

Submitted for the Degree of B.Sc. in Computer Science 2011

ROSS2: Visualisation of Stack and Queue

200735816

Mark Kelly

Except where explicitly stated all the work in this report, including appendices, is my own and was carried out during my final year. It has not been submitted for assessment in any other context.

Student signature: _____

Acknowledgements

I would like to thank my supervisor Isla Ross for providing guidance throughout the creation of this project. Also thanks to Mark Dunlop for providing constructive criticism during the project poster day. This criticism helped me try and make improvements to my time management.

Not really sure what else to put here so I'll just say thanks to whoever reads this for taking the time to read it.

Abstract

This is a report which will go into the details about the development of a piece of software which is supposed to output visualisations of various implementations of stacks and queues. This program was developed in the Java programming language. The software is aimed at being a learning aid for 2nd year computer science students who need to learn about the ins and outs of various stack and queue implementations.

Contents

- Introduction
- Background research
 - stacks and queues
 - similar programs
- Specification
 - project overview
 - desired functionality
 - the specification itself
 - Marking scheme used
- Design
 - design methodology
 - designing a GUI
 - class design
 - how to get some stuff working
- Implementation
 - The GUI
 - The imps
 - Iterators in the implementations
 - Something that wasn't done
- Testing
 - Testing during implementation
 - Testing after implementation
 - what wasn't tested

- Evaluation
- Appendix A: bibliography/other references
- Appendix B: test data
- Appendix C: user guide

Introduction

This is a report which will go into detail about the many aspects of the development of a program for the class 52.497. Every student in 4th year was required to choose from a selection of projects to undertake over the academic year. The project which I was allocated was called “visualisation of stacks and queues” (as you probably gathered from the title page of this project). Here is a description of what the task initially entailed from the project offerings list:

“This project is to produce a piece of software which will act as a visual learning aid for students studying the stack and queue abstract data types and implementations of these. Users should be able to interact with the software in order to view the results of various operations, while providing their own data where appropriate. In addition, a facility should be included to enable the user to predict the outcome of an operation before it takes place and the software should then check whether or not the prediction is correct.

The software should also enable the user to explore a number of possible implementations for stacks and queues in terms of appropriate visualisations of these implementations. The user prediction facility should be included here also.

For a greater challenge, in addition to the above, the visualisations should take the user down to a lower-level still, down to individual lines/small fragments of Java code.“

Throughout this report I will go into details about parts of the project such as the background research, the system specification, the design, testing and overall evaluation of the finished product.

Background Research

Before functionality and a specification can be decided upon, background research has to take place. This is done to find out about programs which are similar to the one which your project is asking you to implement. It is also done to research the various aspects of the subject matter which the project is based upon. In the case of this project, research would have to go into the following subjects:

- The abstract data types stack and queue
- Stack and queue visualisation programs

I will go into some details about each of these two subjects, including some information about the various implementations of each of them which I intend to use.

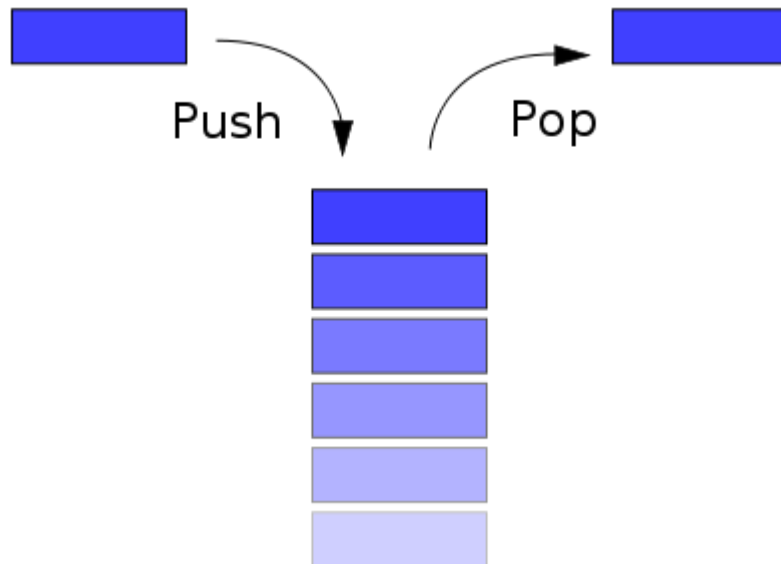
Stacks and Queues

These two things in a programming context are referred to as abstract data types (or ADTs), pretty much meaning that they have a generalised approach which is followed by everyone. They are normally taught to 2nd year computer science students in programming classes and hence are a prime candidate for being the subject of a study aid. I will give a small explanation about each ADT and go into my findings about the various ways which they can be implemented. This was mostly done by looking at slides for the class CS207 Advanced Programming. Mostly the research in this section was for the sake of revision.

Stack

The basic premise of the stack data type is that you pile a piece of data on top of the last piece of data which has been added to the stack. When something is to be

removed from the stack (or popped from the stack), the thing on the top of the stack is removed. This was the last piece of data which was placed in the stack, therefore the stack ADT is a last in first out ADT (or LIFO). A pictorial example of this can be seen below.



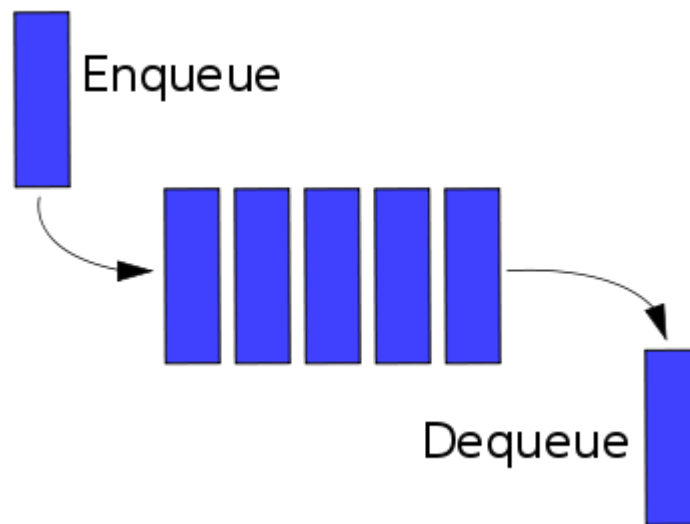
There are two main ways in which this structure can be implemented and these are by making use of an array or creating a linked list. There are other ways to implement a stack (like for example, using an extendible array), however I felt that based upon the slides which I had looked at, it would be wise to stick to the implementation styles which are relevant to the course.

The array way of implementing a stack involves placing data into the slots in a limited list of slots each with their own indices starting from 0. The linked list way of implementing a stack involves the use of connecting nodes through links.

Queue

The queue data type works in a first in first out (or FIFO) way. As data is added

to a queue, the value at the front of the queue (or head of the queue) is stored. Once the value at the head of the queue has been dequeued, the value which was input before it then becomes the head. The way in which this is established is entirely dependent on how the queue is implemented. Below is a pictorial example of the regular functionality of a simple queue.



Like the stack ADT, I've decided that the best implementation choices for a study aid would be to use array and linked list implementations. However when it comes to queues, there are various ways in which an array can be used to implement a queue. This mostly comes down to how the implementation handles the removal of the head of the queue. It can be handled by either sliding all of the values along the array one index to the left. This way is rather inefficient however since a loop has to be applied to get everything to move along every time you remove something from the queue.

A more efficient way to use an array would be to make the array handle numbers in a cyclical way. This means that instead of moving the value in the array, it would be more efficient to change the index value of the head by incrementing it by one.

The linked list way of implementing a queue would involve the linking of nodes for making additions to the queue and removing nodes for removing items from the

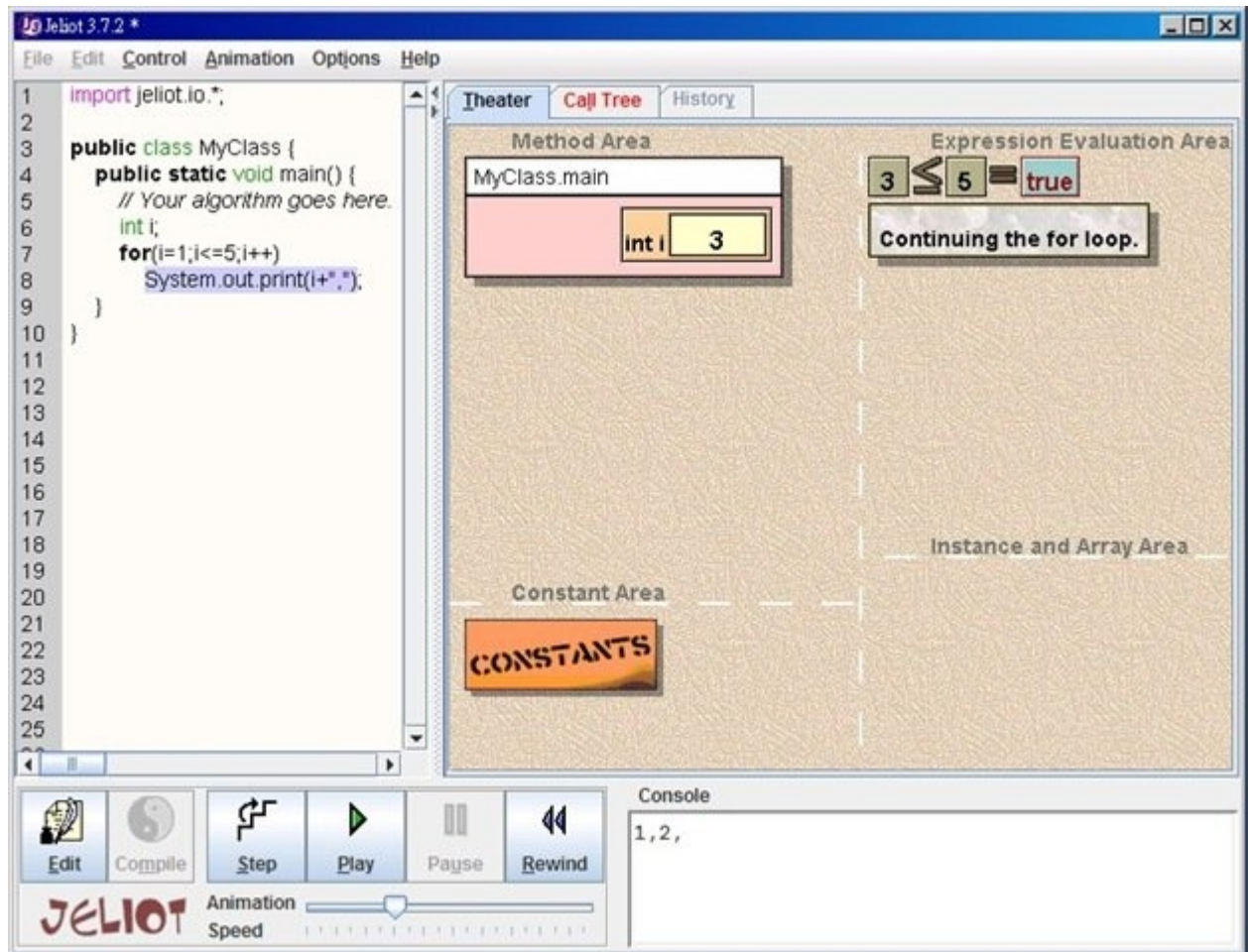
queue. It makes use of a one way link to each node to make the connections to each node. Therefore one node is connected to all the nodes before it as one large node.

Similar programs

The next part of the background research for the project involved looking around for similar programs to get an idea about what would be some useful functionality for my program. This information and research would be valuable in putting together a system specification for the project. I will also come to a conclusion overall about how well these previous programs in general handle visualisation. This will mostly be for programs which specifically deal with stacks and queues however there is a couple which work with general visualisation of code as well.

Jeliot

Jeliot is a visualisation program which takes in code which the user creates and visualises the movement and changes of variables within the program.



It allows the user to step forward or backwards through the code which they have implemented and makes use of animations to display what is happening in the program. It shows the main method, the expression which is being altered, the arrays and the constants in separate areas.

For smaller programs, I would say that this program can be a very useful learning aid and could perhaps be used as a debugging tool since it shows the back end of what is going on in the program in great detail. However, for larger programs I feel that the visualisation area would become overly cluttered and hard to follow. It also doesn't show the structure of implemented abstract data types either. It also doesn't provide any algorithm information since you have to provide the code yourself to get a visualisation to appear.

This example comes from a program created by a student from Bogazici University in Turkey. The program has a few buttons for pushing and popping specific values into a stack or queue. There is also functionality to clear the data structure and to switch what the visualisation represents in real time without having to clear the values from the data structure.


(Block...)

Tom Jerry Tweety Snoopy Plucky Taz Piggy

Jerry	Tom								
-------	-----	--	--	--	--	--	--	--	--

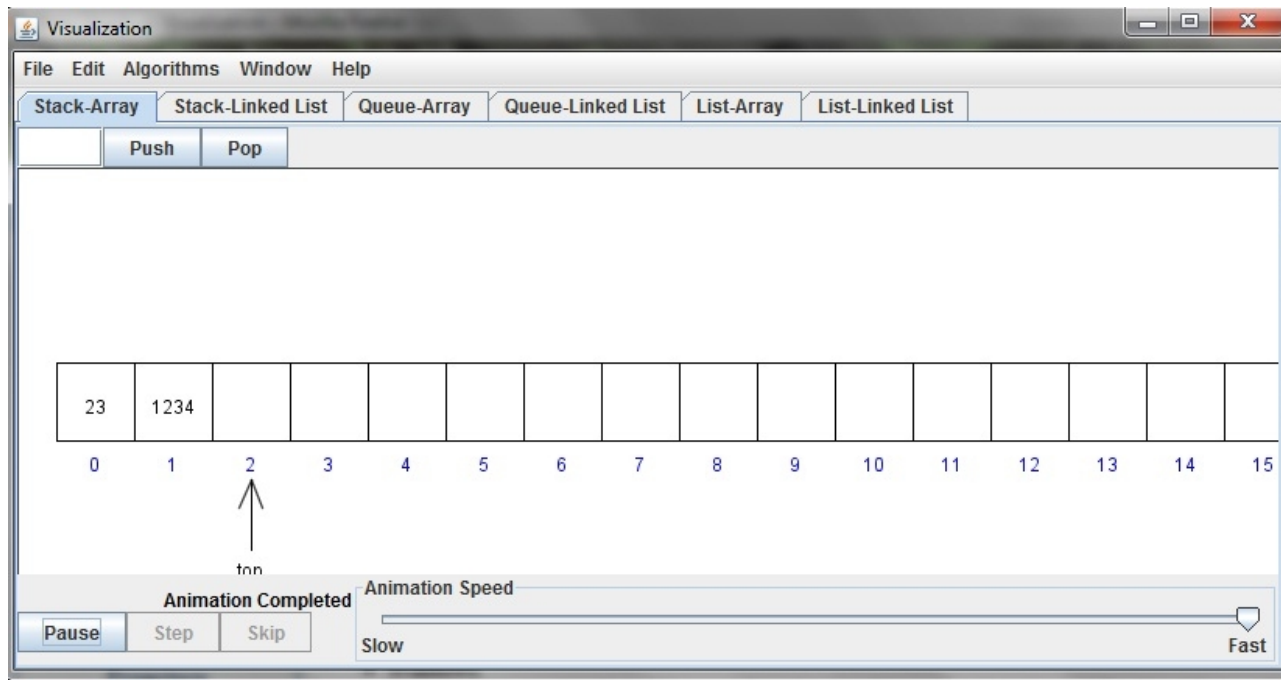
^ ^
Front Rear

add delete reset refresh Selection: Queue ▾



Overall I would say that the program provides a decent visualisation of the high level abstraction of stacks and queues. However the program itself provides no indication about what sort of implementation of either ADT is being used. There also isn't any signal that there could be multiple ways to implement a stack or queue. There is also lack of flexibility on what data can be placed into the data structures since buttons with specific data to be placed into the stacks and queues are used instead of using some means of taking in user data.

This is a program which was developed by an Associate Professor from the University of San Francisco called David Galles. I would assume that he uses it to teach classes, and after working with it for a while it would appear to be a very good program and would be useful to pretty much anyone who needs to learn about a lot of data structures in programming.

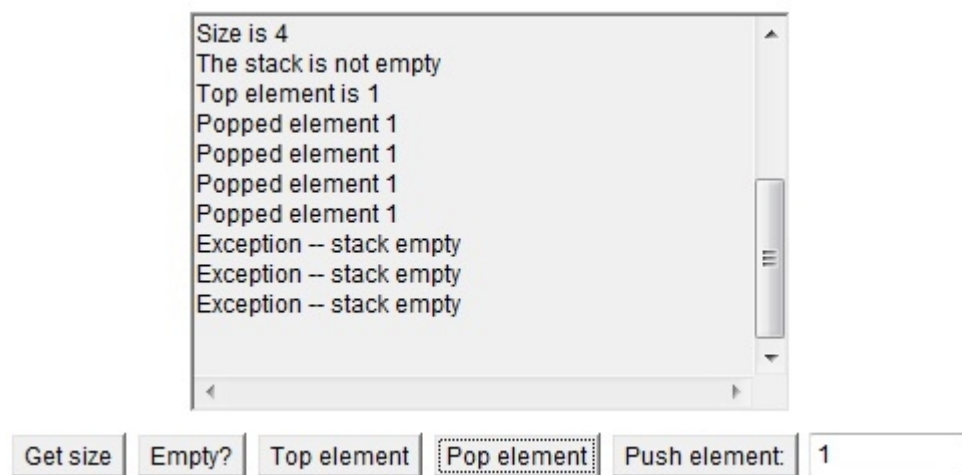


The program provides visualisations for lots of different data structures, including some different implementations of the stack and queue ADTs. The visualisation of the data structures are excellent and are brief while still providing the important required details. There is also a means of altering the speed of the animation of the visualisations. While it isn't essential it is still a nice feature. The program also keeps all of the various implementations of each data structure under their own options and lets the user jump between different implementations by clicking on tabs for each of them. The only minor gripe that I have about the program is that it doesn't provide any code for any of the implementations, however that barely detracts from the usefulness this study aid can provide to a student in understanding the basic concepts of the data

Overall I would say that the programs provided are good for people who are just starting to learn about the data structures, however beyond that I would say that they are not particularly useful.

Java4.datastructures.net

These programs were provided as resources from a book called Data Structures and Algorithms in Java by Michael Goodrich and Roberto Tomassia. This is a set of four Java applets which provides a console based visualisation of stacks and queues using linked lists and arrays.



As you can see, the visualisation isn't particularly amazing since it only provides a console view of what is happening to each data structure. The console seems to be missing a feature to display all of the contents of the data structure which is being worked with and only allows you to see the top value of whatever structure you are working with. There is also very little information about the algorithms which are being used to establish the data structures and there is also no mention of any of the code which is being used in the program.

On the upside however, it provides a basic, uncluttered view of what is happening when it comes to the pushes and pops which are being executed.

General observations

After looking at a few different visualisation programs, I can conclude that while all of them were relatively user friendly there was something wrong with all of them. Some of these items may have been rather trivial, but they were wrong nonetheless. These perceived mistakes were mostly features which were never added. All of the programs did what they did rather well, albeit with a few quirks here and there in each of them.

None of the programs had any means of quizzing the user about the data structures and none of them had much reference to the code which was being used. So in conclusion the programs were all fine but were generally let down by not going into a deep enough level of abstraction.

Specification

The specification of the project is essentially just a set of goals which the programmer aims to achieve with the program which they are creating. This tool is useful in providing the programmer a guideline as to what they should be developing and it also makes for a useful evaluation tool once the programmer has settled upon a final product. Before a final specification can be put together I had to first try and build a general overview of what I would like my program to do and then try and refine my goals from there.

Project overview

At this point I had to take into consideration the general overview of the project (which is provided in the introduction of this report) and then try and come up with my own. Here is what my initial idea for the project was (taken from my initial specification):

“The main point of this project is to produce a piece of software which will act as a visual learning aid for students (generally 2nd year Computer Science students with Java knowledge) studying the stack and queue abstract data types and implementations of these.

The program needs to allow students to interact with it. It needs to let the user be able to make alterations to data sets so that the user can see how each structure alters data (for example, pushing and popping). Also required is the ability to ask the user what they think will happen when they apply a push or pop to either structure, providing feedback on whether or not the user was correct.

Multiple implementations of both of these structures will also be explored (like arrays and linked lists for example) along with quiz based interactivity (such as asking the the user to choose the correct result of whatever process is currently executing)

throughout the visualisation of these implementations to further aid the education of the users. An extra task which has been made available for this project which I wish to do is to provide further in depth visualisation (looking at chunks or individual lines of Java code) of each of the implementations which will be demonstrated.”

From this point I could then take into consideration what sort of functionality I wish for my final program to have.

Desired functionality

The developer must then take into consideration what kind of functionality they would like their system to have. In the case of this program, I decided that I wanted to have the following functionality in my final product:

- There should be the ability to push and pop values from a stack or a queue. This should be done by input provided by a user.
- There should be the ability to undo something which has been previously done and also the ability to clear whole stacks and queues if required by the user.
- There should be some form of visual representation showing what happens when the user executes a push or pop function upon the data set they are currently working with.
- There should be a console to display questions and feedback to the user. This feedback could be used as a basic level of visualisation to the user.
- There should also be a console providing code for each of the various forms of the implementations of each abstract data type.
- There should be a test function which displays questions inside a console for the user to answer. This would further add to the user interactivity of the project

From this set of goals I can then attempt to put together a system specification.

The specification itself

With all of that information, a final specification can be put together. For my specification I decided to settle upon the following:

“The program has to be able to display multiple implementations of the abstract data types stack and queue. These implementations should be the following:

- A simple look at the stack ADT
- A simple look at the queue ADT
- An array implementation of a stack
- Cyclical and sliding array implementations of a queue
- A linked list implementation of a stack
- A linked list implementation of a queue

The program has to allow users to manipulate the contents within each of these implementations. This can be through adding and removing numbers from each of the implementations, moving back a step if the user makes a mistake and totally clearing the contents of a data set for an implementation.

There has to be some form of visual feedback of what the user is doing with each of the implementations. Some form of console will be required to provide the user feedback on what they are doing, along with a pictorial representation of what they are doing.

As an extra piece of user interactivity, a test mode should also be available to provide the users with questions about both of the abstract data types. These questions can be made to display in the same console which feedback is provided in.

For a further level of abstraction, another console which displays the code for the users chosen implementation can be provided.”

This can be used later on as a tool of evaluation for my finished program.

Marking scheme used

Since the main point of this program is to produce a piece of software at the end of the development cycle and there is very little experimentation involved in this project, the marking scheme which was used for this project was the marking scheme for a software development based project.

Design

In this part of the report I will go into the various aspects of the design process I went through for the project.

Design methodology

The first thing which I had to do was to take into consideration which design methodology I would use in the development of my program. Initially I decided not to adhere to one, however since this had to be done in a professional manner I had to eventually pick one.

In the end I decided to use the waterfall methodology for my design. This design style pretty much means doing everything in one go. In this case doing the requirements, then the design, the implementation, the verification(or testing/evaluation) and finally the maintenance. I chose this because I felt that in the end it would be the most time efficient way to handle things.

I had also taken into consideration the iterative methodology. This is the same as the waterfall way, only it allows for the developer to loop back to earlier stages in project if necessary. I eventually decided against this because it seemed to have the potential to lead to a fair bit of wasted effort in the long run if I did something wrong in the implementation and then decided to go all the way back to the requirements stage.

Designing a GUI

The next thing which I decided to design was GUI. I decided to start with this since I thought that it would be the easiest part to design at the start and it seemed wise to get it out of the way quickly.

The following things are what I felt were necessary in my GUI:

- There would have to be buttons to handle the adding and removal of numbers. There would also need to be buttons for the handling of back stepping and clearing a data structure. A quit button would also be useful.
- There would need to be a panel for the visualisation part of the program to happen.
- There would need to be a couple of consoles added to the program. These could be done with the use of text areas attached to scroll panes.
- A menu bar for various options would also be required.

For the menu part I decided to boil the initial menu options to file, implementations and help. Inside of the implementations menu there would be two further menus which would be used called stack and queue.

The file menu would contain all of the options for the altering of a data structure. It would pretty much contain all of the options which the buttons will be used for (add, remove, back, clear and quit).

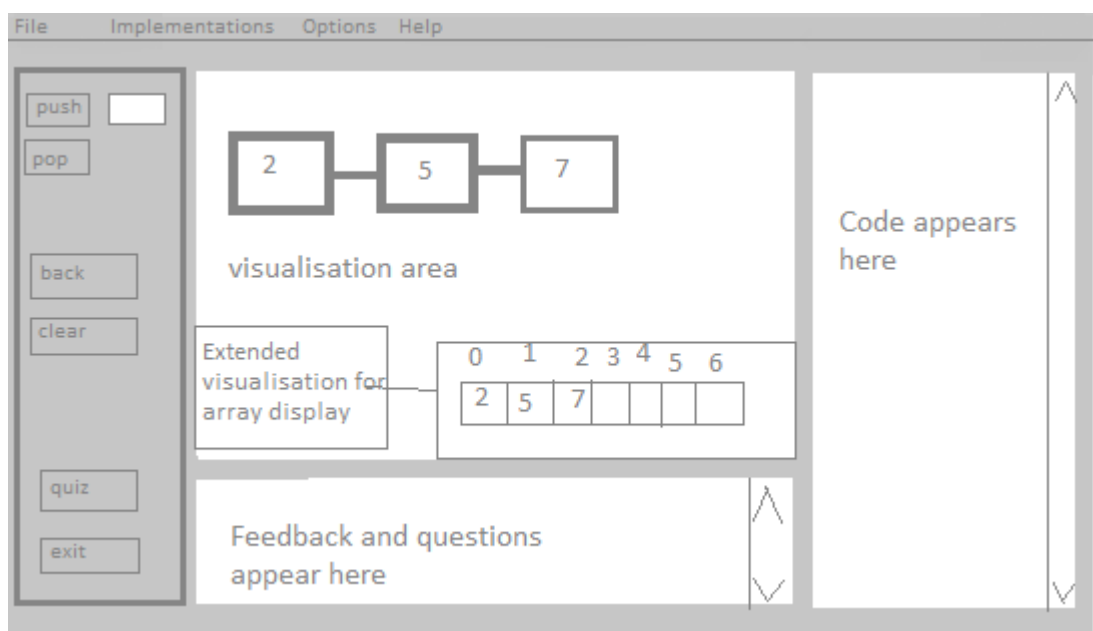
The implementations menu would contain the sub menus stack and queue. Inside of these sub menus would be options to select multiple types of implementations for each of the respective ADTs. These options would be simple stack, array stack, linked list stack, simple queue, inefficient array queue, efficient array queue and linked list queue.

The help menu would contain an option telling the user how to use the program. There would also be a button to tell the user what the project was for.

There was consideration of another menu which would make use of radio boxes

to turn off certain displays and functions (such as turning off and on the code display or turning off and on the question functionality), however these functions wound up being scrapped. The questions one was scrapped due to how I wanted to implement the questions functionality and the code display option was scrapped because the program was just going to display the code and not iterate through it or anything like that so I felt the option would've cluttered the program a bit.

Here is an early concept of how I wanted my GUI to look:

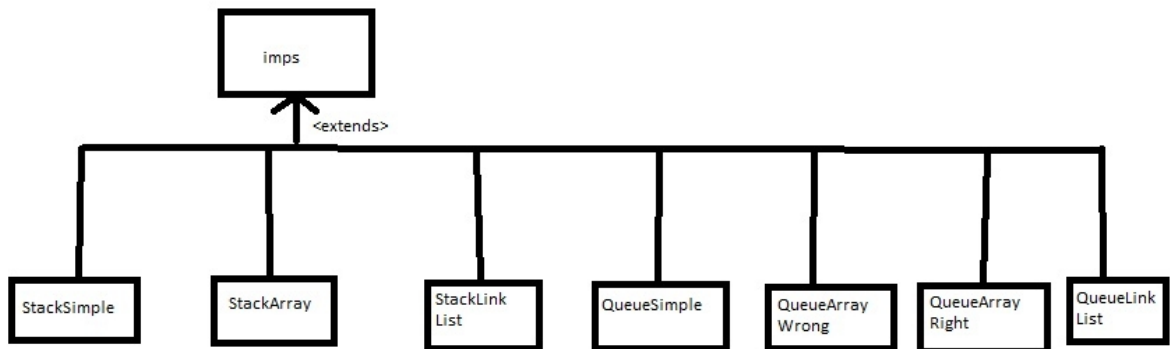


Class design

The next part of the design process which I decided to tackle was what classes my program would've needed and also how I want them to interact with each other.

The first thing I chose to take into consideration was how to separate each of the various implementations I'd need for my project. In the end I settled upon using an abstract class which all of the implementations would extend from. I could've used an interface instead, however I felt that by using an abstract class I allowed myself a little

bit of flexibility to add things to the abstract class if it wound up becoming necessary later on.



For the GUI and the interactivity with the GUI I chose to have a class which would contain methods which the listeners in the GUI class would execute when they were activated. It is good object oriented programming practice to try and keep methods separate from the GUI class. For the painting of the visualisations I felt that it would be best to keep painting methods in their own class as well.

The quiz part of the program could be provided its own class which interacts with the GUI separate from everything else since it doesn't really have to interact with anything else other than the text box inside of the GUI.

There would also need to be a class for creating nodes for linked lists which would only interact with the two linked list classes.

In the end the class list I decided to settle on was the following:

- GUI
- Graphics
- Quiz
- Listeners
- listNode
- imps
- StackSimple
- StackArray
- StackLinkList
- QueueSimple
- QueueArrayWrong
- QueueArrayRight

How to get some stuff working

The next thing I had to do was then take into consideration how I would get everything to work. I say “some” because I couldn't come to a conclusion on how to get everything to work.

The imps

I had to decide what all of the implementations would actually require. I came to the conclusion that all of the implementations would at least require the following things:

- a means of adding a number
- a means of removing a number
- a means of clearing the data structures
- a means of actually accessing each implementation

From this I decided to place add, remove, clear and isSelected methods to my Imps abstract class. This means that everything which extends my Imps class will have to have those methods in them to do anything.

The simple implementations I could use the library classes of Java to put together since very little information will be required to be output for them other than the actual contents of the structures themselves. The other classes involving arrays and linked lists will have to make use of helper local methods to construct the respective data structures.

The isSelected method would also need to handle parsing of a text file so that the code for each implementation can be displayed. Unfortunately due to time constraints I was unable to implement this part of my program.

The listeners

The listeners class will be the main class involved in getting things moving, therefore it'll need to have the most methods. There will need to be methods to get access to each of the implementations and for executing the adding, removing and clearing of numbers. There will also need to be methods for a quit action, a help action and an action to show what the program is about.

In this class there will also need to be a means of differentiating between what implementation has been switched on. There will also need to be something which will check if nothing has been switched on so that the program can have an initial state to display when the program has been turned on.

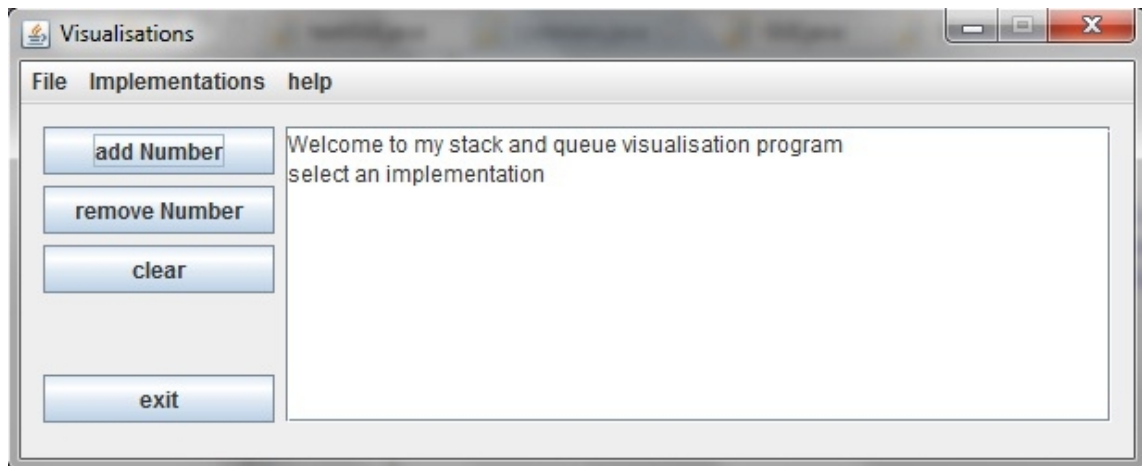
From these conclusions I had come up with a list of methods for the Listeners

class:

- void addNum()
- void removeNum()
- void clear()
- void exitAction()
- void howtoAction()
- void aboutAction()
- void whatImp()
- boolean isNoImps()
- void initialState()

The graphics and quiz

These two parts were where difficulties began to arise. I couldn't figure out a solution for outputting graphics in a way which was tidy and easy to maintain. There were ideas such as trying to create a snap grid in a panel for the graphics. However I couldn't figure out a way to construct a shape with a number inside of it along with a way of connecting to others like it. Eventually I gave up on trying to come up with a solution to this issue and decided to settle on a more console based implementation for my program.



The quiz part was in requirement of a method which would print out a group of questions into the console part of the GUI. There would also need to be some way to take in an answer. Due to time constraints I was unable to implement this part of the program.

Implementation

In this section I will go into some details about the way which I went about implementing my program. I will try and show why I did some of the things that I did and also try and explain why I decided against certain other ideas. There will also be some notes put together about the parts which I was unable to implement.

The GUI

To begin with, the GUI was just a menu bar to make sure the the functionality I was able to implement was working. This piece is known as TestGUI inside my code listing. It doesn't do anything now but it is still in the code.

The main GUI in the final implementation was put together using NetBeans initially. I decided upon this mostly because I was running out of time at the point where I took into consideration how to implement a GUI, but in hindsight it would seem that this was the best solution and would've saved a lot of time if I thought of using NetBeans a lot sooner.

NetBeans creates a GUI by making use of the `groupLayout` style of GUI layouts in swing. This way sets the horizontal layout of the contents of the frame and then sets the vertical layout. After it does this, the contents are then packed using a `pack()` method.

Some changes had to be made to the GUI however. The menus would appear under the buttons for a while. I discovered that this was because NetBeans used AWT buttons instead of Swing buttons, so there was a compatibility issue. After changing all of the buttons to `Jbuttons`, the problem was solved. There was also an issue with nothing appearing inside the console. This problem was alleviated by changing the `JtextArea` console to a static variable. After this everything would print fine to it via use

of `console.append("string")`. There was also the requirement to place a new line after every string. This was done by placing `\n` after everything.

Theimps

To get the implementations to work I first had to develop a way of getting access to each implementation class. It's necessary to come up with a way to get access to one while turning off access to the other as well. The solution I came up with is demonstrated in the following piece of code:

```
ll_queue.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if (l.isNoImps()) {
            l.LLQueueFlag = true;
            l.whatImp();
            ll_queue.setEnabled(false);
        } else {
            l.falseAllFlags();
            resetMenuItems();
            l.LLQueueFlag = true;
            l.whatImp();
            ll_queue.setEnabled(false);
        }
    }
});
```

This is the code which I use in the action listeners in the GUI. A flag is used to represent each implementation. The program first checks if any other implementations are turned on. If there isn't, the program turns on the flag for the respective implementation. Using this, the `whatImp()` method is used. This method is a group of if statements which accesses the `isSelected()` method of the selected implementation.

Also in this section of code is a means of preventing re-clicking of an activated implementation. Using `setEnabled(false)` to make the option inaccessible was the

simplest option for this task. I took into consideration using an extra if statement to provide and output saying that the implementation had already been selected, but that seemed like an overly complicated way to handle the situation. It would've also made the code a lot more unclean.

If an implementation has been selected already, the code does the same thing only it first has to make a few initialisations. This is done through `l.falseAllFlags()` and `resetMenuItems()`. These two set all of the flags for implementations to false and does `setEnabled(true)` to all of the menu items in the implementations menu.

Initially there was a problem with the `isNoImps()` method because the boolean was never changing. This was due to thinking that the boolean would initialise itself after setting it to true. It turned out this was a wrong assumption and it was better to use `if(isNoImps())` instead of `if(isNoImps(true))`. The latter one never reads the true part.

Iterators in the implementations

There were some instances where iterators for things such as the head and tail values were wrong by one. There was even an instance where the index value in the `arrayQueueRight` class which turned to negative one when the tail index became five.

I sorted out this issue by creating a new variable for storing the index. I also used a method to change the tail value to 0 when the tail value reached 5. At this point the cyclical array implementation of a queue was completed. It made the code a little bit more messy but it got everything in that class working properly.

For the `stackArray` I was initially having issues after I did a clear command upon it. This was due to forgetting to initialise the value of the index after doing a clear. It turns out that I made this error in my `queueArrayWrong` as well.

The `queueArrayWrong` remove method was difficult to get working initially but

in the end it works now. Here is the code for that section:

```
private int queuePop() {  
  
    if(queue[head] !=0){  
        int getHead = queue[head];  
        queue[head] = 0;  
  
        if (queueFull() == true){  
            System.out.println("queue full mode");  
            {  
                buffer = 0;  
                for(int j =0; j<4; j++){  
                    queue[j] = queue[j+1];  
                }  
                queue[4] = buffer;  
            }  
        }  
        else{  
            for(int j =0; j<4; j++){  
                queue[j] = queue[j+1];  
            }  
        }  
        tail--;  
        return getHead;  
    }  
    return 0;  
}
```

If I were able to get the code display section to work, this piece of code would've been the perfect way to show why this way of using an array for a queue is really inefficient in comparison to the less code intensive cyclical array style. That code looks like this:

```
private int queuePop() {  
    if (queue[head] != 0){  
        int getHead = queue[head];  
        queue[head] = 0;  
        head++;  
        if (head == size){
```

```
        head = 0;
    }
    return getHead;
}
return 0;
}
```

A lot less lines and a lot easier to read. It also performs less functions since it doesn't have to use a loop at any point.

Something that wasn't done

I was unable to get the linked list implementations working in the program. For some reason which I could not figure out, every time I tried to print the contents of a linked list which was longer than one, it went into an infinite loop. As a place holder I decided to use java api classes like I did for the simple versions of the implementations. It isn't ideal, but at least it still works. I know that the values were going into the linked list without a problem, however outside of that I was unable to make much progress on that.

The code display part of the program was also not implemented. However I know that I could've used a bufferedReader to parse the lines of a text file to display the code in a console box. I was out of time at the point of discovery of this unfortunately.

In all honesty I wasn't even close to being able to implement a graphical visualisation for any of the implementations.

Testing

For thorough testing of my program it is best practice to test your code throughout the implementation process. It is also good practice (and probably common sense) to do a final test upon all of the completed functions once the implementation stage of the project is completed. There are many ways which both of these testing times could've been approached. I shall now explain how I did my testing for each stage. I will also explain what has not been tested.

Testing during Implementation

During the implementation stage I chose to use the console in the Eclipse IDE to make sure that all output was the expected output. The output for each test case were produced by simply using some “System.out.println()” lines of code. I also used this means of testing to debug certain problems I was having during the implementation process. This was mostly to check if certain methods or lines of code were actually being reached by the program. It was also used to check if certain hidden variables were changing like they should be and if they were not it gave me an idea of which piece of code would be in requirement of some debugging. White and black box testing would take place in this phase of the implementation process.

For example, there is a method which checks if no implementations are selected (called “isNoImps()” imaginatively) which isn't intended to output anything in the main program but provides a true or false depending on certain events. Using a “System.out.println()” to show the true or false output, I can see if the boolean is changing at the correct times. This would be an example of white box testing.

There are of course other ways which I could've gone about this testing phase of the project. I chose the method I used for testing because it was the one that I am most comfortable with using. The only downside is that it leaves a considerable amount of

lines in the code which end up needing to be cleaned up at the end of the implementation phase, but that is a trivial issue. There are many extensions available for testing code in Eclipse which I could've made use of instead, such as Junit, however I did not feel comfortable using it since it would require creating a test class for each class as I wrote them and I felt that it would take up too much time. It also felt unnecessary to put too much work into coding which would have no effect upon the finished product.

Testing after Implementation

After I was happy (well, as happy as I could be) that I had completed the implementation phase of the project, further testing would have to take place to make sure everything is working as it should be. This process is also referred to as checking the validation of the program. I would have to test the following things in my program:

- The buttons
- The menu options
- The handling of user input
- The stacks and queues

Testing the buttons and menu options was simply a matter of clicking them and seeing if I got the expected result from pressing them. The input handling however would be a little more complicated and require some more thorough testing than that since there are many possibilities to take into account for that area.

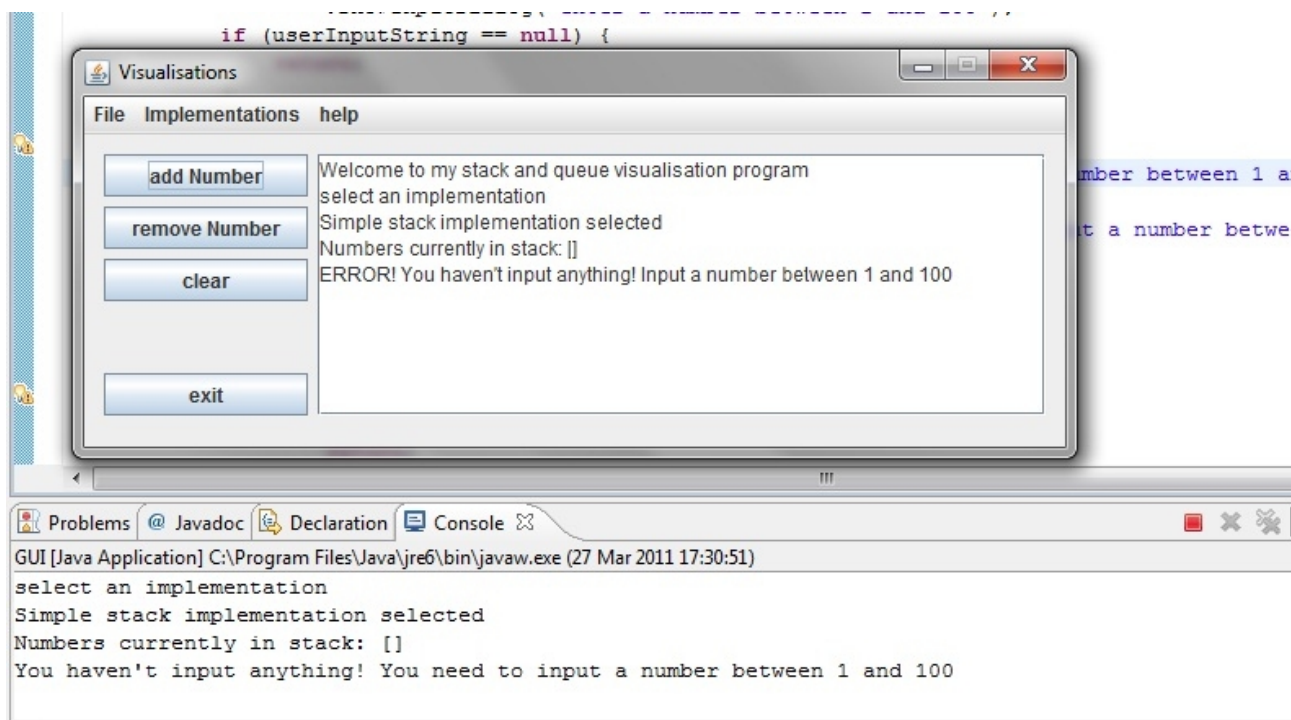
At this point, I chose to test the input handling of the program with the use of normal, extreme and exceptional testing. Normal testing is when the inputs you provide

to a program are well within the limits of what the program is meant to handle. An example of normal testing in the context of this project would be inputting the number 23 to add the number to a simple stack.

Extreme testing is when you provide inputs to the program which are at the limits of what the program should be able to handle. This is used to make sure that the limits which the programmer has imposed upon the system are actually being acknowledged. An example of extreme testing in the context of this project would be inputting the number 100 to add the number to a simple stack.

Exceptional testing is when you provide erroneous data to your program and see how the code handles it. This is done to make sure that the code is robust and capable of carrying on if someone is attempting to cause the program to crash (for example, this is normally done by lecturers during demonstrations of programs). There are many possibilities when it comes down to erroneous input, such as inputting letters, random symbols or even just inputting nothing at all.

Simultaneously I tested the various stack and queue implementations to make sure they took in input correctly and that indices were changing correctly. The testing at this phase would be deemed as successful if the program was able to handle all of the various types of input which I chose to throw at it.



The picture above is showing a small part of the beginning of the final testing process. Test data will be provided in the appendix of this report.

What wasn't tested

There are a couple of things which have not been tested in this program. An example of this would be testing the program to make sure that it worked on other operating systems. The program was developed and proven to work on the Windows 7 Home Premium operating system. It is likely that the program should have no issues working on other Windows based operating systems, however it is not particularly wise to just make assumptions about this sort of thing. The programs status on other types of operating systems (such as MacOS or any of the multitude of Linux variations available) is currently unknown.

There are also probably a few million various combinations of characters and symbols that I have not used to test the robustness of the program. It is impossible to test for absolutely every possibility when it comes to handling input since there are so many possibilities there are for key inputs (potentially an infinite amount if you don't

impose a length limit upon the input).

Evaluation

There are many ways in which to evaluate the final product produced at the end of a project's implementation and testing phase. I chose to evaluate in the following ways:

- Test against the specification
- Test against use cases
- Test upon the students which the program was aimed at (2nd year computer science students)

I also intend to evaluate my own time management and take into consideration things which could've been handled differently throughout the process of building this project.

Evaluate against specification

This means of evaluation is simply to see if the final product does what it is meant to do. This process is also referred to as verification. For the sake of simplicity I shall be referring to the desired functionality which I settled on in the specification section of this report. These were the following:

- There should be the ability to push and pop values from a stack or a queue. This should be done by input provided by a user.
- There should be the ability to undo something which has been previously done and also the ability to clear whole stacks and queues if required by the user.
- There should be some form of visual representation showing what happens when the user executes a push or pop function upon the data set they are currently working with.

- There should be a console to display questions and feedback to the user. This feedback could be used as a basic level of visualisation to the user.
- There should also be a console providing code for each of the various forms of the implementations of each abstract data type.
- There should be a test function which displays questions inside a console for the user to answer. This would further add to the user interactivity of the project

Some of these goals were met in the final program, however some were not met. There is the ability to push and pop values from stacks and queues via the use of input from a user and there is a console to display feedback about what alterations have been made to the data types. There is also a working clear function for all of the various implementations as well. Due to time constraints I was unable to implement the other functions which I aimed to put into my program. I had to try and take into consideration which functions were the most important and the pieces that I didn't get around to implementing seemed comparatively superfluous. Out of the goals which I never got around to achieving I feel that I had the best chance of implementing the code displaying part of the program.

Overall I would say that my program only does about half of what it is supposed to do currently and there is still considerable room for improvement.

Evaluate against use cases

Another way to go about the evaluation of a project is to create a couple of use cases to evaluate your program against. This pretty much means just setting a few goals for your program to reach from an implementation standpoint. There are three different types of use cases which vary mostly by the amount of detail involved in each of them. These are:

- Brief use case
- Casual use case
- Fully dressed use case

The brief use case is simply a few short sentences describing the goal of the use case. The casual use case is simply a larger version consisting of a few paragraphs which summarise the goal of the use case. The fully dressed use case is a formal document which adheres to a template.

Since I created the use cases relatively early in the project's life cycle, they adhere closer to the definition of brief use cases. These use cases were the following:

Actor: student

Interaction: Student wants to see the difference between stack and queue ADTs by interacting with them in some way.

Goal: The student sees a difference between the stack and queue ADTs

Was it successful: Since the final product shows various implementations of each of the two ADTs, presents them in different ways (albeit in a much more minimal way than I would've liked) and allows the use to add and remove numbers from each of the implementations on their own (i.e. Interacting with it) I would say that this has been a successful use case. Numbers remain in each implementation until the student does something to remove them, so this is another point for the interaction part of the use case.

Actor: student

Interaction: student wants to see why a cyclical array queue is more efficient than a sliding array queue.

Goal: the student sees the differences in efficiency between the two implementations.

Was it successful: In a way it is successful since I was able to successfully implement both ways of using an array for a queue and it shows the indices of the head and tail of the queue. However since I was unable to implement a way of displaying the code for each implementation it is rather difficult to see at a glance whether or not either implementation is as efficient as the other. Overall I would say that this use case has only been partially successful.

Evaluation by students

Due to time constraints and other outside factors, I was unable to get around to this part of the evaluation process on time. My intention was to gather a small group of 2nd year computer science students who had taken part in CS207 advanced programming and ask them a few questions based upon the program which I asked them to evaluate. I was going to ask the questions such as the following:

- Did you feel that this program provided you a better understanding on how the stack and queue ADTs work?
- Do you see the difference in efficiency between a cyclical array queue and a sliding array queue thanks to this program?
- Is there any additional features which would've made this program more useful to you?

With some answers to these questions, I feel that I would've gotten a better perspective upon how well my finished program served its purpose (or if it even actually served its purpose at all).

Evaluation of time management

Around the time of the creation of the initial specification, I had to put together a schedule for each of the important milestones of the program. To begin with the schedule I decided to put together was the following (including compulsory deadlines):

- 22nd October: Project specification and plan
- 9th November: Finish research on implementations of structures and finalise specification, including desired functionality.
- 22nd November: Finish use cases and GUI design.
- 3rd December: Project poster
- 22nd December: Finish system design.
- 28th January: Progress report
- 8th February: Finish implementation.
- 22nd February: Finish testing.
- 3rd March: Finish evaluation.
- 13th March: Finish first draft of report, await comments and criticism.
- 27th March: Final draft, get bound and handed in.
- 30th March: Final Report Binding
- 1st April: Final Report

Up until the project poster day, I would say that I was working quite well next to this imposed schedule. However around the design stage, things started to get difficult and I started to fall behind. At the progress report stage of the project, I was given an opportunity to make some alterations to the schedule based upon my progress. These alterations were also based upon criticisms from the poster day. These alterations were the following:

- 28th February: Finish the implementation
- 7th March: Finish testing
- 11th March: Finish evaluation
- 21st March: Finish first draft of report
- 30th March (or preferably before): Finish final draft, bind and hand in report

Even with these time alterations I still struggled to finish an implementation in good time. This also slowed down the rest of the tasks in my schedule.

Overall I would say that my handling of time was initially rather good, however towards the end time management became worse and I fell a lot further behind than I would've liked to (not that I wanted to fall behind at all, but it seems to always be a danger in projects such as this).

Conclusion

This has been a report showing the many aspects of the development of a program for visualising the stack and queue data types. Overall to be honest I am not particularly pleased with how my project turned out in the end, however I am happy that the program is able to serve the desired purpose of showing how the data types handle input, albeit at a very basic level. If I was capable of implementing the other features in my rather ambitious design (and if I wasn't unwell for some of the implementation phase) this project would've been far more successful.

Appendix A: Bibliography/other references

- Head First Java 2nd edition – Bert Bates and Kathy Sierra
- Absolute Java 4th edition – Walter Savitch
- Data Structures and Algorithms in Java – Michael Goodrich and Roberto Tomassia
- Software Engineering 8th Edition – Ian Sommerville
- Writing Effective Use Cases – Alistair Cockburn
- The Java API
- CS207 slides
- queue image by Wikipedia/Vegpuff used under Creative Commons License
http://en.wikipedia.org/wiki/File:Data_Queue.svg

Appendix B: Test data

Testing Buttons(before selected implementation)

Action	Expected result	Success?
Click “add number”	Throw error	Yes
Click “remove number”	Throw error	Yes
Click “clear”	Throw error	Yes
Click “quit”	End program	Yes

Testing menu bar: file (before selected implementation)

Action	Expected result	Success?
Click “file”	Bring up menu	Yes
Click “add number”	Throw error	Yes
Click “remove number”	Throw error	Yes
Click “clear”	Throw error	Yes
Click “quit”	End program	Yes

Testing menu bar: implementations

Action	Expected result	Success?
Click “Implementations”	Bring up menu	Yes
Click “Stack”	Bring up sub menu	Yes
Click “queue”	Bring up sub menu	Yes
Click “simple stack”	Sign that simple stack is selected and show empty stack	Yes
Click “Array stack”	Sign that array stack is selected and show empty stack	Yes
Click “Linked List stack”	Sign that linked list stack is selected and show empty stack	Yes
Click “Simple queue”	Sign that simple queue is selected and show empty queue	Yes
Click “Inefficient Array queue”	Sign that inefficient array queue is selected and show empty queue	Yes
Click “Efficient Array queue”	Sign that efficient array queue is selected and show empty queue	Yes
Click “Linked List Queue”	Sign that linked list queue is selected and show empty queue	Yes

Testing menu bar: help

Action	Expected result	Success?
Click "Help"	Bring up menu	Yes
Click "How to use"	Pop up box with help	Yes
Click "About"	Pop up box with info	Yes

Testing user input(simple stack)

Action	Expected result	Success?
Add "13"	13 added to stack	Yes
Add "101"	Invalid number error	Yes
Add "Derp"	Invalid input error	Yes
Add "<blank>"	No input error	Yes
Add "27"	27 added to stack	Yes
Add "55"	55 added to stack	Yes
Add "77"	77 added to stack	Yes
Add "9"	9 added to stack	Yes
Add "12"	Full stack error	Yes
Press "remove"	9 removed from stack	Yes
Press "remove"	77 removed from stack	Yes
Press "clear"	Stack cleared	Yes
Press "remove"	Nothing there error	Yes
Press "clear"	Nothing there error	Yes

Test user input(array stack)

Action	Expected result	Success?
Add "1"	1 added to stack index 0	Yes
Add "2"	2 added to stack index 1	Yes
Add "3"	3 added to stack index 2	Yes
Press "remove"	3 removed from stack	Yes
Add "4"	4 added to stack index 2	Yes
Add "5"	5 added to stack index 3	Yes
Add "6"	6 added to stack index 4	Yes
Add "7"	Full stack error	Yes
Press "remove"	6 removed from stack	Yes
Press "remove"	5 removed from stack	Yes
Press "clear"	Stack cleared	Yes
Press "remove"	Nothing there error	Yes
Add "8"	8 added to stack index 0	Yes

Test user input(inefficient array queue)

Action	Expected result	Success?
Add "1"	1 added index 0, tail = 1	Yes
Add "2"	2 added index 1, tail = 2	Yes
Add "3"	3 added index 2, tail = 3	Yes
Click "remove"	1 removed, tail = 2	Yes
Add "4"	4 added index 2, tail = 3	Yes
Add "5"	5 added index 3, tail = 4	Yes
Add "6"	6 added index 4, tail = 5	Yes
Click "remove"	2 removed, tail = 4	Yes
Click "remove"	3 removed, tail = 3	Yes
Click "clear"	Queue cleared	Yes
Add "clear"	Nothing there error	Yes
Add "7"	7 added index 0, tail = 1	Yes

Test user input(efficient array queue)

Action	Expected result	Success?
Add "1"	1 added index 0, tail = 1	Yes
Add "2"	2 added index 1, tail = 2	Yes
Add "3"	3 added index 2, tail = 3	Yes
Click "remove"	1 removed, head = 1, tail = 3	Yes
Click "remove"	2 removed, head = 2, tail = 3	Yes
Click "remove"	3 removed, head = 3, tail = 3	Yes
Click "remove"	Nothing there error	Yes
Add "1"	1 added, head = 3, tail = 4	Yes
Add "2"	2 added, head = 3, tail = 0	Yes
Add "3"	3 added, head = 3, tail = 1	Yes
Add "4"	4 added, head = 3, tail = 2	Yes
Add "5"	5 added, head = 3, tail = 3	Yes
Add "6"	Queue full error	Yes
Click "clear"	Queue cleared	Yes
Add "7"	7 added, head = 0, tail = 1	Yes

Appendix C: User Guide

How to get started

To get the program to work, simply double click the file called *visualisations.jar* and the program should then start to run.

How to use

- The first thing you have to do is select an implementation. This is done by going to the *Implementations* option in the menu at the top of the program's screen. You can switch between implementations at any time and still maintain the values which you have input.
- You can add up to five numbers to whatever implementation you have chosen. This can be done by either clicking on the button called *add number* or by going to the *file* and clicking on the option of the same name. From here you can input a number between 1 and 100. Any other form of input is invalid.
- To remove a number, simply click on the button called *remove number* or go to the *file* option and click on the option of the same name within that menu.
- To clear the values from the implementation you are working with, click on the *clear* button or go to the *file* option and click on the option of the same name within that menu. The values in each implementation will remain as you change implementations until you clear the values yourself.
- There is also a *quit* option available by pressing the button of the same name. All work will disappear when you close the program. Not like you'd need to save anything, but I thought I'd say so anyway.