Assignment 2 - Code Evaluation

Mason Nakamura

September 2021

1 Node Class

```
1  public class Node {
2      String name;
3      Node next;
4  }
```

Give object Node attributes String name and Node next (i.e. the proceeding node).

2 List Class

```
1  public class List {
2      Node head;
3
4      public boolean isEmpty(){
5          return(head == null);
6      }
7       // show the elements in the list
8      public void show(){
9          while(!isEmpty()){
10             if(head.next == null){
11                 System.out.print(head.name);
12             }else {
13                 System.out.print(head.name + " | ");
14             }
15             head = head.next;
16         }
17     }
18 }
```

In line 2 we give attribute head to object list to identify the head of the list. We use the isEmpty() method in our show() method which prints all elements in the list object, while isEmpty() finds whether the list has a head or not (i.e. an empty list or broken list). We use a while loop in line 9 since we don't know how long our list is and in it we print every head.name while assigning a new head in every loop.

3 Queue Class

```
1  public class Queue {
2      Node head, tail;
3
4      public boolean isEmpty(){
5          return(head == null);
6      }
7
8      public void enqueue(String s){
9          Node oldTail = tail;
10         tail = new Node();
11         tail.name = s;
12         tail.next = null;
13         if(isEmpty()){
14             head = tail;
15         }else{
16             oldTail.next = tail;
17         }
18     }
19
20     public String dequeue(){
21         String retval;
22         if(!isEmpty()){
23             retval = head.name;
24             head = head.next;
25             if(isEmpty()){
26                 tail = null;
27             }
28         }else{
29             retval = "";
30         }
31         return retval;
32     }
33
34     public void show(){
35         while(!isEmpty()){
36             if(head.next == null){
37                 System.out.print(head.name);
38             }else {
39                 System.out.print(head.name + " | ");
40             }
41             head = head.next;
42         }
43     }
44
45 }
```

We assign attributes head and tail which are Node objects to the Queue object. We assign oldTail to tail to preserve that instance of tail. Next, we create a new Node object and assign to tail where we assign the tail.name a string and tail.next to be null since it is the end of the Queue. In line 13, we see if head is null, and if so, we assign tail to the head. Since most of the cases fall into the else statement, it may be wise to switch the if and else statement to save on effort.

For the dequeue method on line 20, we initialize a string retval which we intend to return later on. Then we check if the head exists; if so, we assign the name of the head to retval, move the head to the next item in the queue, and if the head is null, then we assign the tail to be null. If the head is null to begin with, then we return an empty string. If queue survives all cases it returns retval and changes head to head.next.

On line 34, the show() method prints all elements in the queue for testing.

4 Stack Class

```
1         public class Stack {
2       Node top;
3       String setVal;
4
5       public boolean isEmpty(){
6           return(top == null);
7       }
8
9       public void push(String s){
10          Node oldTop = top;
11          top = new Node();
12          top.name = s;
13          top.next = oldTop;
14      }
15
16      public String pop(){
17          if(!isEmpty()){
18              setVal = top.name;
19              top = top.next;
20          }else{
21              return ("");
22          }
23          return setVal;
24      }
25
26      public String peek(){
27          if(!isEmpty()){
28              setVal = top.name;
29          }else{
30              return ("");
31          }
32          return setVal;
33      }
34
35      public void show(){
36          while(!isEmpty()){
37              if(top.next == null){
38                  System.out.print(top.name);
39              }else {
40                  System.out.print(top.name + " | ");
41              }
42              top = top.next;
43          }
44      }
```

```
45 }
```

We assign attribute top to the Stack object. Next, we define the push() method which pushes a string onto the stack and assigns the string to the top of the stack. On line 16, the pop() method checks if the stack is empty; if not, it assigns setVal to the name of the top of the stack and assigns the top of the stack to the next node. It then returns setVal, taking into account an empty stack. The peek() method on line 26 is very similar to the pop() method, except it does not reassign the top node. The show() method is used for testing.

5 Test Class

```
1      import java.io.*;
2  import java.util.Arrays;
3
4  public class Test {
5
6      public static void main(String[] args) {
7          String[] words = {};
8          // Testing for List class and Node class
9          Node person1 = new Node();
10         person1.name = "Jhon Von Neumann";
11         person1.next = null;
12         Node person2 = new Node();
13         person2.name = "Paul Erdos";
14         person2.next = null;
15         person1.next = person2;
16         Node person3 = new Node();
17         person3.name = "Carl Gauss";
18         person3.next = null;
19         person2.next = person3;
20         Node person4 = new Node();
21         person4.name = "Aristotle";
22         person4.next = null;
23         person3.next = person4;
24
25         List people = new List();
26         people.head = person1;
27         System.out.println("List:");
28         people.show();
29         System.out.println();
30
31         Stack stacked_people = new Stack();
32         stacked_people.top = person1;
33         System.out.println("Stack:");
34         stacked_people.show();
35         System.out.println();
36
37
38         Queue queued_people = new Queue();
39         queued_people.head = person1;
40         queued_people.tail = person4;
41         System.out.println("Queue:");
42         queued_people.show();
```

```
43          System.out.println();
44
45
46          String fileName = "Assignment 1/Test_Cases";
47          File file = new File(fileName);
48          try {
49              FileReader fr = new FileReader(file);
50              BufferedReader br = new BufferedReader(fr);
51              String line;
52              while ((line = br.readLine()) != null) {
53                  // add strings from txt file line by line into
                        array words
54                  words = Arrays.copyOf(words, words.length + 1); //
                        extends memory
55                  words[words.length - 1] = line; //adds word to
                        extra memory
56              }
57          } catch (FileNotFoundException e) {
58              System.out.println("An error occurred.");
59              e.printStackTrace();
60          } catch (IOException e) {
61              e.printStackTrace();
62          }
63          int times = 0;
64          for (String i : words) {
65              // initialize stack and queue for each line of words
66              Stack sword = new Stack();
67              Queue qword = new Queue();
68              // make all letters lowercase
69              i = i.toLowerCase();
70              // remove all spaces
71              i = i.replaceAll(" ", "");
72              // push every character into the stack and enqueue into
                     a queue as strings
73              for(int j = 0; j < i.length(); j++){
74                  //get every character from every line
75                  char k = i.charAt(j);
76                  // convert characters to strings
77                  String string_char = Character.toString(k);
78                  sword.push(string_char);
79                  qword.enqueue(string_char);
80              }
81              // initialize reversed words as empty strings
82              String reversed_word_stack = "";
83              String word_queue = "";
84              // pop every element in the stack to get the reversed
                    word and dequeue in the queue
85              for(int k = 0; k < i.length(); k++){
86                  word_queue += qword.dequeue();
87                  reversed_word_stack += sword.pop();
88              }
89              // see if reversed word is equal to dequeued word(i.e.
                    a palindrome)
90              if (reversed_word_stack.equals(word_queue)){
91                  // print word if palindrome
92                  System.out.println(reversed_word_stack);
93                  times += 1;
```

```
94                  }
95              }
96              System.out.println("Number of Palindromes: " + times);
97          }
98      }
```

From lines 8-43, I tested the Node class, Stack class, Queue class, and List class, using the same elements in all tests. It then prints out each element in the different objects, but does not test the methods in their particular classes. From lines 46-62, I used a file reader to read line by line the strings in the .txt file. On lines 64-97 I iterated through every line in the .txt file, converted all characters to lowercase, removed any spaces, and converted every character to a string. I then pushed and enqueued each character of every line to their respective objects. Then I initialized empty strings to test for palindromes when dequeueing and popping to theses empty strings. Since the dequeueing in the queue will output the original line while the popping from the stack will output the reversed order of the line, we can check for palindromes by using a method from Java to check for equality from two strings. This outputs the palindrome if it exists as well as the number of palindromes at the end.