CMPT 435 - Fall 2021 - Dr. Labouseur

Assignment One

Shannon Cordoni

Shannon.Cordoni@Marist.edu

September 24, 2021

1 Problem One: Palindromes

1.1 The Data Structure

Given a list of strings our job was to create an algorithm to go through this list and print out the palindromes. To do this we were assigned to read each element of the list into an array, and then ignoring case and spaces, put each letter of each element into a stack or queue. Then by popping and dequeue we could compare each letter forward and backward to see if the word was a palindrome.

The data structures used for this lab involve nodes, arrays, stacks, and queues.

1.2 Stack Class

1.2.1 Description

For each element i in the word array we'll make all the letters lowercase and get rid of spaces. Then we will pass each line $wordarray[i]$ to a new method to then push or pop each letter in $wordarray[i]$ to or from a stack.

```

/**
*
* Assignment 1
* Due Date and Time: 9/24/21 before 12:00am
* Purpose: This class creates the stack
* Input: The user will be inputting a word.
* Output: The program will push each letter of the word into the stack.
* @author Shannon Cordoni
*
*/
public class StackCordoni{

    private NodeCordoni myTop = null;
        private String myData;

        //This method takes in a new word or letter and pushes it into the stack
        public  void push(String newword)
        {
            NodeCordoni oldTop = myTop;
            myTop = new NodeCordoni();
            myTop.setData(newword);
            myTop.setNext(oldTop);
            }//push

        //This method removes a letter from the stack and returns it
        public NodeCordoni pop()
        {
            NodeCordoni answer = null;
            if(!isEmpty())
            {
                answer = myTop;
                myTop = myTop.getNext();
            }//if

            else{
                System.out.println("The stack is empty");
            }//else

            return answer;
    }//pop

        //This method checks whether or not the stack is empty
        public boolean isEmpty()
        {
            boolean empty = false;

            if(myTop == null)
            {
            empty = true;
            }//if

            return empty;
        }//empty
}//Stackcordoni
```

1.2.2 Description of Stack Code

The code above is the code inside the stack class, the good parts of the stack class involve the different methods created, such as *push*, *pop*, and *isEmpty*.

The *push* method operates by taking in a new node representing a letter and adding it into the queue. This is accomplished by first setting a temp variable to equal the current top value in the stack this way we do not loose the current top value when we change the top pointer to point to the new node. Then it creates a new node at the top pointer and sets it's data to be the new string, and it's next value to be the old top of the stack or the temp variable we created before.

The *pop* method creates a temp variable *answer* which is the node we hope to remove from the stack to then be compared to the node from the queue. It then checks to see if the stack is empty, if it is empty then we cannot remove anything from an empty stack. If it is not empty then we can set the temp variable to the top of the stack, this way we don't lose the newest node when we move the top pointer, and then set the new top to be the next node in line and then return the temp variable.

The *isEmpty* method checks whether or not the stack is empty, it does this by looking to see if the head or top of the stack is null, this is because stacks are first in last out, meaning that there will always be a top to the stack due to the fact that when something is popped out the rest of the stack moves up to replace the top node.

1.3 Queue Class

1.3.1 Description

For each element i in the word array we'll make all the letters lowercase and get rid of spaces. Then we will pass each line $wordarray[i]$ to a new method to then enqueue or dequeue each letter in $wordarray[i]$ to or from a queue.

```
/**
 *
 * Assignment 1
 * Due Date and Time: 9/24/21 before 12:00am
 * Purpose: This class creates the stack
 * Input: A word/statement from the input file .
 * Output: The program will push each letter of the word/statement into the Queue.
 * @author Shannon Cordoni
 *
 */
public class QueueCordoni {

        private NodeCordoni myHead;
        private NodeCordoni myTail;

        //This method adds a node to the queue, it does so by adding it
        //to the end of the queue
        public void enqueue(String newword)
        {
                //this sets a temp variable to hold the current tail node
                NodeCordoni oldTail = myTail;

                //this sets the tail to be a new node and its data to be the new string
                myTail = new NodeCordoni();
                myTail.setData(newword);

                //This checks to see if the queue is empty
                //if it is not empty then the old tail is set to now point to the new Node
                if (!isEmpty()){
                        oldTail.setNext(myTail);
                }//if

                //if the queue is empty then all variables are the same because there is nothing
                //in the queue. Then the head and tail pointer would be pointing to the same thing.
                else{
                        myHead = myTail;
                }//else

        }//enqueue

        //This method removes a node from the queue
        public NodeCordoni dequeue()
        {
                //This sets the temp variable to null so that it can be set later.
                NodeCordoni answer = null;

                //If the queue is not empty then it will remove the first node from the queue
                if(!isEmpty())
                {
                        //This sets the temp variable to the first node in the
                        //list and then sets the new head pointer to the second
                        //node in the queue
                        answer = myHead;
                        myHead = myHead.getNext();

                        //if the queue is empty then the head is null
                        if(isEmpty()){
                                myHead = null;
                        }//if
                }//if

                else{
                        System.out.println("The Queue is empty");
                }
                return answer;
        }//dequeue

        //This checks to see if the queue is empty
        public boolean isEmpty()
        {
                boolean empty = false;

                if(myHead == null)
                        {
                        empty = true;
                        }//if
                return empty;
        }//empty

}//QueueCordoni
```

1.3.2 Description of Queue Code

The code above is the code inside the Queue class, the good parts of the Queue class involve the different methods created, such as *enqueue*, *dequeue*, and *isEmpty*.

The *enqueue* method takes in a new node representing a letter and adds it into the queue. It does this by first creating a temp variable so that we do not lose the current tail pointer of the queue. We then set the tail pointer to be a new node and its data to be the new string. It then checks to see if the queue is empty, if it is not empty then it takes the new node and adds it to the queue by setting the temp variable or the old tail to now point to the new node. If the queue is empty then that means that the head, and tail would be pointing to or signifying the same node.

The *dequeue* method creates a temp variable *answer* which is the node we hope to remove from the queue to then be compared to the stack. It then checks to see if the queue is empty, if it is empty then we cannot remove anything from an empty queue. If it is not empty then we can set the temp variable to the head or front of the queue and then set the new head to be the next node in line and return the temp variable.

The *isEmpty* method checks whether or not to see if the queue is empty, it does this by looking to see if the head of the list is null, due to the fact that if there is something in the queue then there is always a head to the queue being that queues are first in first out.

1.4 Node Class

1.4.1 Description

For each element or letter of a word in the array a node was created to represent the letter. This was so that the creation of the stacks and queues could run more smoothly and so that each letter of the string would be linked to the next one.

```java
/**
 *
 * Assignment 1
 * Due Date and Time: 9/24/21 before 12:00am
 * Purpose: This class creates the linked list (Node Class)
 * @author Shannon Cordoni
 *
 */

public class NodeCordoni
{
   /**
    * Instance Variable for word data and node
    */
   private String myData;
   private NodeCordoni myNext;

   /**
    * The default Constructor for NodeCordoni
    */
   public NodeCordoni()
       {
       myData = new String();
       myNext= null;
       }//Node Cordoni

   /**
    * The full constructor for NodeCordoni
    * @param newData the incoming data of the item
    */
   public NodeCordoni(String newData)
       {
       myData = newData;
       myNext = null;
       }//NodeCordoni

   /**
    * the setter for the item data
    * @param newData the incoming data of the item
    */
   public void setData(String newData)
       {myData = newData;} //set data

   /**
    * The getter for the item data
    * @return the incoming data of the item
    */
   public String getData()
       {return myData;}//get data

   /**
    * The setter for the node
    * @param NewNext the incoming node data
    */
   public void setNext(NodeCordoni newNext)
       {myNext = newNext;}//set Node

   /**
    * the getter for the node
    * @return the incoming node data
    */
   public NodeCordoni getNext()
       { return myNext;}//get node

}//NodeCordoni
```

1.4.2 Description of Node Code

This code for the Node Class was created by in class lessons but also previous knowledge from Software Development 1. Using the same set up each node was created so that it consisted of a string and a myNext linking each node to the next. Getters and setters were created for both the nodes themselves and the data inside of them so that we would be able to call *node.getNext*(), *node.setNext*(), *node.getData*(), and *node.setData*() in the stack, queue, and main class to make working the stack and queue run more smoothly.

1.5 Main Class

1.5.1 Description

With the *magicitems.txt* file input each line of the file was read into an array. This array was then passed into a method that took each index of the array and took away the spaces and made all the letters the same case. Taking these new found singular words they were then put into another array and passed letter by letter into the stack or queue to then be popped/enqueued and compared.

```

/**
 *
 * Assignment 1
 * Due Date and Time: 9/24/21 before 12:00am
 * Purpose: To see if a word is a palindrome
 * Input: The user will be inputting a file containing a list of words/statements .
 * Output: The program will output the palindromes.
 * @author Shannon Cordoni
 *
 */

import java.io.File;
import java.io.FileNotFoundException;
import java.util.InputMismatchException;
import java.util.Scanner;

public class MainCordoni {

        //Declare keyboard
        static Scanner keyboard = new Scanner(System.in);

        public static void main(String[] args) {

                //Declare and initialize variables
                StackCordoni theStack = new StackCordoni();
                String filename;
                String line;
                String statement;
                String noSpaceStatement;
                NodeCordoni word = null;
                QueueCordoni theQueue = new QueueCordoni();
                String[] wordarray = new String[666];

```

```
            //create new file object
            File myFile = new File("magicitems.txt");

            try
            {
                    //create scanner
                    Scanner input = new Scanner(myFile);
                    line = null;

                    int i = 0;

                    //while there are more lines in the file it inputs them into
                    //a word array
                while(input.hasNext())
                 {
                            //Input into array
                            wordarray[i] = input.nextLine();
                            i++;
                 }//while

                    input.close();

            }//try

            //error for file not found
            catch(FileNotFoundException ex)
        {
          System.out.println("Failed to find file: " + myFile.getAbsolutePath());
        }//catch

            //Error in case of a null pointer exception
        catch(NullPointerException ex)
        {
            System.out.println("Null pointer exception.");
            System.out.println(ex.getMessage());
        }//catch

            //General error message
        catch(Exception ex)
        {
            System.out.println("Something went wrong");
            ex.printStackTrace();
        }//catch

            //Passes word array into the palindrome function to remove spaces and change
            //letter case so that letters can be passed into stack and queue
            palindrome(wordarray);

            }//main


            //this method takes in one element of the array and make all letters
            //the same case and gets rid of spaces
            public static void palindrome( String[] wordarray)
            {

                    //System.out.println(wordarray);
                    String line = "none";
                    String statement = "none";
                    String noSpaceStatement;


                    for(int i = 0; i<wordarray.length; i++){

                            //creation of stack and queue
```

```
                                StackCordoni theStack = new StackCordoni();
                                QueueCordoni theQueue = new QueueCordoni();

                                //takes each index of the array and inputs it into a variable
                                line = wordarray[i];

                                //Takes each letter of the string and makes it lowercase
                                statement = line.toLowerCase();

                                //Takes the string and removes spaces between words
                                noSpaceStatement = statement.replaceAll("\\s", "");

                                //Takes each letter of the string and puts them into an array
                                String[] charArray = noSpaceStatement.split("");

                                //Pushes each letter in the array into the stack
                                pushStack(charArray, theStack);

                                //Enqueues each letter in the array into the queue
                                enqueueQueue(charArray, theQueue);

                                //compares each letter from the stack and queue
                                compare(line, theStack, theQueue);

                        }//for
                }//palindrome

                //This method pushes each letter of the array into the stack
                public static void pushStack(String[] chararray, StackCordoni stack){

                        //goes through the array to push each letter
                        for(int i = 0; i < chararray.length; i++){
                                stack.push(chararray[i]);
                        }//for

                }//pushStack

                public static void enqueueQueue(String[] chararray, QueueCordoni queue){

                        //goes through the array to enqueue each letter
                        for(int i = 0; i < chararray.length; i++){
                                queue.enqueue(chararray[i]);
                        }//for

                }//enqueueQueue

                //This method pops and dequeues a letter from the stack and
                //queue respectively. Then it compares each letter to see
                //if the word is a palindrome
                public static void compare(String chararray, StackCordoni stack,
                                            QueueCordoni queue){

                        NodeCordoni popVal;
                        NodeCordoni dequeueVal;
                        String valPop;
                        String valDequeue;

                        //pop from the queue and store letter in a variable
                        popVal = stack.pop();
                        valPop = popVal.getData();

                        //dequeue from the queue and store letter in a variable
                        dequeueVal = queue.dequeue();
                        valDequeue = dequeueVal.getData();

```

```
            //Looks to see if the letters are the same
            if(valPop.equals(valDequeue)){

                /*
                *while the letters are equal we go through the rest of the
                *stack and queue until we reach letters that are not the
                *same or the stack is empty (since we are putting the same
                *amount of letters into the stack/queue we only have to see
                *if one of them is empty, since i pushed before enqueue I
                *used the stack)
                */

                while((valPop.equals(valDequeue))&&(!(stack.isEmpty()))){
                    popVal = stack.pop();
                    valPop = popVal.getData();
                    dequeueVal = queue.dequeue();
                    valDequeue = dequeueVal.getData();
                }//while

                /*
                *If we reach the end of the stack and all of the letters are
                *the same then the word is a palindrome and we print it out
                */
                if(stack.isEmpty()){
                    System.out.println(chararray);
                }//if
            }//if
        }//compare
}//MainCordoni
```

1.5.2 Description of Main Code

The main class above consists of different methods to help determine if a string is a palindrome. The good parts of the code first include the file sections. While reading the file it goes through an inputs each line into a word array. Then to keep everything out of the main method, different methods were used to help organize the code better. These methods include the *palindrome*, *pushStack*, *enqueueQueue*, and *compare* methods.

The *palindrome* method takes in the word array and then for each index in the array it creates a stack and queue, and inputs the index into a variable(*line*). This variable then has the spaces removed from it and all of the letters changed to lower case. Each letter of the variable is then put into an array (*chararray*) so that it can be passed into the queue and stack letter by letter.

The *pushStack* method and the *enqueueQueue* method each take in this *chararray* and their stack or queue respectively pushes or enqueues each letter.

The *compare* method then pops and dequeues from the stack and queue respectfully and then puts the data from these nodes into a temp variable($popVal$ or $dequeueVal$). The data from temp variables are then compared, if they are the same then the while loop will continue until there is a mismatch letter, meaning there is no use to check the rest of the word, or it reaches the end of the stack. This is because once the stack and the queue are full, both contain the same amount of letters since the same data was put inside them. Since I kept using the stack first, I set the while loop to terminate if the letters did not match and if the end of the stack was reached. If the end of the word or stack is reached then we have a palindrome, if not, then we move back into the *palindrome* method and onto the next word or index in the *wordarray*.