

## ADVANCED JAVA PROGRAMMERS GUIDE

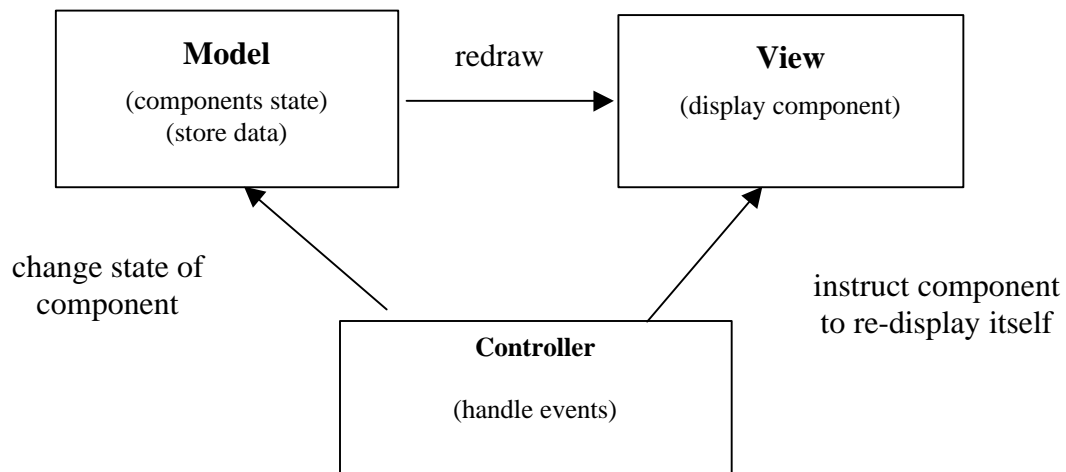
File:	swing.doc
Date Started:	Aug 7, 2000
Last Update:	Jan 6, 2002
Status:	proof

### SWING COMPONENTS I

The Swing components located in package **javax.swing.\*** and are Java answers to create a better Graphical User Interface. The Swing component work a little different then using AWT, but programming is very similar. Swing components work on the **Model-View-Controller** (MVC) principle. A GUI component has three characteristics. Each characteristic is represented by a **class** in the Model-View-Controller (MVC) design approach.

characteristic	class	description
contents	<b>Model</b>	state or data contents of the component (selected, deselected, visible, label name etc)
visual appearance	<b>View</b>	displays components
behavior	<b>Controller</b>	reaction to events like mouse clicks

Using the Model-View-Controller (MVC) approach instead of a component as one class doing every thing, each task divided into separate classes. This is a very good approach when using object oriented programming concepts.



Each component has a separate class for the Model, View and Controller. 3 Separate classes.

## Swing Components

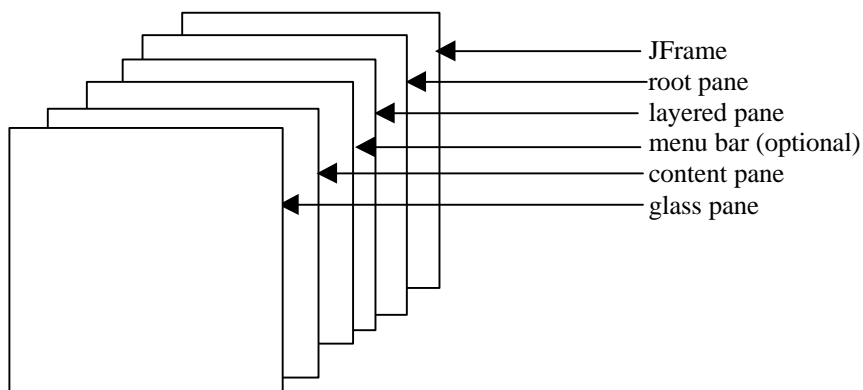
All swing components start with a 'J'. They operate the same but behave a little differently than their AWT counterparts.

component	description
<b>JFrame</b>	display GUI components on a screen
<b>JApplet</b>	display GUI components on a web browser
<b>JDialog</b>	pop up window
<b>JPanel</b>	a container that components can be placed
<b>JLabel</b>	an area where text or icons can be displayed
<b>JButton</b>	a area with a label used to trigger an event when clicked
<b>JTextField</b>	an area where input data from the keyboard can be displayed
<b>JCheckBox</b>	a box area that can be selected or not selected
<b>JComboBox</b>	a drop down list of selections
<b>JList</b>	an area displaying a list of items to be selected
<b>TextArea</b>	an area displaying many lines of text that can be edited
<b>JScrollPane</b>	an area that contains a component that can be scrolled
<b>JSlider</b>	a component that has a slider thumb that points to tick marks

## Drawing objects on the JFrame

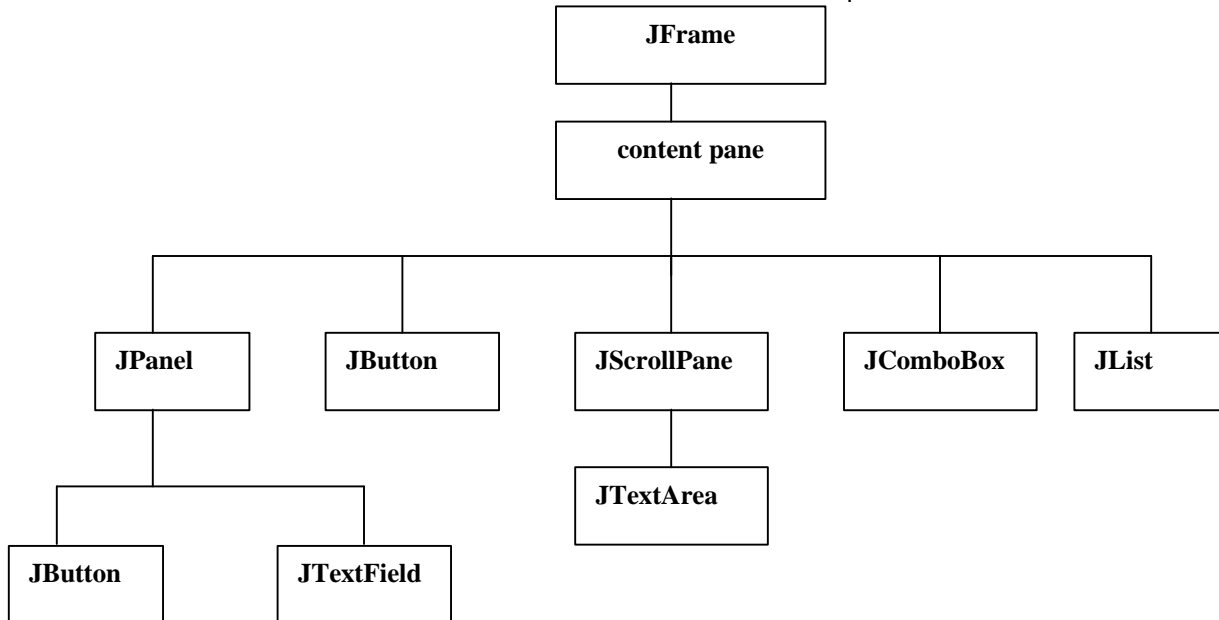
A JFrame has 4 panes. A pane is an area where components are drawn on.

pane	purpose
<b>root pane</b>	made up of a glassPane, an optional menuBar, and a contentPane.
<b>layered pane</b>	the layeredPane manages the menuBar and the contentPane.
<b>glass pane</b>	glassPane sits over the top of everything, so it can receive mouse movements.
<b>content pane</b>	where you add your JComponents to



## containment hierarchy

Here is the Swing component hierarchy. All components attach to the content pane, the content Pane attaches to the JFrame. JPanels can contain other components.



## containers

The frame, dialogs and applets are the **top level** containers. Panels, scrollpanes and tabbed panes are **intermediate** containers. Buttons, labels combo boxes and text fields are **atomic** components. The top level container provides place for the lower level components to paint them selves. Every top level container indirectly contains an intermediate container known as a content pane,. The content pane contains all of the visible components in the GUI . A content pane does not include a menu bar the menu bar is attached to the top level container like the JFrame. Components are added to the panel and panels are added to the content pane.

## Example adding a Swing Component

When using JComponents you must add the component to the **content pane** not the **JFrame**. There are 3 steps to add a Swing component.

(1) You must first get the content pane from the JFrame.

```
Container contentPane = getContentPane(); // get content pane from JFrame
```

(2) Make a button by calling the JButton class constructor.

```
JButton btn = new JButton("Hello"); // create button object
```

(3) use the **add()** method of the Container class to add GUI objects to the frame.

```
contentPane.add(btn); // add button to content pane
```

The contentPane is derived from the Container class located in package java.awt

```

// JButtontest.java
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
class JButtonTest extends JFrame
{
    public JButtonTest(String title)
    {
        super(title); // send title to Frame class
        Container contentPane = getContentPane();
        contentPane.setLayout(new FlowLayout());
        JButton btn = new JButton("Hello");
        contentPane.add(btn);
    }

    // main method to create frame
    public static void main(String[] args)
    {
        // declare and create JButtonTest object
        JFrame jf = new JButtonTest("Frame Test");
        jf.setSize(200,200); // re-size frame
        jf.show(); // display window

        jf. addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
    }
} // end JButtonTest

```



## LAYOUT MANAGERS

The layout managers are used to arrange the components in a container. There are 6 Layout Managers:

manager	description
<b>FlowLayout</b>	default manager for every JPanel, lays out components left to right honors a components preferred size
<b>GridLayout</b>	displays components equal in size on a matrix of rows and columns does not honors a components preferred size all components set to size of grid element
<b>BorderLayout</b>	5 areas north, south, east, west and center for north, south components honors a components height but width set to size of container, for east west components honors a components width but height of component set to container or constraint but north south components
<b>CardLayout</b>	area that contain different components at different times
<b>GridBagLayout</b>	places components into different areas according to weights
<b>BoxLayout</b>	puts components in a single row or column

The layout managers are the same ones used in AWT and operate the same. We do not need to go into a deep discussion in how to use them. When using swing you set the layout to the content pane not the JFrame. The first thing we need to do is to get a reference to the content pane. A content pane is a Container object.

```
Container c = getContentPane(); // get content pane
```

Once we get the content pane we set the content pane to FlowLayout. By using flow layout, each component will be displayed one after each other, row by row.

```
c.setLayout(new FlowLayout()); // set layout manger to flow layout
```

Then you add the component to the contentPane

```
contentPane.add (new Button("Button ")); // add button to content pane
```

JFrames have a default manager of BorderLayout and JPanels and JApplets have default layout manager of FlowLayout. Here are the API's for the common layout managers that you will use.

<b>flow layout</b>	Components are lined up horizontally until one line fills then another line starts, there is a 5 pixel gap between each component	
<b>FlowLayout</b> (int align)	c.setLayout( new FlowLayout())	
	<i>align</i> = FlowLayout.LEFT, FlowLayout.CENTER or FlowLayout.RIGHT	
<b>FlowLayout</b> (int align, int hgap, int vgap )	c.setLayout( new FlowLayout (FlowLayout.CENTER,15,20) )	
	hgap = horizontal pixels between components , vgap = vertical pixels between components	

<b>border layout</b>	The area to be laid out is divided into areas North, South, East, West and Center. The borders are laid out first and the center last.	
<b>BorderLayout</b> ()	c.setLayout(new BorderLayout());	
<b>add</b> (component, "North")	<b>add</b> (component, BorderLayout.NORTH)	add component north
<b>add</b> (component, "South")	<b>add</b> (component BorderLayout.SOUTH)	add component south
<b>add</b> (component, "East")	<b>add</b> (component BorderLayout.EAST)	add component east
<b>add</b> (component, "West")	<b>add</b> (component BorderLayout.WEST)	add component west
<b>add</b> (component, "Center")	<b>add</b> (component BorderLayout.CENTER)	add component center

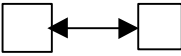
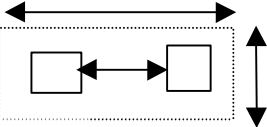
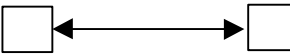
<b>grid layout</b>	specify how many rows and columns of a grid. Each component placed in grid sequentially	
<b>GridLayout</b> (int row,int columns)	setLayout(new GridLayout(3,3));	
<b>GridLayout</b> (int rows, int cols, int hgap, int vgap)	setLayout(new GridLayout(3,3, 10,20));	
<b>add</b> (component)	components are added starting with first row and column	

## Box Layout

The box layout lets you layout single row or column of components. Its like a grid Layout Manager but you get to work with signal rows and columns. There is also a Box class container that has BorderLayout as the default layout manager. The box class is in package javax.swing.\* You can have a horizontal box container or a vertical box container.

```
Box bh = Box.createHorizontalBox();
Box bv = Box.createVerticalBox();
```

A horizontal arranges components left to right. A vertical box arranges components to bottom. There are no spaces between the components. You need to add fillers between the components to add spaces. There are three kinds of fillers:

filler	description	example using
<b>struts</b>	Adds space between adjacent components 	<code>bh.add(new Label("Label "));</code> <code>bh.add(Box.createHorizontalStrut(5));</code> <code>bh.add(new Button("Button "));</code>
<b>rigid areas</b>	Pair of struts, separates adjacent components by a fixed amount. It forces the minimum height or width of a box 	<code>bh.add(new Button("Button "));</code> <code>bh.add(Box.createRigidArea</code> <code>(new Dimension(10,50));</code> <code>bh.add(new TextField("TextField "));</code>
<b>glue</b>	separates components as much as possible 	<code>bv.add(new Button("Button "));</code> <code>bv.add(Box.createGlue());</code> <code>bv.add(new TextField("TextField "));</code>

## JCOMPONENTS

The JComponents are the GUI components that you use to attach to JFrames. We now present a demo program to display all the common components. Writing a swing program is not too different then using AWT. You need to attach components to the content pane and import the **javax.swing** package. If you want to run the test program just type in the blue text. All components are created and attached inside the constructor. The constructor starts by sending a title to the super class JFrame, gets the ContentPane and the sets the layout manager to FlowLayout.

```
// SwingTest.java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class SwingTest extends JFrame
{
    public SwingTest()
    {
        super("Swing Test");

        // get content pane
        Container c = getContentPane();

        // set layout manger to flow layout
        c.setLayout(new FlowLayout());
    }
}
```

## JLabel

A label can display a single line of read-only text. We create a JLabel and add it to the content pane:

```
JLabel lab1 = new JLabel("I am a Label");
c.add(lab1);
```

A JLabel can display a tool tip caption when the user positions the mouse cover the JLabel.

```
JLabel lab2 = new JLabel("I am a Label with a tool tip");
lab2.setToolTipText("you got me");
c.add(lab2);
```

A JLabel may also contain text and an icon, an icon is a small graphic image.

```
JLabel lab3 = new JLabel
("see the stars",new ImageIcon("stars.gif"),SwingConstants.LEFT);
c.add(lab3);
```


The above JLabel constructor has three arguments, test message, ImageIcon and label position. We set the JLabel text to "see the stars" we create an Icon object with new ImageIcon("stars.gif") and we position the test to the Left using SwingConstants.LEFT. The **SwingConstants** interface defines a set of common integer constants to align text etc.

<b>JLabel()</b>		JLabel lab1 = new JLabel();
<b>JLabel(String s)</b>		contentPane.add(lab1);
<b>JLabel(String s, int align)</b>		JLabel lab2 = new JLabel("message");
		contentPane.add(lab2);
<b>s =</b>	text	JLabel lab3 =
<b>align =</b>	SwingConstants. LEFT, SwingConstants. CENTER SwingConstants. RIGHT	JLabel("message",SwingConstants.CENTER);
		contentPane.add(lab3);
String <b>getText()</b>	get text from Label	String s = lab1.getText();
void <b>setText(String text)</b>	put text into Label	lab2.setText("Hello");

### setting the sizes of JLabels

Sometimes you need to set the size of your JLabel rather than depending on the text to set the size. You can use the void setPreferredSize(Dimension preferredSize) for this

```
JLabel lab1 = new JLabel();
lab1.setPreferredSize(new Dimension(40,20);
```

width   height  


where the Dimension object receives the desired width and a height of the JLabel.

## JTextField and JPasswordField

The JTextField is a single line text area which the user types text from the keyboard or simply type in text. A JPasswordField displays text as stars \*\*\*\*\*. The size of the JTextField can be set in the constructor,

```
JTextField text1 = new JTextField(10);
c.add(text1);
```

or set by the initial text message, where the size is set to the number of characters.

```
JTextField text2 = new JTextField("try to change me");
c.add(text2);
```

You can set the JTextField size and also have an initial message :

```
JTextField text3 = new JTextField("I have 20 characters",20);
c.add(text3);
```

By setting the `setEditable()` method argument to false, you can just have the JTextField display a text message but the user cannot change it.

```
JTextField text4 = new JTextField("try to change me");
text2.setEditable(false);
c.add(text4);
```

A JPasswordField just displays stars even if you give it an initial value or when the user types in a message.

```
JPasswordField pass1 = new JPasswordField("what is the password");
c.add(pass1);
```

<b>Jtext field</b> <input type="text"/>	gets a line of text from the user, or put a line of text
<b>TextField()</b> <b>JTextField()</b>	<code>JTextField tf1 = new JTextField();</code> <code>tf1.setText("text field 1");</code> <code>contentPane.add(tf1);</code>
<b>TextField(int cols)</b> <b>JTextField(int cols)</b> <i>cols = number of columns</i>	<code>JTextField tf2 = new JTextField(20);</code> <code>tf2.setText("text field 2");</code> <code>contentPane.add(tf2);</code>
<b>TextField(String text)</b> <b>JTextField(String text)</b>  <i>text = message cols</i>	<code>JTextField tf3 = new TextField</code> <code>("text field 3");</code> <code>contentPane.add(tf3);</code>
<b>TextField(String text, int cols)</b> <b>JTextField(String text, int cols)</b> <i>text = message</i> <i>cols = number of columns</i>	<code>JTextField tf4 = new JTextField</code> <code>("text field 4", 20);</code> <code>contentPane.add(tf4)</code>
<code>String <b>getText()</b></code>	get text from JTextField
<code>void <b>setText</b>(String text)</code>	put text into a JTextField
<code>void <b>setEditable</b>(boolean editable)</code>	true lets user edit text, false user cannot edit text

## JButton

A JButton is an area clicked by a mouse that seems to be pressed that triggers some action to occur in a program. JButtons are rectangle with a text message label.

```
JButton btn1 = new JButton("press me");
c.add(btn1);
```

A JButton can also contain an icon, a graphical image:

```
JButton btn2 = new JButton
("click on a star",new ImageIcon("stars.gif"));
c.add(btn2);
```

An extra feature of a JButton is that it will display another icon when the mouse is over the JButton by calling the `setRolloverIcon()` method with an icon object.

```
JButton btn3 = new JButton
("click on a star",new ImageIcon("stars.gif"));
btn3.setRolloverIcon(new ImageIcon("red-ball.gif"));
c.add(btn3);
```

## JCheckbox

A JCheckBox lets you check a small rectangle with a label as being selected or not. A checkbox also has a label for identification.


```
JCheckBox chk1 = new JCheckBox("check me");
c.add(chk1);
JCheckBox chk2 = new JCheckBox("check me too");
c.add(chk2);
```

<input checked="" type="checkbox"/> <b>check box</b>	yes or no check mark on a box identified by a label, Checkbox can be created with initial state <b>true</b> for <b>checked false</b> for unchecked default is unchecked
<b>JCheckBox</b> (String label);	JCheckBox cb1 = new JCheckBox("Check 1"); contentPane.add(cb1);
label = checkbox label	
<b>JCheckBox</b> (String label, boolean initialState);	JCheckBox cb = new JCheckBox("Check 2", true); contentPane.add(cb2);
label = checkbox label initial state unchecked	
boolean <b>isSelected()</b>	returns state of checkbox true = selected
void <b>setSelected</b> (boolean state)	sets the check box to a new state

## JRadioButtons

Radio buttons are let you select a round circle that is filled for selected. JRadio buttons are grouped together and only one radio button can be selected at a time. To determine which radio button is grouped together a RadioGroup object is constructed that you add radio buttons to it. JRadioButtons also have labels.

```
JRadioButton rb1 = new JRadioButton("A");
c.add(rb1);
JRadioButton rb2 = new JRadioButton("B",true);
c.add(rb2);
JRadioButton rb3 = new JRadioButton("C");
c.add(rb3);
// button group
ButtonGroup grp1 = new ButtonGroup();
grp1.add(rb1); // add buttons to group
grp1.add(rb2);
grp1.add(rb3);
```

 <b>radio button</b>	A filled in circle surrounded by a circle identified by a label, A Radio Button can be created with initial state <b>true</b> for <b>selected</b> or <b>false</b> for unselected default is unselected
<b>JRadioButton</b> (String label); label = radio button label	JRadioButton rb1 = new JRadioButton("Radio 1"); contentPane.add(rb1);
<b>JRadioButton</b> (String label, boolean initialState); label = radio button label (initial state unchecked)	JRadioButton rb2 = new JRadioButton("Radio 2", true); contentPane.add(rb2);
boolean <b>isSelected</b> ()	returns state of radio button true = selected
void <b>setSelected</b> (boolean state)	sets the radio button to a new state

## JComboBox

A JComboBox provides a list of items the user can select. A JComboBox displays the first selection and the user must press an arrow to see the rest to select from. A JComboBox is also called a drop down box. We first initialize an array of strings and then give to the JComboBox constructor.

```
String[] selects = {"red","green","blue","brown","orange","red"};
JComboBox cb1 = new JComboBox(selects);
cb1.setMaximumRowCount(4);
c.add(cb1);
```

The JComboBox uses the **DefaultComboBox** model that supports the following methods for inserting and removing items. These methods are wrapped for you in the JComboBox.

method	description
void <b>addElement</b> (Object anObject)	add element to combo box
Object <b>getElementAt</b> (int index)	get element at specified index
int <b>getIndexOf</b> (Object anObject)	get index of specified object
Object <b>getSelectedItem</b> ()	get selected item
int <b>getSize</b> ()	get number of items in combo box
void <b>insertElementAt</b> (Object anObject,int index)	insert item at specified index
void <b>removeAllElements</b> ()	empties combo box
void <b>removeElement</b> (Object anObject)	remove specified object
void <b>removeElementAt</b> (int index)	remove element at specified index
void <b>setSelectedItem</b> (Object anObject)	set selected item to this specified object

## JList

A JList displays all the selectable items. If there are more items not visible than they can be seen using scrollbars. By default more than 1 Item can be selected at once buy using the shift key or ctrl key and a mouse click.

```
String[] selects2 = {"tigers","elephants","lion","bears","giraff","monkeys"};
JList list1 = new JList(selects2);
list1.setVisibleRowCount(4);
c.add(list1);
```

You can force only 1 item only to be selected by setting the `setSelectionMode()` method to **SINGLE\_SELECTION** mode and setting the visible rows with the `setVisibleRowCount(int rows)` method.

```
JList list2 = new JList(selects2);
list2.setVisibleRowCount(4);
list2.setSelectionMode(ListSelectionMode.SINGLE_SELECTION);
c.add(list2);
```

We have a **MULTIPLE\_INTERVAL\_SELECTION** mode allows selection of a continuous range of items in the list by clicking the first item and then holding the shift key and click the last item. You can also select miscellaneous items to be selected by holding the CTRL key while clicking an item a second time.

```
JList list3 = new JList(selects2);
list3.setVisibleRowCount(4);
list3.setSelectionMode(ListSelectionMode.MULTIPLE_INTERVAL_SELECTION);
c.add(list3);
```

We will use a button and a button event handler to be coded later to copy the selections from the **MULTIPLE\_INTERVAL\_SELECTION** list box to a **SINGLE\_INTERVAL\_SELECTION** list box.

```
btn3 = new JButton(" >> ");
c.add(btn3);
```

A **SINGLE\_INTERVAL\_SELECTION** mode allows selection of a of a continuous range of items in the list by clicking the first item and then holding the shift key and click on the last item.

```
JList list4 = new JList(selects2);
list4.setVisibleRowCount(4);
list4.setSelectionMode(ListSelectionMode.MULTIPLE_INTERVAL_SELECTION);
c.add(list4);
```

### replacing contents of a list box

Once you got values in your list box you can replace them using the `setListData(final Object[] listData)` or the `void setListData(final Vector listData)` methods.

```
String[] replaceString = {"lots","of","work","using","JList"};
list4.setListData(replaceString);
```

### adding and removing data elements from a list box

This is a very difficult thing to do. You first need to construct a JList that has a DefaultListModel object

```
JList list5 = new JList(new DefaultListModel());
c.add(list5);
```

Next you must get the model object from the list box to when you want to add and remove elements

```
DefaultListModel model = (DefaultListModel)list5.getModel();
```

Now you can add and remove items

```
model.addElement("the");
model.addElement("cow");
model.addElement("ate");
model.addElement("the");
model.addElement("pig");
model.remove("ate");
```

That seems like a lot of work to do. You must be very careful the **addElement()** and **removeElement()** methods only work with the **DefaultListModel**. All the other List boxes use the **AbstractListModel**. The **AbstractListModel** does not have the `addElement()` and `remove()` methods.

### getting number of items in a list box

Again you must get the **model object** from the list box and the call the `size()` methods

```
DefaultListModel model = (DefaultListModel)list.getModel();
int size = model.getSize();
```

**JList box methods**

void <b>ensureIndexIsVisible</b> (int index)	if the specified cell isn't completely visible, scroll to cell
void <b>setVisibleRowCount</b> (int nums); set the number of visible rows	list3.setVisibleRowCount(4);
void <b>setSelectionMode</b> (int mode);	list3. setSelectionMode (ListSelectionMode.SINGLE_SELECTION);
	selection modes: ListSelectionMode.SINGLE_SELECTION, ListSelectionMode.SINGLE_INTERVAL_SELECTION, ListSelectionMode.MULTIPLE_INTERVAL_SELECTION
public void <b>setListData</b> (final Vector listData)	listData is a Vector containing the items to be displayed in the list box
void <b>setListData</b> (final Object[] listData)	listData an array of Objects containing the items to be displayed in the list
boolean <b>isSelectionEmpty</b> ()	returns true if no item selected
int[] <b>getSelectedIndices</b> ()	Return an array of all of the selected indices in increasing order.
int <b>getSelectedIndex</b> ()	returns the first selected index. Returns -1 if there is no selected item.
Object <b>getSelectedValue</b> ()	returns first selected value or null, if the selection is empty
Object[] <b>getSelectedValues</b> ()	return currently selected items as an array of objects
void <b>setSelectedIndex</b> (int index)	index = The index of the one cell to select
void <b>setSelectedIndices</b> (int[] indices)	Select a set of cells. indices = The indices of the cells to select
void <b>setSelectedValue</b> (Object anObject, boolean shouldScroll)	Selects the specified object from the list. anObject = the Object to select shouldScroll true if the list should scroll to display
public void <b>setFixedCellWidth</b> (int width)	defines the width of every cell in the list

Here is the **DefaultListModel** API:

**Constructor Summary**

<a href="#">DefaultListModel()</a>	
------------------------------------	--

**Method Summary**

void <a href="#">add</a> (int index, Object element)	Inserts the specified element at the specified position in this list.
void <a href="#">addElement</a> (Object obj)	Adds the specified component to the end of this list.
int <a href="#">capacity</a> ()	Returns the current capacity of this list.
void <a href="#">clear</a> ()	Removes all of the elements from this list.
boolean <a href="#">contains</a> (Object elem)	Tests if the specified object is a component in this list.
void <a href="#">copyInto</a> (Object[] anArray)	Copies the components of this list into the specified array.
Object <a href="#">elementAt</a> (int index)	Returns the component at the specified index.
Enumeration <a href="#">elements</a> ()	Returns an enumeration of the components of this list.
void <a href="#">ensureCapacity</a> (int minCapacity)	Increases the capacity of this list, if necessary, to ensure that it can hold at least the number of components specified by the minimum capacity argument.
int <a href="#">capacity</a> ()	Returns the first component of this list.
Object <a href="#">get</a> (int index)	Returns the element at the specified position in this list.

<code>Object getElementAt(int index)</code>	Returns the component at the specified index.
<code>int getSize()</code>	Returns the number of components in this list.
<code>int indexOf(Object elem)</code>	Searches for the first occurrence of the given argument.
<code>int indexOf(Object elem, int index)</code>	Searches for the first occurrence of the given argument, beginning the search at index.
<code>void insertElementAt(Object obj, int index)</code>	Inserts the specified object as a component in this list at the specified index.
<code>boolean isEmpty()</code>	Tests if this list has no components.
<code>Object lastElement()</code>	Returns the last component of the list.
<code>int lastIndexOf(Object elem)</code>	Returns the index of the last occurrence of the specified object in this list.
<code>int lastIndexOf(Object elem, int index)</code>	Searches backwards for the specified object, starting from the specified index, and returns an index to it.
<code>Object remove(int index)</code>	Removes the element at the specified position in this list.
<code>void removeAllElements()</code>	Removes all components from this list and sets its size to zero.
<code>boolean removeElement(Object obj)</code>	Removes the first (lowest-indexed) occurrence of the argument from this list.
<code>void removeElementAt(int index)</code>	Deletes the component at the specified index.
<code>void removeRange(int fromIndex, int toIndex)</code>	Deletes the components at the specified range of indexes.
<code>Object set(int index, Object element)</code>	Replaces the element at the specified position in this list with the specified element.
<code>void setElementAt(Object obj, int index)</code>	Sets the component at the specified index of this list to be the specified object.
<code>void setSize(int newSize)</code>	Sets the size of this list.
<code>int size()</code>	Returns the number of components in this list.
<code>Object[] toArray()</code>	Returns an array containing all of the elements in this list in the correct order.
<code>String toString()</code>	Returns a string that displays and identifies this object's properties.
<code>void trimToSize()</code>	Trims the capacity of this list to be the list's current size.

## JTEXTAREA AND JSCROLLPANE

JTextArea allows the user to type in and display many lines of text. In the constructor we set with an initial text message and set to 4 rows and 10 columns.

```
String str1 = "testing text area";
JTextArea ta1 = new JTextArea (str1,4,10);
c.add(ta1);
```

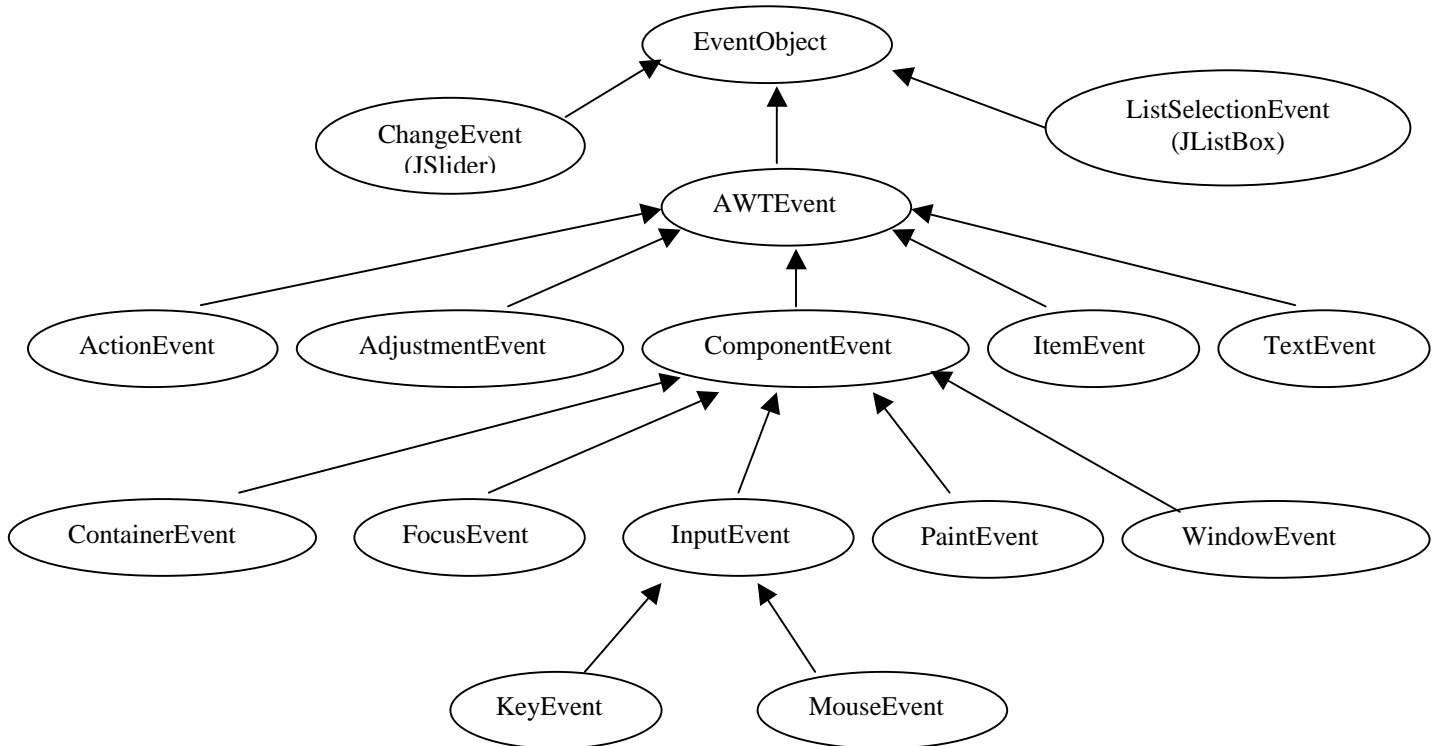
Unfortunately to many programmers and a also big disappointment JTextAreas do not have automatic scroll bars. You have to add a JTextArea to a JScrollPane and then add the JScrollPane to the contentPane.

```
String str2 = "testing text area with scrollers";
JTextArea ta2 = new JTextArea (str2,4,10);
ta2.setEditable(false);
c.add(new JScrollPane(ta2));
```



## EVENT HANDLERS AND LISTENERS

When the user select a component like clicking on a button an event object is created and sent to the program. Swing programs use the same event as AWT inherited from **java.awt.event**. All event objects are derived from EventObject found in **java.event**.



Every event has a corresponding event listener. Event listeners are attached to components originating the event. The event listener calls the event handler to process the event. The following table lists the event with the event listener and a description of the event. Event listeners are interfaces stating which methods the event handler object must implement. All listener interfaces extend `EventListener`. The event classes and listener interfaces are in **java.awt.event** and **javax.swing.event**

For every Event there will be a Listener:

event object	event listener	description of event
ActionEvent	ActionListener	activation of components like clicking a button
AdjustmentEvent	AdjustmentListener	adjustment of adjustable components like scollbars
ContainerEvent	ContainerListener	when components are added or removed from a container
FocusEvent	FocusListener	when a component receives or loses input focus
ItemEvent	ItemListener	when a item is selected from a list, choice or check box
KeyEvent	KeyListener	when keys are pressed on a keyboard
ListSelectionEvent	ListSelectionListener	list box item selected
MouseEvent	MouseListener MouseMotionListener	mouse movement
PaintEvent		when components need repainting
TextEvent	TextListener	when text is modified
WindowEvent	WindowListener	window activity, minimizing, closing etc

A listener is attached to every component. Every component will have a add `XXXListener` method. The following table lists the component with the addlistener methods. The `getSource()` method return the component that generated the event. The `getId()` method return the ID of the event. Every event gets an ID, the event generated has additional methods to identify the component or state of the component. For example the `getActionCommand()` method identified the label of the button that generated the event and the `getStateChange()` method returns the state of a checkbox or radio button.

component	add listener method	event handler method	identification
JTextField JPasswordField	<code>addActionListener()</code>	<code>void actionPerformed();</code>	
JButton	<code>addActionListener()</code>	<code>void actionPerformed();</code>	<code>getActionCommand()</code>
JCheckBox JRadioButton	<code>addItemListener()</code>	<code>void itemStateChanged()</code>	<code>getStateChange()</code>
JComboBox	<code>addItemListener()</code>	<code>void itemStateChanged()</code>	<code>getStateChange()</code>
JList	<code>addListSelectionListener</code>	<code>void valueChanged()</code>	
JTextArea	<code>addActionListener()</code>	<code>void actionPerformed();</code>	

When a event occurs the GUI component notifies the attached event listeners that calls the event handling method. There are

### Three different ways to make event listener objects

(1) make a separate class to handle the component and a separate class that implements the ActionListener interface

```
class ButtonPressTest extends JFrame
{
    ButtonPressTest()
    {
        JButton btn = new JButton("Press me");
        ActionListener listener = new BtnActionListener();
        btn.addActionListener(listener);
        getContentPane().add(btn);
    }
}
class BtnActionListener implements ActionListener
{
    public void actionPerformed(ActionEvent ev)
    {
        JOptionPane.showMessageDialog
        (null,"You pressed button: " + ev.getActionCommand());
    }
}
```

(2 ) make an anonymous class to handle the event

```
class ButtonPressTest2 extends JFrame
{
    ButtonPressTest()
    {
        JButton btn = new JButton("Press me");
        getContentPane().add(btn);
        btn.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent ev)
            {
                JOptionPane.showMessageDialog
                (null,"You pressed button: " + ev.getActionCommand());
            }
        });
    }
}
```

(3) make your application be its own event handler by implementing the event listeners. This is the easiest method but you need to use the `getSource()` methods to find out which component originated the event.

```
class ButtonPressTest3 extends JFrame implements ActionListener
{
    ButtonPressTest()
    {
        JButton btn = new JButton("Press me");
        btn.addActionListener(this);
        getContentPane().add(btn);
    }

    public void actionPerformed(ActionEvent ev)
    {
        JOptionPane.showMessageDialog
        (null,"You pressed button: " + ev.getActionCommand());
    }
}
```

We next write the code for the listener of the above components. We use the anonymous class method. It is easy to identify which component generated the event when each component has its own event handler object. Inside each event handler you will notice we use the `showMessageDialog()` method `JOptionPane` class to print out messages. For each event listener we list the accessor methods to get information about the component. The accessor method may be part of the event listener or to the component.

### Textfield listeners

class	accessor methods	description
ActionEvent	getActionCommand()	get text box message
JTextField	getText()	get text box message
JTextField	setText(String msg)	set text box message

```

text1.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent ev)
    {
        JOptionPane.showMessageDialog
        (null,"You selected text: " + ev.getActionCommand());
    }
});

```

### button listeners

class	accessor methods	description
ActionEvent	getActionCommand();	get button label

```

btn1.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent ev)
    {
        JOptionPane.showMessageDialog
        (null,"You pressed button: " + ev.getActionCommand());
    }
});

```

### checkbox listeners

class	accessor methods	description
ItemEvent	getStateChange()	get state of checkbox 1 = not selected 2 = selected

```

chk1.addItemListener(new ItemListener()
{
    public void itemStateChanged(ItemEvent ev)
    {
        JOptionPane.showMessageDialog
        (null,"The checkbox is: " + ev.getStateChange() + " " +
        ((JCheckBox)ev.getSource()).getText());
    }
});

```

**radio button listeners**

class	accessor methods	description
ItemEvent	getStateChange();	get state of checkbox 1 = not selected 2 = selected

```

class MyRBListener implements ItemListener
{
    public void itemStateChanged(final ItemEvent ev)
    {
        if(ev.getSource() == rb1 && ev.getStateChange() == 1 )
            JOptionPane.showMessageDialog
            (null,"The radio button A is selected");
        else if(ev.getSource() == rb2 && ev.getStateChange() == 1)
            JOptionPane.showMessageDialog
            (null,"The radio button B is selected " );
        else if(ev.getSource() == rb3 && ev.getStateChange() == 1)
            JOptionPane.showMessageDialog
            (null,"The radio button C is: selected" );
    }
}
MyRBListener rbListener = new MyRBListener();

rb1.addItemListener(rbListener);
rb2.addItemListener(rbListener);
rb3.addItemListener(rbListener);

```

**combo box listener**

class	accessor methods	description
ItemEvent	getStateChange()	get state of checkbox 1 = not selected 2 = selected
JComboBox	getSelectedIndex()	returns selected item index

```

cb1.addItemListener(new ItemListener()
{
    public void itemStateChanged(ItemEvent ev)
    {
        JOptionPane.showMessageDialog
        (null,"You have selected: " + cb1.getSelectedItem());
    }
});

```

**list box listener**

class	accessor methods	description
JList	int getSelectedItem()	get selected item
JList	Object getSelectedValue()	get selected value
JList	Object[] getSelectedValues()	get an array of selected values
JList	setListData(Object[] objs)	set list data to this array of objects

```

// multi selected
list1.addListSelectionListener(new ListSelectionListener()
    {
        public void valueChanged(ListSelectionEvent ev)
        {
            Object[] obj = list1.getSelectedValues();
            String values="";
            for(int i=0;i<obj.length;i++)values = values + obj[i];
            JOptionPane.showMessageDialog
            (null,"You have selected: " + values);
        }
    });

// single selected
list2.addListSelectionListener(new ListSelectionListener()
    {
        public void valueChanged(ListSelectionEvent ev)
        {
            JOptionPane.showMessageDialog
            (null,"You have selected: " + list1.getSelectedValue());
        }
    });

```

The following list event handler uses the `getSelectedValues()` method to get all the selected values and then calls the `setListData()` method to set other list box.

```

// copying list boxes
btn3.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent ev)
        {
            list4.setListData(list3.getSelectedValues());
        }
    });

} // end SwingTest constructor

```

### main method

Hers the main function to run the program, which just calls the constructor to build the JFrame.

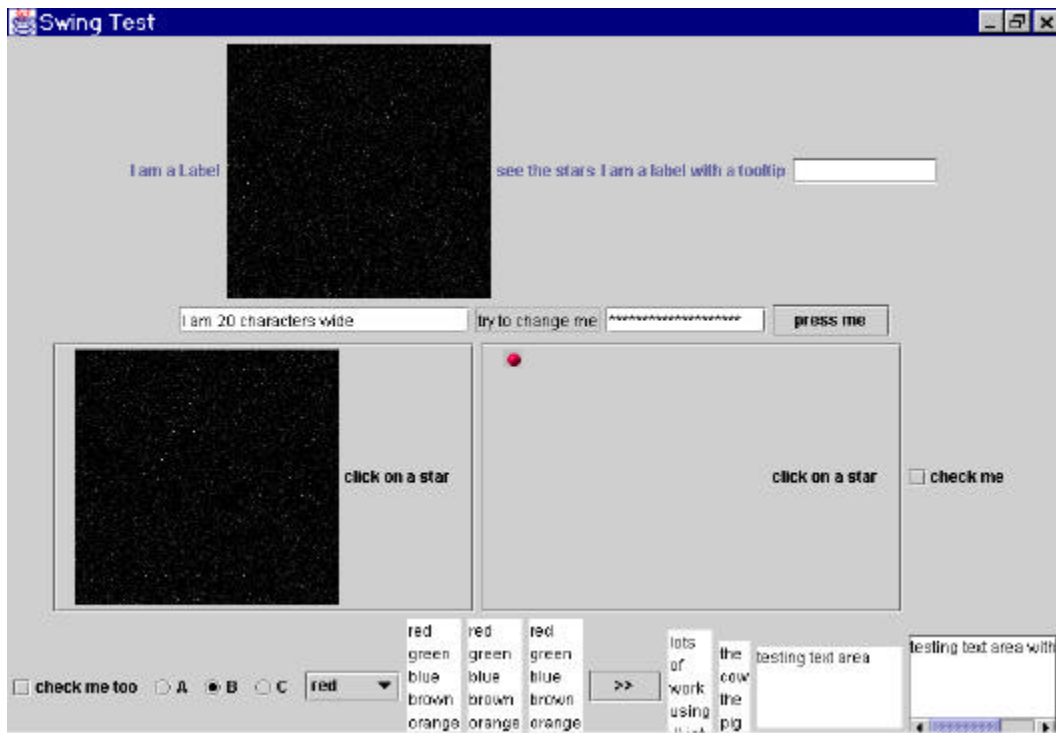
```

public static void main(String[] args)
{
    SwingTest st = new SwingTest();
    st.setSize(600,400);
    st.show();
    st.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
}

```

}

} // end class SwingTest



## SWING 1 EXERCISE 1

You need to make a GUI that allows someone to search, add, change, delete information. This can be inventory of items for sale, a list of employees, student names and marks etc. Use objects stored in Hashtables or Vectors to store the data. Write out all the data to files using object streams. Use a JList box or JTextArea to store the information. Use JButtons and JLabels that use icons. Use different Layout Managers or Horizontal and Vertical Boxes to align and position controls. If you want to you don't need to use a Layout Manager just hardcode dimensions and positions of the Swing components. Call your class Swing1Ex1.

search	add	change	delete	new
Name	<input type="text"/>			
Address	<input type="text"/>			
Phone	<input type="text"/>			
<input type="text"/>				

**IMPORTANT**

You should use all the material in all the lessons to do the questions and exercises. If you do not know how to do something or have to use additional books or references to do the questions or exercises, please let us know immediately. We want to have all the required information in our lessons. By letting us know we can add the required information to the lessons. The lessons are updated on a daily bases. We call our lessons the "living lessons". Please let us keep our lessons alive.

**E-Mail all typos, unclear test, and additional information required to:**

[courses@cstutoring.com](mailto:courses@cstutoring.com)

**E-Mail all attached files of your completed exercises to:**

[students@cstutoring.com](mailto:students@cstutoring.com)

**Order your next Lesson from:**

[www.cstutoring.com/aj.htm](http://www.cstutoring.com/aj.htm)

This lesson is copyright (C) 2000-2002 by The Computer Science Tutoring Center "www.cstutoring.com"  
This document is not to be copied or reproduced in any form. For use of student only.