JAVA NOTES

GRAPHICAL USER INTERFACES

Terry Marris July 2001

8 DROP-DOWN LISTS

8.1 LEARNING OUTCOMES

By the end of this lesson the student should be able to

- understand and use *JLists*
- understand and use combo boxes

8.2 INTRODUCTION

In the previous two lessons we looked at check boxes and radio buttons. The problem with check boxes and radio buttons is that they can take up a lot of space on the screen. A *JList* displays choices in a frame that can be scrolled through. A combo box displays just one choice; when it is selected all other choices are shown in a drop-down list. We look at combo boxes first.

8.3 COMBO BOXES

When a user clicks on the field, a list drops down from which the user can make a selection.

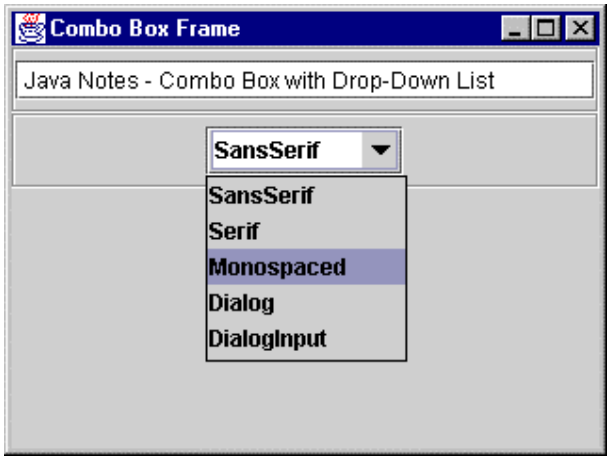

Figure 8.1 *A Combo Box*

When the user clicks on the *SansSerif* field, the list *SansSerif, Serif, Monospaced, Dialog, DialogInput* appears. If the user makes a selection, the text *Java Notes - Combo Box with Drop Down List* is displayed in the chosen font.

We create a new *JComboBox* instance and then add strings to it.

```
JComboBox fontComboBox = new JComboBox();
fontComboBox.addItem("SansSerif");
fontComboBox.addItem("Serif");
fontComboBox.addItem("Monospaced");
...
```

The current field, the one that is displayed is, by default, non-editable; its background colour is the same as its container. So we set it to white before adding the combo box to the container.

```
fontComboBox.setBackground(Color.white);
add(fontComboBox);
```

We register this panel to be the action listener for the combo box.

```
class ControlPanel extends JPanel implements ActionListener {

     ...

  public ControlPanel()
  {
     ...

    comboBoxPanel.getFontComboBox().addActionListener(this);
  }
```

Then retrieve the string that was selected by the user with a call to *getSelectedItem()*.

```
  public void actionPerformed(ActionEvent e)
  {
    JComboBox source = (JComboBox)e.getSource();
    String fontName = (String)source.getSelectedItem();
    textPanel.setTextFont(fontName);
  }
}
```

The complete program is shown below.

```
/* ComboBoxFrame.java
   Terry Marris 2 July 2001
*/

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


class TextPanel extends JPanel {
  private JTextField text;

  public TextPanel()
  {
    Border etched = BorderFactory.createEtchedBorder();
    setBorder(etched);

    text = new JTextField(
          " Java Notes - Combo Box with Drop-Down List ", 28);
    add(text);
  }


  public void setTextFont(String fontName)
  {
    text.setFont(new Font(fontName, Font.PLAIN, 12));
    repaint();
  }
}
```

```
class ComboBoxPanel extends JPanel {
  private JComboBox fontComboBox;

  public ComboBoxPanel()
  {
    Border etched = BorderFactory.createEtchedBorder();
    setBorder(etched);

    fontComboBox = new JComboBox();
    fontComboBox.addItem("SansSerif");
    fontComboBox.addItem("Serif");
    fontComboBox.addItem("Monospaced");
    fontComboBox.addItem("Dialog");
    fontComboBox.addItem("DialogInput");

    fontComboBox.setBackground(Color.white);
    add(fontComboBox);
  }


  public JComboBox getFontComboBox()
  {
    return fontComboBox;
  }
}
```

```
class ControlPanel extends JPanel implements ActionListener {
  private TextPanel textPanel;
  private ComboBoxPanel comboBoxPanel;

  public ControlPanel()
  {
    setLayout(new BorderLayout());

    textPanel = new TextPanel();
    add(textPanel, "North");

    comboBoxPanel = new ComboBoxPanel();
    add(comboBoxPanel, "South");

    comboBoxPanel.getFontComboBox().addActionListener(this);
  }


  public void actionPerformed(ActionEvent e)
  {
    JComboBox source = (JComboBox)e.getSource();
    String fontName = (String)source.getSelectedItem();
    textPanel.setTextFont(fontName);
  }
}
```

```
public class ComboBoxFrame extends JFrame {
  public ComboBoxFrame()
  {
    setTitle("Combo Box Frame");

    Toolkit tk = Toolkit.getDefaultToolkit();
    Dimension screen = tk.getScreenSize();
    setSize(screen.width/2, screen.height/2);
    setLocation(screen.width/4, screen.height/4);

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

    Container contentPane = getContentPane();
    contentPane.add(new ControlPanel(), "North");
  }


  public static void main(String[] s)
  {
    JFrame aFrame = new ComboBoxFrame();
    aFrame.show();
  }
}
```

Exercise: Explain each line of the program shown above.

8.4 LIST BOX

The *JList* component places a list of items inside a single box. An item is selected by clicking on it.

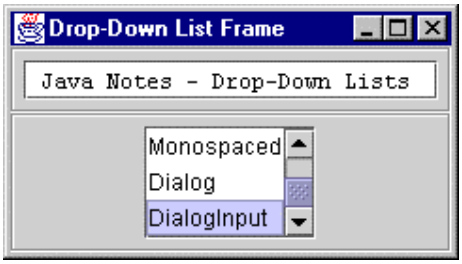

Figure 8.2 *A JListBox*

Figure 8.2 shows a drop down list. The list has too many entries to be shown in the space provided. A scroll bar enables you to scan up and down the entire list. As before, selecting a font changes the appearance of the text in the text box.

First, we create an array of objects. (Arrays are discussed in Part Two. An array is kind of container that holds objects in sequence, one after the other. The objects are indexed 0, 1, 2, ...) In our example, the objects are just strings.

```
String[] fontNameSet = { "Serif", "SansSerif",
                         "Monospaced",
                         "Dialog", "DialogInput" };
```

We create a *JList* instance and set the number of visible items to three.

```
JList fontList = new JList(fontNameSet);
fontList.setVisibleRowCount(3);
```

Now we create a *JScrollPane* instance; we provide our *JList* as a parameter to the *JScrollPane* constructor.

```
JScrollPane scrollPane = new JScrollPane(fontList);
```

And finally add the *scroll pane* (which contains the list) to its panel.

```
add(scrollPane);
```

We implement a *ListSelectionListener* (and not an *ActionListener*).

```
class ControlPanel extends JPanel
                   implements ListSelectionListener {
```

We register this panel as the *JList*'s *selection listener* (and not action listener).

```
  private ListPane listPanel;

     ...

    listPanel.getFontList().addListSelectionListener(this);
  }
```

We implement *valueChanged(ListSelectionEvent)* and not *actionPerformed(ActionEvent).*

```
  public void valueChanged(ListSelectionEvent e)
  {
```

We retrieve the object selected by a call to *getSelectedValue().*

```
    JList source = (JList)e.getSource();
    Object value = source.getSelectedValue();
```

Then we convert it to a string (since we know that we stored strings in the *JList* to begin with).

```
    String fontName = (String)value;
    textPanel.setTextFont(fontName);
```

The entire program is shown below.

```
/* ListFrame.java
   Terry Marris 2 July 2001
*/

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


class TextPanel extends JPanel {
  private JTextField text;

  public TextPanel()
  {
    Border etched = BorderFactory.createEtchedBorder();
    setBorder(etched);

    text = new JTextField(
                        " Java Notes - Drop-Down Lists ", 20);
    add(text);
  }


  public void setTextFont(String fontName)
  {
    text.setFont(new Font(fontName, Font.PLAIN, 12));
    repaint();
  }
}
```

```
class ListPanel extends JPanel {
  private JList fontList;

  public ListPanel()
  {
    Border etched = BorderFactory.createEtchedBorder();
    setBorder(etched);

    String[] fontNameSet = { "Serif", "SansSerif",
                             "Monospaced",
                             "Dialog", "DialogInput" };

    fontList = new JList(fontNameSet);
    fontList.setVisibleRowCount(3);
    JScrollPane scrollPane = new JScrollPane(fontList);

    add(scrollPane);
  }


  public JList getFontList()
  {
    return fontList;
  }
}
```

```
class ControlPanel extends JPanel
                   implements ListSelectionListener {
  private TextPanel textPanel;
  private ListPanel listPanel;

  public ControlPanel()
  {
    setLayout(new BorderLayout());

    textPanel = new TextPanel();
    add(textPanel, "North");

    listPanel = new ListPanel();
    add(listPanel, "South");

    listPanel.getFontList().addListSelectionListener(this);
  }


  public void valueChanged(ListSelectionEvent e)
  {
    JList source = (JList)e.getSource();
    Object value = source.getSelectedValue();

    String fontName = (String)value;
    textPanel.setTextFont(fontName);
  }
}
```

```
public class ListFrame extends JFrame {
  public ListFrame()
  {
    setTitle("Drop-Down List Frame");

    Toolkit tk = Toolkit.getDefaultToolkit();
    Dimension screen = tk.getScreenSize();
    setLocation(screen.width/4, screen.height/4);

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

    Container contentPane = getContentPane();
    contentPane.add(new ControlPanel());

    pack();
  }


  public static void main(String[] s)
  {
    JFrame aFrame = new ListFrame();
    aFrame.show();
  }
}
```

Exercise: Explain each line of the program shown above.

8.5 DOCUMENTATION

javax.swing.JList

Constructors	
JList(Object[] items)	initialises a JList that displays the given items.

Methods		
void	setVisibleRowCount(int c)	sets the number of rows that can be displayed without a scroll bar.
void	setSelectionMode(int mode)	determines whether single or multiple-item selections are allowed. mode is one of SINGLE_SELECTION, SINGLE_INTERVAL_SELECTION MULTIIPLE_INTERVAL_SELECTION
void	addListSelectionListener(ListSelectionListener l)	adds a listener that is notified each time a change in the list's selection is made.
Object[]	getSelectedValues()	returns the selected values or an empty array if the selection is empty.
Object	getSelectedValue()	returns the first selected value or null if the selection is empty.

javax.swing.event.ListSelectionListener

methods		
void	valueChanged(ListSelectionEvent e)	called whenever the list selection changes

javax.swing.JComboBox

Methods		
void	setEditable(boolean b)	b is true if the combo box field can be edited by the user, false otherwise.
void	addItem(Object item)	adds the given item to the list.
void	insertItemAt(Object item, int index)	inserts the given item at the given index position in the list.
void	removeItem(Object item)	removes the given item.
void	removeItemAt(int index)	removes the item at the given index.
void	removeAllItems()	removes all items from the item list.
Object	getSelectedItem()	returns the currently selected item.

8.6 FURTHER READING

HORSTMANN & CORNELL *Core Java 2 Volume 1* pp 420, 438
www.java.sun.com/docs/books/tutorial

In the next lesson we look at layout managers.

8.7 REVIEW

8.8 EXERCISES

1 Try out the *ComboBoxFrame.java* program shown in §8.3 above.

2 Try out the *ListFrame.java* program shown in §8.4 above.