# Data Structure Animation Tutorial

**Sun Yiling**

**A Major Report**

**In**

**The Department**

**Of**

**Computer Science**

**Presented in Partial Fulfilment of the Requirements**
**for the Degree of Master of Computer Science at**
**Concordia University**
**Montreal, Quebec, Canada**

**March 2001**

## ABSTRACT

Data Structure Animation Tutorial

Sun Yiling

This study is an animation tutorial for the people who wants to learn the Data Structure. The emphasis is placed on vivid animations to help the people to understand algorithms for data structure easily. Some of the implementations to be addressed are: stack (Array-Based Stack, Linked Stack), queue (Array-Based Queue), List (Circular Linked List, Double Linked List, Linear Linked List), sort (Quick Sort, Merge Sort, Bubble Sort, Shell Sort, Insertion Sort, Heap Sort, Radix Sort, Selection Sort) , heap (Priority Queue, Heap Build , Heap Sort), recursive ( Tower of Hanio), hashing ( Open Hashing, Close Hashing) binary search (Loop, Recursive), tree( 2-3 Tree, Huffman Tree, Binary Search Tree, Balance Tree). Conclusions are formulated in terms of further work to be accomplished in order to better help understanding the completed algorithm.

1. Introduction ..... 2

1.1 The role of Data Structure ..... 2
1.2 The role of animation ..... 3
1.3 A road map ..... 4

2.Techniques Survey ..... 6

2.1 C++ ..... 6
2.2 Visual Basic ..... 7
2.3 Java ..... 7

3.Interface Design ..... 10

3.1 Main Interface Design ..... 13
3.2 Animation Window Interface Design ..... 17
3.3 Code Animation Window Interface Design ..... 20
3.4 Help Window Interface Design ..... 22

4. System Design ..... 24

4.1 Applet vs. Application ..... 27
4.2 Container and layout manager ..... 29
4.3 JFC/Swing ..... 33
4.4 Multithreading ..... 35
4.5 Animation issue – Where to Animate ..... 44

5. Implementations ..... 46

5.1 Tower of Hanio ..... 46
5.2 Linked List -- Circular Linked List ..... 52

6. Conclusion ..... 61

Appendix ..... 62

1. System Deployment ..... 62
2. How to use JAR file ..... 62

# 1. Introduction

The project is designed to present the "Data Structure" in a right and efficient way to those who have interests in it. People are now able to learn each concept and algorithm in Data Structure on his own by interacting with this software.

In this project, we will not only introduce most basic concepts and algorithms in data structure: stack, queue, list, recursive, binary search, sort, heap and trees, but also apply animated image to simulate each transactions inside algorithm, which is believed to be an interesting experience for your study with data structure.

## 1.1 The role of Data Structure

Representing information is fundamental to computer science. The primary purpose of most computer programs is to store and retrieve information rather than to perform calculations. To be practical in terms of storage requirements and running time, such programs must organize their information in a way that supports efficient processing. For this reason, the study of data structures and the algorithms that manipulate them is at the heart of computer science.

You might think that with ever more powerful computers, program efficiency is becoming less important. However, more powerful computers encourage us to us to attempt more complex problems. More complex problems demand more computation, making the need for efficient programs even greater. As tasks become more complex they become less like or everyday experience. Today's computer scientists must be trained to have a thorough understanding of the principles behind efficient program design, since their ordinary life experiences often do not apply when designing programs.

In the most general sense, a data structure is any data representation and its associated operations. Viewed as a data representation, even an integer or floating point number stored on the computer is a simple data structure. More typically, a data structure is meant to be an organization on structuring for a collection of data items.

Given sufficient space to store a collection of data items, it is always possible to search for specified items within the collection, print or otherwise process the data items in any desired order, or modify the value of any particular data item. Thus, it is possible to perform all necessary operations on any data structure. However, the choice of data structure can make the deference between a program running in a few seconds or requiring many days.

The quest for programming efficiency need not and should not conflict with sound design and clear coding. Creating efficient programs has little to do with "programming tricks" but rather based on well-organized information and clear algorithms. It is unlikely a programmer who has not mastered the basic principles of clear design to write efficient programs. We, therefore realize that a strong emphasis on fundamental software engineering principles should play a key role in developing good programming skill. Once a programmer has learned sufficient principles of clear program design and implementation, it comes to the next step to study the effects of data organization and algorithms on program efficiency.

## 1.2 The role of animation

We understand that learning data structure is really benefit for the programmers, especially for those with limited exposure to computer programming. Then the problem

is how we can grasp data structure efficiently. In view of so many complicated concepts and algorithms in-between, it is a big headache for every learner to have a full understanding and even to remember such boring terms and complicated logic by heart. Admittedly, a good movie, always sourced from a good book, imposes greater impact than the book does on the people. Animation is much easier to access to the one's mind than the abstract words in a book are. Likewise, to alleviate the burden of learning complicated concept and algorithms, we herewith present a vivid animation software just as if to watch a "movie".

Animation plays an import role in this software. In the view of the requirements from this software, we need our animation has the following characters.

- The animation can be taken place either in standalone application or through the web, in another word, it must be portable.
- The animation shall be in a real time, i.e., the people can interrupt it whenever they want.

## 1.3 A road map

Chapter 1 provides an overview of this project, presenting why we need learn data structure, how to present it in an efficient way, why we use animation technology to implement this tutorial.

In chapter 2, we elaborate on a survey regarding to the current techniques, which can be used to implement the animation. By comparison, you will come to understand which is the most suitable technology that we can use for developing this tutorial.

Chapter 3 starts with the discussion of the interface design of this project. This chapter covers the basic principles to design a good interface and detailed description of each part of design, which may help you understand how to use it.

Chapter 4 is the longest and the key part of this document. It covers the system design of this project. Here we introduced each technology in Java that we used in this project, such as how to develop a synchronous animation using multithreading, how to design the graphic with layout control manager, how to use swing technology etc.

In chapter 5, we will focus on some of the typical implementations in this tutorial. We will go through the algorithm with the implementation, and present how we develop it with Java.

Chapter 6 is a conclusion of the experience with this project

Appendix will be attached on the reference how to deploy this system and how to use this system.

# 2.Techniques Survey

At the beginning when we plan to do this project, we could not decide which tool we can use to implement this software more efficiently. Thus we did some investigations on it. There are so many vehicles to implement the animation in software field, such as Visual Basic, C++, and Java, among Java enjoys advantages in implementing the animation with this tutorial. First we will introduce each tool that can be used in developing the animation, and then make a comparison to conclude why we choose Java.

## 2.1 C++

C++ is a big and powerful language. The widespread availability of C++ compilers, tools, libraries and environments makes the decision to choose C++ easier than it used to be, but there are risks as well as benefits. C++ has a lot of feature, but compare with other prior language, the significant one is its object-oriented design.

C++ is a superset of C that provides a very useful framework for object-oriented programming. C++ provides data encapsulation, abstraction, and polymorphism, along with interface definitions to access and manipulate data within objects.

Although C++ classes have a well-defined call-in interface, there is no call-out interface. Classes must be aware of particular instances of other classes, as well as the specific function names of methods being invoked on these other classes. Program flow is still controlled by function calls, since execution is driven by the sequence of method calls.

According to the requirements of our project to implement the animation, C++ is not a wise choice. First, using C++ to design a graphic interface with animation is a complicate job. Second, how to put it on the web is still open to be considered. So, although C++ is

an excellent general-purpose language, but a more specialized language may be appropriate for applications that fit its niche.

## 2.2 Visual Basic

Visual Basic was designed to make developing a graphical Windows application as easy as possible. Visual Basic tales care of the more tedious tasks of creating an application's graphical look. This allows programmers more time to concentrate on an application's features rather than how to style it for windows.

Visual Basic's graphical tools and high-level language constructs make it easy and quick to go from a programming concept to a full-fledged running application. Visual Basic is not only easy to use, but also fun to use. So from here, we can see the more advantages to develop a graphical software on Visual Basic and in the market there are so many animation tutorial software based on Visual Basic.

But there are also many limitations to what Visual Basic can do. For example, font and text manipulations are limited. Since animation is a fantasy a dynamic task, how you make it more synchronous is a big issue for consideration. Also as a tutorial, its non-portability is a big limitation for our project.

## 2.3 Java

The Java language burst onto the scene in 1995 and was given instant celebrity status. While most computer language do not inspire the popular press to grow rhapsodic, Java did. While most computer language do not get coverage on radon and TV, Java did.

The point and promise of Java is that it hopes to be the universal glue connecting users and information. It doesn't matter whether information is stored on Internet Web server, database, information providers, or any other imaginable source, Java will eventually let you use the Web to access this information, and more.

Java is an extremely solid-engineered language. Its syntax seems instantly familiar to most programmers. Its security features are reassuring to both programmers and users of their programs. Java makes it easier to produce bug-free code than C++. Java has build-in support that makes advanced programming task, such as network programming and multithreading, straightforward and convenient.

One obvious advantage is a run time library that gives it platform independence: you can use the same code on Windows 98, Solaris, Unix, Macintosh, and so on. Java is also fully object oriented- even more so than C++. Everything in Java, except for a few basic types like numbers, is an object.

"Computer Animation" is a broad term that could mean anything from a simple animated GIF on a website to a full-blown feature-length movie. Java animation falls somewhere between these two extremes. Unlike animated GIFs, Java provides the flexibility to create truly dynamic animations, that is, animations that change in response to external events or user input. Dynamic animation is necessary for things like real-time displays and simulations. Java is very handy for producing animation that is portable, either as web-based applets or stand-alone application.

This software is implemented both in application and applet using Java technology. To illustrate the advantage of JAVA, herein we make an comparison with C++:

- It is much easier to turn out bug-free code using Java than using C++

- Memory in Java is automatically garbage collected and you will never worry about memory leaks.
- Since it introduces true array and eliminated pointer arithmetic, never will worry about overwriting a key area of memory because of an off-by-one error when working with a pointer.
- It rules out the possibility of confusing an assignment with a test in equal condition.
- It wipes out multiple inheritance, replaced by a new notion of *interface* that derived from Object C.

# 3.Interface Design

When designing the interface, it follows the principles below:

- Consistency, consistency, consistency. The most important thing user interface works consistently. If you can double-click on items in one list and have something happen then you should be able to double-click on items in any other list and have the same sort of thing happen. Put buttons in consistent places on all of windows, use the same wording in labels and messages, and use a consistent color scheme throughout. Consistency in user interface allows users to build an accurate mental model of the way that it works, and accurate mental models lead to lower training and support costs.
- Explain the rules. The users need to know how to work with the application that you built for them. When an application works consistently it means you only have to explain the rules once. This is a lot easier than explaining in detail exactly how to use each and every feature in an application step by step.
- Navigation between screens is important. If it is difficult to get from one screen to another then the users will quickly become frustrated and give up. When the flow between screens matches the flow of the work that the user is trying to accomplish, then the application will make sense to the users. Because different users work in different ways, your system will need to be flexible enough to support their various approaches.
- Navigation within a screen is important. In Western societies people read left to right and top to bottom. Because people are used to this should you design screens that are also organized

left to right and top to bottom. You want to organize navigation between widgets on your screen in a manner that users will find familiar to them.

- Word your messages and labels appropriately. The text that is displayed on the screens is a primary source of information for the users. If the text is worded poorly then the interface will be perceived poorly by the users. Using full words and sentences, as opposed to abbreviations and codes makes the text easier to understand. The messages should be worded positively, imply that the user is in control, and provide insight into how to use the application properly.
- Use color appropriately. Color should be used sparingly in your applications, and if you do use it you must also use a secondary indicator. The problem is that some of your users may be color blind –if you are using color to highlight something on a screen then you need to do something else to make it stand out if you want these people to notice it, such as display a symbol beside it. You also want to use colors in your application consistently so that you have a common look and feel throughout your application. Also, color generally does not port well between platform – what looks good on one system often looks poor on another system. We have all been to presentations where the presenter said "it looks good on my machine at home."
- Use fonts appropriately – Old English fonts might look good on the covers of William Shakespeare's plays, but they are really hard to read on a screen. Use fonts that are easy to read, such as serif fonts like Times Roman. Furthermore, use your fonts consistently and sparingly. A screen using two or three fonts effectively looks a lot better than a screen that uses five or six. Never forget that you are using a different font every time you change the size, style (bold, italics, underlining, ...), typeface, or color.

- Gray things out, do not remove them. You often find that at certain times it is not applicable to give your users access to all the functionality of an application. You need to select an object before you can delete it, so to reinforce your mental model the application should do something with the Delete button and/or menu item. Should the button be removed or grayed out? Gray it out, never remove it. By graying things out when they shouldn't be used people can start building an accurate mental model as to how your application works. If you simply remove a widget or menu item instead of graying it out then it is much more difficult for your users to build an accurate mental model because they only know what is currently available to them, and not what is available. The old adage that out of sight is out of mind is directly applicable here.
- Justify data appropriately. For columns of data it is common practice to right justify integers, decimal align floating-point numbers, and left justify strings.
- Group things on the screen effectively. Items that are logically connected should be grouped together on the screen to communicate that they are connected, whereas items that have nothing to do with each other should be separated. You can use whitespace between collections of items to group them and/or you can put boxes around them to accomplish the same thing.
- Open windows in the center of the action. When your user double-clicks on an object to display its edit/detail screen then his or her attention is on that spot. Therefore it makes sense to open the window in that spot, not somewhere else.
- Pop-up menus should not be the only source of functionality. Your users cannot learn how to use your application if you hide major functionality from them. One of the

most frustrating practices of developers is to misuse pop-up, also called context-sensitive, menus. Typically there is a way to use the mouse on your computer to display a hidden pop-up menu that provides access to functionality that is specific to the area of the screen that you are currently working in.

## 3.1 Main Interface Design

The startup window leaves users with alternative to engage in the topic wanted.

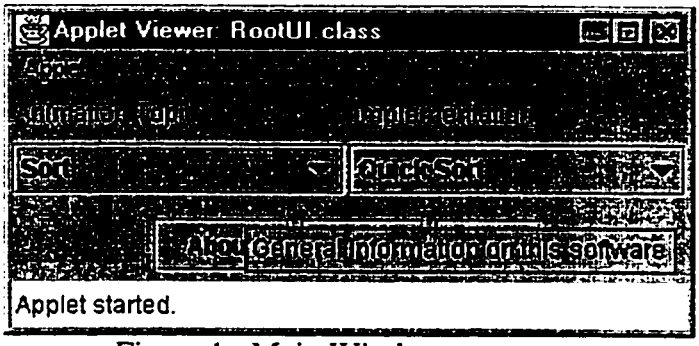



Figure 1 : Main Window

From above figure 1, we can see the main window for this software. There are two menu bars and three buttons. Here we employ the swing technology, by which a tool tip will pop up to state the function of this button when you put your mouse on it. Such technology will help the user to interact with the interface easily and implement the software correctly. For instance, we put the mouse on the "About" button and then it will pop up a text "General information on this software".

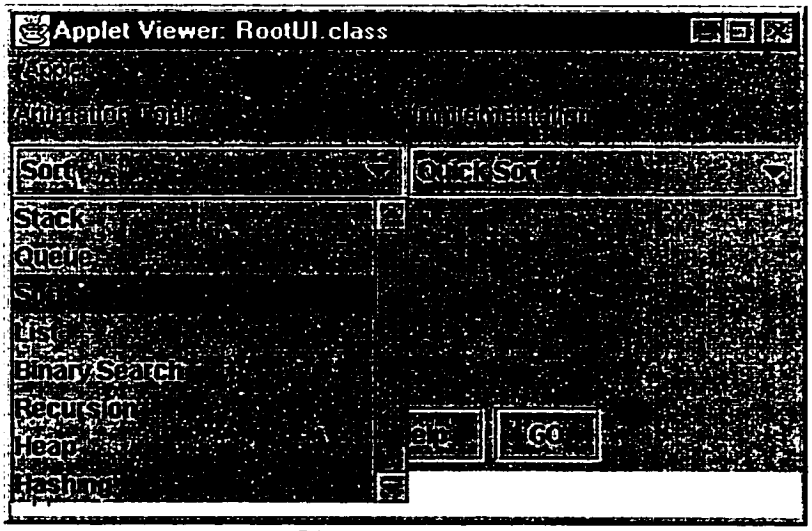



Figure 2

You can choose the topic from menu "Animation Topic", including Stack, Queue, Sort, List, Binary Search, Recursive, Heap, Hashing and Tree. See above figure 2.

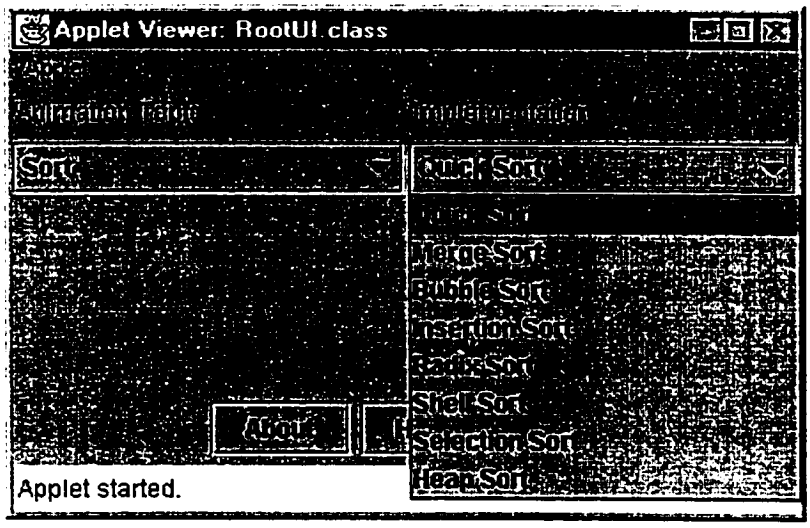



Figure 3

Once you decide the topic you want, you can find various implementations within this topic, which covers:

1. Stack: Array-Based Stack; Linked-List-Based Stack

2. Queue: Array-Based Queue

2 List: Circular-Linked List; Double-Linked List; Linear-Linked List

3 Sort: Quick Sort; Merge Sort; Bubble Sort; Insertion Sort; Radix Sort; Shell Sort; Selection Sort; Heap Sort

4 Recursive: Tower of Hanio

5 Binary Search: Loop; Recursive

6 Hashing: Open Hashing; Close Hashing

7 Heap: Heap Sort; Build Heap; Priority Queue

8 Tree: Binary Search Tree; Balance Tree; 2-3 Tree; Huffman Tree

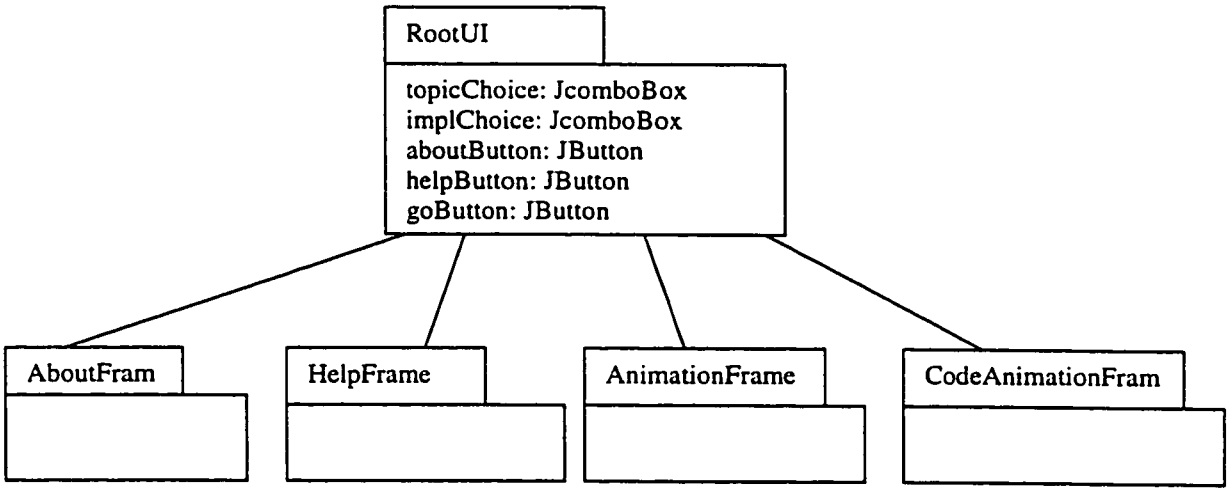
Clicking the "Go" button, a window will pop up for the animation on the currently selected topic and implementation.

For this interface, we defined the RootUI class as the following:

```
public class RootUI extends JApplet implements ActionListener, ItemListener {
                                                    // base class RootUI
  public static boolean isApplication = false; // the flag to determine either application or
                                                    applet for this software
  private static RootUI root = null;
  static HelpFrame helpFrame = null;
  static AboutFrame aboutFrame = null;
  static AnimationFrame animationFrame = null;
  private static JComboBox topicChoice = null;
  private static JComboBox implChoice = null;
  private static JButton aboutButton = null;
```

```
private static JButton helpButton = null;

private static JButton goButton = null;

public static boolean aboutIsOn = false; // the flag to indicate if the about window in
                                         showed or not

public static boolean helpIsOn = false;  // the flag to indicate if the help window is
                                         showed or not

private static String oldTopic = null;

public static CodeAnimationFrame codeAnimFrame = null;

}
```

So from the above pseudo code, all the attributes we defined are clearly .




The following sections we will introduce each class ( aboutFrame, HelpFrame, AnimationFrame, CodeAnimationFrame) in details.

## 3.2 Animation Window Interface Design

Animation window is the mainly interface for each implementation. So a clear and simple design will definitely help users to interact with it friendly.

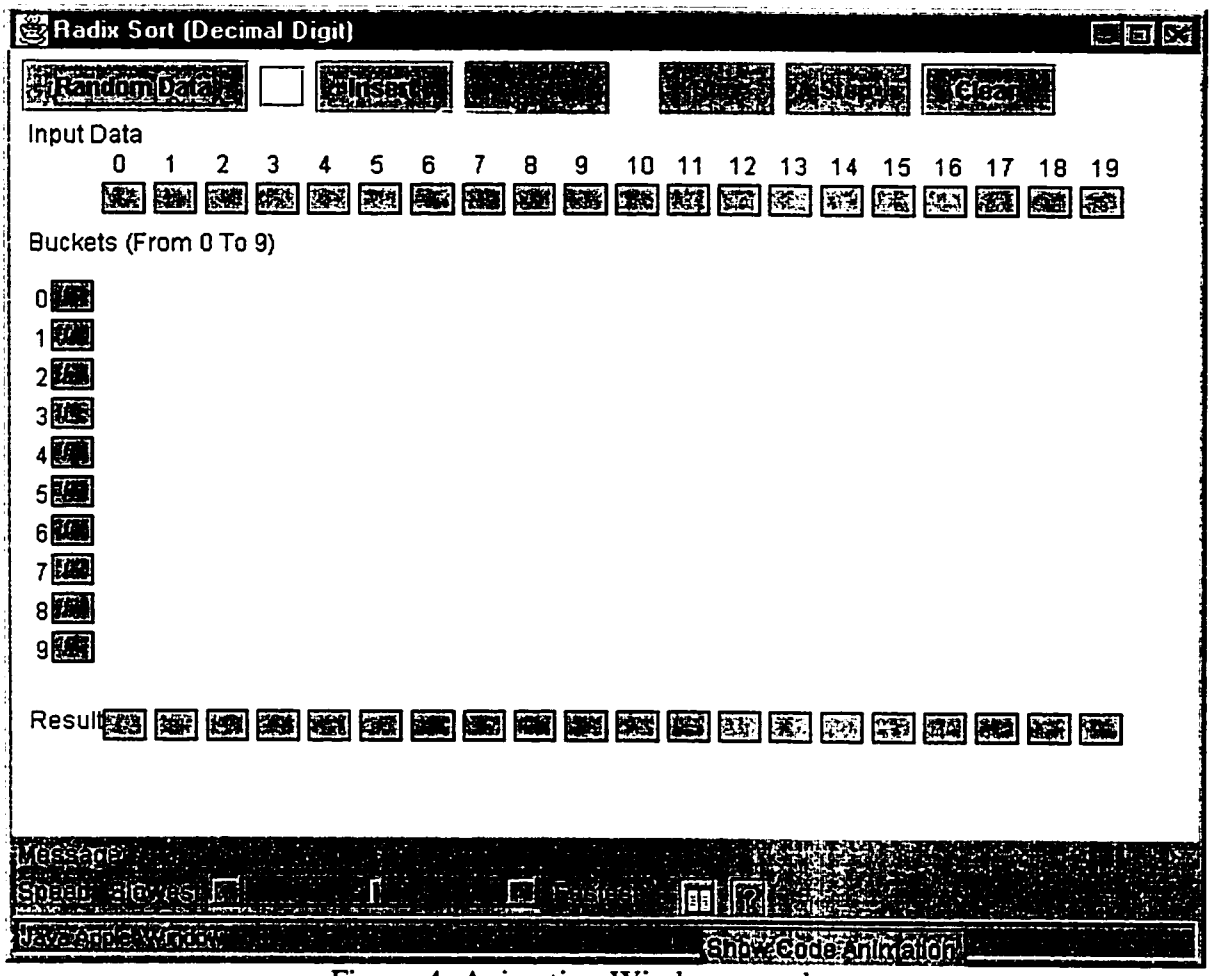



Figure 4: Animation Window sample

In each animation window, there are three common parts: the control panel, animation area, and the speed bar. However, all of them are optional and we design each of animation windows upon specific requirement.

1. Control Panel contains a "Random Data" button, a textField, and "Insert" button, a "Delete" button, a "Run" button, and a "Clear" button. The textField only accepts

integer number from 0 to 99 inclusive. "Random Data" is to randomly generate input data. "Insert" and "Delete" buttons are used for data input before animation starts, or after "Clear" button is clicked to reinitiate the animation. A Click on the "Run" triggers continuous animation, and a click on the "Step" starts heading for the animation step by step. Each click on "Step" button will lead to animation for one step of code execution, which may contain a few sequential steps. Users can switch between the *continuously running* and *stepping* mode at any time. User can also click on the "Clear" button to stop the animation and reinitiate for the new session starting with data input mode.

2. Animation area is where to make the animation. On the top of it, an array "Initial Data " holds the input value. During the animation, after a value is used up, its corresponding box will be shaded while the digits are still visible, on the top of the Initial data array are the index values for the array. At the bottom of the animation area, and array "Result" holds the result data from deleting or traversal. Values are inserted into this array from left to right. The size of the Initial Data array and Result array are determined before data input or after clear based in the window size then, and not changed by the change of window size during animation. In Figure 5, you will see the interface during the animation for the Radix Sort.
3. The Speed bar contains message text, speed scroll bar, code animation button and help button. Message text passes the information and message like warnings, suggestions etc. on each animation steps. The speed scroll bar controls the animation speed when animation is in Run mode. Code Animation button will provide the code animation window. Help button is waiting for your resort.

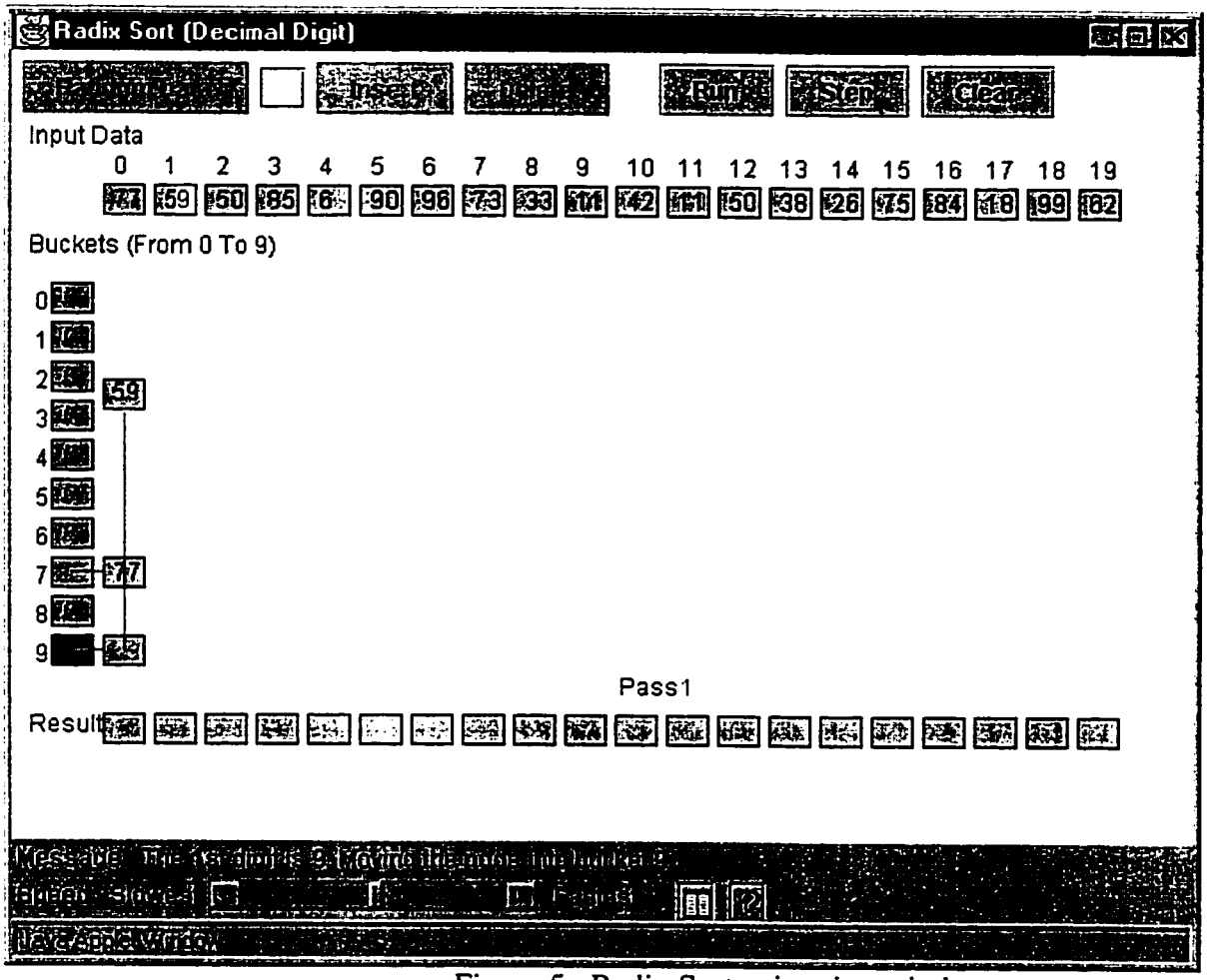



Figure 5 : Radix Sort animation window

```
class AnimationFrame extends Frame implements AdjustmentListener {

  public static Image offScreenImage;

  Graphics offScreenGraphics;

  Dimension windowSize;

  JButton jbCodeAnimation;

  JButton jbHelp;

  public static int delay;  // inverse of speed

  JTextField inputTextField;
```

Scrollbar scrollSpeed;

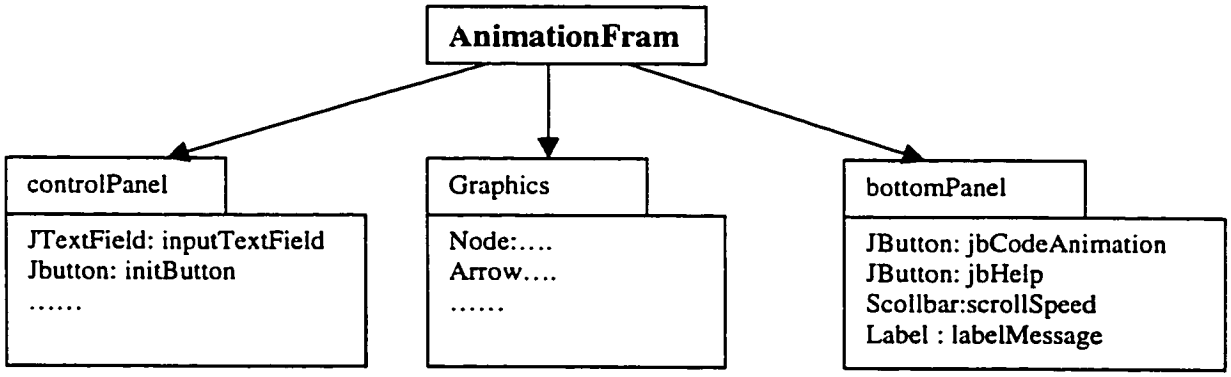
Panel controlPanel;

Panel bottomPanel;

Label labelMessage;

}

The structure for the class of AnimationFrame is depicted as below. Each implementation has its own animation frame that is derived from this class, and in this class, we defined the common parts of each animation window




## 3.3 Code Animation Window Interface Design

This window displays source code on the left and the message from animator on the right. It will pass a line of highlighted number and an optional string as an annotation to the current code source line. The animator is entitled to decide when to invoke highlighting a particular line, or changing the code contents. Below with Figure 6, you will see the sample window when running the Radix Sort animation.

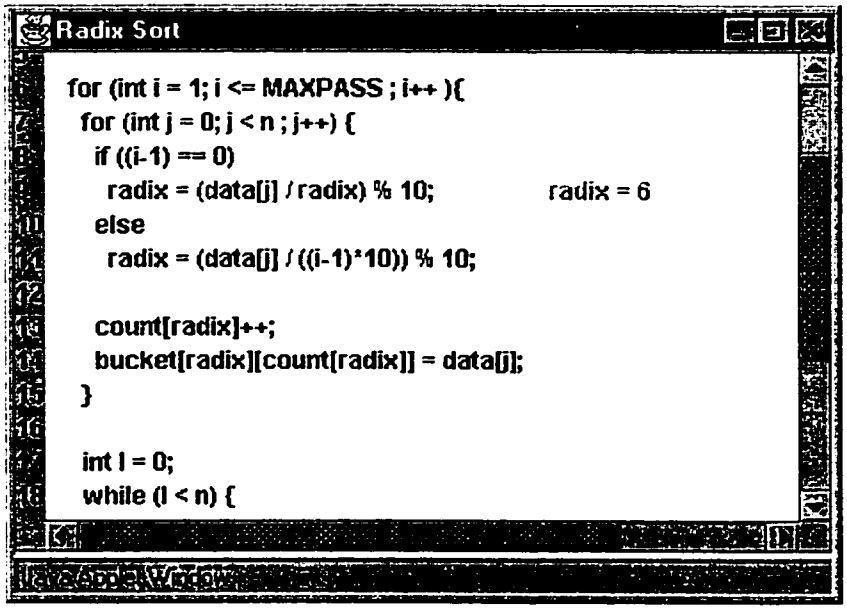



Figure 6: Code Animation Window

The CodeAnimationFrame class is defined as below:

```
class CodeAnimationFrame extends Frame {
  String [] codeArray;   //contain the code strings
  String [] annotation;
  JLabel [] codeLines = new JLabel[256];
  JLabel [] annotationLines = new JLabel[256];
  JLabel [] lineNumber = new JLabel[256];
  JScrollPane codePanelView;
  JPanel codePanel;
  JPanel rowHead;
  JScrollBar verticalScroll;
}
```

## 3.4 Help Window Interface Design

The Help window contains two links and four buttons. The link for the "Concept and Algorithm" passes the information about this particular implementation while the link for "User Interface Explanation" provides the illustration of the information how to deal with this animation window for the users. The four buttons below allows users to move back and forth easily. The Figure 7 shows the sample window for Radix Sort.

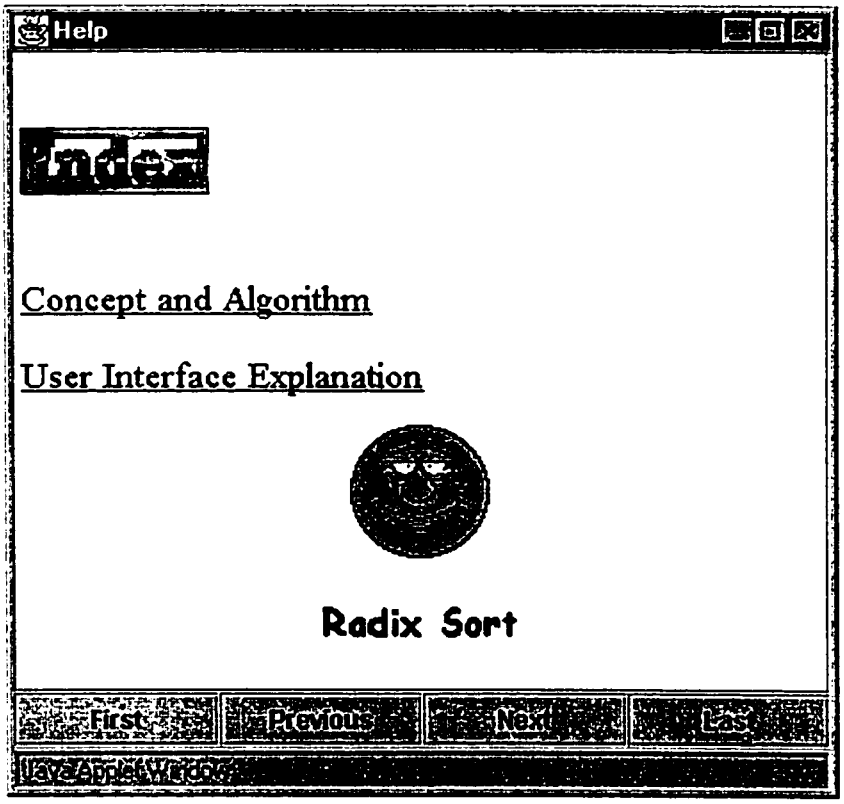



Figure 7: Help window for Radix Sort

The HelpFrame class is defined as below:

```
public class HelpFrame extends JFrame implements WindowListener {
 HelpPanel helpPanel;
 JButton jbtFirst;
 JButton jbtPrevious ;
 JButton jbtNext ;
JButton  jbtLast ;
}
```

So for each implementation, you can get the help content in the helpPanel.

# 4. System Design

As mentioned above, the adoption of Java as our primary technology for this software lies in its powerful virtue.

Java technology is both a programming language and a platform.

*1. The Java Programming Language*

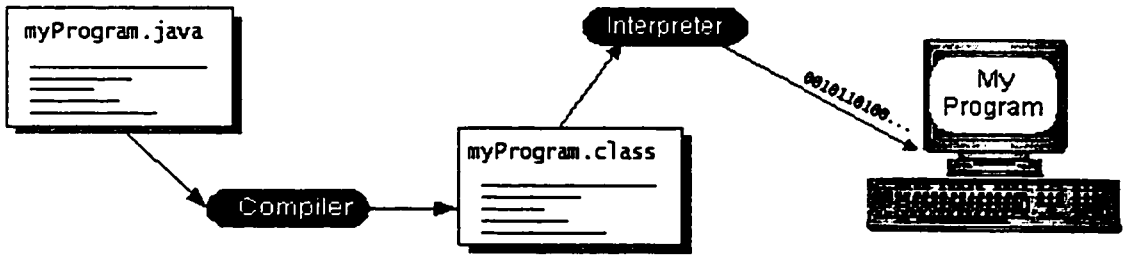
The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

- Simple
- Architecture-neutral
- Object-oriented
- Portable
- Distributed
- High-performance
- Interpreted
- Multithreaded
- Robust
- Dynamic
- Secure

Each of the preceding buzzwords is explained in The Java Language Environment, a white paper written by James Gosling and Henry McGilton. Youcan download a PostScript or PDF version of the paper from http://java.sun.com/docs/white/.

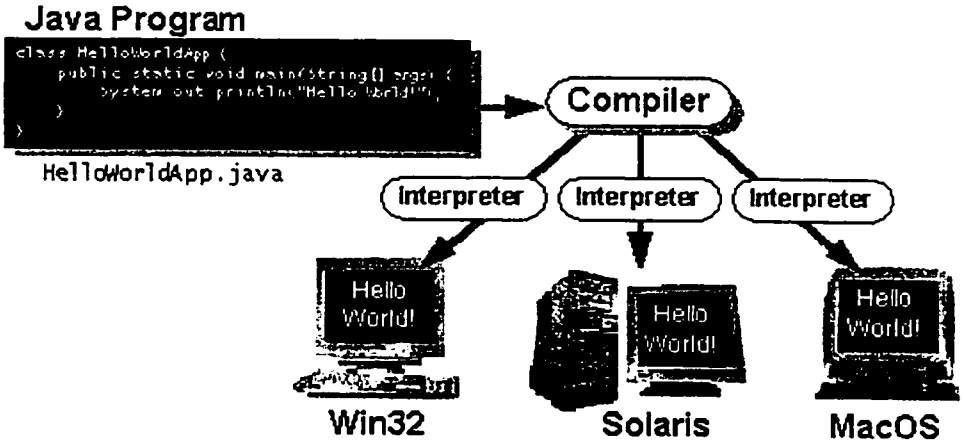
With most advanced programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in

that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called Java bytecodes-the platform-independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java bytecode instruction on the computer. Compilation happens just once; interpretation occurs each time that the program is executed. This below figure illustrates how this works.




You can think of Java bytecodes as the machine code instructions for the Java Virtual Machine (Java VM). Every Java interpreter, whether it's a development tool or a Web browser that can run applets, is an implementation of the Java VM.

Java bytecodes help make "write once, run anywhere" possible. You can compile your program into bytecodes on any platform that has a Java compiler. The bytecodes can then be run on any implementation of the Java VM. That means that as long as a computer has a Java VM, the same program written in the Java programming language can run on Windows 2000, a Solaris workstation, or on an iMac.




*2.The Java Platform*

A platform is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Windows 2000, Linux, Solaris, and MacOS. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

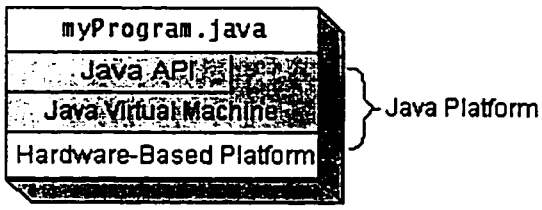
The Java platform has two components:

- The Java Virtual Machine (Java VM)
- The Java Application Programming Interface (Java API)

You've already been introduced to the Java VM. It's the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as packages. The next section, What Can Java Technology Do? , highlights what functionality some of the packages in the Java API provide.

The following figure depicts a program that's running on the Java platform. As the figure shows, the Java API and the virtual machine insulate the program from the hardware.




Native code is code that after you compile it, the compiled code runs on a specific hardware platform. As a platform-independent environment, the Java platform can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time bytecode compilers can bring performance close to that of native code without threatening portability.

Here we introduced some features used in this particular project

## 4.1Applet vs. Application

Quite a few years ago, a "Saturday Night Live" skit poking fun at a television commercial showed a couple arguing about a white, gelatinous substance. The husband said," it's a desert topping." The wife said, "It's a floor wax." And the announcer concluded triumphantly, "It's both!"

1) Java Application:

- A Java application is like any regular program. The only difference is that it is platform independent i.e. anyone on any computer be it a Mac, Win95 machine, Linux or even a Cray supercomputer!, runs the same file.
- It is secure. You need not fear any viruses accompanying the Java you download.

- It runs inside a JVM (Java Virtual Machine) which we can assume is an abstract computer. So this secures your computer from hazardous influence from the Java programs.

2) Java Applet:

- An applet runs inside a browser. It runs under severe network restrictions.
- An applet cannot access URLs of any computer than the one it is running from. For example, if an applet is located on www.ibm.com and tries to get a URL from www.apple.com, the browser will refuse.
- An applet cannot read or write to the local hard disk.

Well, our project is both an applet and an application. That is, you can either load the program with the applet viewer or Netscape, or start it from the command line with the java interpreter. Though we are not sure how often this comes up-we found it interesting that this could be done at all.

Let us see how this can be done. Every class file has exactly one public class. In order for the applet viewer to launch it, that class must derive from *Applet*. In order for *java* to start the application, it must have a static main method. So far, we have

```
class AppletApplication extend Applet
{  public void init() {...}
  ...
  static public void main (string[] args) {...}
}
```

What can we put into *main*? Normally, we make an object of the class and invoke show on it. The applet must be placed inside a frame. And once it is inside the frame, its *init* method needs to be called. The example with our project is done as below:

```
public class RootUI extends JApplet implements ActionListener, ItemListener {
 ...
 public static void main(String[] args) {
   JFrame f = new JFrame();
   f.setResizable(false);
   f.setTitle("Data Structure Animation");
   root = new RootUI();
   f.getContentPane().add(root);
   root.init();
   f.pack();
   f.setVisible(true);
   f.addWindowListener(new WindowClose(f, true));
   isApplication = true;
 }
public void init() {...}
```

## 4.2 Container and layout manager

User interfaces are created by grouping together components and placing them in a logical order inside a container. The container's job is to provide the screen space for its child components and to arrange for them to be drawn onto the screen when they are

visible, while the actual positioning of components within the container is a matter for the container's layout manager.

➢ Container

A container is an extension of Component that has the ability to contain other components. Since a container is itself a component, a container can also be a parent to other containers, making it possible to create hierarchical arrangements of components and most applications have several containers within the top-level frame. The base class from which all containers are derived is jave.awt.Container. Like Component, this class is abstract, so it can not be used directly. Instead, you must either create your own container by subclassing it, in which case you would create a lightweight container, or you can use one of the standard ones, of which JPanel is an example.

➢ Layout manager

Container is a relatively simple class. Aside from assisting with changes in layout when its own size is changed, we have seen just about everything that a container is required to do. It is the layout manager that undertakes the complex task of creating a suitable arrangement of components within a container so that the screen representation appears as we intended. Among various layout managers, we used the following in this project.

1) The flowLayout Manager

The flowLayout manager is a very simple, but effective layout policy. Given a set of components, it lays them out side-by side starting near the top of the container and working downward as each row is filled. Within each row, the components are, by default, arranged as evenly as possible around the center of that row. The most important aspect of FlowLayout's layout policy is the fact that it allows each component to occupy

its ideal space. In our project, you will notice that each button at the top of animation window is side by side as we defined the control_panel to be flowlayout in animation frame.

As the code showed below, you can add the button one by one from left to right.

```
{
controlPanel = new Panel();
controlPanel.setLayout(new FlowLayout(FlowLayout.LEFT));
controlPanel.add(randomButton);
controlPanel.add(inputTextField);
controlPanel.add(insertButton);
controlPanel.add(deleteButton);
deleteButton.addActionListener(this);
controlPanel.add(new Label());
controlPanel.add(sortButton);
controlPanel.add(stepButton);
controlPanel.add(clearButton);
}
```

2) The BorderLayout Manager

As useful as FowLayout is, it isn't really capable of managing a complete user interface, except in the simplest cases. The main shortcoming of FlowLayout is that it is essentially one-dimensional, preferring to lay out components left-to –right and creating vertical distributions only when there is insufficient horizontal space. By contrast, the BorderLayout manager is very suited to more generalized arrangements of components and is the default for the containers.

A Borderlayout can manage from one to five components. Each component is explicitly assigned to one of the four edges of the container or to its center.

For example in our project for the main window, we defined the layout as BorderLayout to divide the container into two parts, north and south.

```
{
rootContainer.setLayout(new BorderLayout());
 rootContainer.add(topPanel, BorderLayout.NORTH);
 rootContainer.add(bottomPanel, BorderLayout.SOUTH);
}
```

3) The GridLayout Manager

The GridLayout is , as its name suggests, used for laying out components in a gridlike configuration. A container managed by GridLayout is divided into a number of equally sized cells, into each of which is placed one component. As the container is resized, the individual cells grow or shrink proportionately, as do the components that they contain.

For example in out project, we defined the tower stack panel on Hanio of Tower implementation as this layout.

```
public TowerStackPanel(Tower f, Tower t, Tower h) {
   from = f;
   to = t;
   help = h;
   index = 0;
   setLayout(new GridLayout(10, 5, 0, 0));
```

```
for(int i=44; i>=0; i--) {
  array[i] = new Button("     ");
  array[i].setBackground(new Color(200, 230, 230));
  add(array[i]);
}
```

## 4.3 JFC/Swing

The biggest difference between the AWT components and Swing components is that the Swing components are implemented with absolutely no native code. Since Swing components aren't restricted to the least common denominator -- the features that are present on every platform -- they can have more functionality than AWT components. Because the Swing components have no native code, they can be be shipped as an add-on to JDK 1.1, in addition to being part of the Java 2 Platform.

Even the simplest Swing components have capabilities far beyond what the AWT components offer:

- Swing buttons and labels can display images instead of, or in addition to, text.
- You can easily add or change the borders drawn around most Swing components. For example, it's easy to put a box around the outside of a container or label.
- You can easily change the behaviour or appearance of a Swing component by either invoking methods on it or creating a subclass of it.
- Swing components don't have to be rectangular. Buttons, for example, can be round.

- Assistive technologies such as screen readers can easily get information from Swing components. For example, a tool can easily get the text that's displayed on a button or label.

Swing lets you specify which look and feel your program's GUI uses. By contrast, AWT components always have the look and feel of the native platform.

Another interesting feature is that Swing components with state use models to keep the state. A JSlider, for instance, uses a BoundedRangeModel object to hold its current value and range of legal values. Models are set up automatically, so you don't have to deal with them unless you want to take advantage of the power they can give you.

If you're used to using AWT components, you need to be aware of a few gotchas when using Swing components:

- Programs should not, as a rule, use "heavyweight" components alongside Swing components. Heavyweight components include all the ready-to-use AWT components (such as Menu and ScrollPane) and all components that inherit from the AWT Canvas and Panel classes. This restriction exists because when Swing components (and all other "lightweight" components) overlap with heavyweight components, the heavyweight component is always painted on top. For more information, see Mixing Heavy and Light Components, an article in The Swing Connection.
- Swing components aren't thread safe. If you modify a visible Swing component -- invoking its setText method, for example -- from anywhere but an event handler, then you need to take special steps to make the modification execute on the event-

dispatching thread. This isn't an issue for many Swing programs, since component-modifying code is typically in event handlers.

- The containment hierarchy for any window or applet that contains Swing components must have a Swing top-level container at the root of the hierarchy. For example, a main window should be implemented as a JFrame instance rather than as a Frame instance.
- You don't add components directly to a top-level container such as a JFrame. Instead, you add components to a container (called the content pane) that is itself contained by the JFrame.

So, the lists above show the feature about the swing technology, that is why we use it for our interface design . It makes our interface more friendly, clear and beautiful.

To implement with swing, first you need add" import javax.swing" at the head of file. As we used in this project, if you need add some help hint for the button, you can use function "setToolTipText(String) " with this particular button. So that when you put mouse on this button, the hint that you explain with this button will be appeared.

## 4.4 Multithreading

Animation in Java should always occur in a separate thread of execution. This allows user to interact with the Java animation program without perceptibly degrading performance. A thread is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently.

Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority. Each thread may or may not also be marked as a daemon. When code running in some thread creates a new Thread object, the new thread has its

priority initially set equal to the priority of the creating thread, and is a daemon thread if and only if the creating thread is a daemon.

When a Java Virtual Machine starts up, there is usually a single non-daemon thread (which typically calls the method named main of some designated class). The Java Virtual Machine continues to execute threads until either of the following occurs:

The exit method of class Runtime has been called and the security manager has permitted the exit operation to take place.

All threads that are not daemon threads have died either by returning from the call to the run method or by throwing an exception that propagates beyond the run method.

In this project, we use the thread to carry out each animation. Hence, the user can control the animation during its processing, like controlling the animation speed, stopping the animation whenever he/she wants, switching the animation mood (automatic and step) smoothly. Here we give a detailed example to elaborate how the thread implements the animation in this project.

1. Basic animation unit -- Node

   As we saw in most of the animation window, there are some small boxes, we called "Node". We defined the common attributes in the node class.

   - Point cord; // the left top point for the node which defined the position of the node
   - Point size; // the length and width for the node which defined the size of the node
   - String label; // the label of the node for specifying the node
   - Color color; // the node color
   - boolean isHiden; // the flag indicates the node is displayed or not
   - boolean isHighlighted; // the flag indicates the node is highlighted or not

- boolean isSelected; // the flag indicates the node is selected or not

- boolean showleftpointer; // the flag indicates the left linker with the node is showed

- boolean showrightpointer;// the flag indicates the right linker with the node is showed

- static final Color NodeDefaultColor = new Color(250, 220, 100);

- static final Font NodeFont12 = new Font("TimesRomana", Font.PLAIN, 12);

- static final Font NodeFont8 = new Font("TimesRomana", Font.PLAIN, 8);

From the above listing attributes of the node, we can images what the node appears, and we can create a node at any position with any size, any colour.... And we can control the statues about the node, either displayed or hidden, selected or unselected.

2. Animation implement

Subsequent to the definition of the element, it comes to make it animate. We implement it in a function named 'move' which you can move a specific node from its original place to wherever you want.

```
public void move(AnimationFrame frame, Point d) {
  Image nodeImage = frame.createImage(size.x + 3, size.y + 3);
  moveBody(d, nodeImage, frame.offScreenImage, frame.graphics);
 }
 public void moveBody(Point d, Image nodeImage, Image offScreenImage,
                 Graphics graphics){
  if (nodeImage == null) {
   System.err.println("move failed to create node image");
   return;
```

```
}
Graphics offg;
offg = nodeImage.getGraphics();
Thread me = Thread.currentThread();
if (d.x>=0)
 for (int i=0; i<d.x; i++) {
  offg.drawImage(offScreenImage, 0, 0, size.x+3, size.y+3,
            cord.x, cord.y-1, cord.x+size.x+3, cord.y+size.y+2, null);
  cord.x++;
  absShow(offg);
  graphics.drawImage(nodeImage, cord.x-1, cord.y-1, null);
  try {
   me.sleep(AnimationFrame.delay);
  } catch (InterruptedException e) { }
 }
else
 for (int i=0; i>d.x; i--) {
  offg.drawImage(offScreenImage, 0, 0, size.x+3, size.y+3,
            cord.x-2, cord.y-1, cord.x+size.x+1, cord.y+size.y+2, null);
  cord.x--;;
  absShow(offg);
  graphics.drawImage(nodeImage, cord.x-1, cord.y-1, null);
  try {
```

```
      me.sleep(AnimationFrame.delay);
    } catch (InterruptedException e) { }
  }
if (d.y>=0)
  for (int i=0; i<d.y; i++) {
    offg.drawImage(offScreenImage, 0, 0, size.x+3, size.y+3,
                   cord.x-1, cord.y, cord.x+size.x+2, cord.y+size.y+3, null);
    cord.y++;
    absShow(offg);
    graphics.drawImage(nodeImage, cord.x-1, cord.y-1, null);
    try {
      me.sleep(AnimationFrame.delay);
    } catch (InterruptedException e) { }
  }
else
  for (int i=0; i>d.y; i--) {
    offg.drawImage(offScreenImage, 0, 0, size.x+3, size.y+3,
                   cord.x-2, cord.y-1, cord.x+size.x+1, cord.y+size.y+2, null);
    cord.y--;
    absShow(offg);
    graphics.drawImage(nodeImage, cord.x-1, cord.y-1, null);
    try {
      me.sleep(AnimationFrame.delay);
```

```
        } catch (InterruptedException e) { }
      }
    }
```

Obviously we draw the node in the new position and make the node hidden in the old place. We defined moving in four directions with four loops in the above code. Also here we use the current thread called 'me' to simulate the animation. The move function above moves the node d units away from its current position. The move is first along axis x, then along axis y. Each step moves 1 unit of distance either horizontally or vertically. First copy a background image of the moving node to a node image that is larger than the moving node by 3 in either direction, and make sure the new position of the node should be at position (1, 1) of the node image, and the boundary of the node image will have a distance of at least 1 from the original or moved position. The node image is then copied back to foreground of the animation frame. The inside function absShow does the same thing as above, but instead of drawing it on its current position, this function draws it at absolute position (1,1). It is very useful for combined double buffering.

3. Using thread

There are two ways to create thread in Java, The first is to extend the *Thread* class and provide a specific implementation for the class's *run* method:

```
Public class MyThread extends Thread
{   public void run ()
      { ... }
}
```

To start the thread, simply instantiated the extended class and call the *Thread* class's *start* method. The *start* method is what actually creates the separate thread of execution, which in turn calls the *run* method. When the *run* method executes to completion the thread is automatically stopped and its resources are freed.

The second method involves implementing the *Runnable* interface on a class and providing a specific implementation for the interface's *run* method. Incidentally, the *Thread* class employs this technique.

To start a thread with this method, use the *Thread* class constructor that accepts a *Runnable* target as a parameter (an instance of a class that implements the *Runnable* interface) and call the *Thread* class's *start* method:

```
public class MyClass implements Runnable
{   public static void main ( String args[] )
        { new Thread ( new MyClass()).start();}
    public void run ()
    {...}
}
```

We herewith gives an example (Tower of Hanio) to understand how to control the animation.

There are three buttons: "Run" is to operate animation automatically. "Step" is to keep animation running step by step. "Clear" n is to stop animation whenever you want. How to synchronize these actions, how we use thread to get these functions done. Here we performed as following.

- initial a thread.

```
myThread = new Thread(this);
myThread.start();
```

- perform action with each button

```
if (src == runButton){
 isStepping = false;
  resume();
 }
// when step button is pushed, allow thread go one step to suspend point
else if(src == stepButton){
 isStepping = true;
resume();
}
```

*// clear button is clicked, here you can stop an alive thread with click this button. One thing shall call your attention is that you can only stop a thread which is alive now, therefore before you stop a thread, make sure of its status from time to time.*

```
else if(src == clearButton){
 if (myThread.isAlive())
 myThread.stop();
 isSuspended = true;
 isStepping = true;
 stackPanel.clear();
```

- implement animation actions with Thread

```
// when you new a thread and start it, the run function below will be active, so in this
function, it will contain the really action about the animation
   public void run() {
   towerCanvas.initialize(nbrDisks);
   char c =(char)(fromTower.numOfDisk + '0');       // change integer to char
   String nbrDisk = new String("  " + c + "  ");
   stackPanel.push(nbrDisk, fromTower.name, toTower.name, helpTower.name, "   ");
   TOH(fromTower.numOfDisk, fromTower, toTower, helpTower);
   stackPanel.pop();
  }
// The resume () suspend() and pause() functions with thread will help you to switch the
running mode( auto or step) , here we use a flag called "isSuspended" to do the switch in
these functions.
  public synchronized void resume() {
    if (isSuspended) {
      isSuspended = false;
      notify();
       }
  }
  public synchronized void suspend() {
    isSuspended = true;
  }
  void pause() {
```

```
  suspend();
  try {
   synchronized(this) {
    while (isSuspended)
     wait();
   }
  } catch (InterruptedException e){ }
 }
```

Above is the typical example to use the thread to simulate animation in our project. The other implementations nearly follow the same way.

## 4.5 Animation issue – Where to Animate

In Java, you draw on a graphics context; therefore, you should do all the animation within a component class's *paint* method, because that method is passed a *Graphics* object by the Java runtime system. The *Graphics* object represents the component's graphics context.

This implies that you should derive you animation class (directly or indirectly) from the Java *Component* class, because Component is the ancestor of all graphics components and is the base class that defines the virtual *paint* method. (If you create an applet, you will automatically be deriving from the *Component* class).

You can override *paint* to perform the desired animation with a multitude of drawing methods from *Graphics* class (e.g., drawRect, drawImage, etc). The animation thread's sole responsibility is to effect a change on the drawn image's state (e.g., change coordinates, change image) and to call the *Component* class's repaint method. Internally,

*repaint* calls the *Component* class's *update* method, which clears the drawing area, and in turn class your component's *paint* method.

# 5. Implementations

In this project, we introduced more than 20 implementations under Data Structure topic. User can understand each algorithm behind the implementation through this software. By telling you the each step inside the algorithm, it is a very useful tool for both learners who intend to study the Data Structure by himself, and teachers who attempt to explain and introduce the Data Structure in a vivid and profound way.

Here we simply introduce some particular implementations in this software.

## 5.1 Tower of Hanio

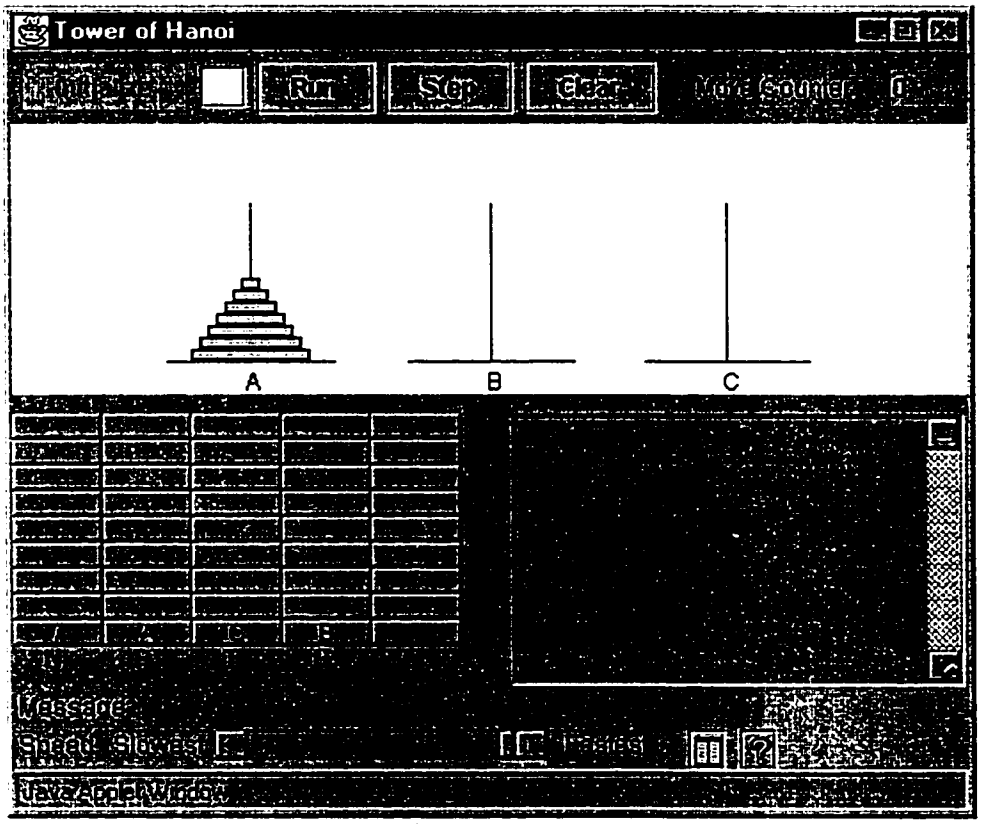


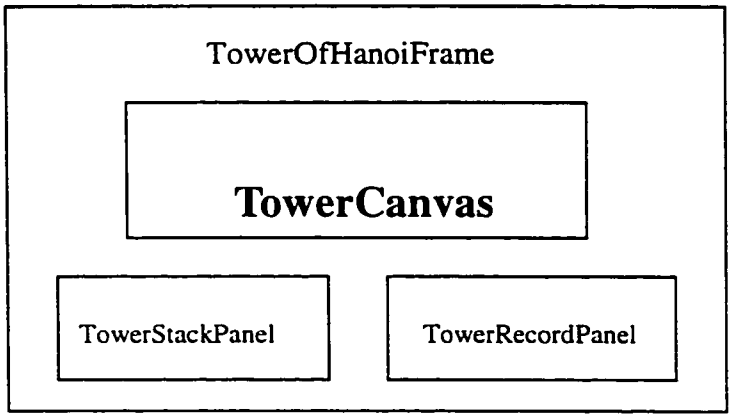

Figure 8: Tower of Hanio

The Towers of Hanio puzzle give us a good example for the recursive algorithm. An algorithm is recursive if it calls itself to do part of its work. For this approach to be successful, the "call to itself" must be on a smaller problem than the one originally

attempted. In general, a recursive algorithm must have two parts: the base case, which handles a simple input that can be solved without resorting to a recursive call, and the recursive part which contains one or more recursive calls to the algorithm where the parameters are in some sense "closer" to the base case than those of the original call.

The Towers of Hanoi puzzle begins with three poles and n rings, where all rings start on the leftmost pole. The rings each have a different size, and are stacked in order of decreasing size with the largest ring at the bottom. The problem is to move the rings from the leftmost pole to the rightmost pole in a series of steps. At each step the top ring on some pole is moved to another pole. There is one limitation on where rings may be moved: A ring can never be moved on top of a smaller ring.

Through playing with these towers, the users can understand the concept of recursive deeply. The above figure 8 shows the main interface for this implementation. When Initialize button is pressed, the number inputted in the text box will be taken as number of disks. A maximum of 9 disks is allowed. Maximum number of lines in the execution stack that located in the left middle of the interface is equal to the number of disks. The stack grows from bottom up. The label are N for number of disks, F for from peg_ ID, T for to to_peg ID, H for help_peg ID, and L for the code line number for this call. The code animation window will help user to understand it with the code accordingly.



1 TowerOfHanoiFrame.class (defined in the file TowerOfHanoiFrame.java):

This class defines the main animation window for this project. It contains and synchronizes three animation parts: TowerCanvas, StackPanel, and RecordPanel.

- This class extends AnimationFrame, implements ComponentListener ActionListener and Runnable.
- Data Members:

```
JButton initButton;                 /*initialize the number of disks*/
JButton runButton;                  /*run animation successtively  */
JButton stepButton;                 /*run animation step by step  */
JButton clearButton;                /*clear towers and internal data*/
Tower From, To, Help;               /*three towers */
int nbrDisks;                       /*user input initial number of disks  */
Dimension stackSize;                /*the size of stack animation panel*/
Dimension recordSize;               /*the size of record area panel */
Panel animationPanel;               /*Display the animation of Towers*/
/*animation panel include a TowerCanvas, a TowerStackPanel and a
TowerRecordPanel */
TowerCanvas animationCanvas;        /* Display the animation of Towers */
TowerStackPanel     stackPanel;     /* Display the dynamical change of stack*/
Tower RecordPanel record;   /* Record the moving sequence      */
Thread myThread;                    /* For synchronize three display panels      */
```

boolean suspended=true;

boolean doStep=true;

- Method Paint

Check if resized window is big enough to hold the all the values of array on screen.

if the window size too small, set error message.

else if window size is big enough, the three animation parts are resized to fit the new screen

- Method actionPerformed

if action is Code Animation, do code animation

if action is initialize, check the range of inputted integer, then initialize the disks

if action is run, run the animation

if action is step, run one step of the animation

if action is clear, remove all disks from the screen

- Method Run

call function TOH(NumberOfDisks, From, To, Help)

- Method TOH(int N, Tower F, Tower T, Tower H)

1) if N= = 0, then return

2) else

2.1) if N > 0, push the string " N-1, F, H, T" into StackPanel

2.2 )call TOH(N-1, F, H, T)

2.3 )if doStep, suspend here

2.4) do move(F, T) by moving a disk on the animationCanvas

2.5) add a record of moving onto RecordPanel

2.6) if N > 0, push the string " N-1, H, T, F" into StackPanel

2.7) call TOH(N-1, H, T, F)

2.8) pop out one line from StackPanel

2 TowerCanvas.class (defined in the file TowerCanvas.java):

This class is used to draw the animation of three towers.

- Data Members:

  Image offScreenImage // for double buffering

  Graphics offScreenGraphics

  Graphics graphics

  Dimension windowSize

  Tower From, To, Help //passed from TowerOfHanoiFrame by reference

- Method Initialize

  1) for( int i = numberOfDisks; i > 0; i--)

  1.1 )set position for each disks

  1.2 )set size for each disks

  1.3 )create each disks

  1.4 )put disks to its position

  2) repaint

- Method clear

  1) Remove all disks

  2) paint

- Method moveDisk(Tower from, Tower to)

  1) set the distance for disk moving

2) call move function from the TowerDisk class

3) repaint

- Method Paint

1) set the center of each towers according the window size

2) draw three towers without disks on the offScreenImage

3) draw all disks for each tower on the offScreenImage

4) draw the image on the screen

3 TowerStackPanel.class (defined in the file TowerStackPanel.java):

This class is used to show the contents of stack that contains the sequence of recursive call of TowerOfHanoi.

- Data Members:

Button array[] = new Button[45] /* 45 nodes used to display max 9 lines of "N, F, T, H, L" */

int index

Tower From, To, Help /* passed be reference*/

- Method push(String s1, String s2, String s3, String s4, String s5)

1) set labels with the input string for five top nodes

2) increase the index

- Method pop

1) Remove labels for five top nodes

2) decrease the index

- Method clear

  Remove label for all nodes in the stack

4 TowerRecordPanel.class (defined in the file TowerRecordPanel.java):

This class shows the disks moving sequence

- Data Members

  TextArea record; /*display the record of moving*/

- Method setRecord(String s1, String s2)

  1) set the top string in the array with s1 and s2

  2) append the top string to the textarea

- Method clearRecord

  Set record to null

5 Tower.class (defined in the file Tower.java)

This class is used to draw a tower in TowerCanvas.

6 TowerDisk.class (defined in the file TowerDisk.java )

A disk is an object of TowerDisk class.

## 5.2 Linked List -- Circular Linked List

A list is a finite, ordered sequence of data items known as elements. We introduced three implementations for the list: Circular Linked List; Double Linked List; Linear Linked List. From Figure 9, we can see the one implementation-"Circular Linked List". Through animation of the nodes and pointers the program shows how to insert a given number into a user-specified position of the list, how to delete a number from user specified position and how to lookup a user specified number in list, how to implement linked list.

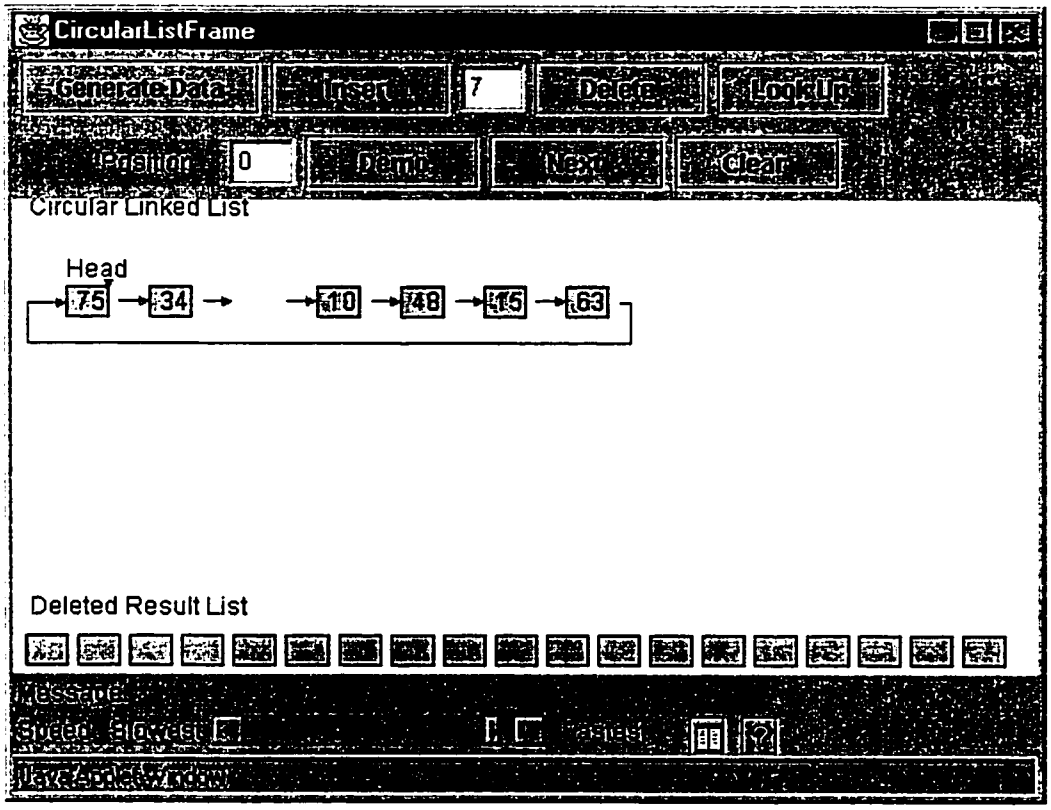



Figure 9: CircularListFrame

1. Design Rationale

The underlay of the program deals with details of how these operations are constructed from simple instructions of a programming language or another. An object-oriented language will implement those operations in a different manner than an older type of programming language. Then, the programmer's choice is also important for these lower level functionality.

Although Java language uses reference for all the objects and variables it does not explicitly support pointers. But for a linked list pointers are really necessary to link all the nodes together. In this program we make a compromise that we implement the list as an array-based list. But on graphic user interface users can only see a linked list represented by images of nodes and pointers.

The problem is that array-based list has a limitation on its maximum size not as linked list does. If we assign a big number to it we may waste most of the space of the array. In contrast if we assign a small number to it, a problem will arise when user resize the window. The list can not grow accordingly when the user wants to insert more data than the maximum size even if the list only occupies a small portion of the window.

The solution for this problem is that when user resize the window we declare a new array for our list according to the window size and copy the contents of the old array to the new one. Since these actions are taken place behind the scene the user will not notice any of these from the user interface. User can only see a linked list growing and shrinking on the behalf of his/her actions.

➢ Double Buffing

This program is a human-computer interactive system. So the program provide a graphic user interface. In the animation part of the interface we have many images to draw, two arrays of nodes and one/two array of pointers. Sometime we have to erase them and repaint them. Moreover, when a node is moving on the screen we have to erase it and repaint it in next position continuously. This will cause flickering problem on screen. In this program we use two techniques to reduce flickering.

- We repaint the only part of the area that we have change and remain the rest of screen untouched.
- Use double-buffering to build an offScreenImage frame. We paint all the images on this offScreenImage, then place the entire offScreenImage to onscreen once.

After using the combination of two techniques user can see animation on screen very smoothly.

➢ Mouse Events

Linked-list has an important variable, curr, which is the current position in list. For insertion operation the new number will be insert into this position and for deletion the number on this position will be deleted. Unlike stack or queue structure, when user wants to insert/delete an element to/from a linked-list, user must specify a position. In this program we provide a text field to user on user interface. User can input an integer as a position then perform insertion or deletion. The program will validate the input to see if the position is in list.

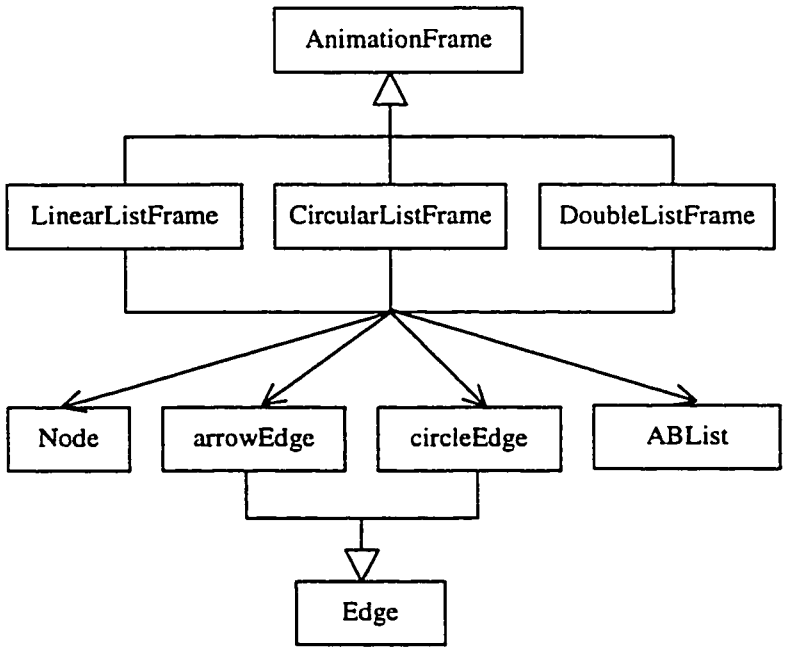
For user's convenience we decide to develop an important feature that user can use mouse to select a position in the linked-list. The only thing user need to do is to point the mouse cursor to one of the nodes and click on it. The selected position will show in the position text field. It is for sure that this position is in the list.

Another feather using mouse event is that user can perform reinsertion from the deletion result list to animation list. By clicking on result list user can select an element that he/she want to insert it into animation list and by clicking on animation list user can select a position. Insertion becomes very easy and error free.

2. Classes Design

LinearListFrame class, CircularListFrame class and DoubleListFrame class are three major classes. They all inherited from AnimationFrame class, which provides a window frame that can handle exit event and resize event. They use Node class, arrowEdge class and circleEdge that provide graphic presentation of nodes, pointers and tail pointers of the lists, respectively. And also LinearListframe classDoubleListFrame class and

CircularListFrame class use LinkedList class to hole nodes and pointers. ArrowEdge class and circleEdge class are inherited from EdgeClass.




Class diagram for LinkedList subsystem

Followings are the detail description about the each class in implementing CircularListFrame as an example.

➢ Class Name: *CircularListFrame*

CircularListFrame class is responsible for handling animation of most important operations such as deletion, insertion and lookup on linear linked list. It also provide other auxiliary features for Demo, which automatically shows user how insertion and deletion work, for clear, which clears the images on animation part of the screen, for Help, which can provide help information to user, and so on.

Methods:

- *init():* Initialize all the instance variables.

- *paint():* Paint all the Nodes and arrowEdges on offScreenImage according to their visibility then place it to onscreen. In case user resizes the window, it declare new list of Nodes and arrowEdges and copies the contents of old list to new list.
- *actionPerformed():* It responses on behalf of user input. It can perform animation of Generate Data, Demo, Insertion, Deletion and Lookup by calling corresponding methods.
- *insertionAnimation():* It handles insertion animation. First it checks the position user wants to insert the data. If the position is inside list this function shifts list Nodes one position to the right and leave a space for insertion Node. Then it moves the animation Node from input text field to the specified position.
- *deletionAnimation():* It handles deletion animation. First it move the Node in user specified position from Node list to dump, the result list. Then shift the Nodes in list, whose positions are greater than the position of the deleted Node, one position to left side. If the dump list is full, it shifts all the Nodes in dump one position to left and leave last space for deleted Node.
- *mouseClicked():* Highlight the node to be selected and set its position into the appropiate text field.
- *blockOutList():* Unhighlight any highlighted Node in Node list.
- *blockOutDump():* Unhighlight any highlighted Node in result list.
- *randomData():* This function generates ten random data and inserts them into Node list at random position.
- *clearAll():* This function clears the animation part of the screen. It erases all the Node in list and their related arrowEdges.

- *clearNode()* and *showNode():* Erase or draw one Node on offscreen then place it to onscreen.
- *clearListEdge()* and *showListEdge():* Erase or draw one ArrowEdge on offscreen then place it to onscreen.
- *drawCurrentFlag()* and *deleteCurrentFlag():* Draw or erase a flag that indicates the current position.
- *drawTailFlag()* and *deleteTAilFlag():* Draw or erase a flag that indicates the tail position.

➢ Class Name: *ArrowEdge*

The class can instanciate arrowEdge objects. An arrowEdge object contains one or more line and has an arrow on one side. It provides functions to draw or erase an arrowEdge on offScreenImage.

Instance variables:

- *Start:* A point object that contains a x-coordinate and a y-coordinate. It is a start point of an arrowEdge.
- *End:* A point object that contains a x-coordinate and a y-coordinate. It is an end point of an arrowEdge.
- *Visible:* A boolean variable that indicates whether an arrowEdge should be erase or not.

Methods:

- *StackShow():* Draw an arrowEdge on offScreenImage.
- *StackErase():* Erase an arrowEdge on offScreenImage.

➢ Class Name: *ArrowEdge*

The class has the same responsibilities as arrowEdge. It is a specialized arrowEdge that draws or erase a pointer from the tail to the head of the list.

Instance variables:

Same as the *ArrowEdge* class.

Methods:

Same as the *ArrowEdge* class.

➢ Class Name: *ABList*

An array-based list uses an array to store the elements of the list. It has a fixed size. Elements can be inserted into or deleted from the list. This class also provides a set of functions to set the current position in list and many other utilities such as searching an element in list.

Instance Variables:

- *msize:* An integer that holds the value of the maximum size of the list.
- *curr:* An integer that holds the current position of the list.
- *length:* An integer that indicates the number of elements in list.
- *listArray:* An integer array that stores the elements of the list.

Methods:

- *insert():* Insert an element into the array-based list.
- *setFirst():* Set current position to the first element.
- *next():* Set current position to next element in list.
- *prev():* Set current position to previous element in list.
- *currValue():* Return the value of element in current position to caller.
- *isInList():* Check curr to if it is in the list.

- *findIndex():* Search for a given value in list. If find this value return its position to caller. If not return –1 to indicate no such a value in list.
- *remove():* Remove a value from the current position of the list and decrements the lengthfunction of stack pushes a value into the top of stack and increment the top index.

# 6. Conclusion

Generally speaking, the project is exciting for me. From it, not only did I derive more knowledge and practice with fantastic JAVA, but also those who want to learn the complex but basic data structure may reap benefits. Although, I present more than 20 implementations for the data structure algorithm in this project, there are still some more for further completion. Here I would like to show my gratitude my professor Dr.Tao Lixin, for his contribution to perfect this project. Should you need any help or information, please feel free to send email :ylsun@cs.concordia.ca.

**Reference Book**


1. Clifford A. Shaffer , Virginia Polytechnic Institute and State University. *A Practical Introduction to DATA STRUCTURES AND ALGORITHM ANALYSIS.*
2. Ben Shneiderman, Addison-Wesley, 1998, *Designing the user interface: Strategies for Effective Human-Computer-Interaction.*
3. Horstmann &Cornell, *Core Java2.*

# Appendix

## 1. System Deployment

Before you run this software, you have to do some preparations with your computer.

With application:

1. Download the package named "JAVA 2 SDK standard Edition 1.3" from the Sun JAVA web page: www.java.sun.com
2. Install the package following the instruction on the web page.
3. Don't forget to add the PATH and CLASS PATH in your PC system setting.
4. Go to the directory where the software located, use command :

   -> java RootUI

With Applet:

1. Download a browser either Netscape or Internet Explore with latest version if you don't have it.
2. download the package named "JAVA 2 SDK standard Edition 1.3" and Java Plug-in from the Sun JAVA web page: www.java.sun.com
3. Go to my web page : www.cs.concordia.ca/~grad/ylsun/Root.html

## 2. How to use JAR file

The Java™ Archive (JAR) file format enables you to bundle multiple files into a single archive file.

- Creating a jar file

1) you need create your own Manifest.txt file which contains:

Main-Class: RootUI.class

**********

Warning: The text file must end with a new line or carriage return. The last line will not be parsed properly if it does not end with a new line or carriage return.

**********

2) create the jar file with command:

jar cvmf Manifest.txt Root.jar *.class *.html help image

Then the Root.jar file is created with all the class files, html file, help directory and image directory.

- Viewing the Contents of a JAR File

Using command: jar tf Root.jar

- Extracting the Contents of a JAR File

Using command: jar xf Root.jar

- Running JAR-Packaged Software

1) with applet

Modify Root.html file with adding :archive="Root.jar"

2) with application

Using command : java –jar Root.jar