Assignment One

Shannon Brady

shannon.brady2@Marist.edu

September 17, 2022

1 Node and LinkedList

1.1 Node Class

A Node has the following attributes:

1. Name (a string value)
2. Next (a Node value indicating the next item in the linked list, stack, or queue)

```
1  class Node {
2
3      private String name = "";
4      private Node next = null;
5
6      /* Node class constructor */
7      public Node(String name) {
8          this.name = name;
9          this.next = null;
10     } // Node
11
12     /* Getters and Setters */
13     public String getName() {
14         return this.name;
15     } // getName
16
17     public Node getNext() {
18         return this.next;
19     } // getNext
20
21     public void setName(String newName) {
22         this.name = newName;
23     } // setName
24
```

```
25      public void setNext(Node newNext) {
26          this.next = newNext;
27      } // setNext
28
29      @Override
30      public String toString() {
31          return("Name: " + this.getName() + "\n");
32      } // toString
33  } // Node
```

1.2 LinkedList Class

The LinkedList class implements a linked list data structure by utilizing the Node class. A LinkedList has the following attributes:

1. head (a Node value indicating the first item in the linked list)
2. length (an integer value indicating list length)

A LinkedList has the following methods:

1. add<string> (add a new Node, line 13)
    a) If the head is null, the new node becomes the head; otherwise, add the node to the end of the list.
2. remove<string> (remove an existing Node, line 33)
    a) First we check if the target string matches a node name in the list. Next, if there's not a previous node, set the head to the current node's next. If there is a previous node, connect the previous node's next to the current node's next.
3. print (print all Nodes in the list, line 58)
4. isEmpty (checks if list is empty, line 74)
    a) If the head is null, then the list is empty.

```
1
2   class LinkedList {
3
4       private Node head = null;
5       private int length = 0;
6
7       /* LinkedList class constructor */
8       LinkedList() {
9           this.head = null;
10          this.length = 0;
11      } // LinkedList
12
13      public void add(String str) {
14          Node newNode = new Node(str);
15
16          if (this.head == null) {
17              // If the list is empty, set the head
18              this.head = newNode;
19          } else {
20              // Otherwise traverse entire list and add newNode to the end
```

```
            Node lastNode = this.head;
            while (lastNode.getNext() != null) {
                lastNode = lastNode.getNext();
            }

            // lastNode.next = newNode;
            lastNode.setNext(newNode);
        }
        //increment length
        this.length++;
    } // add

    public void remove(String str) {
        Node currNode = this.head;
        Node prevNode = null;
        int i =0;
        while (currNode != null) {
            if (str.equals(currNode.getName())) {
                if (prevNode == null) {
                    // Removing the head of the list
                    // Update head node
                    this.head = currNode.getNext();
                } else {
                    // Connect prevNode's next to currNode's next
                    prevNode.setNext(currNode.getNext());
                    // CurrNode is removed
                    currNode.setNext(null);
                }
            }
            i++;
            prevNode = currNode;
            currNode = currNode.getNext();
        }
        //decrement length
        this.length--;
    } // remove

    public void print() {
        // Prints all nodes in the list
        Node currNode = this.head;
        int i = 0;

        while (currNode != null) {
            System.out.println("Index: "+ i + "\n" +
                               currNode.toString());

            // Set the new next
            currNode = currNode.getNext();
            i++;
        }
        System.out.println();
    } // print

    public boolean isEmpty() {
```

```
75          if (this.head == null) {
76              return true;
77          } else {
78              return false;
79          }
80      } // isEmpty
81
82      /* Getters and Setters */
83      public LinkedList getLinkedList() {
84          return this;
85      } // getLinkedList
86
87      public Node getHead() {
88          return this.head;
89      } // getHead
90
91      public void setHead(Node newHead) {
92          this.head = newHead;
93      } // setHead
94
95  }
```

2 Stack Class

The Stack class implements a stack data structure by utilizing the Node class. A Stack has the following attributes:

1. top (a Node value indicating the first item in the stack)
2. length (an integer value indicating stack length)

A Stack has the following methods:

1. pop (remove a Node from the stack, line 11)
    a) If top is not null, then set the top to the next item in the stack.
2. push (add a Node to the stack, line 22)
    a) The new Node becomes the top of the stack.
3. print (print all Nodes in the stack, line 32)
4. isEmpty (checks if stack is empty, line 44)
    a) If the top is null, then the stack is empty.

```
1   class Stack {
2
3       private Node top = null;
4       private int length = 0;
5
6       /* Stack class constructor */
7       Stack() {
8           this.length = 0;
9       } //Stack
```

```
10
11      public Node pop() {
12          Node x = null;
13          if (top != null) {
14              x = this.top;
15              this.top = x.getNext();
16
17              this.length--;
18          }
19          return x;
20      } // pop
21
22      public void push(String str) {
23          Node newNode = new Node(str);
24
25          // newNode.next = this.top;
26          newNode.setNext(this.top);
27          this.top = newNode;
28
29          this.length++;
30      } // push
31
32      public void print() {
33          // Prints all nodes in the stack
34          Node currNode = this.top;
35
36          while (currNode != null) {
37              System.out.println(currNode.toString());
38
39              // Set the new next
40              currNode = currNode.getNext();
41          }
42      } // print
43
44      public boolean isEmpty() {
45          if (this.top == null) {
46              return true;
47          } else return false;
48      } // isEmpty
49
50      /* Getters and Setters */
51      public Stack getStack() {
52          return this;
53      } // getStack
54
55      public Node getTop() {
56          return this.top;
57      } // getTop
58
59      public int getLength() {
60          return this.length;
61      } // getLength
62
63      public void setTop(Node newTop) {
```

```
64          this.top = newTop;
65      } // setTop
66  }
67
```

---

3 Queue Class

The Queue class implements a queue data structure by utilizing the Node class. A Queue has the following attributes:

1. head (a Node value indicating the first item in the queue)
2. tail (a Node value indicating the last item in the queue)
3. length (an integer value indicating the length of the queue)

A Queue has the following methods:

1. enqueue (add a Node to the end of the queue, line 14)
    a) If the queue is empty, then the head and tail both point to the same Node; otherwise only the tail is updated.
2. dequeue (remove a Node from the front of the Queue, line 33)
    a) If the queue is not empty, update the head to the next Node
3. print (print all Nodes in the Queue, line 47)
4. isEmpty (checks if queue is empty, line 63)
    a) If the head is null, then the queue is empty.

```
1   class Queue {
2
3       private Node head = null;
4       private Node tail = null;
5       private int length = 0;
6
7       /* Queue class constructor */
8       Queue() {
9           this.head = null;
10          this.tail = null;
11          this.length = 0;
12      }
13
14      public void enqueue(String str) {
15          Node newNode = new Node(str);
16
17          if (this.head == null) {
18              // Head and tail are the same if queue length is 1
19              this.head = newNode;
20              this.tail = newNode;
21          } else {
22              // Make the tail's next the new node
23              this.tail.setNext(newNode);
```

```
            // Make the new node the tail
            this.tail = newNode;
            // Set the new node next value to null
            newNode.setNext(null);
        }

        this.length++;
    } // enqueue

    public Node dequeue() {
        Node x = null;
        if (this.head != null) {
            // If the queue isn't empty, grab the current head node
            x = this.head;
            // Update the head
            this.head = x.getNext();
            x.setNext(null);
            // Decrement the queue length
            this.length--;
        }
        return x;
    } // dequeue

    public void print() {
        // Prints all nodes in the list
        Node currNode = this.head;
        int i = 0;

        while (currNode != null) {
            System.out.println("Index: "+ i + "\n" +
                               currNode.toString());

            // Set the new next
            currNode = currNode.getNext();
            i++;
        }
        System.out.println();
    }

    public boolean isEmpty() {
        if (this.head == null) {
            return true;
        } else {
            return false;
        }
    } // isEmpty

    /* Getters and Setters */
    public Node getHead() {
        return this.head;
    } // getHead

    public Node getTail() {
        return this.tail;
```

```
78      } // getTail
79
80      public int getLength() {
81          return this.length;
82      } // getLength
83
84      public void setHead(Node newHead) {
85          this.head = newHead;
86      } // setHead
87
88      public void setTail(Node newTail) {
89          this.tail = newTail;
90      } // setTail
91  }
```

4 Test and Main

4.1 Test Class

The Test class uses various methods to test each of the previously outlined data structures.

A Test has the following methods:

1. testMyLinkedList (validates add, remove, and isEmpty methods, line 16)
2. testMyStack (validates pop, push, and isEmpty methods, line 49)
3. testMyQueue (validates enqueue, dequeue, and isEmpty methods, line 94)
4. checkPalindrome (identifies all palindromes in a text file, line 153)
    a) Read an input file and add all lines of text to an array.
    b) Loop through all items in the array and remove all whitespace and special symbols, then make all letters lowercase.
    c) For each line of text, push each character to a stack and a queue. Then loop through either the stack or the queue (does not matter since length is equivalent) and pop and dequeue a letter from the stack and queue, respectively.
    d) If the letters returned do not match at any given point, then the phrase cannot be a palindrome.
    e) Add all palindromes to a linked list and print the results.

```
1   import java.io.File;
2   import java.io.FileNotFoundException;
3   import java.util.Scanner;
4
5   class Test {
6
7       Test() {
8       }
9
10      /* Testing Methods:
11       *      testMyLinkedList()
12       *      testMyStack()
```

```
13          *       testMyQueue()
14          *       checkPalindrome()
15          */
16      public void testMyLinkedList() {
17          // Create a linked list of really awesome songs
18          LinkedList songs = new LinkedList();
19
20          // Validate the list is empty
21          if (songs.isEmpty()) {
22              System.out.println("Your list is empty.");
23          }
24
25          // Add some really awesome songs
26          songs.add("Particles - Nothing But Thieves");
27          songs.add("Sugarcoat - Kid Bloom");
28          songs.add("Little One - Highly Suspect");
29          songs.add("Why Are Sundays So Depressing - The Strokes");
30          songs.add("My Honest Face - Inhaler");
31          songs.add("Safari Song - Greta Van Fleet");
32          songs.add("Down the Road - Dirty Honey");
33
34          // Print the list of really awesome songs
35          System.out.println("Awesome Songs");
36          System.out.println("---------");
37          songs.print();
38
39          // Make sure remove is removing
40          songs.remove("My Honest Face - Inhaler");
41          songs.remove("Particles - Nothing But Thieves");
42          songs.remove("Down the Road - Dirty Honey");
43
44          System.out.println("Updated Awesome Songs");
45          System.out.println("---------");
46          songs.print();
47      }
48
49      public void testMyStack() {
50          // Create a stack of words that rhyme with stack
51          Stack rhymes = new Stack();
52
53          // Validate stack is empty
54          if (rhymes.isEmpty()) {
55              System.out.println("Your stack is empty.");
56          }
57
58          // Add some rhymes
59          rhymes.push("Rack");
60          rhymes.push("Slack");
61          rhymes.push("Whack");
62          rhymes.push("Snack");
63          rhymes.push("Quack");
64          rhymes.push("Hack");
65          rhymes.push(("Soundtrack"));
66
```

```
67          // Print da rhymes
68          System.out.println("Rhymes 1");
69          System.out.println("---------");
70          rhymes.print();
71
72          // Make sure pop() is poppin...
73          System.out.println("\nPop() Test:");
74          System.out.println("---------");
75
76          Stack rhymes_2 = new Stack();
77          int length = rhymes.getLength();
78
79          for (int i=0; i<length; i++) {
80              // Pop and print
81              Node x = rhymes.pop();
82              System.out.println(x.toString());
83
84              // Push to new stack, order is now flipped
85              rhymes_2.push(x.getName());
86          }
87
88          // Print rhymes again, in reverse order
89          System.out.println("\nRhymes 2");
90          System.out.println("---------");
91          rhymes_2.print();
92      }
93
94      public void testMyQueue() {
95          // Create a queue of people waiting to checkout at Stop & Shop
96          Queue checkout = new Queue();
97
98          // Add the peeps
99          checkout.enqueue("Karen");
100         checkout.enqueue("Evan");
101         checkout.enqueue("David");
102         checkout.enqueue("Marco");
103         checkout.enqueue("Shannon");
104         checkout.enqueue("Dan");
105         checkout.enqueue("Alan");
106
107         System.out.println("Checkout:");
108         System.out.println("---------");
109         checkout.print();
110
111         // Karen is now being helped
112         Node karen = checkout.dequeue();
113         System.out.println("Checkout:");
114         System.out.println("---------");
115         checkout.print();
116
117         // Karen, in Karen fashion, made a scene asking for a manager...
118         Queue customerService = new Queue();
119         customerService.enqueue(karen.getName());
120         System.out.println("Customer Service:");
```

```
121         System.out.println("---------");
122         customerService.print();
123 
124         // Alan is now next in line
125         int length = checkout.getLength();
126         for (int i=0; i<length-1; i++) {
127             Node customer = checkout.dequeue();
128         }
129         System.out.println("Checkout:");
130         System.out.println("---------");
131         checkout.print();
132 
133         // Karen was told to get back on the checkout line
134         // She tried to cut Alan but he didn't fall for her tricks
135         checkout.enqueue(karen.getName());
136         System.out.println("Checkout:");
137         System.out.println("---------");
138         checkout.print();
139 
140         // Finally they all go home
141         length = checkout.getLength();
142         for (int i=0; i<length; i++) {
143             Node customer = checkout.dequeue();
144         }
145 
146         System.out.println("Checkout:");
147         System.out.println("---------");
148         if (checkout.isEmpty()) {
149             System.out.println("The checkout line is empty.");
150         }
151     }
152 
153     public void checkPalindrome() {
154         // magicitems.txt
155         String[] items = new String[666];
156 
157         // palindromes.txt
158         // String[] items = new String[13];
159 
160         LinkedList palindromes = new LinkedList();
161         Stack stack = new Stack();
162         Queue queue = new Queue();
163         int i = 0;
164 
165         try {
166             File file = new File("magicitems.txt");
167             // File file = new File("palindromes.txt");
168             Scanner myReader = new Scanner(file);
169 
170             while (myReader.hasNextLine()) {
171             String data = myReader.nextLine();
172             items[i] = data;
173             i++;
174             }
```

```
            myReader.close();
        } catch (FileNotFoundException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }

        for (int w=0; w<items.length; w++) {
            // Normally I'd initialize this to false, but it seems to make more sense to look for all the words that aren't pali
            // A non-palindrome and palindrome can both have matching characters on opposing ends of the string.
            // A palindrome will never have mismatching characters, a non-palindrome will.
            boolean isPalindrome = true;

            // Remove all whitespace and symbols
            items[w] = items[w].replaceAll("[\\s,\\.'\\(\\)+-]", "");

            // Convert to all lower case
            items[w] = items[w].toLowerCase();

            for (int j=0; j<items[w].length(); j++) {
                // Loop through all the character in the string
                // and add each character to a stack and queue, respectively
                stack.push(String.valueOf(items[w].charAt(j)));
                queue.enqueue(String.valueOf(items[w].charAt(j)));
            }

            int length = stack.getLength();
            for (int c=0; c<length; c++) {
                // Pop and dequeue each letter
                String char1 = stack.pop().getName();
                String char2 = queue.dequeue().getName();

                if (!(char1.equals(char2))) {
                    // A word cannot be a palindrome if both chars are not the same
                    isPalindrome = false;
                }
            }
            if (isPalindrome) {
                // Add all palindromes to a separate list
                palindromes.add(items[w]);
            }
        }
        palindromes.print();
    }
}
```

4.2 Main Class

The Main class creates an instance of the Test class and calls each testing method.

```
class Assignment_1 {
    public static void main(String[] args) {

        Test test = new Test();
```

```
5
6           // Test each of the data structures
7           test.testMyLinkedList();
8           test.testMyStack();
9           test.testMyQueue();
10
11          // Check for palindromes
12          test.checkPalindrome();
13      }
14  }
```