```
//Assignment01
//Rocco Piccirillo
//Runner

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Scanner;
import java.util.function.UnaryOperator;

public class Runner
{

        public static void main(String[] args) throws FileNotFoundException
        {
                //the scanner is storing the magicitems.txt file temporarily
                Scanner scanner = new Scanner(new File("magicitems"));
                //made to actually store the magic items
                ArrayList<String> wordList = new ArrayList<String>();

                //while there is still another line of text more keeps getting added
                while(scanner.hasNextLine())
                {
                        wordList.add(scanner.nextLine());
                }

                //upper transforms all of the characters into upperCase
                UnaryOperator<String> upper = (x) -> x.toUpperCase();
                //noSpaces removes all of the spaces from wordList
                UnaryOperator<String> noSpaces = (x) -> x.replace(" ", "");
                wordList.replaceAll(upper);
                wordList.replaceAll(noSpaces);


                //not the most creative names but this is so I don't misplace anything
                Stack stack = new Stack();
                Queue queue = new Queue();

                //made for my printouts
                String pally;
          // loop over arrayList
          for (int i = 0; i < wordList.size(); i++)
          {
              pally = " is a palindrome";
              // Loop over each string in arrayList
              for(int j = 0; j < wordList.get(i).length(); j++)
              {
                  // filling the stack and queue
                  stack.push(wordList.get(i).charAt(j));
                  queue.enQueue(wordList.get(i).charAt(j));
              }

              int stackSize = wordList.get(i).length();
              // looping over the stack and queue to do the check
              for (int z = 0; z < stackSize; z++)
```

```
            {
                // if any of the chars don't match, it is NOT a palindrome
                // this is simultaneously checking and removing from our stack and 
queue
                if (stack.pop().data != (queue.deQueue().data))
                {
                    pally = " is not a palindrome";
                }
            }
            // pally is only set to not a palindrome if we find inequalities, 
otherwise, we found no issues and the default string is good
            System.out.println(wordList.get(i) + pally);

        }
      }
}



//Assignment01
//Rocco Piccirillo
//LinkedList

public class LinkedList
{

	Node head; //refers to the first node

	//want to assign this data to a node
	//gets added at the end of the list
	public void append(char data)
	{
		//creating a new node everytime you insert
		Node node = new Node();

		//whatever data i assign will be in that node
		node.data  = data;
		node.next = null;

		//if we are inserting our first object
		if(head == null)
		{
			head = node;

		} else
		{
			Node n = head;
			while(n.next != null)
			{
				n = n.next;
			}
			n.next = node;
		}
	}
	//this is premade for enQueue
	public void insertAtStart(char data)
	{
		Node node = new Node();
		node.data = data;
		node.next = null;
		node.next = head;
		head = node;

	}
	//this is premade for my pop/deQueue
	//the head value is being replaced with the next value
	public Node delete()
	{
		Node node;
		node = head;
		head = head.next;
		return node;
	}
	//prints out all of the values
	public void show()
	{
		Node node = head;

		while(node.next != null)
		{
			System.out.println(node.data);
			node = node.next;
		}
		System.out.print(node.data);
	}


}


//Assignment01
//Rocco Piccirillo
//Stack

public class Stack
{

	Node head;
	//pushes the newly created node ontop of the stack
	public void push(char data)
	{
		Node node = new Node();
		node.data = data;
		node.next = null;
		node.next = head;
		head = node;
	}

	//changes the address of the head and removes it
```

```java
	public Node pop()
	{
		Node node;
		node = head;
		head = head.next;
		return node;
	}

	//prints outs all the node.data from the stack
	public void display()
	{
		Node node = head;
		while(node.next != null)
		{
			System.out.println(node.data);
			node = node.next;
		}
		System.out.print(node.data);
	}

	//prints the top of the stack
	public Node peek()
	{
		Node node;
		node = head;
		System.out.print(node.data);
		return head;
	}
}



//Assignment01
//Rocco Piccirillo
//Queue

public class Queue {

	Node head;

	//adds to the tail of the queue
	public void enQueue(char data)
	{

		Node node = new Node();
		node.data  = data;
		node.next = null;

		if(head == null)
		{
			head = node;

		} else
		{
```

```
                    Node n = head;
                    while(n.next != null)
                    {
                            n = n.next;
                    }
                    n.next = node;
            }

    }

    //removes from the front of the queue
    public Node deQueue()
    {
            Node node;
            node = head;
            head = head.next;
            return node;
    }

    //prints out all of the values of the queue
    public void show()
    {
            Node node = head;

            while(node.next != null)
            {
                    System.out.println(node.data);
                    node = node.next;
            }
            System.out.print(node.data);
    }

    //prints out the head node
    public Node peek()
    {
            Node node;
            node = head;
            System.out.print(node.data);
            return head;
    }
}
//Assignment01
//Rocco Piccirillo
//Node

public class Node {

    //this is the string that gets stored in the node
    public char data;

    //this is the pointer for the next node
    public Node next;


}
```