Animation of Java Linked Lists

Herbert L. Dershem and Ryan L. McFall
Department of Computer Science
Hope College
Holland, MI 49423
{dershem mcfall}@cs.hope.edu}

Ngozi Uti
Northern Kentucky University
Highland Heights, KY 41099
uti@nku.edu

Abstract

Linked lists are an important component of the computer science curriculum. JVALL is a software package that provides an animation of linked list operations that is fully compatible with the Java LinkedList class. The animations are driven by a client program that can be either an applet or standalone application. It provides an effective way for students to learn, experiment with, and debug linked list based classes.

1 Introduction

One of the most popular models for the implementation of data structures is the linked list. This model and its associated operations are an important topic in the traditional data structures course. The linked list operations are virtually always introduced through a visualization, both in textbooks and on chalkboards. This paper describes a useful tool that has been developed that enables linked list structures to be visualized through animations of the standard operations.

The presentation of algorithm visualizations on the web has many advantages, particularly in distance education settings. This has been the focus of much recent work. [1] [3] [4] [5] Animations that can be generated and controlled by students have also been shown to be effective in instruction. [7] Another technique that has proven to be useful is the visualization of pre-existing code to enhance student understanding. [2]

The present work is intended to provide an effective tool to support student learning and understanding of linked list operations as well as assist students in understanding applications that use linked lists. It is fully compatible with the Java LinkedList class and can be used by client programs that are either Java standalone applications or applets. A general-purpose client program is included that provides an effective user interface for the visualization of all linked list operations under user control.

SIGCSE'02, February 27- March 3, 2002, Covington, Kentucky, USA.

2 Design Objectives

The objective of this project is to produce a linked list visualization tool with the following capabilities:

User-controlled animation of basic linked list algorithms

Operations are animated at a speed that is controlled by the user and allows the user to observe the construction and modification of nodes in the list. Likewise the user determines the implementation model for the linked list, the context where the operations are called, and the colors of the components of the animation.

Animation of pre-existing code that uses linked lists

The animation class provided is an extension of the Java LinkedList class. Therefore, any program that uses the Java class is easily modified to enable its linked list operations to be animated.

Support for multiple linked list implementation models

The present version supports both linear and circular singly-linked lists. Future versions will add support for doubly linked lists and header node lists as well.

Animations within both Java applets and applications

The tool is easily applied in both of these environments via a parameter passed when the animated linked list object is constructed.

3 Design and Features

The package that provides the linked list animation is called JVALL (Java Visual Automated Linked List). Important features of this tool are described below.

Visualization

The JVALL animation window is shown in Figure 1. The window is divided into three areas. The top area contains controls that allow the user to modify the colors, speed, and implementation model of the animation. The middle area is the animation display that contains the nodes in the list. List nodes contain data and reference components. The data component displays the String representation of the node

data object as given by the application of the Java toString method. The reference component contains the source of a pointer from the node to the next node in the linked structure. The bottom area contains text reporting the animation status and buttons to undo and redo operations.

Figure 3 is a snapshot taken in the midst of an insert animation. The inserted node ("Frank O") is moved into place after new references have been assigned. The discarded reference between the previous and succeeding nodes remains beneath the inserted node and is marked as such while the inserted node is being moved into place. This reference disappears upon completion of the animation. Similar animations exist for all Java LinkedList class operations.

Due to the limitations on screen sizes, most animations restrict their users to input only digits or a limited number of characters for display. Instead of limiting user inputs, and in order to be consistent with the flexible nature of the linked list data structure, the "click, hold to expand" node was used. JVALL's node is capable of displaying lengthy values by displaying an abbreviated text with dots at the end to indicate that it contains more characters than can be displayed in the basic code rectangle. To view the entire text, the user can click and hold on a node with a lengthy value.

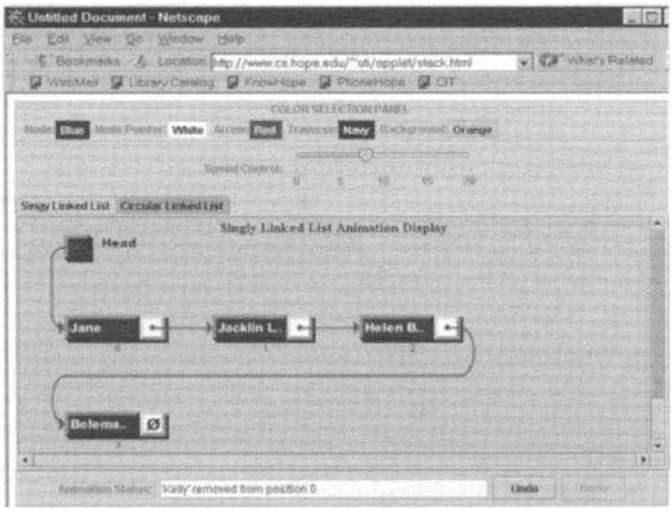

Figure 1

Compatibility with Java LinkedList Class

JVALL was designed to be an animation that can be used for visualization of existing Java programs as well as for teaching and learning the linked list data structure. For easy adaptation by existing Java programs, JVALL is fully compatible with the Java LinkedList class. Each of JVALL's methods animates and carries out the corresponding Java LinkedList method.

JVALL can be used to animate a program by simply replacing the references to the Java LinkedList class with references to Jvall. The remainder of the code is left unchanged. JVALL is designed be driven by a client program that propagates the animation display. "Client" program in this paper refers to a user supplied program using JVALL for its linked list animation. In [8], Stasko and Lawrence affirm that animations are most effective when students are actively engaged. For example, the client program should provide an avenue for a user to type in values to be inserted into the linked list. The code segment below demonstrates how JVALL can be used to animate an existing program.

Existing program:

```
LinkedList mylist = new LinkedList();
```

JVALL animated client program:

```
Jvall mylist = new Jvall(Jvall.STANDALONE);
```

In the code segment above JVALL will appear as a standalone application that will respond to animation calls from the client program. JVALL will provide automatic linked list visualization as the client program requests operations. Using a different parameter in the constructor call enables JVALL to be used in an applet.

Linear and Circular Models

By making a selection on a menu bar, the user can change the visualized model at any point during the animation between a linear and circular list. Figure 2 shows JVALL's view of a circular linked list.

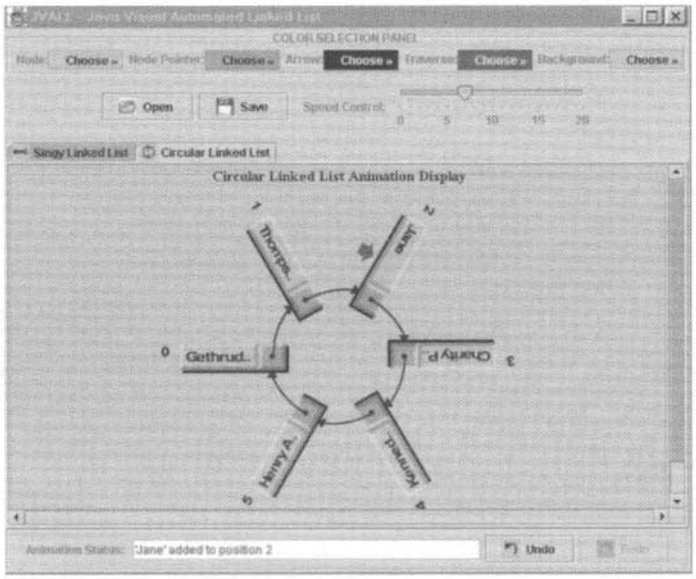

Figure 2

Since the linked list operations are independent of the model used for representation, any list can be viewed with either model. When in the mode to view one model,

operations are animated in a manner consistent with that model.

Animation Status Report

At the bottom of the JVALL window is an animation status report. In this text field, the user can read a description of the operation that is currently being animated and receive reports on any exceptions that are thrown during the operation.

Undo/Redo Capability

A client that uses JVALL automatically receives a set of visualization controls, including unlimited multiple undo's and redo's, speed control, dynamic status updates, and full color customization. A user can utilize the multiple undo's and redo's to rewind and review previous stages of the animation for better understanding. This capability is most appropriate with an interactive client that waits for the user to enter requested operations on the linked list. When used during non-interactive applications, unpredictable actions may result.

File load and save

When declared as standalone, JVALL provides the ability to save or open a text file in order to construct the visualized linked list. The save and open options are not provided when JVALL is used in an applet because of Java security restrictions.

The open and save buttons can be observed in Figure 2. These buttons, when clicked, open a file chooser window allowing the user to load or save the linked list as a text file. The file chooser window has provision for the user to select and view the content of a text file before loading it into the linked list. Only text files with the ".txt" extensions can be loaded. JVALL loads a text file by reading each line as a node into the linked list. The populated linked list is then displayed on the screen.

Linked list operations, such as insert, delete, replace etc., can be carried out on the loaded linked list. While saving, JVALL stores each node as a line in the text file.

While this feature presently saves and reads the nodes as a text file, a future improvement to this software will permit any serialized objects to be saved and restored.

Color Selection

JVALL provides smooth animation and quality graphics that are fully customizable by the user. The default settings are designed to highlight important aspects of the animation. Several colors are used to distinguish among components of the animation including nodes, node pointers, arrows, traversing arrow and background. The color selection panel is designed so that color selections automatically display text in contrasting colors, ensuring that the content of the node is always properly displayed.

Speed Control

The user has the ability to control the speed of the animation. This permits a student to slow down animations to follow each step when first learning the operation, and to speed them up when the process is familiar but the visualization of the result is important.

Applet and Standalone Application

Education today has gone beyond regular classroom instruction to harness the versatility and availability of the Internet. Many students and teachers now use the Internet for learning and teaching purposes. JVALL is well suited for use on web browsers since it can be added to an applet or used as a standalone Java application. Shown below are examples of how a client can declare JVALL to be added to an applet or a standalone application.

As an applet:

```
Jvall mylist = new Jvall(Jvall.MAKEAPPLET);
JPanel panel = mylist.getDisplay();
//returns a JPanel that can be added to an applet
this.getContentPane().setLayout(new
                                BorderLayout());
//set applet Bordered layout
this.getContentPane().add(panel,"Center");
//add JVALL to the center of the applet
```

As a standalone:

```
Jvall mylist = new Jvall(Jvall.STANDALONE);
```

In the case of standalone, JVALL will automatically appear ready to display animations. Figure 3 shows an example of standalone JVALL along with a client program. As an applet, the getDisplay() method returns a JPanel that a client can add to its applet. Figure 1 shows JVALL as an applet appearing within a browser.

JVALL Client

It is essential for JVALL to communicate the vital stages of the animation to the client program. JVALL communicates with clients through an event interface to inform the client of animation stages such as when the animation is running or ending. It also provides the current textual status of the animation. This information will enable the client to track all animation stages and respond to the animation in a synchronized manner.

For example, a client might want to disable certain buttons to prevent the user from clicking them while the animation is running and re-enable them when the animation is ended. Through the event interface, the client can call and execute the code to disable and enable the buttons within the interface methods

4 Educational Uses of JVALL

Interactive Linked List Client

The distribution of the JVALL package includes one generalized, interactive client that provides a user interface that permits the user to interactively perform linked list operations and watch their animation. The client window appears directly beneath the JVALL window as shown in Figure 3.

A user can type into the value and position text boxes to insert nodes into the specified position in the linked list. In standalone mode, a user has the option of opening a text file, loading it into the linked list and continuing with the normal linked list operations, such as insert, delete, replace, etc. Each time a button is clicked, JVALL gets a request from the client and the animation is displayed accordingly. If, for example, a user tries to insert or delete a node from an invalid position in the list, JVALL will throw the appropriate exception and also inform the user of valid positions. The General Linked List Operation client responds in an appropriate way to such an exception. The client also provides a variety of operations including insert position, insert first, insert last, replace, find, delete position, delete first, delete value, and clear all. All of these operations are animated appropriately by JVALL.

Laboratory Activities

This client supports many classroom uses. One such use is as a laboratory activity where the students are asked to perform a sequence of list operations and observe their animations. This can be done with several implementation models to emphasize the different algorithms used. Students could view the animations with code in hand or be asked to generate appropriate code.

Classroom Demonstration

Since JVALL may be used with any user-provided client, it is easy for an instructor to construct an appropriate client for classroom demonstration. In addition, with the open and save file capability, demonstrations can be performed on a substantial list without requiring the tedious process of populating that list.

Debugging Student Programs

JVALL is particularly useful in assisting students when debugging their programs that make use of the Java LinkedList class. In addition to the benefit of being able to visualize LinkedList operations, using JVALL in conjunction with a traditional debugger can be especially powerful in discovering errors in the logic of an application that uses the Linked List class. This enables the student to step through the code with the debugger and watch the results of operations evolve within the JVALL window. This is an improvement over use of a debugger alone because debuggers typically show addresses and not the actual structure of a linked list.

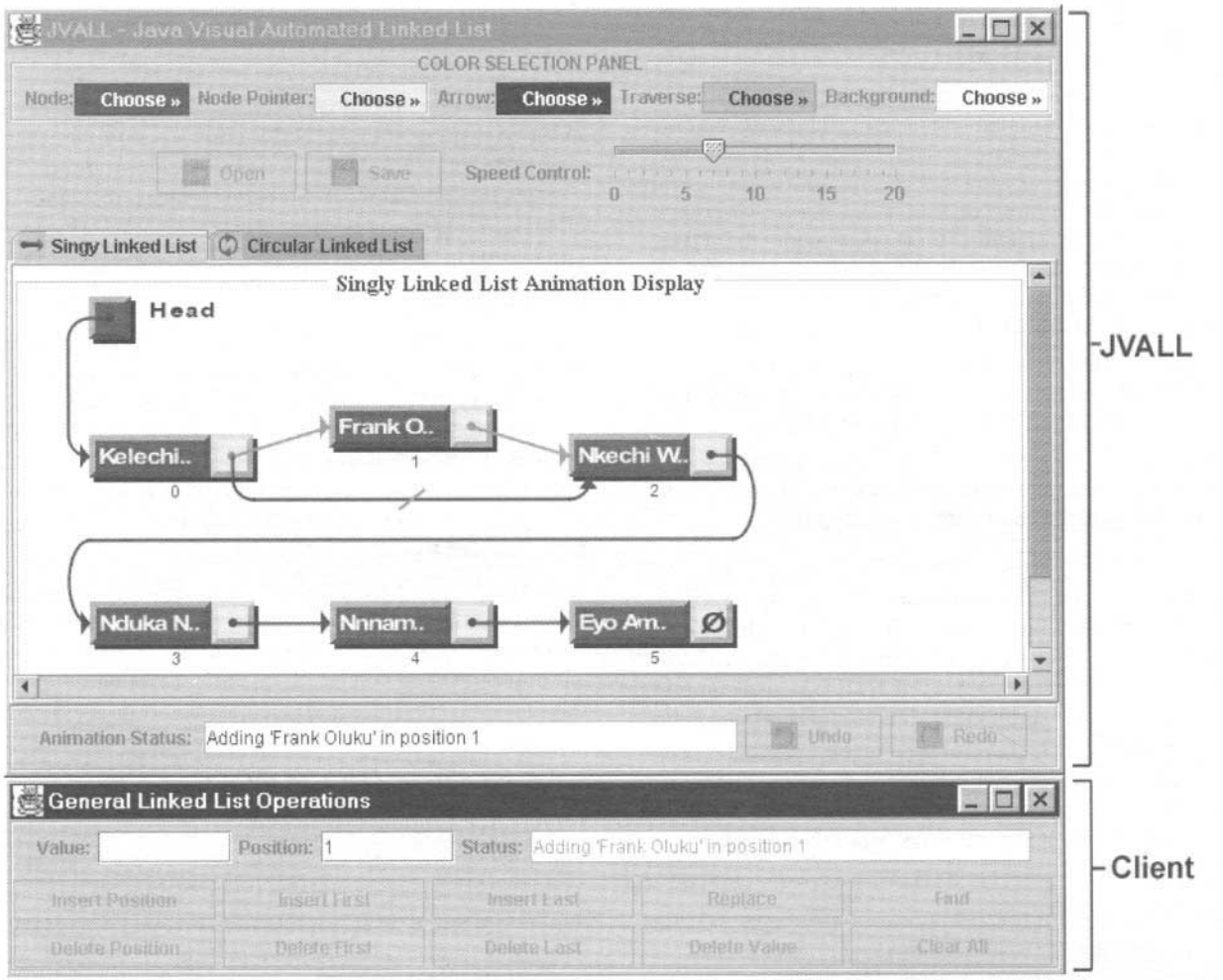

Figure 3

Since any program that uses the LinkedList class can be easily animated, no special code needs to be added to realize this benefit.

Visualization of Classes Implemented Using Linked Lists

A final educational use of JVALL is the visualization of the operations of classes that are implemented using the Java LinkedList class. Complex data are often implemented with linked lists and JVALL provides an effective way for understanding their operations under such an implementation. The distribution of JVALL includes a client that implements the stack ADT using linked lists as an illustration of this capability.

5 Conclusion

Several animation strategies have been developed and implemented for effective use as teaching and learning tools. The qualities that an effective algorithm visualization package should possess have been classified by Cordova in [1]. Below, we apply these five criteria to JVALL.

- **Flexibility**. JVALL is flexible in the sense that it is capable of animating any program that uses the Java LinkedList class. JVALL can be used and represented in different ways to meet the needs of the client program. JVALL can also animate applets as well as standalone applications.
- **Integration of algorithm text and visualization**. JVALL provides dynamic textual updates of the animation stages to the user and the client program. Because JVALL was built upon the Java LinkedList class and is independent of implementation, it is unable to provide code display of the linked list.
- **Ease of modification**. All the components of JVALL are fully customizable. JVALL comes with a full color control panel for customizing all aspects of the animation display. The linked list nodes, pointers, arrows, traverse arrow, and background colors can be changed to suit the user's taste.
- **Execution control features**. JVALL's speed control and multiple undo's and redo's give the user the ability to rewind and adjust the speed of the animation to aid learning at a user's pace.
- **Support for animation of algorithms supplied by the user**. A user can utilize JVALL animation for debugging purposes. JVALL is fully compatible with the Java LinkedList class. Hence, JVALL throws the same exceptions as Java LinkedList and, in addition, suggests possible causes of such exceptions in natural language to aid faster debugging.

According to these criteria, JVALL is a very effective tool to enhance the learning of the algorithms and applications of linked lists.

6 Acknowledgments

This work was supported in part by the National Science Foundation Research Experiences for Undergraduates program through grant #EIA-0097464.

References

[1] Cordova, J.A comparative evaluation of web-based algorithm visualization systems for computer science education, *Journal of Computing in Small Colleges 14,3* (1999), 72-77.

[2] Dershem, H.L. and Vanderhyde, J. Class visualization for teaching object-oriented concepts, *SIGCSE Bulletin 30,1* (1998), 53-57.

[3] Naps, T.L. Algorithm visualization served off the World Wide Web: Why and how. *SIGCSE Bulletin 28,1* (1996), 66-71.

[4] Naps, T.L., Eagan, R., and Norton, L. JHAVE – An environment to actively engage students in web-based algorithm visualization, *SIGCSE Bulletin 32,1* (2000), 109-113.

[5] Rodger, S. JAWAA, 1997. [Online] Available at http://www.cs.duke.edu/~rodger/tools/tools.html.

[6] Roessling, G. and Freisleben, B. AnimalScript: An Extensible Scripting Language for Algorithm Animation, *SIGCSE Bulletin 33,1* (2001), 70-74.

[7] Stasko, J.T. Using student-built algorithm animations as learning aids, *SIGCSE Bulletin 29,1* (1997), 25-29.

[8] Stasko, J.T. and Lawrence, A. *Empirically assessing algorithm animations as learning aids*, in Software Visualization: Programming as a Multimedia Experience, The MIT Press, 1998, 419-438.