CMPT 435 - Fall 2021 - Dr. Labouseur

Assignment One

Nicholas Petrilli

September 24, 2021

1 Node

Listing 1: Node Class

```java
//Node class for singly linked list

public class Node {
    char item;
    Node next;

    // Empty constructor to create a new Node without specifying the attribut
    public Node() {
        item = 0;
        next = null;
    }
    //Constructor to create a node with char item and a pointer to the next n
    public Node(char item, Node next) {
        this.item = item;
        this.next = next;
    }
    //Setter function to set the  of node
    public void setItem(char item) {
        this.item = item;
    }
    //Setter function to set the pointer of next node
    public void setNext(Node next) {
        this.next = next;
    }
    //Getter function to return the name of the node
    public char getItem() {
        return this.item;
    }
    //Getter function to return the next node
    public Node getNext() {
        return this.next;
    }
}
```

The Node class creates a singly linked list which is used in the implementation of the Stack and Queue classes. Nodes have two attributes, the item it is storing which is a character, along with a Node pointer to the next object in the stack or queue.

2 Stack

Listing 2: Stack Class

```
//Stack class

public class Stack {
    Node top;

    //Empty constructor to create a stack
    public Stack() {
        top = null;
    }
    //Constructor specifying the top of the stack
    public Stack(Node top) {
        this.top = top;
    }
    //Returns true if stack is empty, returns false otherwise
    public Boolean isEmpty() {
        if (top == null)
            return true;
        return false;
    }
    //Push function to insert an element into the stack
    public void push (char c) {
        Node oldTop = top;
        top = new Node();
        //Create new node for new top, populates with recieved char and point
        top.item = c;
        top.next = oldTop;
    }
    //Pop function to remove an element from the top of the stack
    public char pop() {
        char returnVal = '0';
        if (!isEmpty()) {
            //Removes top element, and makes new top the next element in the
           returnVal = top.item;
34              top = top.next;
35          }
36          else {
37              System.out.println("Stack Underflow");
38          }
39          return returnVal;
40      }
41
42
43
44
45  }
```

The Stack class uses the push and pop methods in order to manipulate data within the stack. Starting at line 21, the push method takes in a character that is going to be inserted into the top of the stack. Lines 22 and 23 move the element that was previously at the top of the stack, and create a new node for the new top. Lines 25-26 then insert the char value that was sent in to the function to the new top, along with the pointer node pointing to the previous top. The pop function removes an element from the top of the stack, which is shown in the function header on line 29. First it checks if the stack is empty for a stack underflow error, then on lines 33-34 the top element is removed from the stack and the next element is made the new top.

3 Queue

Listing 3: Queue Class

```
1   //Queue class with head and tail attributes as nodes
2   public class Queue {
3       Node head;
4       Node tail;
5
6       public Queue() {
7           head = null;
8           tail = null;
9       }
10      //Returns true when queue is empty, false otherwise
11      public Boolean isEmpty() {
12          if (head == null)
13              return true;
14          return false;
15      }
16      //Adds another character to the back of the queue
17      public void enqueue(char c) {
18          Node oldTail = tail;
19          tail = new Node();
20          //Creates a new node for the new tail, populates it with the recieved
21          tail.item = c;
22          tail.next = null;
23          if (isEmpty()) {
24              head = tail;
25          }
26          else {
27              oldTail.next = tail;
28          }
29      }
30      //Removes the char element from the front of the queue
31      public char dequeue() {
32          char returnVal = 0;
33          if (!isEmpty()) {
34              //If the queue isn't empty, remove the head and change it to the
35              returnVal = head.item;
36              head = head.next;
37              if (isEmpty()) {
38                  tail = null;
39              }
40          }
41          else {
42              System.out.println("Underflow");
43          }
44          return returnVal;
45      }
46
47  }
```

Similar to the stack class, the queue class uses the enqueue and dequeue functions to manipulate data in the queue. On line 17, the enqueue function takes in a character that is going to be inserted into the back of the queue. Lines 18-19 move the element that was previously the tail, and creates a new node for the new tail. Then, in lines 21-22 the new tail is assigned the character that was sent into the function, and makes the tail pointer null because it is the last element in the queue. As shown on line 31, the dequeue function removes an element from the front of the queue. It does this by first checking to make sure the queue isn't empty, and then on lines 35-36 the item at the head of the queue is removed and makes the next element the new head.

4 Main

Listing 4: Main Class

```
1   import java.io.*;
```

```java
import java.util.*;

public class Palindrome {

    public static void main(String[] args) {

        Stack myStack = new Stack();
        Queue myQueue = new Queue();

        String normal = "";
        String reverse = "";

        int numItems = 666;
        String[] magicItems = new String[numItems];
        int i = 0;

        File myFile = new File("magicitems.txt");

        try {
            Scanner fileScan = new Scanner(myFile);
        //Populates the array with the lines from text file
        while (fileScan.hasNext()) {
            magicItems[i] = fileScan.nextLine().toLowerCase().replace("␣", ""
            i++;
        }
        fileScan.close();
        }
        catch (FileNotFoundException e) {
            e.printStackTrace();

        }
        //For loop for every element (every line)
        for (int j = 0; j < numItems; j++) {
            //For loop to push and enqueue every character of each line
            for (int k = 0; k < magicItems[j].length(); k++) {
                myStack.push(magicItems[j].charAt(k));
                myQueue.enqueue(magicItems[j].charAt(k));
            }
            //Pop and dequeue each character until the stack (and queue) is e
            while (!myStack.isEmpty()) {
                reverse += myStack.pop();
                normal += myQueue.dequeue();
            }
            //Check to see if the element is equal to itself reversed
            if (reverse.equals(normal)) {
                System.out.println(normal + "␣is␣a␣palindrome.");
48                  }
49                  //reset the strings to empty for the next element
50                  normal = "";
51                  reverse = "";
52
53              }
54
55
56          }
57
58  }
```

The for loop on line 34 is used to traverse through every element of the array which is the strings from the text file. Using another for loop on line 36, each character of each element is traversed. Lines 37-38 push each character onto a stack and enqueue each character onto a queue. After every character of a single element is inserted into the stack and queue, still inside the first for loop, lines 42-43 pop and dequeue each character and store them into two strings, normal and reverse. This is done until the stack is empty, and now the two strings hold the original string and it reversed to check if its a palindrome.