Journal of Information Systems Education, Vol. 17(4)

Teaching Tip

Object-Oriented Programming Principles and the Java Class Library

Thomas P. Cavaiani
Department of Networking, Operations, and Information Systems
College of Business and Economics
Boise State University
Boise, Idaho 83725
tcavaiani@boisestate.edu

ABSTRACT

For novices, learning an object-oriented programming language can be a daunting task. Not only do students need to learn basic programming concepts, but they are also confronted with object modeling concepts as well. Learning Java presents an additional difficulty. Students must learn how to use the Java Class Library to locate the details of classes, methods, and toolkits that they can use in their own classes. One of my primary goals in teaching Java to novices is helping them acquire an understanding of a specific subset of tools in the Java Class Library. This goal is addressed by emphasizing the use of inheritance and a specially designed set of exercises. To demonstrate this approach, this paper will outline the creation of a simple text editor. This example illustrates how considerable functionality can be added to complex programs by using existing classes documented in the Java Class Library.

Keywords: Object-Oriented Programming, Guided Instruction, Java Programming, Class Libraries

1. INTRODUCTION

Novice programming students have minimal difficulty gaining a conceptual understanding of Object-Oriented Programming (OOP) principles such as code reuse, inheritance, and overloading, but typically experience considerable difficulty applying these principles when coding. Over the last four semesters I have devised a list of objectives that address the role that OOP principles, such as code reuse and inheritance, play in searching the Java Class Library. These objectives are as follows:

- Assist students in learning how to apply the concept of inheritance to practical examples.
- Assign practical examples that require students to use and extend existing classes.
- Deliver specific instruction designed to familiarize students with the packages documented by the Java Class Library.
- Assist students in learning searching techniques for locating specific classes and methods in the Java Class Library.

These objectives have led to the development of specific teaching methods and exercises used in my introductory Java programming course. The teaching methods address how to appropriately structure the learning environment, create programming exercises that pertain to "real world" applications, and provide students with a guided approach that helps them discover how to solve problems. A brief discussion of the basis for this approach follows.

Several researchers have discussed the need to appropriately structure the learning environment (Bandura, 1977; Bruner, 1966; Bruner, 1986; Vygotsky, 1978). Continually adjusting the level of help in response to a learner's level of performance not only produces immediate results, but also instills individuals with the skills necessary for independent problem solving later on (Vygotsky, 1978). Bandura (1977) indicates that most human behavior is learned observationally through modeling. By observing others one learns how new behaviors are performed. Later this coded information is used by an individual to as a guide for action. A major theme described by Bruner (1966) is that learning is an active process in which learners construct new ideas or concepts based upon their current and/or past knowledge. The learner selects and transforms information, constructs hypotheses, and makes decisions, relying upon schemas and mental models, to provide meaning and organization to experiences that allows the individual to go beyond the given information. With regard to a guided approach to instruction, Bruner (1966, 1986) indicates that instructors should try and

encourage students to discover principles by themselves. Curriculum should be organized in a spiral manner so that students continually build upon what they have already learned. With regard to programming the research indicates that individuals typically do not learn just to program, but that they learn to program *something* (Adelson and Soloway, 1990; Harel and Papert, 1985). This conclusion implies that students will be more successful in learning to program if programming concepts are presented in the context of practical examples.

2. PROGRAMMING EXERCISES

Six programming assignments are assigned during the semester in my introductory Java programming course. The first four are practical examples that cover topics on code reuse, creating an applet, graphical user interfaces, event-driven programming, decision-making, looping, arrays and maps, file creation, and data input and output. The first program is an animation program built completely from existing classes provided to students from a sample program discussed early in the semester. The second exercise requires creation an applet that displays a digital clock. Exercises three and four require students to create an airline ticketing system to assign seats to passengers, and then write a seat map to secondary storage. Exercises five and six pertain to creating a simple text editor; exercise five focuses on creating the graphical user interface (GUI) while exercise six adds functionality to the GUI, including opening and saving files, and editing and formatting text.

Each of these assignments pertains to practical examples that most students have become familiar with before entering this course. The Airline booking program and text editor exercises have been designed to provide students with insights into how real-world programs are coded. The simple text editor is designed specifically to introduce students to the use high-level objects in their programs as described in the Java Class Library, and is the focus of the remainder of this paper. Complete descriptions of these exercises can be found on the on-line course website at http://cispom.boisestate.edu/cis125tcavaiani/cis125crsotl.htm

3. BUILDING A SIMPLE TEXT EDITOR

The simple text editor project helps students learn how to create a functional word processor with a graphical user interface. The Java Class Library contains packages that facilitate building the GUI from objects such as frames, menus, dialog boxes, buttons, and drop down lists. After creating the GUI, students add *ActionListeners* to these GUI objects to "listen" for user-generated events during program execution, and code methods that act upon the events.

The Java Class Library (http://java.sun.com/j2se/1.5.0/docs/api/) is an extensive set of documentation for all the Java packages that have been developed by the Sun Corporation. Without some guidance students would have no idea which library class to use to create the GUI. The *JFrame* class provides the required functionality. The following set of guidelines for searching the Java Class Library is demonstrated to students:

1. Select the *All Classes* pane in the Java Class Library window.
2. Select the *Find on this page...* option from the browser *Edit* menu. The *Find* dialog box appears.
3. Type jframe in *Find what* text area of the *Find* dialog box.
4. Click on the *JFrame* class name that appears at the bottom of the *All Classes* pane. Documentation for the JFrame class appears in the *main* pane (See Figure 2).

3.1 Creating the Interface

The GUI for the simple text editor is modeled after MS Notepad (See Figure 1).

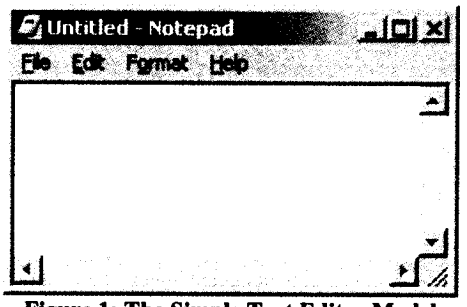


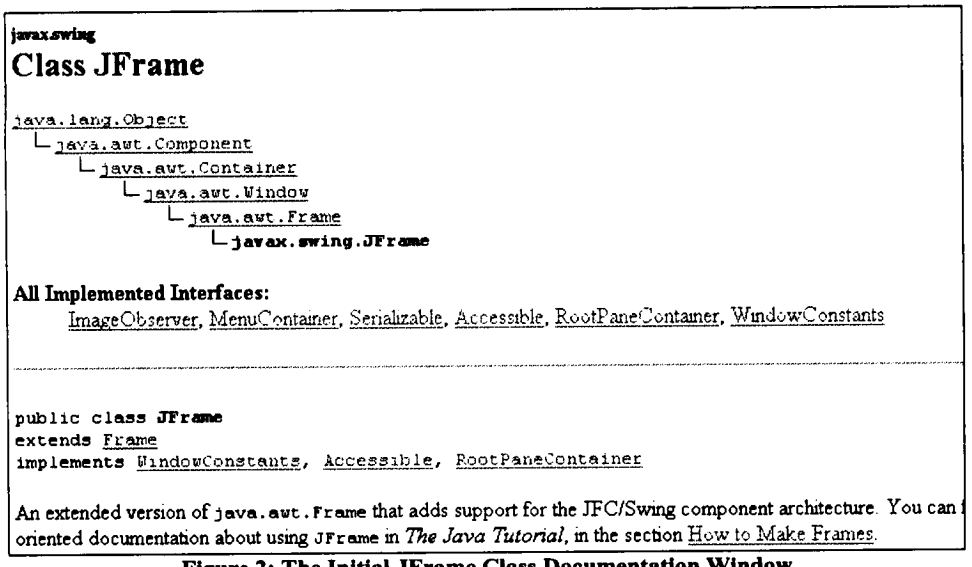
Figure 1: The Simple Text Editor Model

Considerable information is provided by this page. The class hierarchy displays the list of parent classes from which JFrame inherits functionality. Students are encouraged to investigate the *Component* and *Container* classes, to become familiar with methods that they make available to the JFrame class. They are also encouraged to investigate the *MenuContainer* link to determine which classes implement this interface (See Figure 3). Lastly they are encouraged to investigate the *How to Make Frames* tutorial, which provides a good starting point for developing a GUI. Students quickly become aware that many of the classes required for the simple text editor (JFileChooser, JMenu, JMenuItem, and JTextArea) are listed as classes that implement the MenuContainer interface.

The GUI window is created using a *JFrame*. Editing functionality is obtained by adding a *JTextArea* to the frame. A menu bar, menus, and menu items are created using the intuitively named *JMenuBar*, *JMenu*, and *JMenuItem* classes. Methods to *add* menus and menu items to the JFrame, and to make the JFrame object visible (*setVisible*) are all easily located using the *Edit Find* option described earlier. These methods are found in either the *Component* or *Container* classes, so it is important to emphasize to students that extend their search beyond the Method Summary (See Figure 5) for a given class.

3.2 Adding Editing Functionality to the GUI

Upon completion of the graphical user interface, students will have a text editor with numerous menu options, including an Edit menu that includes Cut, Copy, and Paste options. Adding functionality for cut, copy, and paste is a trivial task, once students discover that JTextArea inherits functionality from the *JTextComponent* class (See Figure 4).



javax.swing

Class JFrame

java.lang.Object
  └ java.awt.Component
    └ java.awt.Container
      └ java.awt.Window
        └ java.awt.Frame
          └ **javax.swing.JFrame**

All Implemented Interfaces:

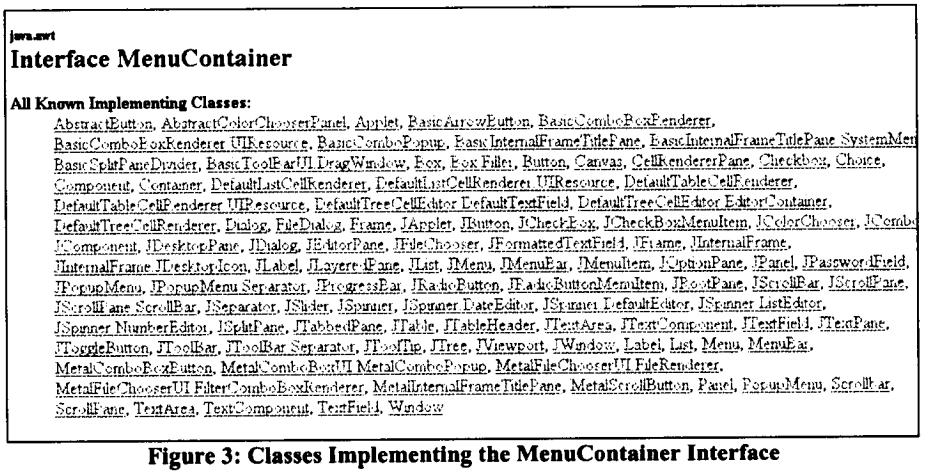
ImageObserver, MenuContainer, Serializable, Accessible, RootPaneContainer, WindowConstants

---

```
public class JFrame
extends Frame
implements WindowConstants, Accessible, RootPaneContainer
```

An extended version of `java.awt.Frame` that adds support for the JFC/Swing component architecture. You can f
oriented documentation about using `JFrame` in *The Java Tutorial*, in the section How to Make Frames.

Figure 2: The Initial JFrame Class Documentation Window



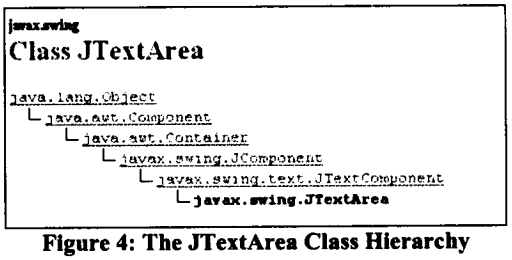
java.awt

Interface MenuContainer

All Known Implementing Classes:

AbstractButton, AbstractColorChooserPanel, Applet, BasicArrowButton, BasicComboBoxRenderer,
BasicComboBoxRenderer.UIResource, BasicComboPopup, BasicInternalFrameTitlePane, BasicInternalFrameTitlePane.SystemMen
BasicSplitPaneDivider, BasicToolBarUI.DragWindow, Box, Box.Filler, Button, Canvas, CellRendererPane, Checkbox, Choice,
Component, Container, DefaultListCellRenderer, DefaultListCellRenderer.UIResource, DefaultTableCellRenderer,
DefaultTableCellRenderer.UIResource, DefaultTreeCellEditor.DefaultTextField, DefaultTreeCellEditor.EditorContainer,
DefaultTreeCellRenderer, Dialog, FileDialog, Frame, JApplet, JButton, JCheckBox, JCheckBoxMenuItem, JColorChooser, JComb
JComponent, JDesktopPane, JDialog, JEditorPane, JFileChooser, JFormattedTextField, JFrame, JInternalFrame,
JInternalFrame.JDesktopIcon, JLabel, JLayeredPane, JList, JMenu, JMenuBar, JMenuItem, JOptionPane, JPanel, JPasswordField,
JPopupMenu, JPopupMenu.Separator, JProgressBar, JRadioButton, JRadioButtonMenuItem, JRootPane, JScrollBar, JScrollPane,
JScrollPane.ScrollBar, JSeparator, JSlider, JSpinner, JSpinner.DateEditor, JSpinner.DefaultEditor, JSpinner.ListEditor,
JSpinner.NumberEditor, JSplitPane, JTabbedPane, JTable, JTableHeader, JTextArea, JTextComponent, JTextField, JTextPane,
JToggleButton, JToolBar, JToolBar.Separator, JToolTip, JTree, JViewport, JWindow, Label, List, Menu, MenuBar,
MetalComboBoxButton, MetalComboBoxUI.MetalComboPopup, MetalFileChooserUI.FileRenderer,
MetalFileChooserUI.FilterComboBoxRenderer, MetalInternalFrameTitlePane, MetalScrollButton, Panel, PopupMenu, Scrollbar,
ScrollPane, TextArea, TextComponent, TextField, Window

Figure 3: Classes Implementing the MenuContainer Interface

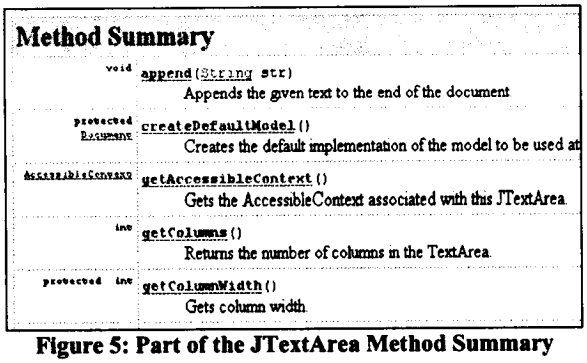


javax.swing

Class JTextArea

java.lang.Object
  └ java.awt.Component
    └ java.awt.Container
      └ javax.swing.JComponent
        └ javax.swing.text.JTextComponent
          └ **javax.swing.JTextArea**

Figure 4: The JTextArea Class Hierarchy



Method Summary

void	**append**(String str) Appends the given text to the end of the document
protected Document	**createDefaultModel**() Creates the default implementation of the model to be used at
AccessibleContext	**getAccessibleContext**() Gets the AccessibleContext associated with this JTextArea.
int	**getColumns**() Returns the number of columns in the TextArea.
protected int	**getColumnWidth**() Gets column width.

Figure 5: Part of the JTextArea Method Summary

Searching the *JTextArea Method Summary* (See Figure 5) does not reveal cut, copy, or paste methods. (Method names are in alphabetical order).

Further searching of the class hierarchy indicates that cut, copy, and paste methods are available in *JTextComponent* class (See Figure 6).

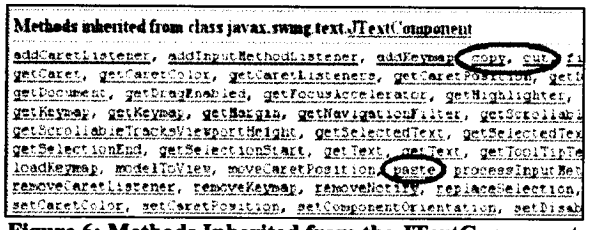


Figure 6: Methods Inherited from the JTextComponent Class

3.3 Adding a File Dialog to the GUI

Implementing File-Open and File-Save is nearly as easy. A File Chooser tutorial is available at http://java.sun.com/docs/books/tutorial/uiswing/components/filechooser.html. This link (also available on the course web site) describes the use of predefined dialog boxes that can be used to display File-Open and File-SaveAs dialog boxes.

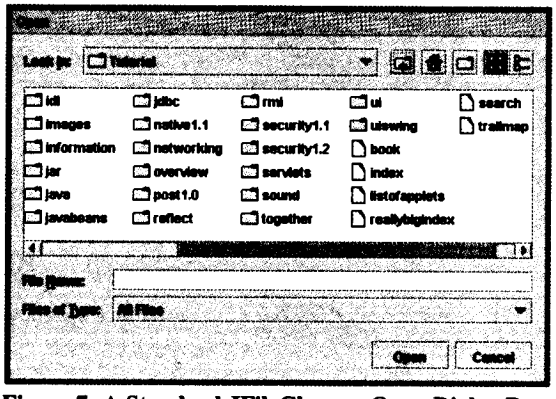


Figure 7: A Standard JFileChooser Open Dialog Box

Bringing up a standard open dialog (See Figure 7) requires only the following two lines of code (the first to create a file choose object and the second to associate the dialog box object with a File-Open menu click action):

```
JFileChooser fc = new JFileChooser();
...
int returnValue = fc.showOpenDialog(frame);
```

Displaying a SaveAs dialog box is similar. A statement like *File file = fc.getSelectedFile();* can be used to associate the file selected in the dialog box with an actual file on disk. Code to open and read the file can be included to transfer the contents of the file into a *JtextArea* frame.

4. CONCLUSION

This paper has provided an overview of the types of exercises students participate in during the semester. It attempts to illustrate the role of the Java Class Library in helping programmers create their own programs, and how inheritance affects the search techniques that students must learn to be successful in obtaining information necessary to locate and reuse classes documented in the library.

Providing instruction for incorporating use of the Java Class Library into an introductory Java programming course can be quite valuable to students as they attempt to gain an understanding of Object Oriented Programming principles such as inheritance and code reuse. Based upon research in learning theory, novice student programmers will not discover these techniques without some guidance. An understanding of these principles and the ability to search for and locate existing packages can greatly streamline the programs they write and save them from "reinventing the wheel". If this is the case students will gain confidence in their ability to write fairly complex programs knowing that much of the hard work has been done for them and that documentation is available that explains how to incorporate these classes into the programs they create.

5. REFERENCES

Adelson, B. and Soloway, E. (1985), The Role of Domain Experience in Software Design. IEEE Transactions on Software Engineering SE-1, no.11, pp.1351-1360.

Bandura, A. (1977), Social Learning Theory. General Learning Press, New York.

Bruner, J. (1966), Toward a Theory of Instruction, Harvard University Press, Cambridge, MA.

Bruner, J. (1986), Actual Minds, Possible Worlds. Harvard University Press, Cambridge, MA.

Harel, I. and Papert, S. (1990), "Software Design as a Learning Environment." Interactive Learning Environments, Vol. 1, no. 1, pp. 1-32.

Vygotsky, L.S. (1978), Mind in Society: The Development of Higher Psychological Processes. Harvard University Press, Cambridge, MA.

AUTHOR BIOGRAPHY

Thomas P. Cavaiani received his Ph.D. in Mathematics Education from Oregon State University in 1988. Currently, he teaches in the Department of Networking, Operations, and Information Systems at Boise State University. He has published in the *American Technical Education Association Journal*, the *Journal of Research on Computing in Education,* and the *Journal of Information Systems Education*. His teaching interests include Java programming and computer networking.

Information Systems & Computing
Academic Professionals

STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the Journal of Information Systems Education have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

ISSN 1055-3096