Java Help Files

by
Peter Lavin

May 22, 2004

Overview

Help screens are a necessity for making any application user-friendly. This article will show how the JEditorPane and JFrame classes, along with HTML files, can be used to create help windows for any Java application. This is a project-oriented article that will walk the reader through all the code needed for development. Some familiarity with Java is assumed.

Introduction

HTML files are often used to display help for desktop applications. For example, the help files that accompany many Windows applications are usually made up of compiled HTML files. Using HTML with Java is relatively easy because Java has built-in HTML capabilities. You can control the way text appears in components by using HTML coding. For example, if you want to bold the text of a JButton the following code will do this for you:

```
JButton mybutton = new JButton();
mybuttom.setText("<html><b>Press Me</b></html>");
```

Given that Java has this innate capability it is not difficult to develop a class that will display help files in HTML format. In this article we will take you through the steps necessary to create such a class and, because the actual help files will be independent of the application, you will be able to use this class with any Java application.

Before looking at the code let's clearly state what we want our class to do. It will provide help information for any application by browsing local HTML files. To keep things simple we will navigate using a home page that is a list of hyperlinks to specific help files. Pages may exceed the size of the screen so our application will need to be able to scroll.

The Code

Find below a complete listing of the code in our class. Have a quick look at it now but don't worry if you don't understand all the details. Some basic knowledge of Java is assumed so not all the lines of code will be discussed. Relevant sections will be explained in detail.

```
 1://////////////////////////////////////////////////////////////////
 2:/**
 3:* This class creates a frame with a JEditorPane for loading HTML
 4:* help files
 5:*/
 6://package goes here
 7:import java.io.*;
 8:import javax.swing.event.*;
 9:import javax.swing.*;
10:import java.net.*;
11:import java.awt.event.*;
12:import java.awt.*;
13:
14:public class HelpWindow extends JFrame implements ActionListener{
```

```
   15:    private final int WIDTH = 600;
   16:    private final int HEIGHT = 400;
   17:    private JEditorPane editorpane;
   18:    private URL helpURL;
   19:////////////////////////////////////////////////////////////////
   20:/**
   21: * HelpWindow constructor
   22: * @param String and URL
   23: */
   24:public HelpWindow(String title, URL hlpURL) {
   25:    super(title);
   26:    helpURL = hlpURL;
   27:    editorpane = new JEditorPane();
   28:    editorpane.setEditable(false);
   29:    try {
   30:        editorpane.setPage(helpURL);
   31:    } catch (Exception ex) {
   32:        ex.printStackTrace();
   33:    }
   34:    //anonymous inner listener
   35:    editorpane.addHyperlinkListener(new HyperlinkListener() {
   36:        public void hyperlinkUpdate(HyperlinkEvent ev) {
   37:            try {
   38:                if (ev.getEventType() ==
HyperlinkEvent.EventType.ACTIVATED) {
   39:                    editorpane.setPage(ev.getURL());
   40:                }
   41:            } catch (IOException ex) {
   42:                //put message in window
   43:                ex.printStackTrace();
   44:            }
   45:        }
   46:    });
   47:    getContentPane().add(new JScrollPane(editorpane));
   48:    addButtons();
   49:    // no need for listener just dispose
   50:    setDefaultCloseOperation(DISPOSE_ON_CLOSE);
   51:    // dynamically set location
   52:    calculateLocation();
   53:    setVisible(true);
   54:    // end constructor
   55:}
   56:/**
   57: * An Actionlistener so must implement this method
   58: *
   59: */
   60:public void actionPerformed(ActionEvent e) {
   61:    String strAction = e.getActionCommand();
   62:    URL tempURL;
   63:    try {
   64:        if (strAction == "Contents") {
   65:            tempURL = editorpane.getPage();
   66:            editorpane.setPage(helpURL);
   67:        }
   68:        if (strAction == "Close") {
```

```
  69:             // more portable if delegated
  70:             processWindowEvent(new WindowEvent(this,
  71:                 WindowEvent.WINDOW_CLOSING));
  72:         }
  73:     } catch (IOException ex) {
  74:         ex.printStackTrace();
  75:     }
  76:}
  77:/**
  78: * add buttons at the south
  79: */
  80:private void addButtons() {
  81:     JButton btncontents = new JButton("Contents");
  82:     btncontents.addActionListener(this);
  83:     JButton btnclose = new JButton("Close");
  84:     btnclose.addActionListener(this);
  85:     //put into JPanel
  86:     JPanel panebuttons = new JPanel();
  87:     panebuttons.add(btncontents);
  88:     panebuttons.add(btnclose);
  89:     //add panel south
  90:     getContentPane().add(panebuttons, BorderLayout.SOUTH);
  91:}
  92:/**
  93: * locate in middle of screen
  94: */
  95:private void calculateLocation() {
  96:     Dimension screendim =
Toolkit.getDefaultToolkit().getScreenSize();
  97:     setSize(new Dimension(WIDTH, HEIGHT));
  98:     int locationx = (screendim.width - WIDTH) / 2;
  99:     int locationy = (screendim.height - HEIGHT) / 2;
 100:     setLocation(locationx, locationy);
 101:}
 102:public static void main(String [] args){
 103:     URL index = ClassLoader.getSystemResource("index.html");
 104:     new HelpWindow("Test", index);
 105:
 106:}
 107:}//end HelpWindow class
 108:////////////////////////////////////////////////////////////////
```

The HelpWindow Class

The first thing to notice about our class is that it extends JFrame (line 14). We need not have done things this way. We could simply have included an instance of the JFrame class as a data member. However, we want our class to have all the functionality of the JFrame class. By using inheritance we will be able to directly change the title or the size of our frame by using the parent methods, “setTitle” and “setSize”. After all, this is the whole point of object-oriented languages.

ActionListener

Our class will also implement the interface ActionListener (line 14) so that it can easily react to events, principally mouse clicks. This is the most commonly used listener and requires that we implement the “actionPerformed” method.

Lines 60 through 76 implement this method and process button clicks. This method will be dealt with in detail shortly.

Data Members & Constructor

Only four data members are included in our class (Lines 15 through 18). Two are simple integers used to set the size of the frame. These variables are declared as “final” and will be used as default values. Using variables instead of literals makes for easier code maintenance and declaring them “final” means that they cannot be changed.

The two remaining data members are URL and JEditorPane objects respectively. An URL is fairly self-explanatory, it is constructed from an HTML page in the same directory as our application, but the JEditorPane is a bit more interesting.

However, before moving on to a discussion of JEditorPane a few comments about the constructor (lines 24 - 55). are in order. The constructor accepts two arguments, a title for the application and a home page for our help files. The first line of the constructor code (line 25) is a call to the parent constructor in order to set the title of our application. The home page should be a list of hyperlinks to specific help files, although for testing purposes any HTML page will do. This URL is assigned to a class variable.

JEditorPane Class

This is the class that will be our help file browser. It can handle content formatted as text, rich text or HTML. We don’t have to do anything special to enable it to handle HTML files. As the Sun online tutorial says, “It effectively morphs into the proper kind of text editor for the kind of content it is given”. This happens on line 30 where the content is set to an URL.

One important thing that we do need to do with this component is add a HyperTextListener. Lines 35 – 46 create an anonymous inner listener of this class so that information is updated if a hyperlink is clicked. This listener simply

calls the “setPage” method to change URLs. If you need more information about listeners see the article “Listeners In Java” also found on this website.

The JEditorPane class is a component that gets added to our main, JFrame-derived class. This happens on line 47. Because our parent class is a JFrame it comes with a BorderLayout as its default layout manager. When an item is added to a JFrame’s contentpane its default location will be at the centre of the JFrame and this is exactly what we want.

The actionPerformed Method

Buttons are added to our application in the addButtons method (starting on line 80), and the application itself is added as an ActionListener to both buttons. All this means is that our buttons are able to react to events. The code that processes events is the “actionPerformed” method (line 60 and following).

Our “Contents” button functions as a Home Page button by returning to the list of hyperlinks to specific help files.

The “Close” button simply disposes of our help window but perhaps a few comments are in order. The line:

```
processWindowEvent(new WindowEvent(this, WindowEvent.WINDOW_CLOSING));,
```

could just as easily have been replaced with,

```
dispose();
```

However, creating a window closing event makes for more robust code. If we decide to add a window listener that handles the window closing event then we will ensure that the same code will execute regardless of whether the user shuts down the application from the title bar or by pressing the “Close” button. In other words we will handle the closing event in one location only.

It is also worth noting that our help window is disposed of when it is closed. This behaviour is set on line 50. It is important that the default close operation not be set to “`EXIT_ON_CLOSE`” because the class described here is an ancillary class and closing it should not end the application. Also, with this in mind, don’t forget to remove the “main” method before you incorporate this class into another class. The “main” method is included here simply for testing purposes.

Enhancements

The class we've created is a very basic use of JEditorPane to display HTML help files. There are numerous ways in which it could be improved and I will make a few suggestions here. Buttons to navigate to previously visited URLs might be helpful. A menubar might provide a more elegant means of navigation.

Remember also that this class retains all the functionality of its parent. If you are not satisfied with the size of the window you can simply use the "setSize" method inherited from the JFrame class. Likewise with methods such as "setBackground". Furthermore, the functionality of this class could be extended by deriving other classes from it. That said, it is best to remember that Java only supports HTML version 3.2 and this fact will certainly restrict what can be achieved.

Finally, the way in which exceptions are handled could be improved. In a follow-up article we will show how to construct a generic class to handle exceptions thrown by any application.

To summarise, we have presented a class that creates a basic help window and can be incorporated into any Java application. In so doing we have achieved one of the goals of object-oriented programming – namely creating a reusable class.

About the Author

Peter Lavin runs a Web Design/Development firm in Toronto, Canada. For more information visit http://www.softcoded.com. He may be reached at peterlavin@sympatico.ca.