Assignment One - Data Structures and Magic Items

Charlie Schmitz

September 16th 2020

---

1 Algorithms Assignment One Main Class File

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.Collections;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Scanner;
import java.util.*;
import java.io.*;

public class Algorithms_Assignment_One {

  public static void main(String[] args) {
    Singly_Linked_List linkedList = new Singly_Linked_List();
    Singly_Linked_List palindrome = new Singly_Linked_List();
    Queue myQueue = new Queue();
    Stack myStack = new Stack();
    int arrayCounter = 0;
    boolean isPalindrome = true;


  /***************File System Import*****************************
    */
    String content = new String();
    int count=1;
    File file = new File("magicItems.txt");

     try {
    Scanner sc = new Scanner(new FileInputStream(file));

    while (sc.hasNextLine()){

      content = sc.nextLine();
      System.out.println("------------------------------");
      System.out.print("Linked List : ");
```

```
35        System.out.println("Element - " + (linkedList.size()+1));
36          linkedList.addHead(content);
37
38
39          for(int i = 0; i < content.length(); i++) {
40            //get each character and then cast character to string
    for push and enqueue
41            char c = content.charAt(i);
42            String s = String.valueOf(c);
43
44            //break down each line into characters then convert that
    to String to push and enqueue...
45            //output in console WILL SHOW EACH CHARACTER TWICE AS IT
    SHOWS THAT EACH HAS BEEN ADDED TO STACK/QUEUE CORRECTLY, COULD
    COMMENT OUT CONFIRMATION MESSAGE IF DESIRED
46
47            if(!checkEmptyString(s) && !checkBlankString(s)) {
48              System.out.print("Stack - ");
49                myStack.push(s);
50              System.out.print("Queue - ");
51                myQueue.enqueue(s);
52            }
53          }
54
55          int tempSize = 0;
56          isPalindrome = true;
57          while(tempSize < content.replace(" ", "").length()) {
58            System.out.println("--------");
59            String tempCharStack = myStack.pop();
60              String tempCharQueue = myQueue.dequeue();
61
62
63
64              if(isPalindrome == true) {
65                if(tempCharStack.toLowerCase().equals(tempCharQueue.
    toLowerCase())) {
66                  isPalindrome = true;
67
68                }//if
69                else {
70                  isPalindrome = false;
71                }
72              }
73            tempSize++;
74            //System.out.println(tempSize);
75            //System.out.println(content.length());
76          }//while
77
78          if(isPalindrome == true) {
79            System.out.println(content + " is  a Palindrome");
80            palindrome.addHead(content);
81            arrayCounter++;
82          }//if
83          else {
84            System.out.println(content + " is NOT a Palindrome");
85
86          }//else
```

```
87
88
89
90
91      }//while
92      sc.close();
93
94      }catch(FileNotFoundException fnf){
95      fnf.printStackTrace();
96      }
97
98      catch (Exception e) {
99      e.printStackTrace();
100     System.out.println("\nProgram terminated Safely...");
101     }
102
103
104      //Use to Remove All Items from Linked List
105      System.out.println("\n--------Removes All Elements in List
        Once Processing Complete----------");
106     while(!linkedList.isEmpty()) {
107       linkedList.removeHead();
108     }//while
109
110     palindrome.printList(palindrome);
111
112   }//main
113
114   public static boolean checkEmptyString(String string) {
115     return string == null || string.length() == 0;
116   }//used to check to make sure not pushing empty spaces into
        stacks and queues
117   public static boolean checkBlankString(String string) {
118       return string == null || string.trim().isEmpty();
119   }////used to check to make sure not pushing blank spaces into
        stacks and queues
120
121
122 }//Algorithms_Assignment_One
123
124
125   /*******************************************/
126
127     // -Testing if each character is read properly -
128     //myStack.getEachCharacter("12345");
129
130     /*******************************************
131      * Initial Testing For Singly Linked List
132     linkedList.addHead("my first element");
133     linkedList.addHead("my second element");
134     linkedList.addHead("my third element");
135     linkedList.addTail("my Four element");
136     linkedList.addTail("my Five element");
137     linkedList.addTail("my Six element");
138
139     linkedList.removeElement("my Six element");
140
```

```
141     while(!linkedList.isEmpty()) {
142       linkedList.removeHead();
143     }
144      ******************************************/
145
146     /******************************************
147      * Initial Testing For Queue
148     myQueue.enqueue("person 1");
149     myQueue.enqueue("person 2");
150     myQueue.enqueue("person 3");
151     myQueue.enqueue("person 4");
152     myQueue.enqueue("person 5");
153     myQueue.enqueue("person 6");
154     myQueue.dequeue();//should be person 1, 2, ...
155     myQueue.dequeue();
156     myQueue.dequeue();
157     myQueue.dequeue();
158     myQueue.dequeue();
159     myQueue.dequeue();
160     ******************************************/
161
162     /******************************************
163      * Initial Testing For Stack
164     myStack.push("person 1");
165     myStack.push("person 2");
166     myStack.push("person 3");
167     myStack.push("person 4");
168     myStack.pop();
169     myStack.pop();
170     myStack.pop();
171     myStack.pop();
172     ******************************************/
173
```

2 Singly Linked List

```
1
2  public class Singly_Linked_List {
3
4    //Node Class Used Created for Linked List
5    private static class Node {
6      private String myData;
7      private Node myNext;
8
9      public Node(String data, Node next) {
10       myData = data;
11       myNext = next;
12     }//NodeConstructor
13
14     public String getData(){
15       return myData;
16     }//getData
17
18     public Node getNext() {
19       return myNext;
20     }//getNext
21
22     public void setNext(Node newNext) {
23       myNext = newNext;
24     }//setNext
25
26   }//NodeClass
27
28   //Linked List
29   private Node head = null;
30   private Node tail = null;
31   private int size = 0;
32
33   public Singly_Linked_List() {
34
35   } //SinglyLinkedList()
36
37   public int size() {
38     return size;
39   }//size of list
40
41   public boolean isEmpty() {
42     if (size == 0) {
43       return (true);
44     }//if
45     else {
46       return (false);
47     }//else
48   }//Checks if list is empty
49
50
51   public String head() {
52     if(isEmpty()) {
53       return null;
54     }//if list is empty return null
55     else {
```

```
56        return head.getData();
57      }//if list is not empty return the value from the head
58    }//first or head of the list is checked and value returned if
      there is one
59
60    public String tail() {
61      if(isEmpty()) {
62        return null;
63      }//last element of list checked and if empty returns null
64      else {
65        return tail.getData();
66      }//otherwise, return the element at the tail
67    }//last or tail of the list is checked and value returned if
      there is one
68
69  //you can add an element to the beginning (head) of the list or to
      the end (tail) done
70    public void addHead(String element) {
71      head = new Node(element, head);
72      //create a new node, takes the element passed to the function,
      creates pointer at the head
73
74      if(size == 0) {
75        tail = head;
76      }//if the size of the list is 0 the value serves as both the
      tail and head values of the list
77
78      size++; //up's the size with each addition
79      System.out.println("Added Node Element '" + head.getData() + "'
       to First (head) Position");
80    }//addHead
81
82    public void addTail(String element) {
83      Node newNode = new Node(element, null);
84      if(isEmpty()) {
85        head = newNode;
86      }//if the list is empty the head will become this element
87      else {
88        tail.setNext(newNode);
89      }//else not empty the tail will point to the new Node created
90
91      tail = newNode;
92      size++;
93      System.out.println("Added Node Element '" + tail.getData() + "'
       to Last (tail) Position");
94
95    }//addTail
96
97    public String removeHead() {
98      if(isEmpty()) {
99        return null;
100     }//if
101     else {
102       String ans = head.getData();
103       head = head.getNext(); //set the head to the new next value
      in the list
104
```

```
105       size--;
106       if(size==0) {
107         tail = null;
108       }//if no removed all items in the list assign null value to
      tail to show that
109
110       System.out.println("Removed Node Element '" + ans + "' From
      First (head) Position");
111
112       return ans;
113     }//else
114
115
116   }//removes the Node at the head of the list
117
118   public String removeElement(String element) {
119     Node current = head;
120     Node previous = head;
121     int position = 0;
122     while((current != null) && (current.getData() != element)) {
123       previous = current;
124       current = current.getNext();
125       position++;
126     }//while we are not at the end of the list or we have not found
       the element being searched
127     //keeps track of where we are in the list and what is in front/
      behind us
128
129     if(current==null) {
130       return null;
131     }//if current element is null then it doesn't exist
132     else {
133       if(head==current) {
134         head = current.getNext();
135       }//if
136       else if(tail==current) {
137         tail = previous;
138         tail.setNext(null);
139       }//else if
140       else {
141         previous.setNext(current.getNext());
142       }//else
143
144       System.out.println("Elementwas Found and Removed at position
      " + position);
145       size--;
146       return current.getData();
147     }
148
149   }//Deletes a Node at a certain location/Finds position of
      specific element in the list
150
151   public static void printList(Singly_Linked_List list) {
152
153     Node currNode = list.head;
154
155       System.out.print("\nPalindromes: ");
```

```
156
157         // Traverse through the LinkedList
158         while (currNode != null) {
159             // Print the data at current node
160             System.out.print(currNode.getData() + ", ");
161 
162             // Go to next node
163             currNode = currNode.getNext();
164         }
165 
166 
167 
168   }//toString
169 
170 
171 }//Singly Linked List
```

---

3 Stack

```
1
2  public class Stack {
3    private Singly_Linked_List list = new Singly_Linked_List();
4
5    public Stack() {
6
7    }//constructor for the stack
8
9    public int getSize() {
10     return list.size();
11   }//get size
12
13   public boolean isEmpty() {
14     return list.isEmpty();
15   }//isEmpty
16
17 //First In - Last Out
18   public void push(String element) {
19     list.addHead(element);
20   }//push
21
22   public String pop() {
23     return list.removeHead();
24   }//pop
25
26   public String getTop() {
27     return list.head();
28   }//getTop, gets the element on the top of the stack
29
30   public void getEachCharacter(String myString) {
31
32     System.out.println("Initial String = " + myString);
33
34     //Cycle through each character of the string and assign to
       constant c to be printed/compared
35     for(int i = 0; i < myString.length(); i++) {
36       char c = myString.charAt(i);
37       System.out.println(c);
38     }//for
39
40
41
42     /*
43      * //declare a new stack to read into
44       String newString = "";
45       Stack myStack = new Stack();
46       for(int i = 0; i < myString.length(); i++) {
47         myStack.push(myString.substring(i,i+1));
48       }
49
50       for(int i = 0; i < myString.length(); i++) {
51         myStack.push(myString.substring(i,i+1));
52
53       }//loops through all the characters in the string and push's
     each individual character
```

```
54
55        while(!myStack.isEmpty()) {
56          newString += myStack.pop();
57        }
58
59        System.out.println("Final String = " + newString);
60
61        return newString;
62        */
63
64
65    }//getEachCharacter returns each individual character of a string
        put in the stack
66
67
68
69  }//Stack
```

---

4 Queue

```
1
2  public class Queue {
3
4    private Singly_Linked_List list = new Singly_Linked_List();
5
6    public Queue() {
7
8    }//constructor
9
10   public int getSize() {
11     return list.size();
12   }//get Size
13
14   public boolean isEmpty() {
15     return list.isEmpty();
16   }//isEmpty
17
18 //First in - First Out
19   public void enqueue(String element) {
20     list.addTail(element);
21   }//adding item onto the tail of the line
22
23   public String dequeue() {
24     return list.removeHead();
25   }//removing item from front of line
26
27   public String head() {
28     return list.head();
29   }//see what is at the front of the line
30
31 }//Queue
```