Stacks and Queues

In this assignment, you will be implementing Stacks and Queues. You will then use stacks and queues to test if words are palindromes, and use stacks to decompose phrases into words (to find new palindromes).

- Implement `MyStack` (use the included `MyLinkedList` to store the stack elements)
  - Implements `StackInterface`
  - Throws a `StackException` if peek or pop are called on an empty stack
- Implement `MyQueue` (use `MyLinkedList` to store the stack elements)
  - Implements `QueueInterface`
  - Throws a `QueueException` if peek or dequeue are called on an empty queue
- Test your MyStack and MyQueue **thoroughly**
- In `Palindrome.java`
  - Implement `stackToReverseString()` using MyStack
  - Implement `reverseStringAndRemoveNonAlpha()` using `MyStack`
  - Implement `isPalindrome()`, which returns true if a word or phrase is a palindrome, using `MyStack` and `MyQueue`
  - CHALLENGE: Implement `explorePalindrome()` which lists all possible backwards decompositions of a phrase (e.g. “evil guns” => snug live“), a common trick to make new palindromes (uses MyStack)
- Style requirements **(NEW!)**
  - Indent your code properly
    - There are several mainstream “styles” of indentation, pick one and be consistent. EG: https://javaranch.com/styleLong.jsp
  - Name your variables with helpful and descriptive names
    - Whats a good variable name? Here is a guide
  - Add comments before every function, and in your code
    - Comments should say **what you are trying to do** and **why**
    - Consult this guide to commenting

Point breakdown

Stacks - 20 points
Queues - 20 points
Style - 15 points

Implementing the functions:
`String stackToReverseString(MyStack)` - 10 points
`String reverseStringAndRemoveNonAlpha(String)` - 5 points
`Boolean isPalindrome(String)` - 10 points

`void explorePalindrome()` (and its helper) - 20 points

Implementing MyStack and MyQueue

In this assignment, we will be making heavy use of the classes `MyStack` and `MyQueue`. You have already implemented `MyLinkedList.java` in a previous assignment. You can use your code, or the sample `MyLinkedList.java` provided.

Implement `MyStack.java` and `MyQueue.java` using the provided interfaces and exceptions.

Make sure you have implemented a `public String toString()` method on MyStack and MyQueue that prints out the contents of the stack from the top down and prints out the queue from the front to back. So, for example, after the following code:

```
MyStack stack = new MyStack();
MyQueue queue = new MyQueue();
stack.push("Hello");
queue.enqueue("Hello");
stack.push("big");
queue.enqueue("big");
stack.push("world");
queue.enqueue("world");
```

```
System.out.println("Stack = " + stack);
System.out.println("Queue = " + queue);
```

Then the output would be:

```
Stack = (world, big, hello)
Queue = (hello, big, world)
```

Test your code thoroughly!! We have provided `TestQueuesAndStacks.java` as an example of some ways to test your code, but you will want to edit it to try out many possible situations. Make sure your code behaves **exactly** as you expect it to, before starting the second half of the assignment.

Is this a palindrome?

A palindrome is a word or phrase that reads the same backwards and forwards, if you ignore punctuation and spaces. For example:

- A dog! A panic in a pagoda!

- ABBA
- Cigar? Toss it in a can. It is so tragic.
- Yo, bottoms up! (U.S. motto, boy.)
- Stressed was I ere I saw desserts.

(from http://www.palindromelist.net/palindromes-y/)

In this part of the assignment, you will be writing several functions in `Palindrome.java` to test and create palindromes using your stack and queue implementations.

Example Input & Output

`Palindrome.java` takes as input: a **mode**, and some **words.** The mode is either “test” or “expand”, so the function call will be:

Test for palindromes
Are these phrases palindromes?

```
javac Palindrome.java && java Palindrome test "oboe" "ABBA" "I'm alas, a
salami" "evil liver"
'oboe': false
'ABBA': true
'I'm alas, a salami': true
'evil liver': false
```

Expand palindromes
Which words could be added to make this a palindrome?

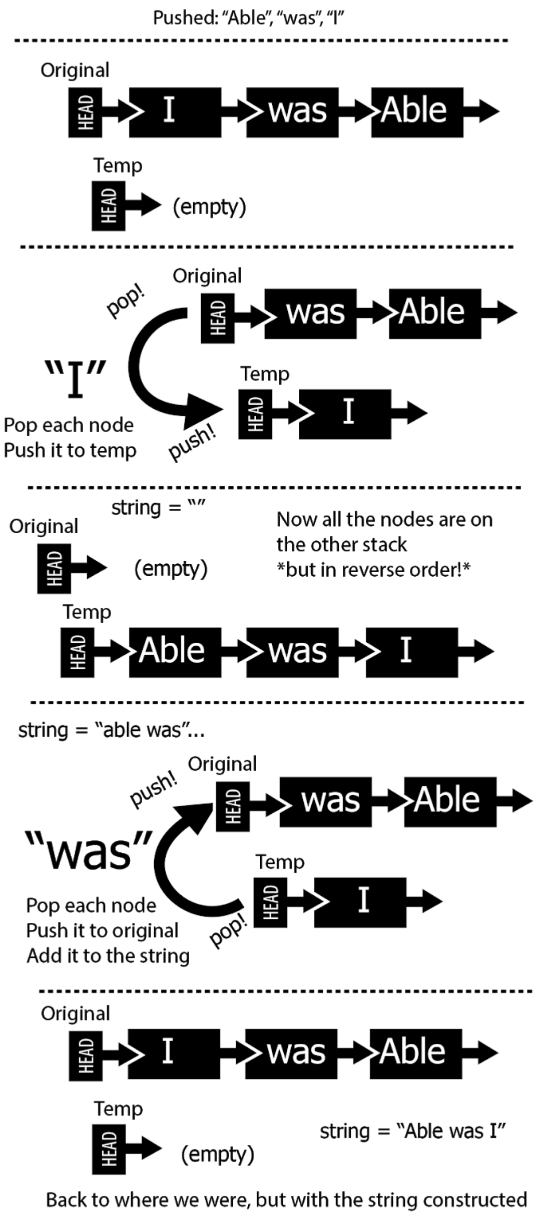
```
javac Palindrome.java && java Palindrome expand "an era live" "mug god"
an era live:  evil a ren a
an era live:  evil aren a
an era live:  evil arena
mug god:  dog gum
```

Functions to implement:

`String stackToReverseString(MyStack):` your toString function in your stack class prints out the stack in the order that things would be popped from it. What if we want to turn it into a string in the opposite order (the order that things were **pushed** to it)?

In Palindrome.java, we do not have access to the internal representation of the stack's list. So instead, we have to pop everything off the stack, read it in order and push it pack onto the stack in the original order so that the stack is unchanged (*Whew!*)

- Create an empty string
- Create a new temporary stack
- Pop everything from the original stack onto the new stack
- Pop everything from the new stack back onto the original stack **and** add it to the string



```
String reverseStringAndRemoveNonAlpha (String)
```

Use a stack to reverse a string. Similar to before. Also, we want to not only

- Create a new stack.
- Iterate through the string, and push each character from the string onto the stack, but **only if** they are alphabetic (ignore spaces and punctuation)
    - `Character.isAlphabetic` will be a useful function here!
- Pop everything from the stack to reconstruct the string in reverse.

- Note: your stack can only contain Objects. Java's `char` datatype isn't an object though! This means that you will have to wrap it (and later cast it) as a `Character` type. Look up the Character class in Java's documentation, or find out more about wrapper classes here.

```
Boolean isPalindrome(String)
```

Implement this function using **both a stack and a queue.**
To test if a string is a palindrome:

- Convert the string to lowercase (we don't care about uppercase vs lowercase characters being different)
- Create a new stack and queue.
- Enqueue and push each character (if it is alphabetic, we don't want to look at white space or punctuation)
- Pop each character from the stack and dequeue each character from the queue until one is empty
- Notice how in our above functions, pushing and then popping from a stack **reversed the order.** How will you use this to test whether this string is a palindrome?

```
void explorePalindrome(String)
```

This function lists all possible endings that would make this string a palindrome, e.g.:

```
javac Palindrome.java && java Palindrome expand "an era live" "mug god"
an era live:  evil a ren a
an era live:  evil aren a
an era live:  evil arena
```

First, convert the string to lowercase and use your `reverseStringAndRemoveNonAlpha` function to reverse it and remove non-alphabetical characters.

Now, most of the work will be done by a recursive helper function to decompose this new string ("evilarena") into words. Takes the original string, the reversed string, an index, and the current stack of words we are building up

```
 public static void decomposeText(String originalText, String
textToDecompose, int index, MyStack decomposition)
```

We have provided a function `String[] getWords(String text, int index)` that uses a dictionary to find which words could be created from a string at a given index. For example `getWords("isawere", 0)` could find the words "i" and "is", `getWords("isawere", 2)` could find the words ("a", "aw" and "awe").

A recursion step:

- **If the index is at the end of the word, we are finished, print out the words (using reverse print) and the original text.**
- Else: Find the potential words at that index
- For each word:
    - Push it to the stack
        - Recurse at the next index (not *just* i++)
        - If it was part of a correct solution, it will print itself out in a subsequent recursion step (if it reaches a conclusion)
    - Pop it from the stack
- Confused? See below for a visual explanation of this approach.
    - As usual, println statements may help you understand your code's operation if you get lost. Consider outputting the list of words from getWords, or printing out the `decomposition` stack each time you push or pop from it. Just remove your debug statements before turning in your code.

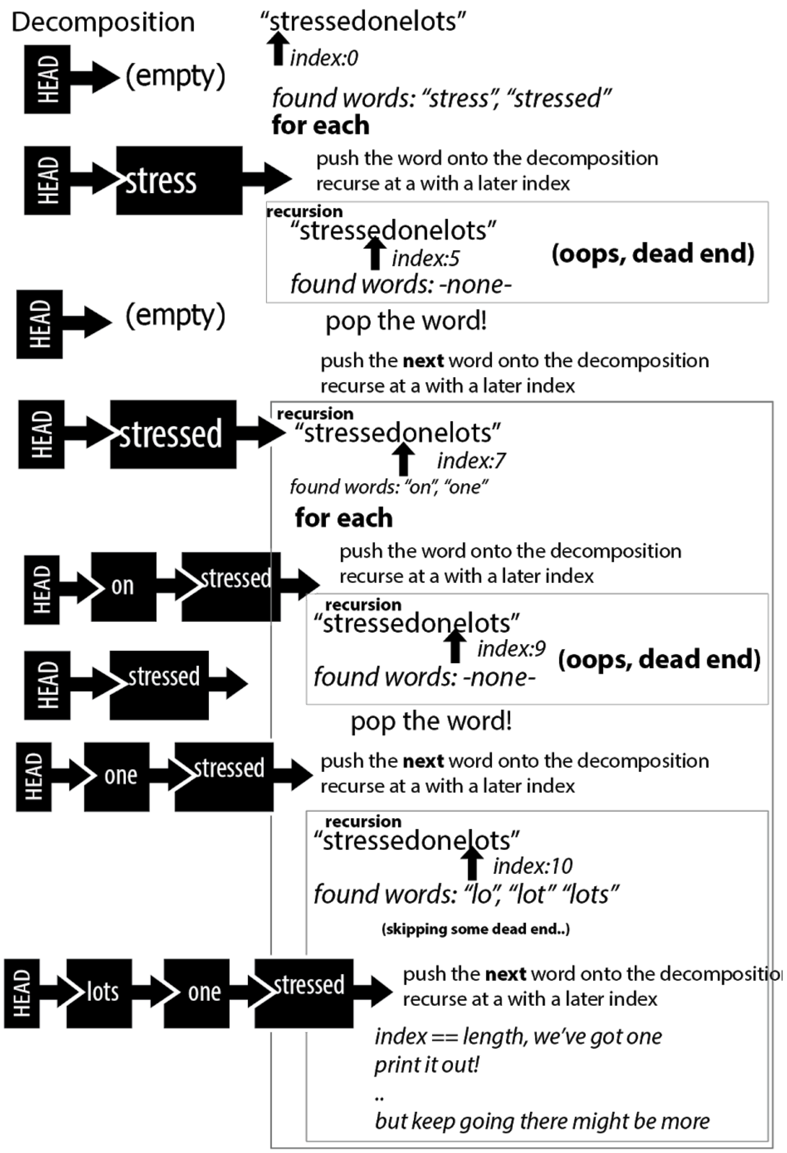
Turning the code in

- Create a directory with the following name: <student ID>_assignment3 where you replace <student ID> with your actual student ID. For example, if your student ID is 1234567, then the directory name is 1234567_assignment3
- Put a copy of your edited files in the directory (MyQueue.java, MyStack.java, and Palindrome.java)
- Compress the folder using zip. Zip is a compression utility available on mac, linux and windows that can compress a directory into a single file. This should result in a file named <student ID>_assignment3.zip (with <student ID> replaced with your real ID of course).
- Double-check that your code compiles and that your files can unzip properly. You are responsible for turning in working code.
- Upload the zip file through the page for [Assignment 3 in canvas]

Visual example of palindrome search with stacks

String: “stole no desserts”

String: “stressedonelots”



Decomposition

“stressedonelots”

index:0

found words: “stress”, “stressed”

for each

push the word onto the decomposition
recurse at a with a later index

recursion

“stressedonelots”

index:5

found words: -none-

(oops, dead end)

pop the word!

push the **next** word onto the decomposition
recurse at a with a later index

recursion

“stressedonelots”

index:7

found words: “on”, “one”

for each

push the word onto the decomposition
recurse at a with a later index

recursion

“stressedonelots”

index:9

found words: -none-

(oops, dead end)

pop the word!

push the **next** word onto the decomposition
recurse at a with a later index

recursion

“stressedonelots”

index:10

found words: “lo”, “lot” “lots”

(skipping some dead end..)

push the **next** word onto the decompositio
recurse at a with a later index

index == length, we’ve got one
print it out!
..
but keep going there might be more