# GUI Programming in JAVA

# Using Swing Control -III

Prepared By:

( Cs Basics Team)

# Objective

In this presentation you will learn about the following swing controls i.e. Purpose, their design time Properties and Run time Methods.

- List (jList)
- Combo (List+Text Field) (jComboBox)
- Timers
- Random Number Generation & use

# Working with jList control

A List (or List box) is box shaped control containing list of objects, from which single or multiple selection can be made. jList control offers the following features- A box shaped control capable to displaying a list of choices (Text or graphics/images) It Allows single or multiple selection of items using mouse.

- Equipped with built-in scroll bar to view a large list. valueChanged() method of ListSelection Listener is used to handle the JList events

# How to add jList control with Frame

A jList can be attached with frame in following way-

- Select List control from Swing palette , and drag and drop on the frame.
- Select the jList control and click on (□) button of Model property. A dialog box appears.
- Type list of choices in ListModelEditor dialog box. Finally press OK button.
- List Box is displayed on the frame with typed choices.

# Commonly used Properties of jList

| Properties | Value | Description |
|---|---|---|
| **Background** | Color | Sets the background color. |
| **Border** | Border setting as in Jpanel | Specifies the type of Border applies on the boundary of the panel. |
| **Foreground** | Color | Sets the foreground color. |
| **model** | Items for Choice | Specifies the items to be displayed as a choice. |
| **selectionMode** | SINGLE<br>SINGLE_INTERVAL<br><br>MULTIPLE_INTERVAL | User may select single item.<br>User may select Single range of items by holding SHIFT key.<br>User may select Multiple range of Items by holding CTRL key |
| **selectedIndex** | Value | Specifies the index of Items to appear selected (default is -1 since no items is selected.) |
| **selectedIndices** | values | Specifies the indices of selected items in form of an array to show multiple items. (default -1) |
| **Visible RowCount** | Value | Sets the number of rows to display without scroll bar. (default is 8) |
| **font** | Font name | Specifies font's name and size etc. |
| **enabled** | True/False | Specifies that list will be active or not. |

Other properties are ***Focusable, NextFocusableComponent Tooltip Text*** etc.

# Commonly used Methods of jList

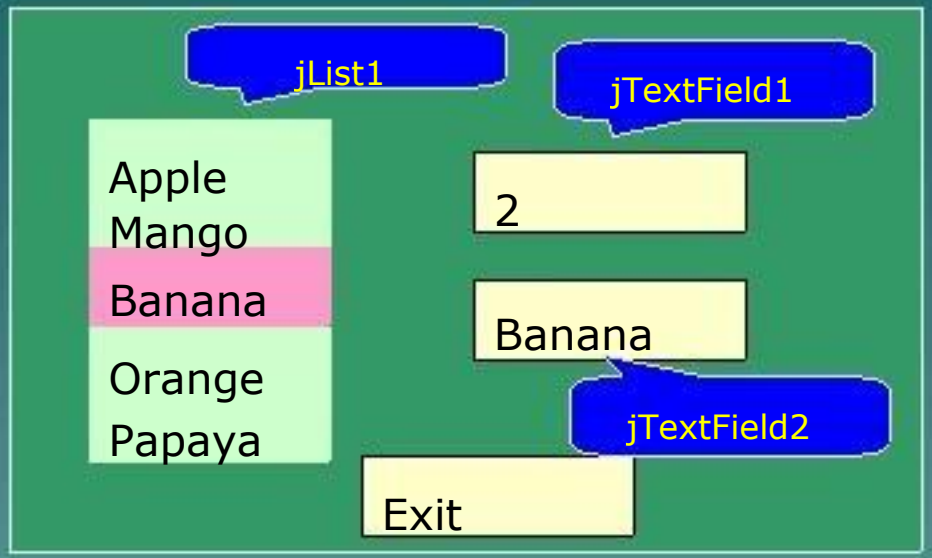
| Methods | Description |
|---|---|
| **Void clearSelection()** | Clears the selection in the list. Ex. jList1.clearSelection(); |
| **int getMaxSelectionIndex()** | Returns the largest selected cell index, or -1 if the selection is empty. Int x=jList1.getMaxSelectionIndex(); |
| **int getMinSelectionIndex()** | Returns the smallest selected cell index, or -1 if the selection is empty. |
| **int getSelectedIndex()** | Returns the index of selected items in single selection mode. Returns -1, if selection is not made. |
| **Object getSelectedValue()** | Returns the value or selected items. String st= (String) jList1.getSlectedValue(); |
| **Object[] getSelectedValues()** | Returns the values of selected items (multiple selection) String st[]= (string) jList1.getSlectedValues(); |
| **Boolean isSelectedIndex(int)** | Returns True if given Index is selected. Ex: if (jList1.isSlectedIndex(2)) {□ } |
| **Boolean isSelectionEmpty()** | Returns True, if nothing is selected. Ex: if (jList1.isSelectedEmpty()) {□ .} |
| **Void setSelectedIndex(int)** | Selects a single cell as per given Index. Does nothing if given index is not in range. Ex. jList1.setSelectedIndex(2); |
| **Void setSelectedIndices(int[])** | Selects a multiple cell as per given Indices in an array. |

Other methods are- isEnabled(), setVisible(), setEnabled() etc. as used earlier.

# How to handle Selections in the jList

## Single Selection

Suppose we want to display the index and value of selected items i.e. 2 and 'Banana' in Text Fields when user selects an items in the list.

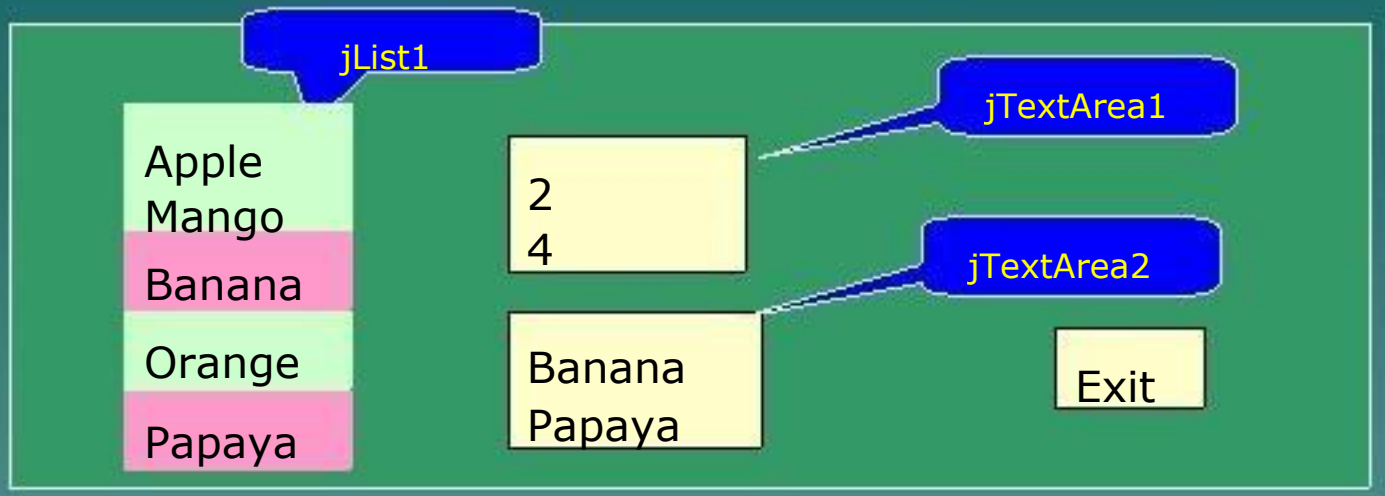



Just Right Click on jList control and choose
Events->ListSelection->ValueChanged
Write the following code in //TO DO Section .

```
int x=jList1.getSelectedIndex();
String st= (String ) jList1.getSelectedValue();
jTextField1.setText(""+x);
jTextField2.setText(st);
```

# How to handle Selections in the jList

## Multiple Selection




Go to Events->ListSelection->ValueChanged of JList1 control
Write the following code in //TO DO Section .

```
int x[]=jList1.getSelectedIndices();
Object st[]= jList1.getSelectedValues();
for(i=0;i< x.length ;i++)
    jTextArea1.append(""+x[i]+'\n');
for(i=0;i<st.length ;i++)
    jTextArea2.append((String) st[i]+'\n');
```

# Adding/Removing Items at run time (Dynamic List)

**An items in the list can be deleted or inserted run time by using special methods available in DefaultListModel class. To implement Dynamic list follow the following steps-**

**To import the DefaultListModel class, type the line at top -**

**import javax.swing.DefaultListModel;**

**Select jList control and go to Model ( ) Property and choose Custom code in the dialog box.**

- **Type the following line in the dialog box and Press OK button.**

  **new DefaultListModel()**

- **The following Methods may be used to manipulate the list at run time-**

| **Methods** | **Description** |
|---|---|
| **Int size()** | Returns the number of items currently in the list. Ex. int x=jList1.size(); |
| **Boolean isEmpty()** | Returns True if the list is empty (no items) |
| **Void addElement (String)** | Adds given string at the end of the list. |
| **Void insertElementAt(string.position)** | Adds given item at given index. |
| **Boolean removeElement(string/index)** | Removes specified items from the list |
| **Void removeElementAt(object,position)** | Removes given object from given index. |
| **Void removeAllElements()** | Removes all entries from the list. |
| **Object getModel()** | Returns the ListModel of the list. |

# Adding/Removing Items at run time (Dynamic List)

**The following steps should be followed in the event handler of the button, where you want to implement the command.**

**1. Make a model object as-**

```
DefaultListModel dlm = (DefaultListModel) jList1.getModel();
```

**2. Use following command as per requirement-**

```
// add item in the end of the list//
dlm.addElement("Apple");
// Insert item at given index in the list//
dlm.addElementAt("Apple",2);
// Remove selected item from the list//
dlm.removeElement(jList1.getSelectedIndex());
// Remove item from given position from the list//
dlm.removeElementAt(4);
// Remove given item from from the list//
dlm.removeElement("Banana");
```

**3. Reflect all changes to the list**

```
jList1.setModel(dlm)
```

# Working with Image List (List of Images)

**The following steps should be followed to implement the jList containing Images choices instead of Text choices.**

1. **Add JList Control on the Frame.**
2. **Right Click on the jList and choose Customised Code from the pop up menu.**
3. **Select Custom Property from the second drop down box of Code Customizer dialog Box.**
4. **Now replace the following pre written line with the following code.**

```
String[] strings={"Item 1", "Item 2", "Item 3", □ };

 // Write New code like this //
ImageIcon[] String = {new ImageIcon("c:\\aaa.jpg"),
                      new ImageIcon("c:\\bbb.jpg"),
                      new ImageIcon("c:\\ccc.jpg")
                     };
```

5. **Press Ok Button to closed the dialog box.**
6. **Open Source editor and write the following code at the top.**

```
import javax.swing.ImageIcon;
```

7. **Now save the application and run.**

# Working with jComboBox control

A jComboBox (TextField + List Box) is a control which offers the list of choice which can be selected through a drop-down list.

By default it is an un-editable control, but we can make it editable by setting 'editable' property as True.

jComboBox1ActionPerformed(..) method can be handled when user selects an item from the combo.

**Difference Between List & Combo Box**

- In Combo Box, user can select and edit an item but in List, Items can be selected and can not be edited.
- List does not offers Text Field where as Combo Box offers Text Field.
- Combo Box offers a Drop-down list, so that it takes less space in the frame, but List this feature is not available.

List allows more than one items to be selected, but in Combo Box, only one item can be selected.

# Commonly used Properties of jComboBox

| Properties | Value | Description |
|---|---|---|
| **Background** | Color | Sets the background color. |
| **Foreground** | Color | Sets the foreground color. |
| **Border** | Border as in jPannel | Specifies the Border setting. |
| **model** | Items for Choice | Specifies the items to be displayed as a choice. |
| **selectedIndex** | Value | Specifies the index of Items to appear selected (default is -1 since no items is selected.) |
| **selectedItem** | String/values | Specifies the indices of selected items. |
| **font** | Font name | Specifies font's name and size etc. |
| **editable** | True/False | If True, you can edit/type new value or choice in the Combo Box. |
| **enabled** | True/False | Specifies that list will be active or not. |
| Other properties are ***Focusable, NextFocusableComponent Tooltip Text*** etc. | | |

# Commonly used Methods of jComboBox

| Methods | Description |
|---|---|
| **Void addItem(string)** | Adds an item to the choice list at the end.<br>Ex. jComboBox1.addItem("Banana"); |
| **Void insertItemAt(string,int)** | Insert a given item at given index.<br>Ex. jComboBox1.insertItemAt("Banana",2); |
| **int getItemCount()** | Returns the number of items in the combo Box.<br>int x=jComboBox1.getItemCount(); |
| **String getItemAt(int)** | Returns the items at specified index.<br>String st=jComboBox1.getItemAt(2); |
| **int getSelectedIndex()** | Returns the index of selected items. |
| **String getSelectedItem()** | Returns the selected items.<br>String st= (String) jComboBox1.getSlectedItem(); |
| **Boolean isEditable()** | Returns True, if Combo box is editable. |
| **Void removeAllItems()** | Removes all the items from combo. |
| **Void removeItem(String)** | Removes specified items from the combo.<br>Ex. jComboBox1.removeItem("Banana"); |
| **Void removeItemAt(index)** | Removes items for given index from the combo.<br>Ex. jComboBox1.removeItemAt(2); |
| **Void setSelectedIndex(int)** | Selects an item as per given Index.<br>Ex. jComboBox1.setSelectedIndex(2); |
| **Void setSelectedItem(String)** | Selects an specified item. |

# Working with Swing Timer control

A Timer control is invisible control, which can be used to repeat a task after specified Interval.

The following steps may be followed to implement a timer in the application.

- Add following code at the top-

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

- Declare a timer object and handler method as-

  Timer<timer_object>=new Timer(<interval>, new ActionListener() { public void actionPerformed(ActiveEvent e) {<commands to be executed>.} });

- Now start the timer as-

  <timer_Object>.start();

- Stop the timer whenever you need.

  <timer_Object>.stop

# Working with Random Numbers

A random number is system generated number, which can be utilized in the application in various ways.

The following steps may be followed to get random number.

- Add following code at the top-

  import javax.util.Random;

- Declare an object as-

  Random <object>=new Random( );

  Ex. Random randobj = new Random();

- Get number as -

  <int var>=<object>.nextInt(<range>);

  // to generate a number between 10 to 50 //

  Ex. int x= randobj.nextInt((50-10+1))+10;

- Use the number as -

  System.out.println("Random number="+x);