Sean Poston

CS300

Project 2 Report

https://repl.it/@seanposton4/CS300Project2

https://youtu.be/RYK6Z7IJGHQ

Abstract

This project had the intent to further the students' knowledge about data structures and how to better interact with them. While the data structures picked (LinkedList, Stack, Queue) were simpler than some other data structures (AVL Tree, HashMap), the project still proved tedious in execution. Each of the three chosen data structures are required to have four functions to accompany them: add, remove, modify, and search. Inserting data into the structures was by far the easiest task in this project. The project ended up being very education and even enjoyable and rewarding at times.

Introduction

This project was meant to exemplify knowledge of basic data structures, and one's ability to program and showcase them. The project had new challenges, as it's almost 500 lines of code. Much of it is repeated, but much of the repetition is necessary because of the iterative way it's written.

There were many realizations of foolishness mid-way through the project that would make it easier to do a second time. Making the code more object oriented with 3 wrapper classes for each of the data structures would have saved much space and possibly even time. It could even be taken a step further to create a parent class (abstract or concrete) that is extended to the data structure wrapper classes.

Another reason for the length of the code is input control. All user input (except for memory related issues, mostly overflow) is safe to put in. There are if statements and loops that require a certain range of inputs for the program to move forward. Everything is variable, so if memory allows it, the program can handle the input.

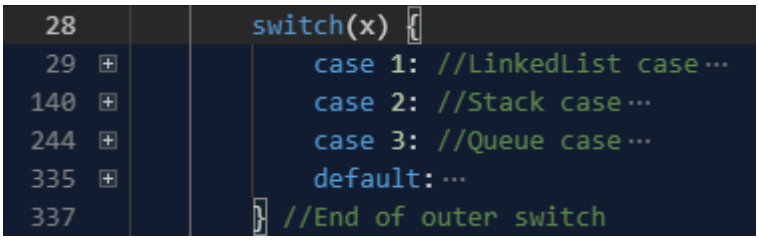
The project was not clear (or I misunderstood) about the way to handle the modifying and removal of data from the data structures. The program has extra variability (which took extra work in the end) which allows the user to do things like remove data from the middle of a stack or queue, for example. Whereas the base data structure wouldn't allow it, the extra work on this project will allow that (as well as base operation of the data structure with a proper index input).

As expressed, the program is inefficient and repetitive. However, on a program of this size and complexity there is a lot of room for inefficiency because it is only simple calculations being made. At worst, the program is running O(n) complexity in a few functions as the data structures are cycled through being searched or modified.

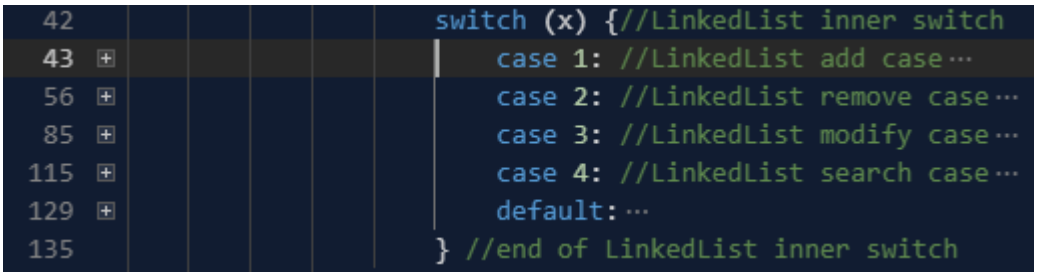
Outline

These switches are the main body of the program. They control the flow of everything within the program. Each “main switch” is a data structure that has 4 functions within: add, remove, modify, and search. There is a total of 4 switches: the main switch, the LinkedList switch, the Stack switch, and the Queue switch, and a total of 12 operable functions between the 3 main structures.

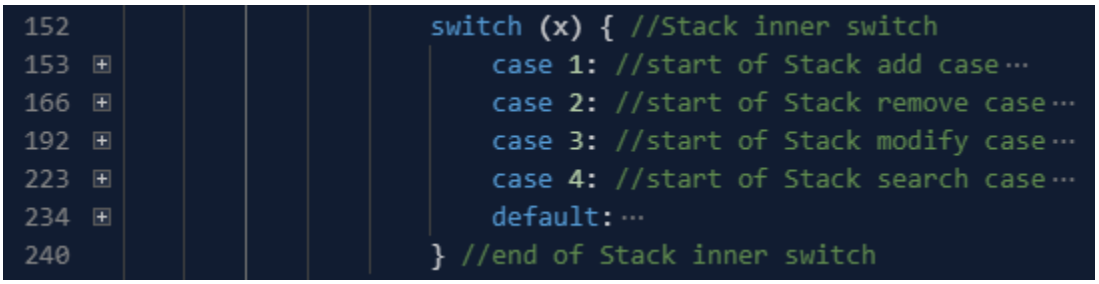
Main Switch



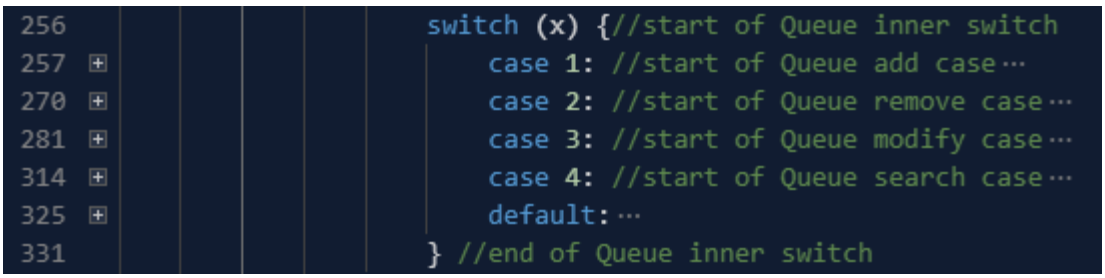
LinkedList Inner Switch



Stack Inner Switch



Queue Inner Switch



User Interface

Perhaps the best part of this program is the user interface. The program makes use of a function to clear the console to keep a clean, sleek look [1].

Another feature of it is the input blocking. When a user first starts the program, there is not any information in the data structure, so the only path available to the user is to input data into it. Whenever the data structure is not empty, the program will allow the user to use the remove, modify, and search functions. Here are examples from the LinkedList structure:

```
53                  case 2: //LinkedList remove case
54                      if (list.size() != 0) {
```

```
82                  case 3: //LinkedList modify case
83                      if (list.size() != 0) {
```

```
112                 case 4: //LinkedList search case
113                     if (list.size() != 0) {
```

Here's what is looks like for the user:

```
--------------------
0 to exit
1 to add an integer
--------------------
```

These will be the only options while the structure is empty. After there is data input, it will look like this, also including an always updated look at the data structure:

```
Current LinkedList:
  0 ->
Head: 0    Tail: 0    Size: 1

--------------------
0 to exit
1 to add an integer
2 to remove an integer
3 to modify an integer
4 to search for an integer
--------------------
```

As for the remove and modify functions, they require an index input from the user. This index is very easy to mess up and cause an out of bounds error. As such, the program doesn't allow any inputs outside the range of the data structure's size. Here is another example from the LinkedList structure:

```
58                  index = 0;
59                  indexFlag = 0;
60                  do {
61                      if (indexFlag == 0) {
62                          System.out.print("Enter the index to remove: ");
63                          index = input.nextInt();
64                      }
65                      else {
66                          clear();
67                          printLinkedList(list);
68                          System.out.print("\n***Out of Bounds Exception. Use a smaller number.***\n");
69                          System.out.print("\nEnter the index to modify: ");
70                          index = input.nextInt();
71                      }
72
73                      indexFlag = 1;
74                  } while (index >= list.size());
```

The search function not only tells the user that their desired input is in the structure, but it also shows the user at exactly what index to find it.

```
350     public static void searchLinkedList(LinkedList<Integer> list, int element) {
351         LinkedList<Integer> indexArray = new LinkedList<>();
352         if (list.contains(element)) {
353             for (int i = 0; i < list.size(); i++) {
354                 if (list.get(i).equals(element)) {
355                     indexArray.add(i);
356                 }
357             }
358             System.out.printf("\nInteger %d is at index(es): ", element);
359             for (int i = 0; i < indexArray.size(); i++) {
360                 System.out.printf("[%d] ", indexArray.get(i));
361             }
362         }
363         else { System.out.println("The integer is not in the list.\n"); }
364     }
```

If the user inputs 3 for the search function, this is what they will see:

```
Current LinkedList:
  0 ->  1 ->  2 ->  3 ->  4 ->  5 ->  3 ->
Head: 0    Tail: 3    Size: 7

Integer 3 is at index(es): [3] [6]
--------------------
0 to exit
1 to add an integer
2 to remove an integer
3 to modify an integer
4 to search for an integer
--------------------
```

As a video presentation is also due, most of the user interface will be left to be shown there.

Conclusion

This project was quite time consuming, totaling near 8 hours. Doing this project was excellent for solidifying knowledge of LinkedList, Stack, and Queue. These data structures are very important in the programming world, and they should be treated as such in the classroom. They have many applications that go beyond their basic uses as data structures. A lot of wrapper classes will use them as a base and allow for even more complex operations. For this reason, this project is one of the most invaluable that we have done during this semester.

Source Code: https://repl.it/@seanposton4/CS300Project2

```
/*
Author: Sean Poston
Purpose: To create a program that will help visualize different data structures.
Date: 4/4/2020
*/

import java.util.LinkedList;
import java.util.Stack;
import java.util.Scanner;
import java.util.ArrayList;
import java.util.Collections;
import java.util.ArrayDeque;

class Main {
    public static void main(String[] args) {
        clear();
        Scanner input = new Scanner(System.in);

        System.out.println("--------------------\n1 for Linked List\n2 for
Stack\n3 for Queue\n--------------------");
        System.out.print("Enter: ");
        int x = input.nextInt(); //get choice for switch
        int element; int index = 0; //to hold elements before adding them to the
objects
        int indexFlag = 0;

        switch(x) {
            case 1: //LinkedList case

                LinkedList<Integer> list = new LinkedList<>();
                do {
                    if (list.size() == 0) {
                        clear();
                        System.out.println("\n--------------------\n0 to exit\n1
to add an integer\n--------------------");
                    }
                    else
                        System.out.println("\n--------------------\n0 to exit\n1
to add an integer\n2 to remove an integer\n"
                            + "3 to modify an integer\n4 to search for an
integer\n--------------------");

                    x = input.nextInt();
```

```
                    switch (x) {//LinkedList inner switch
                        case 1: //LinkedList add case
                            clear();
                            if (list.size() != 0)
                                printLinkedList(list);

                            System.out.print("\nEnter the integer to add: ");
                            element = input.nextInt();

                            clear();
                            list.add(element);
                            printLinkedList(list);
                            break;//end of LinkedList add case

                        case 2: //LinkedList remove case
                            if (list.size() != 0) {
                                clear();
                                printLinkedList(list);

                                index = 0;
                                indexFlag = 0;
                                do {
                                    if (indexFlag == 0) {
                                        System.out.print("Enter the index to
remove: ");

                                        index = input.nextInt();
                                    }
                                    else {
                                        clear();
                                        printLinkedList(list);
                                        System.out.print("\n***Out of Bounds
Exception. Use a smaller number.***\n");
                                        System.out.print("\nEnter the index to
modify: ");

                                        index = input.nextInt();
                                    }

                                    indexFlag = 1;
                                } while (index >= list.size());

                                clear();
                                list.remove(index);
                                printLinkedList(list);
                            }
                            break;//end of LinkedList remove case
```

```
                        case 3: //LinkedList modify case
                            if (list.size() != 0) {
                                index = 0;
                                indexFlag = 0;
                                do {
                                    if (indexFlag == 0) {
                                        System.out.print("Enter the index to
modify: ");
                                        index = input.nextInt();
                                    }
                                    else {
                                        clear();
                                        printLinkedList(list);
                                        System.out.print("\n***Out of Bounds
Exception. Use a smaller number.***\n");
                                        System.out.print("\nEnter the index to
modify: ");
                                        index = input.nextInt();
                                    }

                                    indexFlag = 1;
                                } while (index >= list.size());

                                System.out.printf("Enter the integer to place
into index [%d]: ", index);
                                element = input.nextInt();

                                clear();
                                list.set(index, element);
                                printLinkedList(list);

                            }
                            break;//end of LinkedList modify case

                        case 4: //LinkedList search case
                            if (list.size() != 0) {
                                clear();
                                printLinkedList(list);

                                System.out.print("\nEnter the integer to search
for: ");
                                element = input.nextInt();

                                clear();
```

```
                            printLinkedList(list);
                            searchLinkedList(list, element);
                        }
                        break;//end of LinkedList search case

                    default:
                        clear();
                        printLinkedList(list);
                        if (x != 0)
                            System.out.println("\nInvalid input.");

                } //end of LinkedList inner switch
            } while (x != 0);//end of while loop

            break; //end LinkedList case

        case 2: //Stack case
            Stack<Integer> stack = new Stack<>();
            do {
                if (stack.size() == 0) {
                    clear();
                    System.out.println("\n--------------------\n0 to exit\n1
to add an integer\n--------------------");
                }
                else
                    System.out.println("\n--------------------\n0 to exit\n1
to add an integer\n2 to remove an integer\n"
                        + "3 to modify an integer\n4 to search for an
integer\n--------------------");
                x = input.nextInt();

                switch (x) { //Stack inner switch
                    case 1: //start of Stack add case
                        clear();
                        if (stack.size() != 0)
                            printStack(stack);
                        System.out.print("Enter the integer to add: ");
                        element = input.nextInt();

                        clear();
                        stack.push(element);
                        printStack(stack);

                        break;//end of Stack add case
```

```
                        case 2: //start of Stack remove case
                            if (stack.size() != 0) {
                                index = 0;
                                indexFlag = 0;
                                do {
                                    if (indexFlag == 0) {
                                        System.out.print("Enter the index to
remove: ");
                                        index = input.nextInt();
                                    }
                                    else {
                                        clear();
                                        printStack(stack);
                                        System.out.print("\n***Out of Bounds
Exception. Use a smaller number.***\n");
                                        System.out.print("\nEnter the index to
modify: ");
                                        index = input.nextInt();
                                    }

                                    indexFlag = 1;
                                } while (index >= stack.size());

                                clear();
                                removeStackElement(stack, index);
                                printStack(stack);
                            }
                            break;//end of Stack remove case

                        case 3: //start of Stack modify case
                            if (stack.size() != 0) {
                                index = 0;
                                indexFlag = 0;
                                do {
                                    if (indexFlag == 0) {
                                        System.out.print("Enter the index to
modify: ");
                                        index = input.nextInt();
                                    }
                                    else {
                                        clear();
                                        printStack(stack);
                                        System.out.print("\n***Out of Bounds
Exception. Use a smaller number.***\n");
```

```
                                        System.out.print("\nEnter the index to
modify: ");
                                        index = input.nextInt();
                                    }

                                    indexFlag = 1;
                                } while (index >= stack.size());

                                System.out.printf("Enter the integer to place
into index [%d]: ", index);
                                element = input.nextInt();



                                clear();
                                modifyStackElement(stack, index, element);
                                printStack(stack);
                            }
                            break;//end of Stack modify case

                        case 4: //start of Stack search case
                            if (stack.size() != 0) {
                                System.out.print("Enter the integer to search
for: ");
                                element = input.nextInt();

                                clear();
                                printStack(stack);
                                searchStack(stack, element);
                            }
                            break;//end of Stack search case

                        default:
                            clear();
                            printStack(stack);
                            if (x != 0)
                                System.out.println("\nInvalid input.");

                    } //end of Stack inner switch
                } while (x != 0);
                break; //end Stack case

            case 3: //Queue case
                ArrayDeque<Integer> queue = new ArrayDeque<>();
                do {
```

```
                    if (queue.size() == 0) {
                        clear();
                        System.out.println("\n--------------------\n0 to exit\n1
to add an integer\n--------------------");
                    }
                    else
                        System.out.println("\n--------------------\n0 to exit\n1
to add an integer\n2 to remove an integer\n"
                            + "3 to modify an integer\n4 to search for an
integer\n--------------------");
                    x = input.nextInt();

                    switch (x) {//start of Queue inner switch
                        case 1: //start of Queue add case
                            clear();
                            if (queue.size() != 0)
                                printQueue(queue);
                            System.out.print("Enter the integer to add: ");
                            element = input.nextInt();

                            clear();
                            queue.add(element);
                            printQueue(queue);

                            break;//end of Queue add case

                        case 2: //start of Queue remove case
                            if (queue.size() != 0) {
                                System.out.print("Enter the integer to remove:
");
                                element = input.nextInt();

                                clear();
                                queue.remove(element);
                                printQueue(queue);
                            }
                            break;//end of Queue remove case

                        case 3: //start of Queue modify case
                            if (queue.size() != 0) {
                                index = 0;
                                indexFlag = 0;
                                do {
                                    if (indexFlag == 0) {
```

```
                                    System.out.print("Enter the index to
modify: ");
                                    index = input.nextInt();
                                }
                                else {
                                    clear();
                                    printQueue(queue);
                                    System.out.print("\n***Out of Bounds
Exception. Use a smaller number.***\n");
                                    System.out.print("\nEnter the index to
modify: ");
                                    index = input.nextInt();
                                }

                                indexFlag = 1;
                            } while (index >= queue.size());

                            System.out.printf("Enter the integer to place
into index [%d]: ", index);
                            element = input.nextInt();

                            clear();
                            modifyQueue(queue, index, element);
                            printQueue(queue);


                            //System.out.println("\n***Out of Bounds
Exception. Use a smaller number.***\n");

                        }
                        break;//end of Queue modify case

                    case 4: //start of Queue search case
                        if (queue.size() != 0) {
                            System.out.print("Enter the integer to search
for: ");
                            element = input.nextInt();

                            clear();
                            printQueue(queue);
                            searchQueue(queue, element);
                        }
                        break;//end of Queue search case

                    default:
```

```
                            clear();
                            printQueue(queue);
                            if (x != 0)
                                System.out.println("\nInvalid input.");

                        } //end of Queue inner switch
                    } while (x != 0);
                    break; //end Queue case

                default:
                    System.out.println("\nInvalid input.");
            } //End of outer switch
            System.out.println("Bye");
        }

//start of methods

        public static void printLinkedList(LinkedList<Integer> list) {
            System.out.println("\nCurrent LinkedList: ");
            for (int i = 0; i < list.size(); i++) {
                System.out.printf("%3d", list.get(i));
                System.out.print(" ->");
            }
            if (list.size() != 0)
                System.out.printf("\nHead: %-4d Tail: %-4d Size: %-4d\n",
list.getFirst(), list.getLast(), list.size());
        }

        public static void searchLinkedList(LinkedList<Integer> list, int element) {
            LinkedList<Integer> indexArray = new LinkedList<>();
            if (list.contains(element)) {
                for (int i = 0; i < list.size(); i++) {
                    if (list.get(i).equals(element)) {
                        indexArray.add(i);
                    }
                }
                System.out.printf("\nInteger %d is at index(es): ", element);
                for (int i = 0; i < indexArray.size(); i++) {
                    System.out.printf("[%d] ", indexArray.get(i));
                }
            }
            else { System.out.println("The integer is not in the list.\n"); }
        }

        public static void printStack(Stack<Integer> stack) {
```

```
        Stack<Integer> tempStack = (Stack<Integer>)stack.clone();

        System.out.println("\nCurrent Stack: ");
        while (!tempStack.empty()) {
            System.out.printf("%3d\n", tempStack.pop());
        }
        if (stack.size() != 0)
            System.out.printf("First: %-4d Last: %-4d Size: %-4d\n",
stack.firstElement(), stack.lastElement(), stack.size());
    }

    public static void removeStackElement(Stack<Integer> stack, int index) {
        Stack<Integer> temp = new Stack<>();

        int targetIndex = stack.size() - index;
        for (int i = 0; i < targetIndex - 1; i++) {
            temp.push(stack.pop());
        }

        stack.pop();

        while (!temp.empty())
            stack.push(temp.pop());
    }

    public static void modifyStackElement(Stack<Integer> stack, int index, int
element) {
        Stack<Integer> temp = new Stack<>();

        int targetIndex = stack.size() - index;
        for (int i = 0; i < targetIndex - 1; i++) {
            temp.push(stack.pop());
        }

        stack.pop();
        stack.push(element);

        while (!temp.empty())
            stack.push(temp.pop());
    }

    public static void searchStack(Stack<Integer> stack, int element) {
        if (stack.contains(element)) {
            ArrayList<Integer> indexList = new ArrayList<>();
            Stack<Integer> temp = (Stack<Integer>)stack.clone();
```

```
            int i = 0;
            while(!temp.empty()) {
                if (temp.pop() == element) indexList.add(stack.size() - i - 1);
                i++;
            }

            Collections.sort(indexList);

            System.out.printf("\nInteger %d is at index(es): ", element);
            for (i = 0; i < indexList.size(); i++) {
                System.out.printf("[%d] ", indexList.get(i));
            }
        }
        else { System.out.println("The integer is not in the stack.\n"); }
    }

    public static void printQueue(ArrayDeque<Integer> queue) {
        ArrayDeque<Integer> temp = (ArrayDeque<Integer>)queue.clone();

        System.out.println("\nCurrent Queue: ");
        for (int i = 0; i < queue.size(); i++) {
            System.out.printf("%3d", temp.poll());
            System.out.print(" <-");
        }
        if (queue.size() != 0)
            System.out.printf("\nHead: %-4d Tail: %-4d Size: %-4d\n",
queue.peekFirst(), queue.peekLast(), queue.size());
    }

    public static void modifyQueue(ArrayDeque<Integer> queue, int index, int
element) {
        ArrayDeque<Integer> temp = new ArrayDeque<>();

        for (int i = 0; i < index; i++) {
            temp.add(queue.poll());
        }

        queue.pop();
        queue.offerFirst(element);

        while(!temp.isEmpty()) {
            queue.offerFirst(temp.removeLast());
        }
    }
```

```
    public static void searchQueue(ArrayDeque<Integer> queue, int element) {
        if (queue.contains(element)) {
            ArrayDeque<Integer> temp = (ArrayDeque<Integer>)queue.clone();
            ArrayList<Integer> indexList = new ArrayList<>();
            int i = 0;

            while (!temp.isEmpty()) {
                if (temp.poll() == element) {
                    indexList.add(i);
                }
                i++;
            }

            System.out.printf("\nInteger %d is at index(es): ", element);
            for (i = 0; i < indexList.size(); i++) {
                System.out.printf("[%d] ", indexList.get(i));
            }
        }
        else { System.out.println("The integer is not in the stack.\n"); }
    }

    public static void clear() {
        System.out.print("\033[H\033[2J");
        System.out.flush();
    }
}
```

Sources

[1] Java Clear Console Function. https://stackoverflow.com/questions/2979383/java-clear-the-console

[2] ArrayDeque Class. https://docs.oracle.com/javase/7/docs/api/java/util/ArrayDeque.html

[3] ArrayDeque Clone Method in Java. https://www.geeksforgeeks.org/arraydeque-clone-method-in-java/

[4] Object Deep Copy. https://stackoverflow.com/questions/64036/how-do-you-make-a-deep-copy-of-an-object

Team Member

Sean Poston