SWING - JLIST CLASS

Introduction

The class **JList** is a component which displays a list of objects and allows the user to select one or more items. A separate model, ListModel, maintains the contents of the list.

Class declaration

Following is the declaration for **javax.swing.JList** class:

```
public class JList
   extends JComponent
      implements Scrollable, Accessible
```

Field

Following are the fields for **javax.swing.JList** class −

- **static int HORIZONTAL_WRAP** − Indicates a "newspaper style" layout with cells flowing horizontally then vertically.
- **static int VERTICAL** − Indicates a vertical layout of cells, in a single column; the default layout.
- **static int VERTICAL_WRAP** − Indicates a "newspaper style" layout with cells flowing vertically then horizontally.

Class constructors

S.N.	Constructor & Description
1	**JList** Constructs a JList with an empty, read-only, model.
2	**JList***ListModeldataModel* Constructs a JList that displays elements from the specified, non-null, model.
3	**JList***Object*[]*listData* Constructs a JList that displays the elements in the specified array.
4	**JList***Vector* < ? > *listData* Constructs a JList that displays the elements in the specified Vector.

Class methods

S.N.	Method & Description
1	**void addListSelectionListener***ListSelectionListenerlistener* Adds a listener to the list, to be notified each time a change to the selection occurs; the

preferred way of listening for selection state changes.

2 **void addSelectionInterval*intanchor, intlead***

Sets the selection to be the union of the specified interval with current selection.

3 **void clearSelection**

Clears the selection; after calling this method, isSelectionEmpty will return true.

4 **protected ListSelectionModel createSelectionModel**

Returns an instance of DefaultListSelectionModel; called during construction to initialize the list's selection model property.

5 **void ensureIndexIsVisible*intindex***

Scrolls the list within an enclosing viewport to make the specified cell completely visible.

6 **protected void fireSelectionValueChanged*intfirstIndex, intlastIndex, booleanisAdjusting***

Notifies ListSelectionListeners added directly to the list of selection changes made to the selection model.

7 **AccessibleContext getAccessibleContext**

Gets the AccessibleContext associated with this JList.

8 **int getAnchorSelectionIndex**

Returns the anchor selection index.

9 **Rectangle getCellBounds*intindex*0, *intindex*1**

Returns the bounding rectangle, in the list's coordinate system, for the range of cells specified by the two indices.

10 **ListCellRenderer getCellRenderer**

Returns the object responsible for painting list items.

11 **boolean getDragEnabled**

Returns whether or not automatic drag handling is enabled.

12 **JList.DropLocation getDropLocation**

Returns the location that this component should visually indicate as the drop location during a DnD operation over the component, or null if no location is to currently be shown.

13 **DropMode getDropMode**

Returns the drop mode for this component.

14 **int getFirstVisibleIndex**

Returns the smallest list index that is currently visible.

15 **int getFixedCellHeight**

Returns the value of the fixedCellHeight property.

16 **int getFixedCellWidth**

Returns the value of the fixedCellWidth property.

17 **int getLastVisibleIndex**

Returns the largest list index that is currently visible.

18 **int getLayoutOrientation**

Returns the layout orientation property for the list: VERTICAL if the layout is a single column of cells, VERTICAL_WRAP if the layout is "newspaper style" with the content flowing vertically then horizontally, or HORIZONTAL_WRAP if the layout is "newspaper style" with the content flowing horizontally then vertically.

19 **int getLeadSelectionIndex**

Returns the lead selection index.

20 **ListSelectionListener[] getListSelectionListeners**

Returns an array of all the ListSelectionListeners added to this JList by way of addListSelectionListener.

21 **int getMaxSelectionIndex**

Returns the largest selected cell index, or -1 if the selection is empty.

22 **int getMinSelectionIndex**

Returns the smallest selected cell index, or -1 if the selection is empty.

23 **ListModel getModel**

Returns the data model that holds the list of items displayed by the JList component.

24 **int getNextMatch***Stringprefix, intstartIndex, Position. Biasbias*

Returns the next list element whose toString value starts with the given prefix.

25 **Dimension getPreferredScrollableViewportSize**

Computes the size of viewport needed to display visibleRowCount rows.

26 **Object getPrototypeCellValue**

Returns the "prototypical" cell value -- a value used to calculate a fixed width and height for cells.

27 **int getScrollableBlockIncrement***RectanglevisibleRect, intorientation, intdirection*

Returns the distance to scroll to expose the next or previous block.

28 **boolean getScrollableTracksViewportHeight**

Returns true if this JList is displayed in a JViewport and the viewport is taller than the list's preferred height, or if the layout orientation is VERTICAL_WRAP and visibleRowCount <= 0; otherwise returns false.

29 **boolean getScrollableTracksViewportWidth**

Returns true if this JList is displayed in a JViewport and the viewport is wider than the list's preferred width, or if the layout orientation is HORIZONTAL_WRAP and visibleRowCount <= 0; otherwise returns false.

30 **int getScrollableUnitIncrement***RectanglevisibleRect, intorientation, intdirection*

Returns the distance to scroll to expose the next or previous row *forverticalscrolling* or column *forhorizontalscrolling*.

31 **int getSelectedIndex**

Returns the smallest selected cell index; the selection when only a single item is selected in the list.

32 **int[] getSelectedIndices**

Returns an array of all of the selected indices, in increasing order.

33 **Object getSelectedValue**

Returns the value for the smallest selected cell index; the selected value when only a single item is selected in the list.

34 **Object[] getSelectedValues**

Returns an array of all the selected values, in increasing order based on their indices in the list.

35 **Color getSelectionBackground**

Returns the color used to draw the background of selected items.

36 **Color getSelectionForeground**

Returns the color used to draw the foreground of selected items.

37 **int getSelectionMode**

Returns the current selection mode for the list.

38 **ListSelectionModel getSelectionModel**

Returns the current selection model.

39 **String getToolTipText***MouseEventevent*

Returns the tooltip text to be used for the given event.

40 **ListUI getUI**

Returns the ListUI, the look and feel object that renders this component.

41 **String getUIClassID**

Returns "ListUI", the UIDefaults key used to look up the name of the javax.swing.plaf.ListUI class that defines the look and feel for this component.

42 **boolean getValueIsAdjusting**

Returns the value of the selection model's isAdjusting property.

43 **int getVisibleRowCount**

Returns the value of the visibleRowCount property.

44 **Point indexToLocation***intindex*

Returns the origin of the specified item in the list's coordinate system.

45 **boolean isSelectedIndex***intindex*

Returns true if the specified index is selected, else false.

46 **boolean isSelectionEmpty**

Returns true if nothing is selected, else false.

47 **int locationToIndex***Pointlocation*

Returns the cell index closest to the given location in the list's coordinate system.

48 **protected String paramString**

Returns a String representation of this JList.

49 **void removeListSelectionListener***ListSelectionListenerlistener*

Removes a selection listener from the list.

50 **void removeSelectionInterval***intindex***0, *intindex***1**

Sets the selection to be the set difference of the specified interval and the current selection.

51 **void setCellRenderer***ListCellRenderercellRenderer*

Sets the delegate that is used to paint each cell in the list.

52 **void setDragEnabled***booleanb*

Turns on or off automatic drag handling.

53 **void setDropMode***DropModedropMode*

Sets the drop mode for this component.

54 **void setFixedCellHeight***intheight*

Sets a fixed value to be used for the height of every cell in the list.

55 **void setFixedCellWidth***intwidth*

Sets a fixed value to be used for the width of every cell in the list.

56 **void setLayoutOrientation***intlayoutOrientation*

Defines the way list cells are layed out.

57 **void setListData***Object***[]***listData*

Constructs a read-only ListModel from an array of objects, and calls setModel with this model.

58 **void setListData***Vector* **< ? >** *listData*

Constructs a read-only ListModel from a Vector and calls setModel with this model.

59 **void setModel***ListModelmodel*

Sets the model that represents the contents or "value" of the list, notifies property change listeners, and then clears the list's selection.

60 **void setPrototypeCellValue***ObjectprototypeCellValue*

Sets the prototypeCellValue property, and then *ifthenewvalueisnon* − *null*, computes the fixedCellWidth and fixedCellHeight properties by requesting the cell renderer component for the given value *andindex*0 from the cell renderer, and using that component's preferred size.

61 **void setSelectedIndex***intindex*

Selects a single cell.

62 **void setSelectedIndices***int***[]***indices*

Changes the selection to be the set of indices specified by the given array.

63 **void setSelectedValue***ObjectanObject, booleanshouldScroll*

Selects the specified object from the list.

64 **void setSelectionBackground***ColorselectionBackground*

Sets the color used to draw the background of selected items, which cell renderers can use fill selected cells.

65 **void setSelectionForeground***ColorselectionForeground*

Sets the color used to draw the foreground of selected items, which cell renderers can use to render text and graphics.

66	**void setSelectionInterval***intanchor, intlead* Selects the specified interval.
67	**void setSelectionMode***intselectionMode* Sets the selection mode for the list.
68	**void setSelectionModel***ListSelectionModelselectionModel* Sets the selectionModel for the list to a non-null ListSelectionModel implementation.
69	**void setUI***ListUIui* Sets the ListUI, the look and feel object that renders this component.
70	**void setValueIsAdjusting***booleanb* Sets the selection model's valueIsAdjusting property.
71	**void setVisibleRowCount***intvisibleRowCount* Sets the visibleRowCount property, which has different meanings depending on the layout orientation: For a VERTICAL layout orientation, this sets the preferred number of rows to display without requiring scrolling; for other orientations, it affects the wrapping of cells.
72	**void updateUI** Resets the ListUI property by setting it to the value provided by the current look and feel.

Methods inherited

This class inherits methods from the following classes:

- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

JList Example

Create the following java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingControlDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingControlDemo {

   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
```

```
   private JPanel controlPanel;

   public SwingControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingControlDemo  swingControlDemo = new SwingControlDemo();
      swingControlDemo.showListDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);

      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showListDemo(){

      headerLabel.setText("Control in action: JList");

      final DefaultListModel fruitsName = new DefaultListModel();

      fruitsName.addElement("Apple");
      fruitsName.addElement("Grapes");
      fruitsName.addElement("Mango");
      fruitsName.addElement("Peer");

      final JList fruitList = new JList(fruitsName);
      fruitList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
      fruitList.setSelectedIndex(0);
      fruitList.setVisibleRowCount(3);

      JScrollPane fruitListScrollPane = new JScrollPane(fruitList);

      final DefaultListModel vegName = new DefaultListModel();

      vegName.addElement("Lady Finger");
      vegName.addElement("Onion");
      vegName.addElement("Potato");
      vegName.addElement("Tomato");

      final JList vegList = new JList(vegName);
      vegList.setSelectionMode(
         ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
      vegList.setSelectedIndex(0);
      vegList.setVisibleRowCount(3);

      JScrollPane vegListScrollPane = new JScrollPane(vegList);

      JButton showButton = new JButton("Show");
```

```
      showButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            String data = "";
            if (fruitList.getSelectedIndex() != -1) {
               data = "Fruits Selected: " + fruitList.getSelectedValue();
               statusLabel.setText(data);
            }
            if(vegList.getSelectedIndex() != -1){
               data += " Vegetables selected: ";
               for(Object vegetable:vegList.getSelectedValues()){
                  data += vegetable + " ";
               }
            }
            statusLabel.setText(data);
         }
      });

      controlPanel.add(fruitListScrollPane);
      controlPanel.add(vegListScrollPane);
      controlPanel.add(showButton);

      mainFrame.setVisible(true);
   }
}
```

Compile the program using command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingControlDemo.java
```

If no error comes that means compilation is successful. Run the program using following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingControlDemo
```

Verify the following output