Assignment One

Veronica Longley

September 24, 2021

Our Goal: In this assignment we will utilize Stacks and Queues to locate and output palindromes from a list of magic items.

1 Node Class

```
1  public class NodeLongley
2  {
3          //myData holds the char for each Node
4          //myNext references the next Node in the stack or queue
5          private char myData;
6          private NodeLongley myNext;
7
8          public NodeLongley()
9          {
10                 myData = '␣';
11                 myNext = null;
12         }//NodeLongley
13
14         public NodeLongley(char newData)
15         {
16                 myData = newData;
17                 myNext = null;
18         }//NodeLongley
19
20         //getter for myData
21         public char getData()
22         {
23                 return myData;
24         }//getData
25
26         //getter for myNext
27         public NodeLongley getNext()
28         {
29                 return myNext;
30         }//getNext
31
32         //setter for myData
33         public void setData(char newData)
34         {
35                 myData = newData;
36         }//setData
37
38         //setter for myNext
39         public void setNext(NodeLongley newNext)
40         {
41                 myNext = newNext;
42         }//setNext
43 }//NodeLongley
```

To begin, the Node Class, which is used by Stack and Queue, has two private variables: myData (Line 5) and myNext (Line 6). MyData includes the character while myNext is a reference to the next character in the stack or queue. There is also getters and setters for both of these variables (Lines 20-42) in this class so we are able to access and assign these values.

2 Queue Class

```
1  public class Queue {
2          //initialize Node for head and tail of queue
3          private NodeLongley head;
4          private NodeLongley tail;
5
6          //checks if queue is empty and returns boolean
7          public boolean isEmpty()
8          {
9                  return(head == null);
10         }//isEmpty
11         // enqueue method adds char c to the end of the line
12         public void enqueue (char c)
13         {
14                 NodeLongley oldtail = tail;
15                 tail = new NodeLongley();
16                 tail.setData(c);
17                 tail.setNext(null);
18                 if (isEmpty())
19                         {head = tail;
20                         }//if
21
22                 else
23                         oldtail.setNext(tail);
24         }//enqueue
25
26         // dequeue method removes the char at the front of the line
27         // to be used for comparison
28         public char dequeue()
29         {
30                 char returnval = 'a';
31                 if(!isEmpty())
32                 {
33                         returnval = head.getData();
34                         head = head.getNext();
35                         if (isEmpty())
36                                 tail = null;
37                 }//if
38                 else
39                         System.out.println("Handle Underflow");
40
41                 return returnval;
42         }//dequeue
43 }//queue
```

The Queue Class first initializes a Node to be the head/'front of the line' (Line 3) and a Node to be the tail/'end of the line' (Line 4). We have the method isEmpty to check if the queue is empty and return the appropriate boolean. If the head of the list is assigned to null, the queue is empty and the method returns true, otherwise it is false (Line 9). The enqueue method is how we add characters to the end of the line. We first store what is currently at the end of the queue (Line 14), then create a new node tail (Line 15), set tail to be the character we are adding (Line 16), set its next element to be null as it is the end of the queue (Line 17), then if it is the only element in the queue we make the head equal the tail so isEmpty knows the queue is no longer empty (Line 19) otherwise, we set the

next element of the Node that was the tail before this addition to be the new tail we just added (Line 23). Dequeue is how we remove characters from the queue. We will always be removing the character that was enqueued first. We first check if the queue is empty (Line 31), then store the character we want to return (the head) in returnval (Line 33), and we set head to be the next character in the queue (Line 34). If this was the last element in the queue, we set tail to be null (Line 36). Lastly we must return the value we stored in returnval.

3 Stack Class

```
1  public class Stack {
2          //initialize Node for top of stack
3          private NodeLongley top;
4
5          public Stack()
6          {
7                  top = null;
8          }//Stack
9          //push method which will add char c to the top of the stack
10         public void push (char c)
11         {
12                 NodeLongley oldTop = top;
13                 top = new NodeLongley();
14                 top.setData(c);
15                 top.setNext(oldTop);
16
17         }//push
18
19         //checks to see if the stack is empty and returns true if it is
20         // returns false if it is not
21         public boolean isEmpty()
22         {
23                 if (top == null)
24                         return true;
25                 else
26                         return false;
27         }// isEmpty
28
29         //pop method to remove the pop char from the stack to be used
30         //for comparison
31         public char pop()
32         {
33                 char returnval = 'a';
34                 if (!isEmpty())
35                 {
36                         returnval = top.getData();
37                         top = top.getNext();
38                 }//if
39                 else
40                         System.out.println("Handle Underflow");
41                 return returnval;
42         }//pop
43 }//Stack
```

The Stack class begins by initializing a Node called top (Line 3). This will always be the last character pushed onto the stack. The push method is how we can add characters to the top of the stack. We do so by storing the element on the top of the stack (Line 12), creating a new node (Line 13), storing the character we want to push in this new node (Line 14), then setting the 'next' element to be what was on the top of the stack before this addition (Line 15). The isEmpty method (Lines 21-27) will check if the stack is empty by checking if the top of the stack is null. If that is true it will return true, and if that is false it will return false. Lastly, the pop method allows us to use the values, or in this case characters, by taking them off the stack. We first check if the stack is empty and if it is not, we store the top of the list in returnval (Line 36) and assign top to be the next element in the stack (Line 37). Then we return the value we stored in returnval.

4 Main Class

```
1  import java.io.File;
2  import java.io.FileNotFoundException;
3  import java.util.InputMismatchException;
4  import java.util.Scanner;
5
6  public class main
7  {
8
9  static Scanner keyboard = new Scanner(System.in);
10
11         public static void main(String[] args)
12         {
13                 // TODO Auto-generated method stub
14                 // TODO Auto-generated method stub
15
16                 //initialize variables used to read in items from list
17                 String fileName = "";
18                 int numofItems = 666;
19                 String [] magicItems = new String [666];
20                 String theitem = null;
21                 fileName = "magicitems";
22
23                 //input item names from file and store in array
24                 try
25                         {
26                         Scanner readFile = new Scanner (new File (fileName));
27                         int i = 0;
28                         while(readFile.hasNextLine())
29                         {
30                                 theitem = readFile.nextLine();
31                                 magicItems[i] = theitem;
32                                 i++;
33                         }//while
34                         readFile.close();
35                         }//try
36                 catch(FileNotFoundException ex)
37                         {
38                         System.out.println("Failed to find file: "+ fileName);
39                         }//catch
40                 catch(InputMismatchException ex)
```

```
                {
                System.out.println("Type mismatch. ");
                System.out.println(ex.getMessage());
                }//catch
        catch(NumberFormatException ex)
                {
                System.out.println("Failed to convert String into an int. ");
                System.out.println(ex.getMessage());
                }//catch
        catch(NullPointerException ex)
                {
                System.out.println("Null pointer exception. ");
                System.out.println(ex.getMessage());
                }//catch
        catch(Exception ex)
                {
                System.out.println("Oops, something went wrong. ");
                ex.printStackTrace();
                }//catch

        //for loop to go through the list of magic items
        for (int i = 0; i < numofItems; i++)
        {
                //create stack and queue used to compare the string
                //forwards and backwards
                Stack stackversion = new Stack();
                Queue queueversion = new Queue();
                String word = "";

                //eliminate spaces and lower case letters
                // store in 'word' so we can print the possible
                //palindrome correctly
                word = magicItems[i].toUpperCase();
                word = word.replaceAll(" ", "");

                //for loop to go through the chars of the magic item
                //pushes and enqueues the chars one by one
                for(int j = 0; j <= word.length()-1; j++)
                {
                                char c = word.charAt(j);
                                stackversion.push(c);
                                queueversion.enqueue(c);
                }//for

                //initialize variables used for comparison
                boolean palindrome = true;
                int k = 0;

                //while loop to check for palindromes by popping
                //and dequeuing
                //the same string and comparing those chars
                while ( palindrome == true && k<= word.length()-1 )
                {
                        char s = 'a';
                        char q = 'b';
                        s = stackversion.pop();
                        q = queueversion.dequeue();
                        if(Character.compare(s, q) == 0 )
```

```
99                                      {
100                                             k++;
101                                     }//if
102                                     else
103                                     {
104                                             palindrome = false;
105                                     }//else
106
107                             }//while
108
109                             //Print magic item if palindrome is true
110                             if(palindrome == true)
111                                     System.out.println(magicItems[i]);
112                     }//for
113
114             }//main
115
116     }//main
```

In main we will be using the Stack Class and the Queue Class to find palindromes, words spelled the same forwards and backwards, in a list of items and print them out. Pushing the string character by character onto a stack will reverse the word's order, so when it is compared to the queue we will be able to tell of the string is a palindrome as a queue will preserve the words order. First we take input from the user for the name of the file (Line 24). We then use a while loop (Line 31) to go through the list and store each item in our array magicItems (Line 34). We are using try and catch to read the file and store the items so that if an exception is thrown, we can try and avoid crashing the program. To find the palindromes we must go through each item in the list, push it into a stack, enqueue it into a queue, pop and dequeue, and compare characters. To go through the list item by item, we have a for loop (Line 65). To begin the loop we create a new stack (Line 69) and a new queue (Line 70). We then make the entire string uppercase (Line 76) and eliminate spaces (Line 77) to make comparisons easier. Here we are also storing the string without spaces and in all caps in our variable 'word'. This is so that if the string is a palindrome we can properly print it without missing spaces or having it be all capital letters. We then have another for loop that goes character by character through the string and pushes (Line 84) and enqueues (Line 85) it. Next we use a while loop to pop from the stack (Line 99), dequeue from the queue (Line 100), and compare the characters (Line 101). If the charaters are the same, we incriment k (Line 103), the counter that moves through the string. If they are different we set the boolean palindrome to be false (Line 107). This, along with k getting to the end of the string, will exit the while loop. If k gets to the end of the string and palindrome is never set to false, the string is in fact a palindrome and it is printed out (Line 114). We then go back to the first for loop and repreat this process for each item in the array from the file.