CMPT 435 - Fall 2021 - Dr. Labouseur

Assignment One – Palindrome Scanner

C. Marcus DiMarco
C.DiMarco1@Marist.edu

September 24, 2021

1 Objectives

- Create a program that prints all palindromes within a given text file.

2 Conditions

- A palindrome is defined as a String that is equal to itself reversed without considering special characters.
- Only palindromes should be printed.
- The program must only use custom data structures for Node, Stack and Queue. Any of these structures which already exist in the language may not be used.

3 Overview

Starting with the most elemental unit of the program, MarcusNode is the Node foundation for our custom LinkedList. Using MarcusNode, we are able to create our custom Stack and Queue, aptly named MarcusStack and MarcusQueue.

3.1 MarcusNode

We want to keep track of a list of Strings by pairing each String with a reference to the next String in the list. The MarcusNode class does this with its two field declarations on lines 7-8 and encapsulates them to preserve their contents. The class does not declare a default constructor, ensuring that each MarcusNode object will have an associated String field.

```java
 1 /**
 2  * Custom Node class which will serve as the container for
 3  * individual characters from the strings to be tested.
 4  * Must point to another Node object or null.
 5  */
 6 public class MarcusNode {
 7   private String item;
 8   private MarcusNode next;
 9 
10   // Not having a default constructor forces String input
11   public MarcusNode(String item) {
12     this.item = item;
13     this.next = null;
14   }
15 
16   public MarcusNode(String item, MarcusNode next) {
17     this.item = item;
18     this.next = next;
19   }
20 
21   // Setters and getters for private fields
22   public void setItem(String item) {
23     this.item = item;
24   }
25 
26   public String getItem() {
27     return this.item;
28   }
29 
30   public void setNext(MarcusNode next) {
31     this.next = next;
32   }
33 
34   public MarcusNode getNext() {
35     return this.next;
36   }
37 }
```

3.2 MarcusStack

Building off of the list made possible by MarcusNode, MarcusStack organizes the list in a ”first in, last out” sequence. Whichever MarcusNode was added last will be the first removed. We’ll use this in the program to output a String in reverse.

```java
/**
 * Custom Stack class which will take Nodes as input and will output
 * those Nodes in reverse order.
 */
public class MarcusStack {
  private MarcusNode head;

  // Default constructor
  public MarcusStack() {
    this.head = null;
  }

  // Setters and getters for private field head
  public void setHead(MarcusNode head) {
    this.head = head;
  }

  public MarcusNode getHead() {
    return this.head;
  }

  // Pushes a node on top of the stack
  public void push(MarcusNode node) {
    node.setNext(this.head);  // Will set null if stack is empty
    this.head = node;
  }

  // Pops a node off the top of the stack
  public MarcusNode pop() {
    MarcusNode returnNode = this.head;
    if (this.isEmpty()) {    // Catch stack underflow
      return returnNode;
    } else {
      this.head = head.getNext();
      return returnNode;
    }
  }

  // Returns true if stack is empty
  public boolean isEmpty() {
    if (this.head == null) {
      return true;
    } else {
      return false;
    }
  }
}
```

3.3 MarcusQueue

In stark contrast to MarcusStack, MarcusQueue will organize a list in a ”first in, first out” sequence. This will preserve the same order in which data was entered. We will use this not only to output our forwards-comparison String, but also to house the input list from the text file.

```java
/**
 * Custom Queue class which will take Nodes as input and will output
 * those Nodes in same order.
 */
public class MarcusQueue {
  private MarcusNode head;
  private MarcusNode tail;

  // Default constructor
  public MarcusQueue() {
    this.head = null;
    this.tail = null;
  }

  // Setters and getters for private fields
  public void setHead(MarcusNode head) {
    this.head = head;
  }

  public MarcusNode getHead() {
    return this.head;
  }

  public void setTail(MarcusNode tail) {
    this.tail = tail;
  }

28   public MarcusNode getTail() {
29     return this.tail;
30   }
31
32   // Enqueues an item to the end of the queue
33   public void enqueue(MarcusNode node) {
34     if (this.isEmpty()) {
35       this.head = node;
36       this.tail = node;
37     } else {
38       this.tail.setNext(node);
39       this.tail = node;
40     }
41   }
42
43   // Dequeues a node from the front of the queue
44   public MarcusNode dequeue() {
45     MarcusNode returnNode = this.head;
46     if (this.isEmpty()) {    // Checks for stack underflow
47       return returnNode;
48     } else {
49       this.head = returnNode.getNext();
50       if (this.isEmpty()) {  // Checks if tail needs to be assigned null
51         this.tail = null;
52       }
53       return returnNode;
54     }
55   }
56
57   // Returns true if queue is empty
58   public boolean isEmpty() {
59     if (this.head == null) {
60       return true;
61     } else {
62       return false;
63     }
64   }
65 }
```

Note that the method enqueue() (line 33) only assigns the head field if the queue is empty. A queue uses the head field to track which node is next to be output, so we only need to assign it during dequeue() unless we are enqueueing the first node.

3.4 Assignment1

With the structures we have just created, we are ready to create our palindrome scanner. We'll need some imported libraries: namely, java.io.File to import a file, java.io.FileNotFoundException to account for errors finding the input file, and java.util.Scanner to read the file.

For better readability, we'll also create a separate method here called onlyLettersToLowercase(), which will take a String and return the same String as only lowercase letters without special characters or spaces. Using this, we can compare our forwards and backwards Strings directly using the internal String.equals() method.

```java
/**
 * A program which reads a String from a file and uses a Stack and Queue
 * to compare it to its reverse order. If the Stack output and Queue output
 * match, prints the String and increments a palindrome counter.
 */
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

class Assignment1 {
  // Method to return a String with only lowercase letters
  public static String onlyLettersToLowercase(String string) {
    String returnString = string.toLowerCase();
    returnString = returnString.replaceAll("[^a-z]", "");
    return returnString;
  }

  public static void main(String[] args) {
    MarcusStack stack = new MarcusStack();      // Stack for testing palindromes
    MarcusQueue queue = new MarcusQueue();      // Queue for testing palindromes
    MarcusQueue masterQueue = new MarcusQueue();   // For Strings from File

    // Try/catch block for file import and reading
    try {
      File file = new File("./Assignment1/magicitems.txt");
      Scanner read = new Scanner(file);
      while (read.hasNextLine()) {
        MarcusNode item = new MarcusNode(read.nextLine());
        masterQueue.enqueue(item);
      }
      read.close();
    } catch (FileNotFoundException e) {
      System.out.println("Whoops! Couldn't find magicitems.txt");
      e.printStackTrace();
    }

    // Iterate through list
    while (!masterQueue.isEmpty()) {
      String string = masterQueue.dequeue().getItem();  // Original String
      String lowercase = onlyLettersToLowercase(string); // Parsed
      String forwardsString = "";
      String backwardsString = "";

      // Load a stack and a queue with the lowercase parsed String by character
      for (int i = 0; i < lowercase.length(); i++) {
        MarcusNode stackNode = new MarcusNode(lowercase.substring(i, i + 1));
        MarcusNode queueNode = new MarcusNode(lowercase.substring(i, i + 1));
        stack.push(stackNode);
        queue.enqueue(queueNode);
      }

      // Create forwards String from queue and backwards String from stack
      while(!stack.isEmpty()) {
        forwardsString += queue.dequeue().getItem();
        backwardsString += stack.pop().getItem();
      }

      // Compare forwards and backwards for equality, print original if equal
      if (forwardsString.equals(backwardsString)) {
        System.out.println(string);
      }
    }
  }
}
```