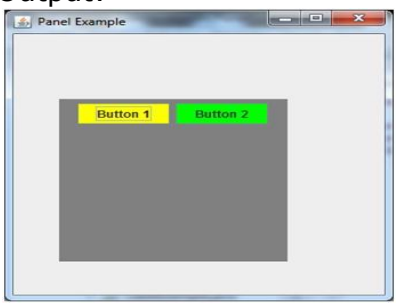
| Constructor | Description |
|---|---|
| JPanel() | It is used to create a new JPanel with a double buffer and a flow layout. |
| JPanel(boolean isDoubleBuffered) | It is used to create a new JPanel with FlowLayout and the specified buffering strategy. |
| | |

| | |
|---|---|
| JPanel(LayoutManager layout) | It is used to create a new JPanel with the specified layout manager. |

**Java JPanel Example**

```
import java.awt.*;  import
javax.swing.*;  public
class PanelExample {
    PanelExample()
      {
      JFrame f= new JFrame("Panel Example");

      JPanel panel=new JPanel();
panel.setBounds(40,80,200,200);
panel.setBackground(Color.gray);

     JButton b1=new JButton("Button 1");
b1.setBounds(50,100,80,30);
      b1.setBackground(Color.yellow);

      JButton b2=new JButton("Button 2");
b2.setBounds(100,100,80,30);          b2.setBackground(Color.green);

     panel.add(b1); panel.add(b2);
f.add(panel);
            f.setSize(400,400);
            f.setLayout(null);
            f.setVisible(true);
}
      public static void main(String args[])
      {
      new PanelExample();
      }
  }
```

Output:




**Java JTextField**

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

### JTextField class declaration

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

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

### Commonly used Constructors:

| Constructor | Description |
|---|---|
| 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:

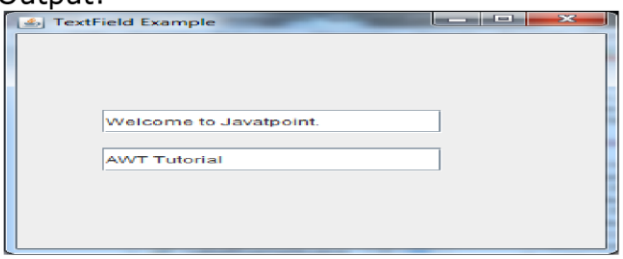
| Methods | Description |
|---|---|
| void addActionListener(ActionListener l) | It is used to add the specified action listener to receive action events from this textfield. |

| | |
|---|---|
| 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

```
import javax.swing.*;
class TextFieldExample
{
public static void main(String args[])
    {
    JFrame f= new JFrame("TextField Example");

    JTextField t1,t2;
    t1=new JTextField("Welcome to Javatpoint.");
    t1.setBounds(50,100, 200,30);

    t2=new JTextField("AWT Tutorial");
    t2.setBounds(50,150, 200,30);

    f.add(t1); f.add(t2);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
    }
    }
```

Output:




## Java JPasswordField

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

### JPasswordField class declaration

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

1. **public class** JPasswordField **extends** JTextField

**Commonly used Constructors:**

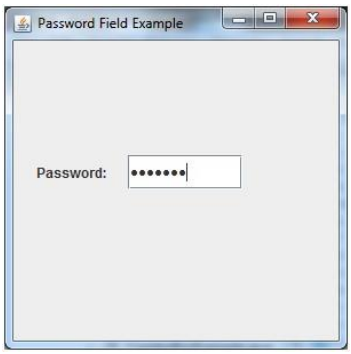
| Constructor | Description |
|---|---|
| JPasswordField() | Constructs a new JPasswordField, with a default document, null starting text string, and 0 column width. |
| JPasswordField(int columns) | Constructs a new empty JPasswordField with the specified number of columns. |
| JPasswordField(String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

**Java JPasswordField Example**

```
public class PasswordFieldExample {
   public static void main(String[] args) {
   JFrame f=new JFrame("Password Field Example");

  JPasswordField value = new JPasswordField();

  JLabel l1=new JLabel("Password:");

        l1.setBounds(20,100, 80,30);
         value.setBounds(100,100,100,30);
            f.add(value);  f.add(l1);
            f.setSize(300,300);
            f.setLayout(null);
            f.setVisible(true);
}
}
```

Output:

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




## Java JTextArea

The object of a JTextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits JTextComponent class

### JTextArea class declaration

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

1. **public class** JTextArea **extends** JTextComponent **Commonly used Constructors:**

| Constructor | Description |
|---|---|
| JTextArea() | Creates a text area that displays no text initially. |
| JTextArea(String s) | Creates a text area that displays specified text initially. |
| JTextArea(int row, int column) | Creates a text area with the specified number of rows and columns that displays no text initially. |
| JTextArea(String s, int row, int column) | Creates a text area with the specified number of rows and columns that displays specified text. |

### Commonly used Methods:

| Methods | Description |
|---|---|

| | |
|---|---|
| void setRows(int rows) | It is used to set specified number of rows. |
| void setColumns(int cols) | It is used to set specified number of columns. |
| | |
| void setFont(Font f) | It is used to set the specified font. |
| void insert(String s, int position) | It is used to insert the specified text on the specified position. |
| void append(String s) | It is used to append the given text to the end of the document. |

**Java JTextArea Example**

```
import javax.swing.*;   public
class TextAreaExample
{
    TextAreaExample(){
      JFrame f= new JFrame();
      JTextArea area=new JTextArea("Welcome to javatpoint");
```

```
        area.setBounds(10,30, 200,200);
        f.add(area);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
public static void main(String args[])
    {
   new TextAreaExample();
    }}
```

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:**

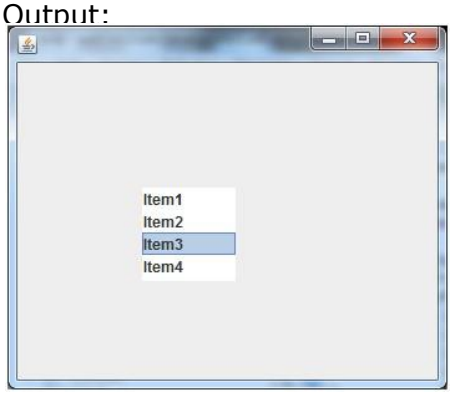
| Methods | Description |
|---|---|
| Void addListSelectionListener(ListSelectionListener listener) | It is used to add a listener to the list, to be notified each time a change to the selection occurs. |
| int getSelectedIndex() | It is used to return the smallest selected cell index. |
| ListModel 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**

```
import javax.swing.*;   public
class ListExample
{
    ListExample(){
       JFrame f= new JFrame();
       DefaultListModel<String> l1 = new DefaultListModel<>();
l1.addElement("Item1");          l1.addElement("Item2");
l1.addElement("Item3");          l1.addElement("Item4");
```

```
        JList<String> list = new JList<>(l1);
list.setBounds(100,100, 75,75);          f.add(list);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
public static void main(String args[])
   {
  new ListExample();
   }}
```

Output:




## 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, Act ionListener, 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. |

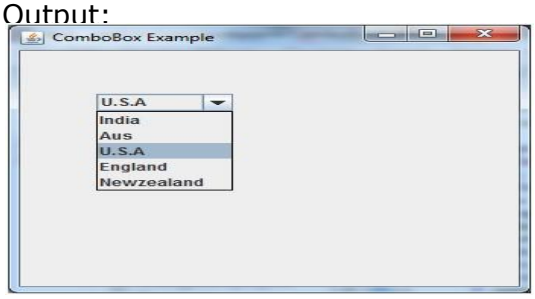
**Commonly used Methods:**

| 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

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

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:

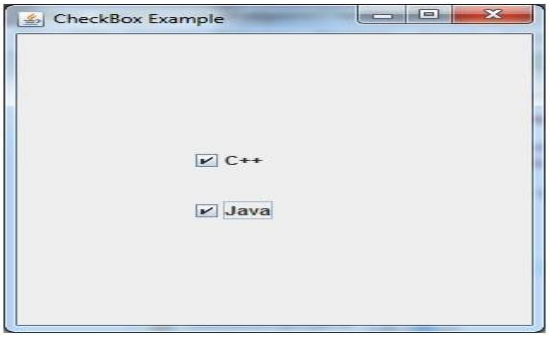
| Methods | Description |
|---|---|
| AccessibleContext getAccessibleContext() | It is used to get the AccessibleContext associated with this JCheckBox. |
| protected String paramString() | It returns a string representation of this JCheckBox. |

### Java JCheckBox Example

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

       JCheckBox checkBox2 = new JCheckBox("Java", true);
checkBox2.setBounds(100,150, 50,50);
```

```
        f.add(checkBox1);
        f.add(checkBox2);

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

Output:




### Java JRadioButton

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

### JRadioButton class declaration

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

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

### Commonly used Constructors:

| Constructor | Description |
|---|---|
| JRadioButton() | Creates an unselected radio button with no text. |
| JRadioButton(String s) | Creates an unselected radio button with specified text. |

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

**Commonly used Methods:**

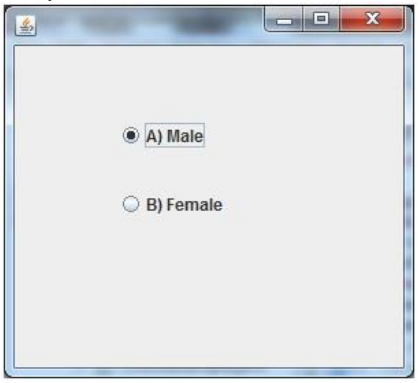
| Methods | Description |
|---|---|
| void setText(String s) | It is used to set specified text on button. |
| String getText() | It is used to return the text of the button. |
| void setEnabled(boolean b) | It is used to enable or disable the button. |
| void setIcon(Icon b) | It is used to set the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

**Java JRadioButton Example**

```
import javax.swing.*;
public class RadioButtonExample {
JFrame f;
RadioButtonExample(){     f=new
JFrame();

JRadioButton r1=new JRadioButton("A) Male");
JRadioButton r2=new JRadioButton("B) Female");

r1.setBounds(75,50,100,30);     r2.setBounds(75,100,100,30);

ButtonGroup bg=new ButtonGroup();

bg.add(r1);bg.add(r2);
f.add(r1);f.add(r2);

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

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

Output:




## Java JScrollPane

A JscrollPane is used to make scrollable view of a component. When screen size is limited, we use a scroll pane to display a large component or a component whose size can change dynamically.

## Constructors

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

## Useful Methods

| Modifier | Method | Description |
|---|---|---|
| void | setColumnHeaderView(Component) | It sets the column header for the scroll pane. |
| void | setRowHeaderView(Component) | It sets the row header for the scroll pane. |
| void | setCorner(String, Component) | It sets or gets the specified corner. |

**Constructor**

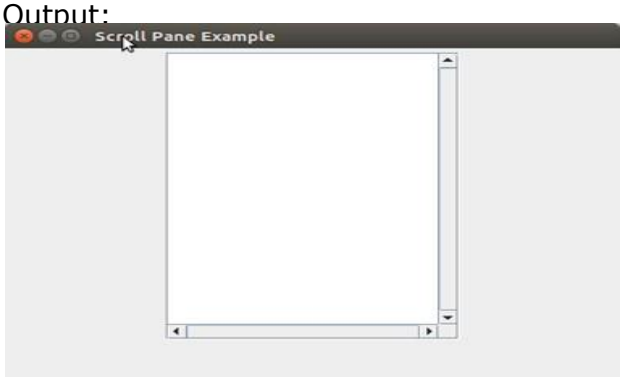
| | | |
|---|---|---|
| Component | getCorner(String) | The int parameter specifies which corner and must be one of the following constants defined in ScrollPaneConstants: UPPER_LEFT_CORNER, UPPER_RIGHT_CORNER, LOWER_LEFT_CORNER, LOWER_RIGHT_CORNER, LOWER_LEADING_CORNER, LOWER_TRAILING_CORNER, UPPER_LEADING_CORNER, UPPER_TRAILING_CORNER. |
| void | setViewportView(Component) | Set the scroll pane's client. |

## JScrollPane Example

```
import java.awt.FlowLayout;
import javax.swing.JFrame;   import
javax.swing.JScrollPane;   import
javax.swing.JtextArea;

public class JScrollPaneExample {      private
static final long serialVersionUID = 1L;

   private static void createAndShowGUI() {

      // Create and set up the window.
      final JFrame frame = new JFrame("Scroll Pane Example");

      // Display the window.          frame.setSize(500, 500);
frame.setVisible(true);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      //          set          flow          layout          for          the          frame
frame.getContentPane().setLayout(new FlowLayout());

      JTextArea textArea = new JTextArea(20, 20);
      JScrollPane scrollableTextArea = new JScrollPane(textArea);


scrollableTextArea.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR
_ALWAYS);
```

```
        scrollableTextArea.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALW
AYS);

        frame.getContentPane().add(scrollableTextArea);
    }
    public static void main(String[] args) {


        javax.swing.SwingUtilities.invokeLater(new Runnable() {

            public void run() {
                createAndShowGUI();
            }
        });
    }
}
```

Output:




## Java JSplitPane

JSplitPane is used to divide two components. The two components are divided based on the look and feel implementation, and they can be resized by the user. If the minimum size of the two components is greater than the size of the split pane, the divider will not allow you to resize it.

The two components in a split pane can be aligned left to right using JSplitPane.HORIZONTAL_SPLIT, or top to bottom using JSplitPane.VERTICAL_SPLIT. When the user is resizing the components the minimum size of the components is used to determine the maximum/minimum position the components can be set to.

### Nested Class

| Modifier and Type | Class | Description |
|---|---|---|

| | | |
|---|---|---|
| protected class | JSplitPane.AccessibleJSplitPane | This class implements accessibility support for the JsplitPane class. |

### Useful Fields

| Modifier and Type | Field | Description |
|---|---|---|
| static String | BOTTOM | It use to add a Component below the other Component. |
| static String | CONTINUOUS_LAYOUT_PROPERTY | Bound property name for continuousLayout. |
| static String | DIVIDER | It uses to add a Component that will represent the divider. |
| static int | HORIZONTAL_SPLIT | Horizontal split indicates the Components are split along the x axis. |
| protected int | lastDividerLocation | Previous location of the split pane. |
| protected Component | leftComponent | The left or top component. |
| static int | VERTICAL_SPLIT | Vertical split indicates the Components are split along the y axis. |
| protected Component | rightComponent | The right or bottom component. |
| protected int | orientation | How the views are split. |

### Constructors

| Constructor | Description |
|---|---|
| JSplitPane() | It creates a new JsplitPane configured to arrange the child components side-by-side horizontally, using two buttons for the components. |
| JSplitPane(int newOrientation) | It creates a new JsplitPane configured with the specified orientation. |

| | |
|---|---|
| JSplitPane(int newOrientation, boolean newContinuousLayout) | It creates a new JsplitPane with the specified orientation and redrawing style. |
| JSplitPane(int newOrientation, boolean newContinuousLayout, Component newLeftComponent, Component newRightComponent) | It creates a new JsplitPane with the specified orientation and redrawing style, and with the specified components. |
| JSplitPane(int newOrientation, Component newLeftComponent, Component newRightComponent) | It creates a new JsplitPane with the specified orientation and the specified components. |

**Useful Methods**

| Modifier and Type | Method | Description |
|---|---|---|
| protected void | addImpl(Component comp, Object constraints, int index) | It cdds the specified component to this split pane. |
| AccessibleContext | getAccessibleContext() | It gets the AccessibleContext associated with this JSplitPane. |
| int | getDividerLocation() | It returns the last value passed to setDividerLocation. |
| int | getDividerSize() | It returns the size of the divider. |

| | | |
|---|---|---|
| Component | getBottomComponent() | It returns the component below, or to the right of the divider. |
| Component | getRightComponent() | It returns the component to the right (or below) the divider. |
| SplitPaneUI | getUI() | It returns the SplitPaneUI that is providing the current look and feel. |
| boolean | isContinuousLayout() | It gets the continuousLayout property. |
| boolean | isOneTouchExpandable() | It gets the oneTouchExpandable property. |
| void | setOrientation(int orientation) | It gets the orientation, or how the splitter is divided. |

**JSplitPane Example**

```
import java.awt.FlowLayout;   import java.awt.Panel;   import
javax.swing.JComboBox;   import javax.swing.JFrame;
import javax.swing.JSplitPane;   public class
JSplitPaneExample {       private static void createAndShow()
{          // Create and set up the window.          final JFrame
frame = new JFrame("JSplitPane Example");          // Display
the window.          frame.setSize(300, 300);
frame.setVisible(true);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       // set flow layout for the frame
       frame.getContentPane().setLayout(new FlowLayout());
       String[] option1 = { "A","B","C","D","E" };
       JComboBox box1 = new JComboBox(option1);
String[] option2 = {"1","2","3","4","5"};
       JComboBox box2 = new JComboBox(option2);
Panel panel1 = new Panel();          panel1.add(box1);
       Panel panel2 = new Panel();
panel2.add(box2);
       JSplitPane splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, panel1, panel
2);
       // JSplitPane splitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
       // panel1, panel2);
frame.getContentPane().add(splitPane);
   }
   public static void main(String[] args) {
       // Schedule a job for the event-dispatching thread:
       // creating and showing this application's GUI.
javax.swing.SwingUtilities.invokeLater(new Runnable() {
public void run() {                createAndShow();
          }
       });
   }
}
```

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.

### JTabbedPane class declaration

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

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

### Commonly used Constructors:

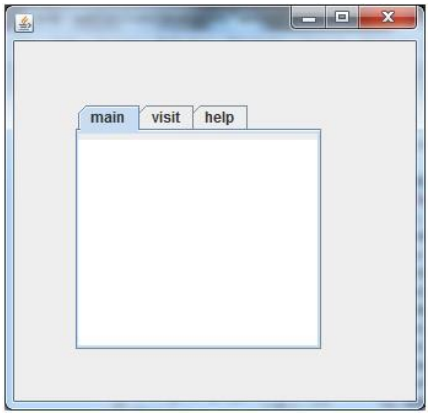
| Constructor | Description |
|---|---|
| JTabbedPane() | Creates an empty TabbedPane with a default tab placement of JTabbedPane.Top. |
| JTabbedPane(int tabPlacement) | Creates an empty TabbedPane with a specified tab placement. |
| JTabbedPane(int tabPlacement, int tabLayoutPolicy) | Creates an empty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

### Java JTabbedPane Example

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

```
TabbedPaneExample(){
f=new JFrame();
    JTextArea ta=new JTextArea(200,200);

   JPanel p1=new JPanel();
p1.add(ta);
    JPanel p2=new JPanel();
    JPanel p3=new JPanel();

   JTabbedPane tp=new
JTabbedPane();
tp.setBounds(50,50,200,200);
tp.add("main",p1);
tp.add("visit",p2);
tp.add("help",p3);        f.add(tp);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);   }   public static
void main(String[] args) {       new
TabbedPaneExample();
}}
```

Output:




## Java JOptionPane

The JOptionPane class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box. These dialog boxes are used to display information or get input from the user. The JOptionPane class inherits JComponent class.

### JOptionPane class declaration

1. **public class** JOptionPane **extends** JComponent **implements** Accessible

## Common Constructors of JOptionPane class

| Constructor | Description |
|---|---|
| JOptionPane() | It is used to create a JOptionPane with a test message. |
| JOptionPane(Object message) | It is used to create an instance of JOptionPane to display a message. |
| JOptionPane(Object message, int messageType | It is used to create an instance of JOptionPane to display a message with specified message type and default options. |

## Common Methods of JOptionPane class

| Methods | Description |
|---|---|
| JDialog createDialog(String title) | It is used to create and return a new parentless JDialog with the specified title. |
| static void showMessageDialog(Component parentComponent, Object message) | It is used to create an informationmessage dialog titled "Message". |
| static void showMessageDialog(Component parentComponent, Object message, String title, int messageType) | It is used to create a message dialog with given title and messageType. |
| static int showConfirmDialog(Component parentComponent, Object message) | It is used to create a dialog with the options Yes, No and Cancel; with the title, Select an Option. |
| static String showInputDialog(Component parentComponent, Object message) | It is used to show a question-message dialog requesting input from the user parented to parentComponent. |

| | |
|---|---|
| void setInputValue(Object newValue) | It is used to set the input value that was selected or input by the user. |

### Java JOptionPane Example: showMessageDialog()

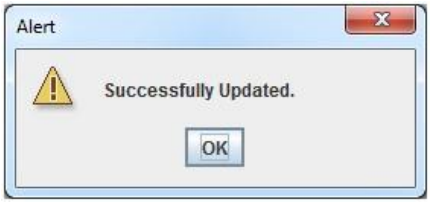
```
import javax.swing.*;
public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
f=new JFrame();
   JOptionPane.showMessageDialog(f,"Hello, Welcome to Javatpoint.");
}
public static void main(String[] args) {
   new OptionPaneExample();
}
}
```

Output:


### Java JOptionPane Example: showMessageDialog()

```
import javax.swing.*;
public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
f=new JFrame();
   JOptionPane.showMessageDialog(f,"Successfully Updated.","Alert",JOptionPane.WARNING_MESSAGE);

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

}
}
```

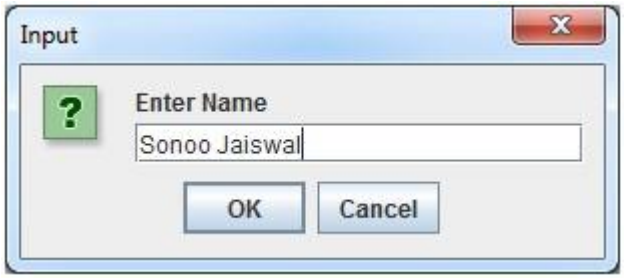
Output:




## Java JOptionPane Example: showInputDialog()

```
import javax.swing.*;
public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
f=new JFrame();
    String name=JOptionPane.showInputDialog(f,"Enter Name");
}   public static void main(String[]
args) {       new OptionPaneExample();
}
}
```

Output:




## Java JOptionPane Example: showConfirmDialog()

```
import javax.swing.*;
import java.awt.event.*;
public class OptionPaneExample extends WindowAdapter{
JFrame f;
OptionPaneExample(){
    f=new JFrame();
    f.addWindowListener(this);
    f.setSize(300, 300);
    f.setLayout(null);
    f.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    f.setVisible(true);
}
public void windowClosing(WindowEvent e) {
    int a=JOptionPane.showConfirmDialog(f,"Are you sure?");
if(a==JOptionPane.YES_OPTION){
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
public static void main(String[] args) {
    new  OptionPaneExample();
}
}
```

Output:


### Java JProgressBar

The JProgressBar class is used to display the progress of the task. It inherits JComponent class.

### JProgressBar class declaration

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

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

**Commonly used Constructors:**

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

**Commonly used Methods:**

| Method | Description |
|---|---|
| void setStringPainted(boolean b) | It is used to determine whether string should be displayed. |
| void setString(String s) | It is used to set value to the progress string. |

| | |
|---|---|
| void setOrientation(int orientation) | It is used to set the orientation, it may be either vertical or horizontal by using SwingConstants.VERTICAL and SwingConstants.HORIZONTAL constants. |
| void setValue(int value) | It is used to set the current value on the progress bar. |

### Java JProgressBar Example

```
import javax.swing.*;
public class ProgressBarExample extends JFrame{
JProgressBar jb;    int i=0,num=0;
ProgressBarExample(){
jb=new
JProgressBar(0,2000);
jb.setBounds(40,40,160,30);
jb.setValue(0);
jb.setStringPainted(true);
add(jb);    setSize(250,150);
setLayout(null);
}
public void iterate(){    while(i<=2000){
jb.setValue(i);     i=i+20;
try{Thread.sleep(150);}catch(Exception e){}
}
}
public static void main(String[] args) {
   ProgressBarExample m=new ProgressBarExample();
m.setVisible(true);
   m.iterate();
}
```

```
}
```

Output: