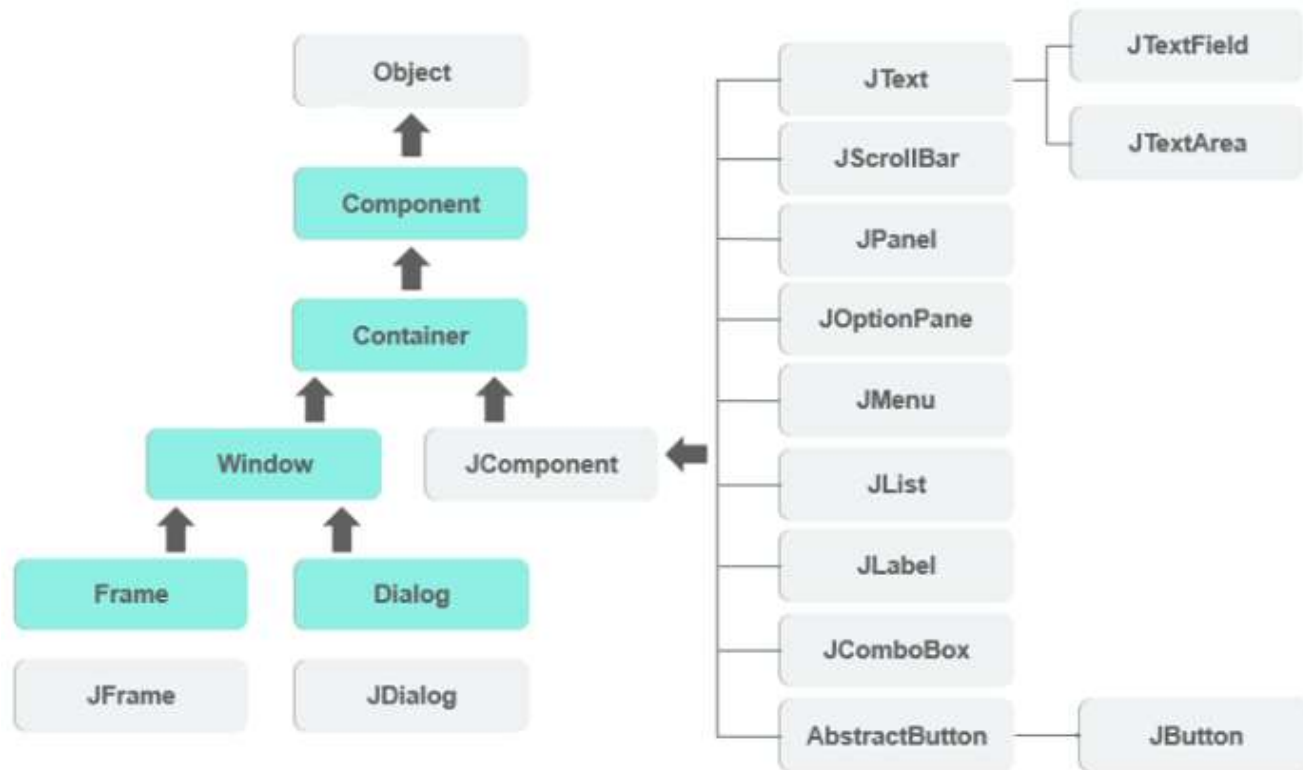


JComponents

JComponents





Java Swing Class Hierarchy



- All the components in swing like JButton, JComboBox, JList, JLabel are inherited from the **JComponent** class which can be added to the container classes.
- **Containers** are the windows like frame and dialog boxes. Basic swing components are the building blocks of any gui application. Methods like setLayout override the default layout in each container.
- Containers like JFrame and JDialog can only add a component to itself.

Java Swing Class Hierarchy



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.

JFrame



JFrame Constructor

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



JFrame 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(Container contentPane)	It sets the contentPane property
static void	setDefaultLookAndFeelDecorated(boolean defaultLookAndFeelDecorated)	Provides a hint as to whether or not newly created JFrames should have their Window decorations (such as borders, widgets to close the window, title...) provided by the current look and feel.
void	setIconImage(Image image)	It sets the image to be displayed as the icon for this window.
void	setJMenuBar(JMenuBar menubar)	It sets the menubar for this frame.
void	setLayeredPane(JLayeredPane layeredPane)	It sets the layeredPane property.
JRootPane	getRootPane()	It returns the rootPane object for this frame.
TransferHandler	getTransferHandler()	It gets the transferHandler property.



Sample Code

```
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

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



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.



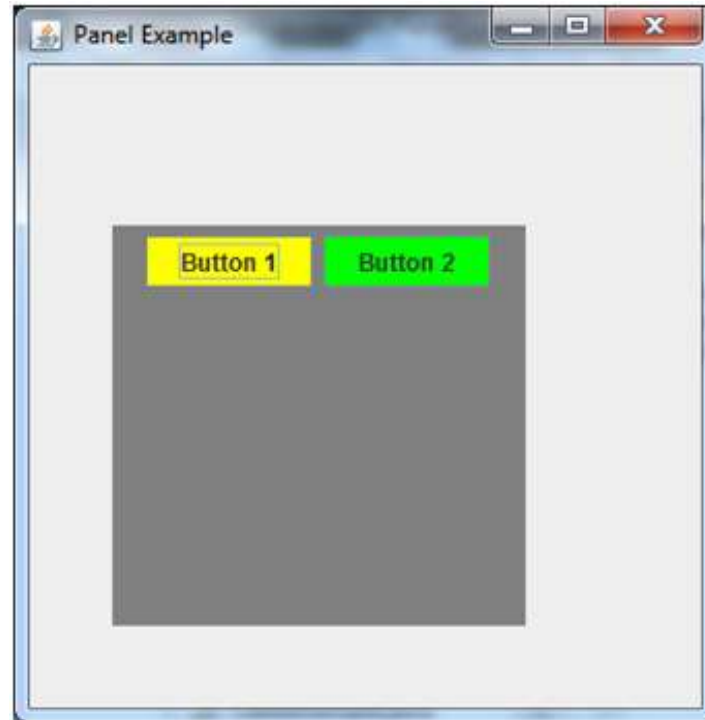
JPanel

Sample Code

```
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();
    }
}
```



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

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

JComponent



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



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.

Commonly used Constructors:

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.

JButton



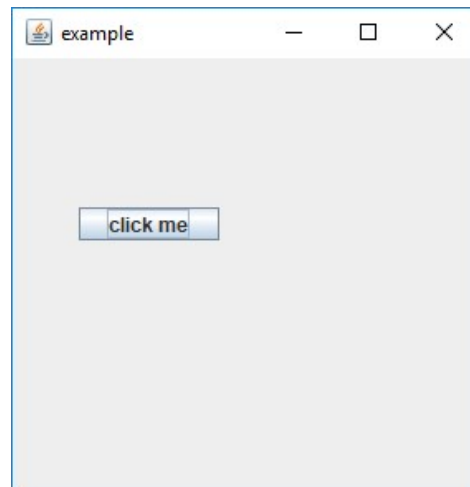
Commonly used Methods of AbstractButton class:

1. **void setText(String s)** - It is used to set specified text on button
2. **String getText()** - It is used to return the text of the button.
3. **void setEnabled(boolean b)** - It is used to enable or disable the button.
4. **void setIcon(Icon b)** - It is used to set the specified Icon on the button.
5. **Icon getIcon()** - It is used to get the Icon of the button.
6. **void setMnemonic(int a)** - It is used to set the mnemonic on the button.
7. **void addActionListener(ActionListener a)** - It is used to add the action listener to this object.

JButton



```
import javax.swing.*;  
public class example{  
    public static void main(String args[]) {  
        JFrame a = new JFrame("example");  
        JButton b = new JButton("click me");  
        b.setBounds(40,90,85,20);  
        a.add(b);  
        a.setSize(300,300);  
        a.setLayout(null);  
        a.setVisible(true);  
    }  
}
```



Sample code



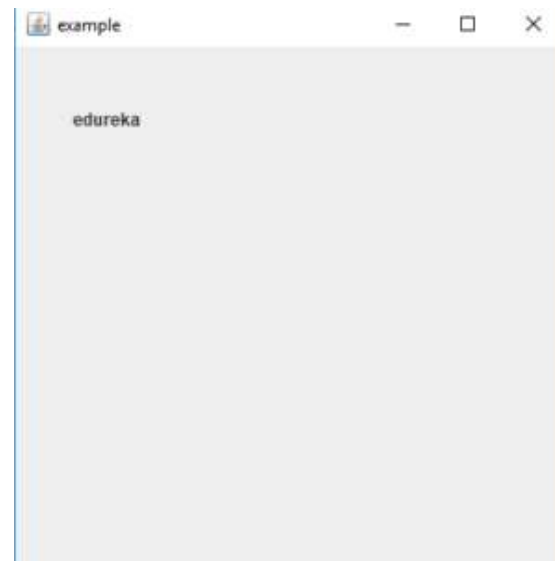
The object of **JLabel** class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly. It inherits **JComponent** class.

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.

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.




```
import javax.swing.*;
public class Example{
public static void main(String args[])
{
JFrame a = new JFrame("example");
JLabel b1;
b1 = new JLabel("edureka");
b1.setBounds(40,40,90,20);
a.add(b1);
a.setSize(400,400);
a.setLayout(null);
a.setVisible(true);
}
}
```



Sample code



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

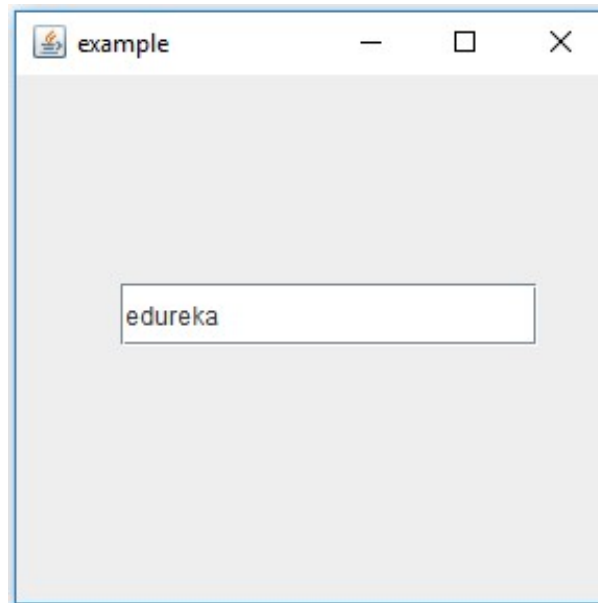
Constructor	Description
JTextField()	Creates a new TextField
JTextField(String text)	Creates a new TextField initialized with the specified text.
JTextField(String text, int columns)	Creates a new TextField initialized with the specified text and columns.
JTextField(int columns)	Creates a new empty TextField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

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.

JTextField



```
import javax.swing.*;  
public class example{  
    public static void main(String args[]) {  
        JFrame a = new JFrame("example");  
        JTextField b = new JTextField("edureka");  
        b.setBounds(50,100,200,30);  
        a.add(b);  
        a.setSize(300,300);  
        a.setLayout(null);  
        a.setVisible(true);  
    }  
}
```



Sample code



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.

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.

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.

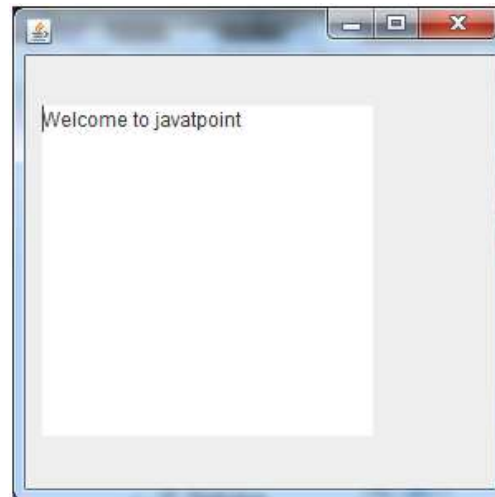
JTextArea




```
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();
    }
}
```



Sample code



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.

Constructor	Description
JCheckBox()	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.

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.

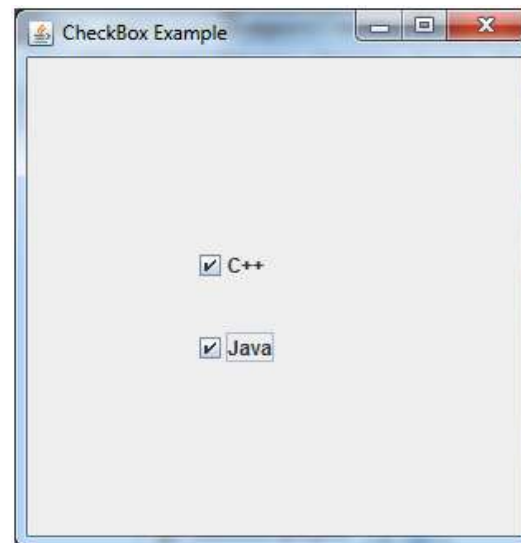
JCheckBox



```
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();
    }
}
```



Sample code



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.

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.

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

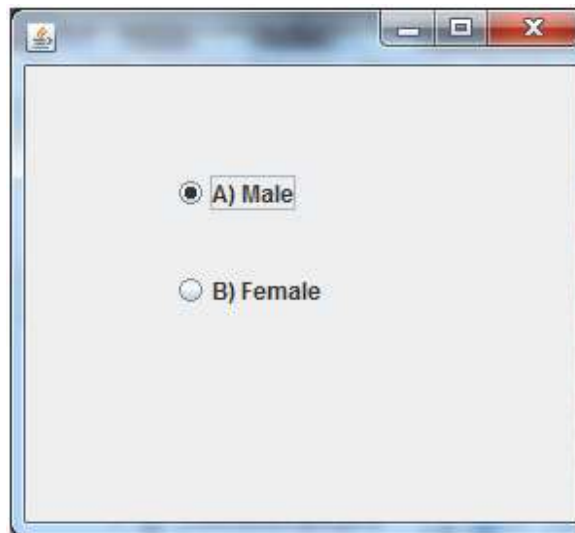



```
import javax.swing.*;

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();
    }
}
```



Sample code



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.

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 .

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 .

JComboBox



```

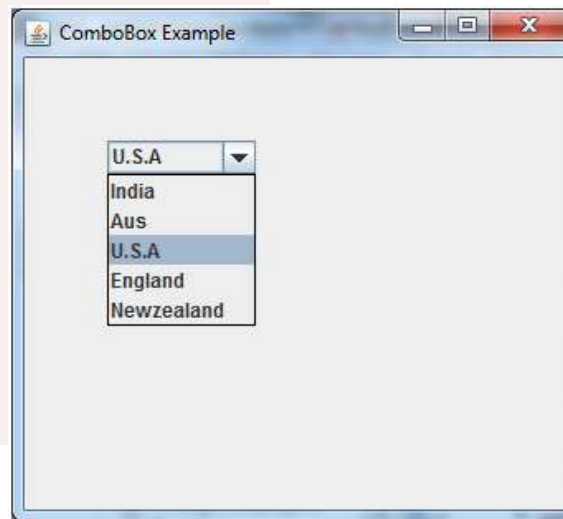
import javax.swing.*.*;

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();
    }
}

```



Sample code



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

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

JTable




```

import javax.swing.*.*;

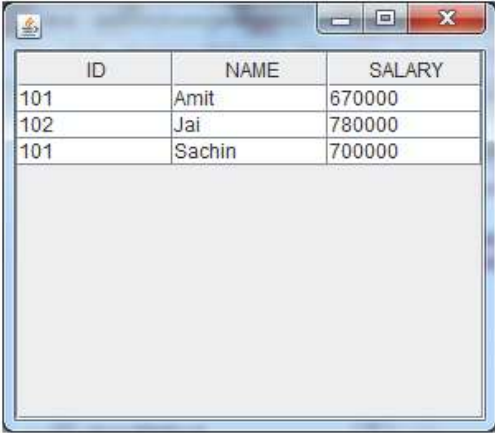
public class TableExample {
    JFrame f;

    TableExample(){
        f=new JFrame();
        String data[][]={ {"101","Amit","670000"},
                           {"102","Jai","780000"},
                           {"101","Sachin","700000"} };

        String column[]={"ID","NAME","SALARY"};
        JTable jt=new JTable(data,column);
        jt.setBounds(30,40,200,300);
        JScrollPane sp=new JScrollPane(jt);
        f.add(sp);
        f.setSize(300,400);
        f.setVisible(true);
    }

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

```



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

Sample code



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.

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

Methods	Description
<code>Void addListSelectionListener(ListSelectionListener listener)</code>	It is used to add a listener to the list, to be notified each time a change to the selection occurs.
<code>int getSelectedIndex()</code>	It is used to return the smallest selected cell index.
<code>ListModel getModel()</code>	It is used to return the data model that holds a list of items displayed by the JList component.
<code>void setListData(Object[] listData)</code>	It is used to create a read-only ListModel from an array of objects.

JList



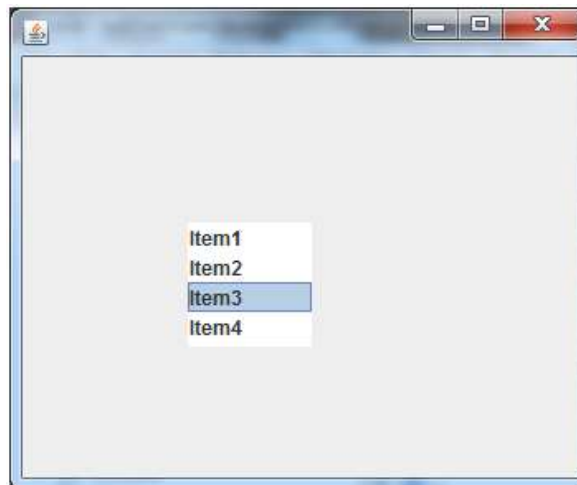
```

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();
    }
}

```



Sample Code



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.

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.

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 information-message 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.

JOptionPane



Message Dialog Box

```
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();
    }
}
```



Confirm Dialog Box

```
import javax.swing.*;

public class OptionPaneExample {
    JFrame f;

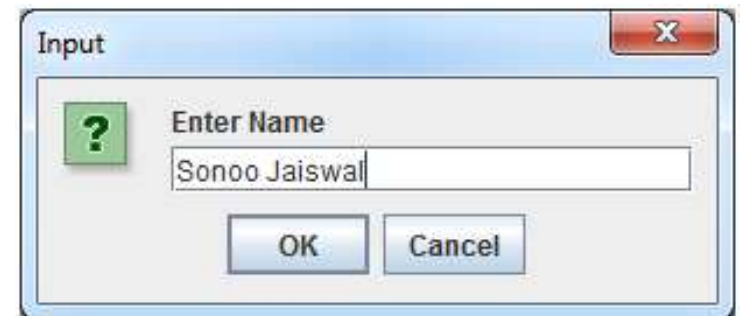
    OptionPaneExample(){
        f=new JFrame();
        JOptionPane.showMessageDialog(f,"Successfully Updated.", "Alert",JOptionPane.WARNING_MESSAGE);
    }

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



Input Dialog Box

```
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();
    }
}
```



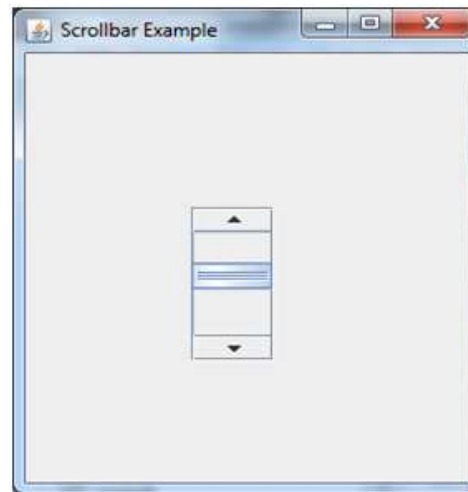
The object of **JScrollbar** class is used to add horizontal and vertical scrollbar. It is an implementation of a scrollbar. It inherits **JComponent** class.

Constructor	Description
JScrollbar()	Creates a vertical scrollbar with the initial values.
JScrollbar(int orientation)	Creates a scrollbar with the specified orientation and the initial values.
JScrollbar(int orientation, int value, int extent, int min, int max)	Creates a scrollbar with the specified orientation, value, extent, minimum, and maximum.

JScrollbar



```
import javax.swing.*.*;
class ScrollBarExample
{
    ScrollBarExample(){
        JFrame f= new JFrame("Scrollbar Example");
        JScrollBar s=new JScrollBar();
        s.setBounds(100,100, 50,100);
        f.add(s);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new ScrollBarExample();
    }
}
```



Sample Code



- 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, JMenu and JMenuItem



- public class JMenuBar extends JComponent implements MenuElement, Accessible
- public class JMenu extends JMenuitem implements MenuElement, Accessible
- public class JMenuitem extends AbstractButton implements Accessible, MenuElement

JMenuBar, JMenu and JMenuitem



```
class MenuExample
```

```
{
```

```
    JMenu menu, submenu;
```

```
    JMenuItem i1, i2, i3, i4, i5;
```

```
    MenuExample(){
```

```
        JFrame f= new JFrame("Menu and MenuItem Example");
```

```
        JMenuBar mb=new JMenuBar();
```

```
        menu=new JMenu("Menu");
```

```
        submenu=new JMenu("Sub Menu");
```

```
        i1=new JMenuItem("Item 1");
```

```
        i2=new JMenuItem("Item 2");
```

```
        i3=new JMenuItem("Item 3");
```

```
        i4=new JMenuItem("Item 4");
```

```
        i5=new JMenuItem("Item 5");
```

```
        menu.add(i1); menu.add(i2); menu.add(i3);
```

```
        submenu.add(i4); submenu.add(i5);
```

```
        menu.add(submenu);
```

```
        mb.add(menu);
```

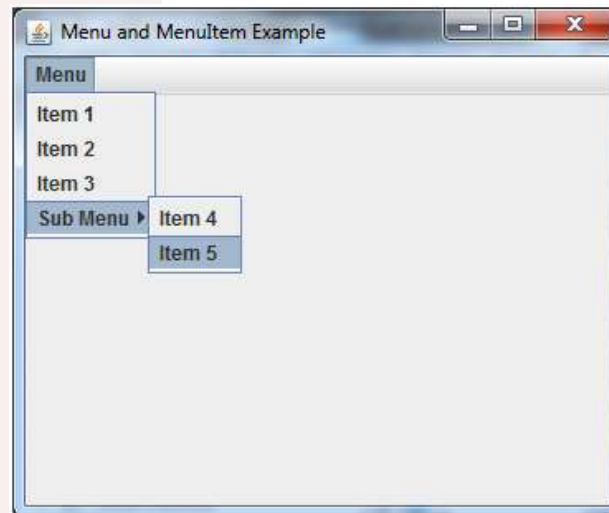
```
        f.setJMenuBar(mb);
```

```
        f.setSize(400,400);
```

```
        f.setLayout(null);
```

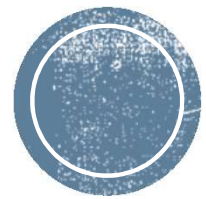
```
        f.setVisible(true);
```

```
}
```



Sample Code





Event Handling

Intro

- **Change in the state of an object** is known as Event, i.e., event describes the change in the state of the source.
- Events are generated as a result of user interaction with the graphical user interface components.
- For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from the list, and scrolling the page are the activities that causes an event to occur.

What is an Event?



The events can be broadly classified into **two** categories:

- **Foreground Events** – These events require direct interaction of the user. They are generated as consequences of a person interacting with the graphical components in the Graphical User Interface.

For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page, etc.

- **Background Events** – These events require the interaction of the end user. Operating system interrupts, hardware or software failure, timer expiration, and operation completion are some examples of background events

Types of Event



Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has a code which is known as an **event handler**, that is executed when an event occurs.

Java uses the **Delegation Event Model** to handle the events. This model defines the standard mechanism to generate and handle the events.

The Delegation Event Model has the following key participants.

- **Source** – The source is an **object on which the event occurs**. Source is responsible for providing information of the occurred event to its handler. Java provides us with classes for the source object.
- **Listener** – It is also known as **event handler**. The listener is responsible for generating a response to an event. From the point of view of Java implementation, the listener is also an object. The listener waits till it receives an event. Once the event is received, the listener processes the event and then returns.

What is Event Handling?



1. The user clicks the button and the event is generated.
2. The object of concerned event class is created automatically and information about the source and the event get populated within the same object.
3. Event object is forwarded to the method of the registered listener class.
4. The method is gets executed and returns.

Steps Involved in Event Handling



ActionEvent	generated when button is pressed, menu-item is selected, list-item is double clicked	ActionListener
MouseEvent	generated when mouse is dragged, moved, clicked, pressed or released and also when it enters or exits a component	MouseListener
KeyEvent	generated when input is received from keyboard	KeyListener
ItemEvent	generated when check-box or list item is clicked	ItemListener
TextEvent	generated when value of textarea or textfield is changed	TextListener

Important Event Classes and Interface



MouseEvent	generated when mouse wheel is moved	MouseListener
WindowEvent	generated when window is activated, deactivated, deiconified, iconified, opened or closed	WindowListener
ComponentEvent	generated when component is hidden, moved, resized or set visible	ComponentEventListener
ContainerEvent	generated when component is added or removed from container	ContainerListener
AdjustmentEvent	generated when scroll bar is manipulated	AdjustmentListener
FocusEvent	generated when component gains or loses keyboard focus	FocusListener

Important Event Classes and Interface





Next – Event Handling

