CMPT 435 - Fall 2022 - Dr. Labouseur

Assignment One – Palindromes

Genevieve Anderson
Genevieve.anderson1@marist.edu

September 23, 2022

1 Node Class

The node class is where I build the framework for my linked list. The linked list allows for each phrase of the magic items file to be put in their own node. Inside the Node class I am building the framework for each node, the linked list is made up of several nodes.

I declared that inside each node there would be a name, and a next pointer [lines 5, 6] which will be initialized in the constructor. In the constructor [line 12] I initialized the name to "n and the next pointer to null to begin with. The name will eventually contain each phrase or character of the magic items. The pointer is what connects the nodes in the linked list.

Within my Node class I have two "getters", and two "setters". The "getters" [lines 18, 22] will provide information for the node, and the "setters" [lines 28, 32] will actually set the name and the pointer in the node. I built "getName()" and "getNext()" which get the name variable, and the pointer for the node. Then I built "setName()" and "setNode()". These are specifically meant for setting the name in the node to a string value, and setting the pointer to the next node.

Lastly, in the Node class I created a "toString()" method [line 37] in order for the program to set the name of the node to the result. This is so that when it comes time to print the magic items phrases from the node it will return the actual phrase rather than the object identifier.

```
1 //This is the class that builds the framework for the linekd list
2 //Building the framework for each node. the linked list is going to
      be made up of several nodes.
3
4 public class Node { //Creating the node for the linked list
```

```
5       String name = ""; //Declaring and initializing the name inside
        the node
6       Node next = null; //Declaring and initializing the node pointer
7
8
9       //Node constructor
10      //Uppercase N is referring to the node class
11      //Lowercase n is referring to the actual node with the
        information in it (could be called anything)
12      public Node(String n, Node node) { //The first parameter is for
         the information the node is holding, the second parameter is
        for the pointer
13          this.name = n; //Initializing the name
14          this.next = null; //Initializing the pointer
15      }
16
17      //I will now build two getters: information for the node,
        information for the pointer
18      public String getName() { //Getting the name variable from the
        node, returns a string
19          return name; //Returns the information from the variable
        name
20      }
21
22      public Node getNext() { //Returning a node
23          return next; //Next because we are calling the pointer next
        , returns the node from the variable next
24      }
25
26      //I will now build two setters: setting the name, and setting
        the pointer
27      //Void because it isnt going to return anything
28      public void setName(String n) { //Parameters always go inside
        parenthesis, I am only updating the name value, so we only need
         one parameter
29          name = n;
30      }
31
32      public void setNode(Node m) { //"m" is the node that we are
        going to set next equal to, for the pointer
33          next = m; //I am using m so that the pointer is not null.
        the pointer will not be null until the end of the linked list.
34      }
35
36      //toString so that the program prints what is actually inside
        the node rather than the object identifier
37      public String toString() {
38          String result = name; //setting a string equal to what is
        inside the node
39          return result;
40      }
41  }
```

2 Stack Class

The stack class is where I build the framework for the stack that will be used in the main program. The stack class has three functions, push, pop, and isEmpty, which will help to determine whether or not a word is a palindrome.

I declared that there would be a top to the stack [line 4], which will be initialized in the constructor. In the constructor [line 7] I initialized that the stack is always empty to begin by stating that the top is null.

I created a push function [line 17] which adds an element to the top of the stack. In this function, there will be a new node added which will take the place of the node that was previously the top of the function. In my push function the pointer that was pointing to the previous top of the stack will now point to the new top of the stack.

I created a pop function [line 23] which removes an element from the top of the stack. When the top is removed, it will be stored into a variable "prevTop" [line 24] so that the node is not completely lost after being removed. In my pop function, the pointer that was pointing to the top of the stack moves so that it is pointing to the next node down, which will be the new top of the stack.

Lastly in my stack class I have an isEmpty function [line 33]. This simple function helps to check whether or not the stack is empty.

```
1  //Building framework for the stack
2  public class Stack {
3      //I am declaring that there is a head in the stack, the head
       will be initialized in the constructor
4      Node top; //Uppercase Node is referring to the Node class I
       made prior
5
6      //Stack constructor
7      public Stack() { //This is a default constructor since there
       are no parameters
8          top = null; //Initiailizing that the stack is empty to
       begin
9      }
10
11     //I will build a function for push, pop, and seeing if the
       stack is empty
12     //The stack is made up of nodes, the node is made up of
       information and information and a pointer
13
14     //Function for push
15     //Nothing will be returned in this function since we are only
       adding a node to the stack
16     //For the push function, we needed to return a node becuase
       that is how we will check whether it is a palindrome
17     public void push(Node newNode) { //Parameter is Node because
       there is a new node coming in to the top of the stack, and we
       must take it from the linked list
18         newNode.next = top; //Telling the new node to point to the
       next node, so that the new node can soon become the new top
19         top = newNode; //Setting the new node to the top of the
       stack
```

```
20      }
21
22      //Function for pop
23      public Node pop() { //No parameter needed because the function
        is only using information from the class
24          Node prevTop = null; //If there is nothing in the stack,
        then we will return null, prevTop will save the old node so
        that we don't completely lose it
25          if(isEmpty() == false) {
26              prevTop = new Node(top.name, top.next); //I created a
        new node in order to make sure there were not any extra
        pointers pointing to something other than the next node in the
        stack
27              top = top.getNext(); //Setting the new head to the next
         node in the stack, since it is now on the top
28          }
29          return prevTop; //Returning the node that was taken off
30      }
31
32       //Function for checking if the stack is empty
33       public Boolean isEmpty() {
34          if(top == null) {
35              return true;
36          }
37          else {
38              return false;
39          }
40      }
41  }
```

3 Queue Class

The Queue class is where I build the framework for the Queue's that are used in the main program. The Queue class has three functions, queue, dequeue, and isEmpty, which will help determine whether or not a word is a palindrome.

I declared that there would be a head and tail for the queue [lines 4, 5], which will be initialized in the constructor. In the constructor [line 8] I initialized that the queue is always empty to begin by stating that the head and tail are null. The queue has a head and a tail, meanwhile in the stack we only indicate the top. This is because we will need to add things to the back of the queue, and we don't add things to the bottom of the stack. Initializing a tail makes it so that the program does not have to calculate how long the queue actually is, but rather can just add a node to the back by putting it behind the tail.

I created an enqueue function [line 16] which will add elements to the back of the queue. If a new node is added when the queue is empty, then the new node will become both the head and the tail [lines 18-20]. Otherwise, when a new node is added to the back of the queue and will become the new tail. The previous tail will point to the new tail [lines 22-24].

I created a dequeue function [line 29] which will remove elements from the front of the queue. When removing the node from the queue, we don't want to completely lose it so I stored it in "oldHead" [line 30]. When the node is removed, the node that was origianlly next in line, will become the new head [line 31]. To make sure that the program doesn't break, I ensured that if the head was null, then the tail should also be null [lines 32, 33]. This is because there must always be both a head and a tail in a properly functioning queue.

Lastly in my queue class I have an isEmpty function [line 39]. This simple function helps to check whether or not the queue is empty.

```
1  //Building the framework for the queue
2  public class Queue {
3      //I am declaring the head and tail inside the queue, then
       initialize them in the constructor
4      Node head; //Queue needs both head and tail, whereas the stack
       only needed a head
5      Node tail;
6
7      //Queue constructor
8      public Queue() {
9          head = null; //Initializing the head
10         tail = null; //Initializing the tail
11     }
12
13     //I will build a function for pop, push, and seeing if the
       stack is empty
14     //The stack is made up of nodes, the node is made up of
       information and information and a pointer
15     //Function for enqueue
16     public void enqueue(Node y) { //Parameter is a node because we
       will be a new node being enqueued in the queue
```

```
17          Node node = new Node(y.getName(), y.getNext()); //I created
     a new node in order to make sure there were not any extra
    pointers pointing to something other than the next node in the
    queue
18          if (head == null) { //If the queue was empty, then the if
    loop will make it so that the incoming node to the queue will
    become both the head and the tail
19              tail = node; //Setting the incoming node to the tail
20              head = tail;  //Setting the head to the tail
21          }
22          else { //The else statement is for if there are already
    nodes in the queue
23              tail.setNode(node); //Opposite of the stack because in
    the stack we are bringining in a new node which must point to
    the old head, but in the queue the tail is already there and
    needs to be pointed to the incoming node
24              tail = node; //Setting the incoming node to the tail
25          }
26      }
27
28      //Function for dequeue
29      public Node dequeue() { //No parameter needed because the
    function is only using information from the class
30          Node oldHead = head; //Creating old head makes it so that
    we don't completly lose the node that is getting removed from
    the queue
31          head = head.getNext(); //The new head is goig to be the
    head that was originally next in line
32          if (head == null) { //If the head is null, then the tail
    should also be null
33              tail = null; //Doing this because otherwise, the tail
    would still be pointing at the node we are returning
34          }
35          return oldHead; //Return the old head to help us verify
    whether or not it is a palindrome in the main program
36      }
37
38      //Function for checking is the queue is empty
39      public Boolean isEmpty() {
40          if (head == null) {
41              return true;
42          }
43          else {
44              return false;
45          }
46      }
47  }
```

4 Main Program

The main program is where I will test if the list of 666 magic items contains any palindromes!

Prior to testing whether or not the magic items are palindromes, I must put the entire list of magic items into a queue so that they can eventually be tested one by one [line 14]. To actually upload the file, I created a try and catch statement [lines 15-29]. This is also where I changed all of the letters to uppercase and removed all spaces from the magic items. I put all of the phrases into their own node, because a linked list is made up of nodes [line 24]. I then enqueued each node into the queue to prepare for testing of palindromes [line 25].

Next, I created a while loop [line 33] to ensure that when the program is running, the entire list of magic items will be accounted for, not just one. The loop runs while the queue is not empty so that we know that we can continue to take from it. Inside this loop, the phrases are each passed into "palindromeCheck()" which is the function where we finally compare whether or not the magic item is a palindrome! After going through the checker and being determined whether or not the magic item is a palindrome, the loop will print out the magic item only if it is a palindrome.

Now, I will go into further detail on the "palindromeCheck()" function [line 43]. I initialized the checker to true [line 44], and while checking for palindromes the checker will change to false if the magic item is not a palindrome. Inside this function, I also converted the nodes into strings [line 45] so that I am able to separate the magic items character by character to check whether or not they are a palindrome. I created a new queue and a new stack [lines 46, 47] which will be the key to checking the magic items for palindromes. Inside of a for loop, I then used the node that was converted into a string a made a sub-string of each of the characters [line 49]. I then put the single characters each into their own node [line 50] because my stack and queue classes only accept nodes. Then, I pushed the single characters into the stack and enqueued them into the queue [lines 51, 52].

At the end "palindromeCheck()" function, I created a while loop which simultaneously dequeues and pops the single characters from the queue and stack. Then, still inside the loop, the single characters are compared against each other [line 58]. The checker is set to false if the word is not a palindrome [line 59].

I included a note that I tested the program on a blank file as well. This was successful and did not break the program. I also included my old code in comment form from when I tested my stack, queue, and node class on a simple example of "abc", to see if "cba" would be returned.

```
1 import java.io.File;
2 import java.io.FileNotFoundException;
3 import java.util.Scanner;
4
5 //Thought process prior to writing main program
```

```
6       //Put all the magic items into their own queue so that we can
        access them in the program
7       //Simultaneously push charachters into stack, queue charachters
         into queue
8       //Simultaneously pop charachters out of stack, dequeue
        charachters out of queue
9       //While popping and dequeueing, compare results to see whether
        or not the word is a palindrome
10
11  public class  Main {
12      public static void main (String[] args) { //Write this
        everytime in a main program
13          //Create a new queue, declare and initialize for holding
        all the magic items
14          Queue magicItems = new Queue(); //Making a variable
        magicItems of the queue data type
15          try { //Trying to find the file
16              File file = new File("magicitems.txt");
17              Scanner sc = new Scanner(file);
18
19              while (sc.hasNextLine()) {
20                  String item = sc.nextLine();
21                  item = item.toUpperCase(); //Changing everything to
         upper case
22                  item = item.replaceAll("\\s+",""); //Replacing the
        spaces with no space
23                  //We must put the string of magic items in a node
        because the queue method takes nodes not strings
24                  Node magicItemsNode = new Node(item, null); //
        Parameters are the string of magic items, then the null pointer
         - null because this node is only a list of the magic items.
        Making a new node to put the phrase into.
25                  magicItems.enqueue(magicItemsNode); //Calling
        enqueue from Queue class, the parameter is because we must
        indicate that each phrase is on its own
26              }
27          }
28         catch (FileNotFoundException e) { //If we cant find the file
29              e.printStackTrace();
30          }
31
32         //While loop so the program accounts for the entire file of
        magic items - not just one phrase
33         while (magicItems.isEmpty() == false) { //While the queue is
         not empty, so that we know that we can take charachters from
        it
34              Node phraseNode = magicItems.dequeue(); //Creating a
        new node to put the dequeed magic items in
35              Boolean palindromeBoolean = palindromeCheck(phraseNode)
        ; //Passing the phrase into the check method to see if its a
        palindrome and returns the result of whether or not it was a
        palindrome
36              if (palindromeBoolean == true) {
37                  System.out.println(phraseNode.getName()); //If it
        is a palindrome, then we will print it
38              }
39          }
```

```
40      }
41
42      //Function to check whether or not the phrase is a palindrome
43      public static Boolean palindromeCheck(Node y) { //Parameter is
        a node, because we are using dequeue items which are in a Node
44          Boolean palindromeCheck = true; //Initializing the checker
        to true, while checking for palindromes it will change based on
         whether or not it is a palindrome
45          String phraseString = y.getName(); //Putting the node that
        contains the dequeued items into a string so that we can
        seperate it charachter by charachter
46          Stack palindromeStack = new Stack(); //intializing the
        stack to check whether its palindrome
47          Queue palindromeQueue = new Queue(); //initializing the
        queue to check whether its a palindrom
48          for (int i = 0; i < phraseString.length(); i++) { //int i =
         0 so that it starts at the beginning of the string, i < length
         to make sure it doesn't go past one phrase at a time, i++ so
        that increments by 1
49              String phraseCharachter = phraseString.substring(i, i
        +1); //String that contains the charachters of each phrase, i
        +1
50              Node charachterNode = new Node(phraseCharachter, null);
         //Creating a new node with only the charachter
51              palindromeStack.push(charachterNode); //Pushing the
        single charachter node into the stack
52              palindromeQueue.enqueue(charachterNode); //Enqueueing
        the single charachter node into the queue
53          }
54
55          while (palindromeStack.isEmpty() == false) { //While the
        stack with the charachters is not empty, continue to check the
        word for whether or not it is a palindrome
56              Node checkQueue = palindromeQueue.dequeue(); //
        Simultaneously dequeuing and popping from the stack and queue
        to check whether its a palindrome
57              Node checkStack = palindromeStack.pop();
58              if (!(checkStack.getName().equals(checkQueue.getName())
        )) { //.equals function for comparing strings
59                  palindromeCheck = false; //Only one equal sign
        because we are actually setting the checker to false
60              }
61          }
62          return palindromeCheck; //Providing the result of the
        comparison
63      }
64  }
65
66  //Note: tested if program worked on blank.txt file
67
68  //Old code from when I was checking if my stack queue and node
        classes worked
69  /*Node nodeOne = new Node("a", null);
70          Node nodeTwo = new Node("b", null);
71          Node nodeThree = new Node("c", null);
72
73          Node nodeFour = new Node("a", null);
```

```
74          Node nodeFive = new Node("b", null);
75          Node nodeSix = new Node("c", null);
76
77
78          Stack stackOne = new Stack(); //initializing a stack
79          stackOne.push(nodeOne);
80          stackOne.push(nodeTwo);
81          stackOne.push(nodeThree);
82
83          while (stackOne.isEmpty() == false) {
84              Node popped = stackOne.pop(); //Telling the system we
        are popping a node
85              System.out.println(popped); //Printing what was in the
        node that was popped
86          }
87
88          Queue queueOne = new Queue(); //initializing a queue
89          queueOne.enqueue(nodeFour);
90          queueOne.enqueue(nodeFive);
91          queueOne.enqueue(nodeSix);
92
93          while (queueOne.isEmpty() == false) {
94              Node queued = queueOne.dequeue();
95              System.out.println(queued);
96          } */
97
98  //Miscellaneous comments
99      //Constructors turn the frameowork from the different classes
        into actual objects
100     //Two equal signs if boolean
101     //&& is how you say and (Boolean)
102     //When biulding a function, after "public", always say the data
         type we are going to return (Node, Boolean, String, etc.)
103     //global variable - applies to the entire class
104     //regular variable - only applies to the specific functin its
        within
```

5 LaTeX Source Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% CMPT 435
% Lab Zero
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Short Sectioned Assignment
% LaTeX Template
% Version 1.0 (5/5/12)
%
% This template has been downloaded from: http://www.LaTeXTemplates.com
% Original author: % Frits Wenneker (http://www.howtotex.com)
% License: CC BY-NC-SA 3.0 (http://creativecommons.org/licenses/by-nc-sa/3.0/)
% Modified by Alan G. Labouseur  - alan@labouseur.com
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%
    ----------------------------------------------------------------------------------------

% PACKAGES AND OTHER DOCUMENT CONFIGURATIONS
%
    ----------------------------------------------------------------------------------------


\documentclass[letterpaper, 10pt]{article}

\usepackage[english]{babel} % English language/hyphenation
\usepackage{graphicx}
\usepackage[lined,linesnumbered,commentsnumbered]{algorithm2e}
\usepackage{listings}
\usepackage{fancyhdr} % Custom headers and footers
\pagestyle{fancyplain} % Makes all pages in the document conform to the custom headers and footers
\usepackage{lastpage}
\usepackage{url}

\fancyhead{} % No page header - if you want one, create it in the same way as the footers below
\fancyfoot[L]{} % Empty left footer
\fancyfoot[C]{page \thepage\ of \pageref{LastPage}} % Page numbering for center footer
\fancyfoot[R]{}

\renewcommand{\headrulewidth}{0pt} % Remove header underlines
\renewcommand{\footrulewidth}{0pt} % Remove footer underlines
\setlength{\headheight}{13.6pt} % Customize the height of the header

\usepackage{xcolor}

```

```
\definecolor{codegreen}{rgb}{0,0.6,0}
\definecolor{codegray}{rgb}{0.5,0.5,0.5}
\definecolor{codepurple}{rgb}{0.58,0,0.82}
\definecolor{backcolour}{rgb}{0.95,0.95,0.92}

\lstdefinestyle{mystyle}{
    backgroundcolor=\color{backcolour},
    commentstyle=\color{codegreen},
    keywordstyle=\color{magenta},
    numberstyle=\tiny\color{codegray},
    stringstyle=\color{codepurple},
    basicstyle=\ttfamily\footnotesize,
    breakatwhitespace=false,
    breaklines=true,
    captionpos=b,
    keepspaces=true,
    numbers=left,
    numbersep=5pt,
    showspaces=false,
    showstringspaces=false,
    showtabs=false,
    tabsize=2
}

\lstset{style=mystyle}
%
    ----------------------------------------------------------------------------------------

% TITLE SECTION
%
    ----------------------------------------------------------------------------------------


\newcommand{\horrule}[1]{\rule{\linewidth}{#1}} % Create horizontal
     rule command with 1 argument of height

\title{
   \normalfont \normalsize
   \textsc{CMPT 435 - Fall 2022 - Dr. Labouseur} \\[10pt] % Header
    stuff.
   \horrule{0.5pt} \\[0.25cm]   % Top horizontal rule
   \huge Assignment One -- Palindromes \\           % Assignment
    title
   \horrule{0.5pt} \\[0.25cm]   % Bottom horizontal rule
}

\author{Genevieve Anderson \\ \normalsize Genevieve.anderson1
    @marist.edu}

\date{\normalsize\today}  % Today's date.

\begin{document}

\maketitle % Print the title

%
    ----------------------------------------------------------------------------------------
```

```
%   CONTENT SECTION
%
    ------------------------------------------------------------------------------------------------------------


% - -- -  - -- -  - -- -  -

\section{Node Class}

\noindent

    \hspace*{1.5em} The node class is where I build the framework
    for my linked list. The linked list allows for each phrase of
    the magic items file to be put in their own node. Inside the
    Node class I am building the framework for each node, the
    linked list is made up of several nodes.  \\
    \hspace*{1.5em} I declared that inside each node there would be
     a name, and a next pointer [lines 5, 6] which will be
    initialized in the constructor. In the constructor [line 12] I
    initialized the name to "n and the next pointer to null to
    begin with. The name will eventually contain each phrase or
    character of the magic items. The pointer is what connects the
    nodes in the linked list. \\
    \hspace*{1.5em} Within my Node class I have two "getters", and
    two "setters". The "getters" [lines 18, 22] will provide
    information for the node, and the "setters" [lines 28, 32]
    will actually set the name and the pointer in the node. I built
     "getName()" and "getNext()" which get the name variable, and
    the pointer for the node. Then I built "setName()" and "setNode
    ()". These are specifically meant for setting the name in the
    node to a string value, and setting the pointer to the next
    node. \\
    \hspace*{1.5em} Lastly, in the Node class I created a "toString
    ()" method [line 37] in order for the program to set the name
    of the node to the result. This is so that when it comes time
    to print the magic items phrases from the node it will return
    the actual phrase rather than the object identifier. \\

\begin{lstlisting}[language = Java]
//This is the class that builds the framework for the linekd list
//Building the framework for each node. the linked list is going to
     be made up of several nodes.

public class Node { //Creating the node for the linked list
    String name = ""; //Declaring and initializing the name inside
    the node
    Node next = null; //Declaring and initializing the node pointer


    //Node constructor
    //Uppercase N is referring to the node class
    //Lowercase n is referring to the actual node with the
    information in it (could be called anything)
    public Node(String n, Node node) { //The first parameter is for
     the information the node is holding, the second parameter is
    for the pointer
```

```
121         this.name = n; //Initializing the name
122         this.next = null; //Initializing the pointer
123     }
124 
125     //I will now build two getters: information for the node,
    information for the pointer
126     public String getName() { //Getting the name variable from the
    node, returns a string
127         return name; //Returns the information from the variable
    name
128     }
129 
130     public Node getNext() { //Returning a node
131         return next; //Next because we are calling the pointer next
    , returns the node from the variable next
132     }
133 
134     //I will now build two setters: setting the name, and setting
    the pointer
135     //Void because it isnt going to return anything
136     public void setName(String n) { //Parameters always go inside
    parenthesis, I am only updating the name value, so we only need
     one parameter
137         name = n;
138     }
139 
140     public void setNode(Node m) { //"m" is the node that we are
    going to set next equal to, for the pointer
141         next = m; //I am using m so that the pointer is not null.
    the pointer will not be null until the end of the linked list.
142     }
143 
144     //toString so that the program prints what is actually inside
    the node rather than the object identifier
145     public String toString() {
146         String result = name; //setting a string equal to what is
    inside the node
147         return result;
148     }
149 }
150 \end{lstlisting}
151 
152 
153 
154 \newpage
155 
156 \section{Stack Class}
157      \hspace*{1.5em} The stack class is where I build the framework
     for the stack that will be used in the main program. The stack
     class has three functions, push, pop, and isEmpty, which will
    help to determine whether or not a word is a palindrome. \\
158      \hspace*{1.5em} I declared that there would be a top to the
    stack [line 4], which will be initialized in the constructor.
    In the constructor [line 7] I initialized that the stack is
    always empty to begin by stating that the top is null. \\
159      \hspace*{1.5em} I created a push function [line 17] which adds
     an element to the top of the stack. In this function, there
```

```
    will be a new node added which will take the place of the node
    that was previously the top of the function. In my push
    function the pointer that was pointing to the previous top of
    the stack will now point to the new top of the stack. \\
160  \hspace*{1.5em} I created a pop function [line 23] which
    removes an element from the top of the stack. When the top is
    removed, it will be stored into a variable "prevTop" [line 24]
    so that the node is not completely lost after being removed. In
     my pop function, the pointer that was pointing to the top of
    the stack moves so that it is pointing to the next node down,
    which will be the new top of the stack. \\
161  \hspace*{1.5em} Lastly in my stack class I have an isEmpty
    function [line 33]. This simple function helps to check whether
     or not the stack is empty. \\
162
163 \begin{lstlisting}[language = Java]
164 //Building framework for the stack
165 public class Stack {
166     //I am declaring that there is a head in the stack, the head
    will be initialized in the constructor
167     Node top; //Uppercase Node is referring to the Node class I
    made prior
168
169     //Stack constructor
170     public Stack() { //This is a default constructor since there
    are no parameters
171         top = null; //Initiailizing that the stack is empty to
    begin
172     }
173
174     //I will build a function for push, pop, and seeing if the
    stack is empty
175     //The stack is made up of nodes, the node is made up of
    information and information and a pointer
176
177     //Function for push
178     //Nothing will be returned in this function since we are only
    adding a node to the stack
179     //For the push function, we needed to return a node becuase
    that is how we will check whether it is a palindrome
180     public void push(Node newNode) { //Parameter is Node because
    there is a new node coming in to the top of the stack, and we
    must take it from the linked list
181         newNode.next = top; //Telling the new node to point to the
    next node, so that the new node can soon become the new top
182         top = newNode; //Setting the new node to the top of the
    stack
183     }
184
185     //Function for pop
186     public Node pop() { //No parameter needed because the function
    is only using information from the class
187         Node prevTop = null; //If there is nothing in the stack,
    then we will return null, prevTop will save the old node so
    that we don't completely lose it
188         if(isEmpty() == false) {
189             prevTop = new Node(top.name, top.next); //I created a
```

```
    new node in order to make sure there were not any extra
    pointers pointing to something other than the next node in the
    stack
190             top = top.getNext(); //Setting the new head to the next
     node in the stack, since it is now on the top
191         }
192         return prevTop; //Returning the node that was taken off
193     }
194 
195      //Function for checking if the stack is empty
196      public Boolean isEmpty() {
197         if(top == null) {
198             return true;
199         }
200         else {
201             return false;
202         }
203     }
204 }
205 \end{lstlisting}
206 \newpage
207 
208 \section{Queue Class}
209 
210     \hspace*{1.5em} The Queue class is where I build the framework
    for the Queue's that are used in the main program. The Queue
    class has three functions, queue, dequeue, and isEmpty, which
    will help determine whether or not a word is a palindrome. \\
211     \hspace*{1.5em} I declared that there would be a head and tail
    for the queue [lines 4, 5], which will be initialized in the
    constructor. In the constructor [line 8] I initialized that the
     queue is always empty to begin by stating that the head and
    tail are null. The queue has a head and a tail, meanwhile in
    the stack we only indicate the top. This is because we will
    need to add things to the back of the queue, and we don't add
    things to the bottom of the stack. Initializing a tail makes it
     so that the program does not have to calculate how long the
    queue actually is, but rather can just add a node to the back
    by putting it behind the tail. \\
212     \hspace*{1.5em} I created an enqueue function [line 16] which
    will add elements to the back of the queue. If a new node is
    added when the queue is empty, then the new node will become
    both the head and the tail [lines 18-20]. Otherwise, when a new
     node is added to the back of the queue and will become the new
     tail. The previous tail will point to the new tail [lines
    22-24]. \\
213     \hspace*{1.5em} I created a dequeue function [line 29] which
    will remove elements from the front of the queue. When removing
     the node from the queue, we don't want to completely lose it
    so I stored it in "oldHead" [line 30]. When the node is removed
    , the node that was origianlly next in line, will become the
    new head [line 31]. To make sure that the program doesn't break
    , I ensured that if the head was null, then the tail should
    also be null [lines 32, 33]. This is because there must always
    be both a head and a tail in a properly functioning queue. \\
214     \hspace*{1.5em} Lastly in my queue class I have an isEmpty
    function [line 39]. This simple function helps to check whether
```

```
     or not the queue is empty. \\
215
216 \begin{lstlisting}[language = Java]
217 //Building the framework for the queue
218 public class Queue {
219     //I am declaring the head and tail inside the queue, then
    initialize them in the constructor
220     Node head; //Queue needs both head and tail, whereas the stack
    only needed a head
221     Node tail;
222
223     //Queue constructor
224     public Queue() {
225         head = null; //Initializing the head
226         tail = null; //Initializing the tail
227     }
228
229     //I will build a function for pop, push, and seeing if the
    stack is empty
230     //The stack is made up of nodes, the node is made up of
    information and information and a pointer
231     //Function for enqueue
232     public void enqueue(Node y) { //Parameter is a node because we
    will be a new node being enqueued in the queue
233         Node node = new Node(y.getName(), y.getNext()); //I created
     a new node in order to make sure there were not any extra
    pointers pointing to something other than the next node in the
    queue
234         if (head == null) { //If the queue was empty, then the if
    loop will make it so that the incoming node to the queue will
    become both the head and the tail
235             tail = node; //Setting the incoming node to the tail
236             head = tail;  //Setting the head to the tail
237         }
238         else { //The else statement is for if there are already
    nodes in the queue
239             tail.setNode(node); //Opposite of the stack because in
    the stack we are bringining in a new node which must point to
    the old head, but in the queue the tail is already there and
    needs to be pointed to the incoming node
240             tail = node; //Setting the incoming node to the tail
241         }
242     }
243
244     //Function for dequeue
245     public Node dequeue() { //No parameter needed because the
    function is only using information from the class
246         Node oldHead = head; //Creating old head makes it so that
    we don't completly lose the node that is getting removed from
    the queue
247         head = head.getNext(); //The new head is goig to be the
    head that was originally next in line
248         if (head == null) { //If the head is null, then the tail
    should also be null
249             tail = null; //Doing this because otherwise, the tail
    would still be pointing at the node we are returning
250         }
```

```
251         return oldHead; //Return the old head to help us verify
    whether or not it is a palindrome in the main program
252     }
253
254     //Function for checking is the queue is empty
255     public Boolean isEmpty() {
256         if (head == null) {
257             return true;
258         }
259         else {
260             return false;
261         }
262     }
263 }
264 \end{lstlisting}
265
266 \newpage
267
268 \section{Main Program}
269
270     \hspace*{1.5em} The main program is where I will test if the
    list of $666$ magic items contains any palindromes! \\
271     \hspace*{1.5em} Prior to testing whether or not the magic items
     are palindromes, I must put the entire list of magic items
    into a queue so that they can eventually be tested one by one [
    line 14]. To actually upload the file, I created a try and
    catch statement [lines 15-29]. This is also where I changed all
     of the letters to uppercase and removed all spaces from the
    magic items. I put all of the phrases into their own node,
    because a linked list is made up of nodes [line 24]. I then
    enqueued each node into the queue to prepare for testing of
    palindromes [line 25]. \\
272     \hspace*{1.5em} Next, I created a while loop [line 33] to
    ensure that when the program is running, the entire list of
    magic items will be accounted for, not just one. The loop runs
    while the queue is not empty so that we know that we can
    continue to take from it. Inside this loop, the phrases are
    each passed into "palindromeCheck()" which is the function
    where we finally compare whether or not the magic item is a
    palindrome! After going through the checker and being
    determined whether or not the magic item is a palindrome, the
    loop will print out the magic item only if it is a palindrome.
    \\
273     \hspace*{1.5em} Now, I will go into further detail on the "
    palindromeCheck()" function [line 43]. I initialized the
    checker to true [line 44], and while checking for palindromes
    the checker will change to false if the magic item is not a
    palindrome. Inside this function, I also converted the nodes
    into strings [line 45] so that I am able to separate the magic
    items character by character to check whether or not they are a
     palindrome. I created a new queue and a new stack [lines 46,
    47] which will be the key to checking the magic items for
    palindromes. Inside of a for loop, I then used the node that
    was converted into a string a made a sub-string of each of the
    characters [line 49]. I then put the single characters each
    into their own node [line 50] because my stack and queue
    classes only accept nodes. Then, I pushed the single characters
```

```
     into the stack and enqueued them into the queue [lines 51,
    52].  \\
274 \hspace*{1.5em} At the end "palindromeCheck()" function, I
    created a while loop which simultaneously dequeues and pops the
     single characters from the queue and stack. Then, still inside
     the loop, the single characters are compared against each
    other [line 58]. The checker is set to false if the word is not
     a palindrome [line 59]. \\
275 \hspace*{1.5em} I included a note that I tested the program on
    a blank file as well. This was successful and did not break the
     program. I also included my old code in comment form from when
     I tested my stack, queue, and node class on a simple example
    of "abc", to see if "cba" would be returned. \\
276
277 \begin{lstlisting}[language = Java]
278 import java.io.File;
279 import java.io.FileNotFoundException;
280 import java.util.Scanner;
281
282 //Thought process prior to writing main program
283     //Put all the magic items into their own queue so that we can
    access them in the program
284     //Simultaneously push charachters into stack, queue charachters
     into queue
285     //Simultaneously pop charachters out of stack, dequeue
    charachters out of queue
286     //While popping and dequeueing, compare results to see whether
    or not the word is a palindrome
287
288 public class  Main {
289     public static void main (String[] args) { //Write this
    everytime in a main program
290         //Create a new queue, declare and initialize for holding
    all the magic items
291         Queue magicItems = new Queue(); //Making a variable
    magicItems of the queue data type
292         try { //Trying to find the file
293             File file = new File("magicitems.txt");
294             Scanner sc = new Scanner(file);
295
296             while (sc.hasNextLine()) {
297                 String item = sc.nextLine();
298                 item = item.toUpperCase(); //Changing everything to
     upper case
299                 item = item.replaceAll("\\s+",""); //Replacing the
    spaces with no space
300                 //We must put the string of magic items in a node
    because the queue method takes nodes not strings
301                 Node magicItemsNode = new Node(item, null); //
    Parameters are the string of magic items, then the null pointer
     - null because this node is only a list of the magic items.
    Making a new node to put the phrase into.
302                 magicItems.enqueue(magicItemsNode); //Calling
    enqueue from Queue class, the parameter is because we must
    indicate that each phrase is on its own
303             }
304         }
```

```
305         catch (FileNotFoundException e) { //If we cant find the file
306             e.printStackTrace();
307          }
308 
309         //While loop so the program accounts for the entire file of magic items - not just one phrase
310         while (magicItems.isEmpty() == false) { //While the queue is not empty, so that we know that we can take charachters from it
311             Node phraseNode = magicItems.dequeue(); //Creating a new node to put the dequeed magic items in
312             Boolean palindromeBoolean = palindromeCheck(phraseNode); //Passing the phrase into the check method to see if its a palindrome and returns the result of whether or not it was a palindrome
313             if (palindromeBoolean == true) {
314                 System.out.println(phraseNode.getName()); //If it is a palindrome, then we will print it
315             }
316         }
317     }
318 
319     //Function to check whether or not the phrase is a palindrome
320     public static Boolean palindromeCheck(Node y) { //Parameter is a node, because we are using dequeue items which are in a Node
321         Boolean palindromeCheck = true; //Initializing the checker to true, while checking for palindromes it will change based on whether or not it is a palindrome
322         String phraseString = y.getName(); //Putting the node that contains the dequeued items into a string so that we can seperate it charachter by charachter
323         Stack palindromeStack = new Stack(); //intializing the stack to check whether its palindrome
324         Queue palindromeQueue = new Queue(); //initializing the queue to check whether its a palindrom
325         for (int i = 0; i < phraseString.length(); i++) { //int i = 0 so that it starts at the beginning of the string, i < length to make sure it doesn't go past one phrase at a time, i++ so that increments by 1
326             String phraseCharachter = phraseString.substring(i, i+1); //String that contains the charachters of each phrase, i+1
327             Node charachterNode = new Node(phraseCharachter, null); //Creating a new node with only the charachter
328             palindromeStack.push(charachterNode); //Pushing the single charachter node into the stack
329             palindromeQueue.enqueue(charachterNode); //Enqueueing the single charachter node into the queue
330         }
331 
332         while (palindromeStack.isEmpty() == false) { //While the stack with the charachters is not empty, continue to check the word for whether or not it is a palindrome
333             Node checkQueue = palindromeQueue.dequeue(); //Simultaneously dequeuing and popping from the stack and queue to check whether its a palindrome
334             Node checkStack = palindromeStack.pop();
```

```
            if (!(checkStack.getName().equals(checkQueue.getName())
    )) { //.equals function for comparing strings
                palindromeCheck = false; //Only one equal sign
    because we are actually setting the checker to false
            }
        }
        return palindromeCheck; //Providing the result of the
    comparison
    }
}

//Note: tested if program worked on blank.txt file

//Old code from when I was checking if my stack queue and node
    classes worked
/*Node nodeOne = new Node("a", null);
        Node nodeTwo = new Node("b", null);
        Node nodeThree = new Node("c", null);

        Node nodeFour = new Node("a", null);
        Node nodeFive = new Node("b", null);
        Node nodeSix = new Node("c", null);


        Stack stackOne = new Stack(); //initializing a stack
        stackOne.push(nodeOne);
        stackOne.push(nodeTwo);
        stackOne.push(nodeThree);

        while (stackOne.isEmpty() == false) {
            Node popped = stackOne.pop(); //Telling the system we
    are popping a node
            System.out.println(popped); //Printing what was in the
    node that was popped
        }

        Queue queueOne = new Queue(); //initializing a queue
        queueOne.enqueue(nodeFour);
        queueOne.enqueue(nodeFive);
        queueOne.enqueue(nodeSix);

        while (queueOne.isEmpty() == false) {
            Node queued = queueOne.dequeue();
            System.out.println(queued);
        } */

//Miscellaneous comments
    //Constructors turn the frameowork from the different classes
    into actual objects
    //Two equal signs if boolean
    //&& is how you say and (Boolean)
    //When biulding a function, after "public", always say the data
     type we are going to return (Node, Boolean, String, etc.)
    //global variable - applies to the entire class
    //regular variable - only applies to the specific functin its
    within
\end{lstlisting}
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The commands below print the source code starting
% on a new page. Comment out or delete if you do not want to
% include the source code in your document.
%
\newpage
\section{\LaTeX \hspace{.25em} Source Code}
\lstset{basicstyle=\footnotesize\ttfamily, breaklines=true,
    language=[LaTeX]{TeX}}
\lstinputlisting{main.tex} % <---CHANGE TO CORRECT FILENAME
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\end{document}
```