COMP 282 Summer 2019 **Project 1**

Due Friday, June 14th 2019 23:55

Requirements: To complete this project you will write and submit 1 file: `Project1.zip`

Project1.zip is a zip-file archive containing all of your source code, as .java files. You do not need to include .class files in your Project1.zip. You must also include a plain-text file inside the zip-file named `Readme.txt`

Each of these files will be created as an ASCII file (i.e. a plain text document with a .java or .txt extension). You may create your source code and `Readme.txt` with any editor or IDE but you must ensure that they are plain text files. In other words, they should not contain anything except ASCII characters.

`Readme.txt` must contain your answers to the questions at the bottom of this assignment. See section named "QUESTIONS FOR README.TXT"

NOTE: you may only use java.util.Scanner for this project. No other use of java.util classes is allowed!

IMPLEMENTATION

Your project is broken into parts as follows…

PART 1 – Creating String Container classes (data structures)

Create each of the following data structures as its own class/file.

Each of these data structures must contain a `print()` method that traverses the data structure's contents and prints them neatly to System.out (using System.out.println). This can be used to debug your code by inspecting contents of the data structures.

A. A resizable string array (aka a vector), which is initialized to 10 elements. Every time the array grows beyond its current capacity, it is copied to a new array of twice the old capacity, and the array reference is updated. This array class must be defined in its own class/file named **StringArray.java**. (Notice, no template/generic code is being used on this first assignment.)
B. A **singly linked list with a head pointer and tail pointer**. This linked list class must be defined in its own class/file named **StringLinkedList.java**. (Notice, no template/generic code is being used on this first assignment.)
C. A Queue implemented using StringLinkedList.java. This class must be defined in its own class/file named **StringQueue.java**. (Notice, no template/generic code is being used on this first assignment.)
D. A Stack of strings using StringLinkedList.java. This class must be defined in its own class/file named **StringStack.java**. (Notice, no template/generic code is being used on this first assignment.)

PART 2 - Reading input from a file into different data structures

In your main routine, which can be put in a file named **Main.java**,
Using java.util.Scanner, read each word from a file into each of the following data structures…

- An instance of StringArray
- An instance of StringLinkedList
- An instance of StringQueue

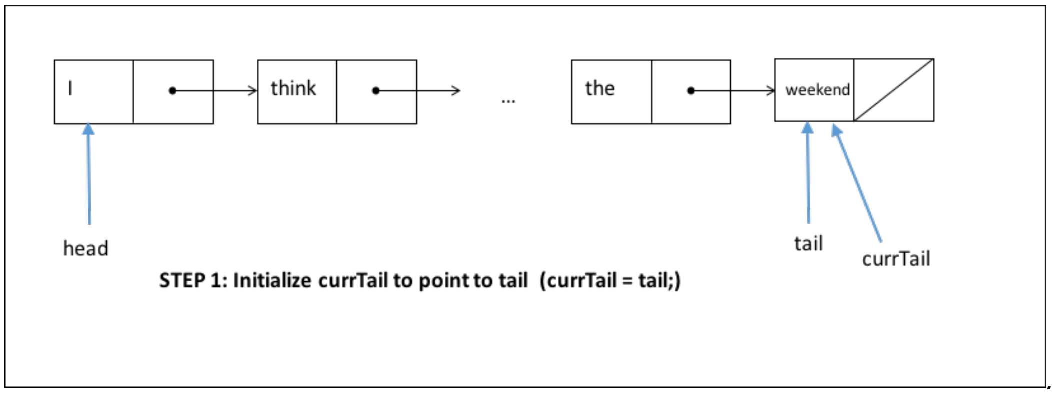
(Don't worry about StringStack for now. We will use it soon in Step 5.)

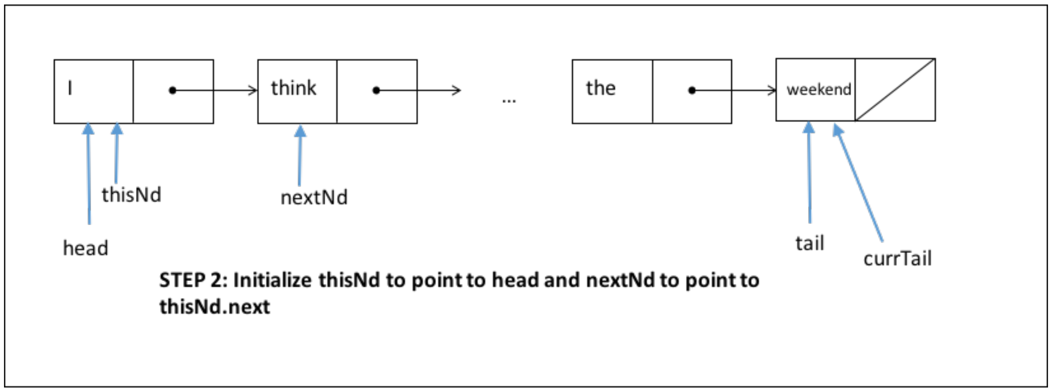
PART 3 – Reversing the list of Strings in-place in an Array

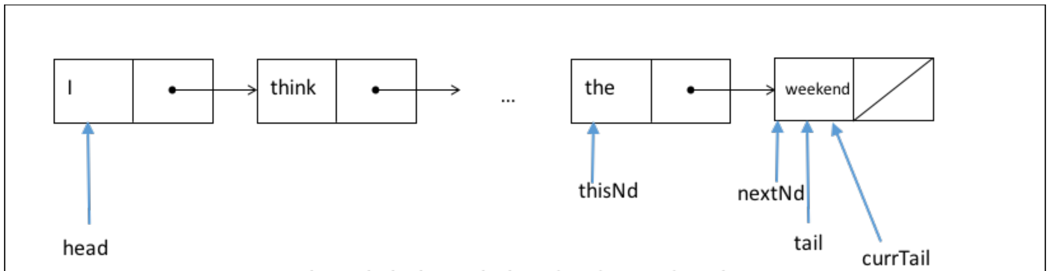
Reverse the strings of the StringArray in place. No other data structure should be needed to perform this action.

PART 4 – Reversing the list of Strings by traversing a Linked List

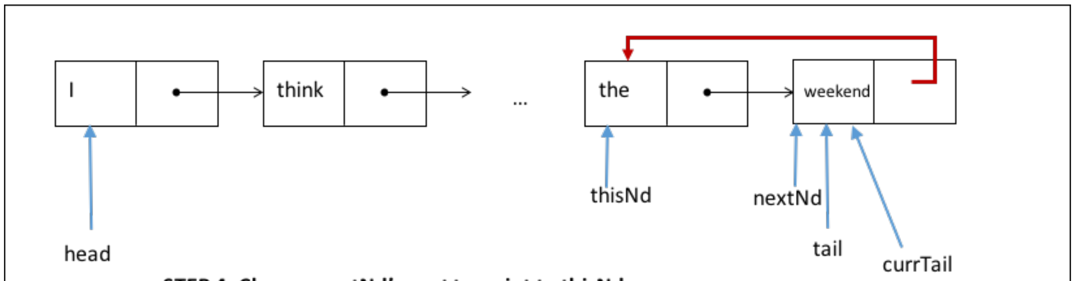
Use the following iterative algorithm to reverse the linked list. You will notice this algorithm is O(n^2)


I
think
...
the
weekend
head
tail
currTail
STEP 1: Initialize currTail to point to tail (currTail = tail;)

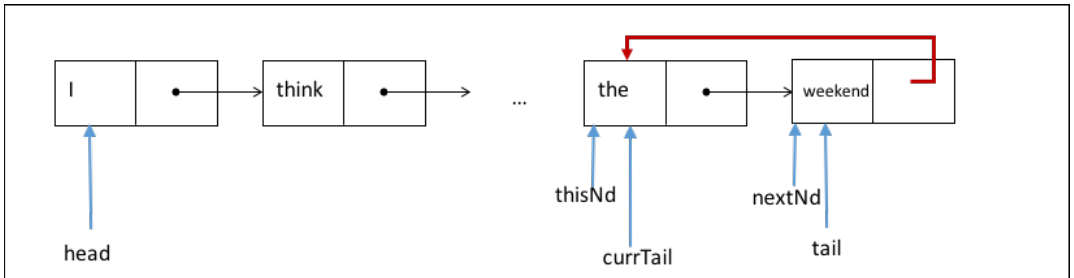

I
think
...
the
weekend
thisNd
nextNd
head
tail
currTail
STEP 2: Initialize thisNd to point to head and nextNd to point to thisNd.next



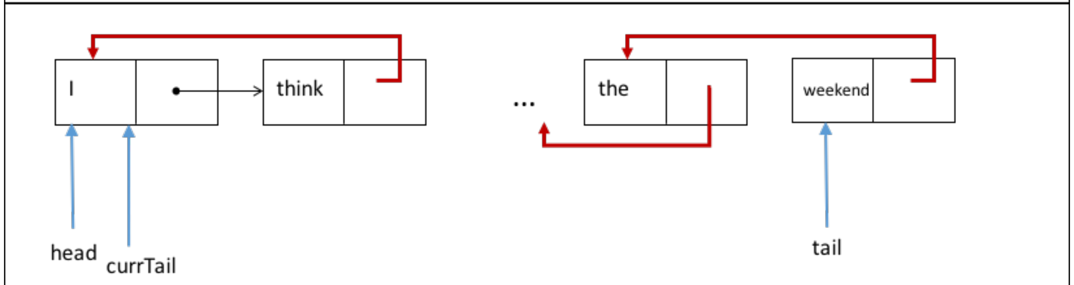
STEP 3: Iterate through the list with thisNd and nextNd until nextNd == currTail



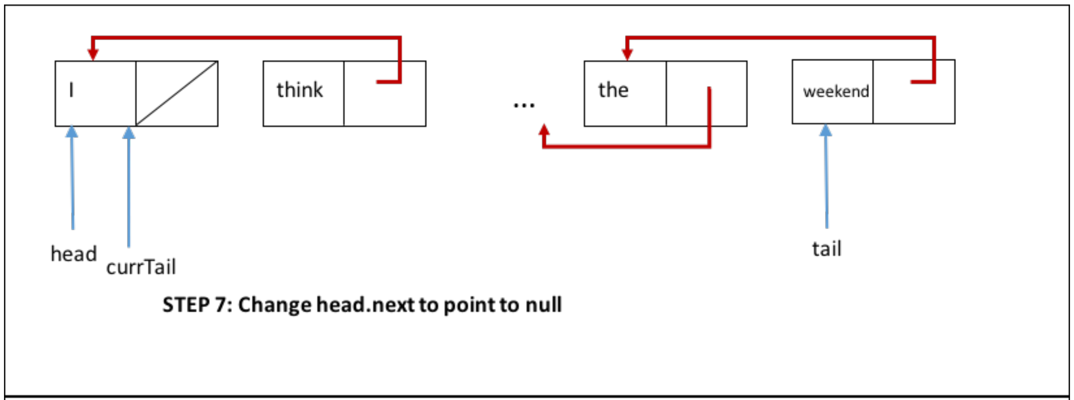
STEP 4: Change nextNd's next to point to thisNd



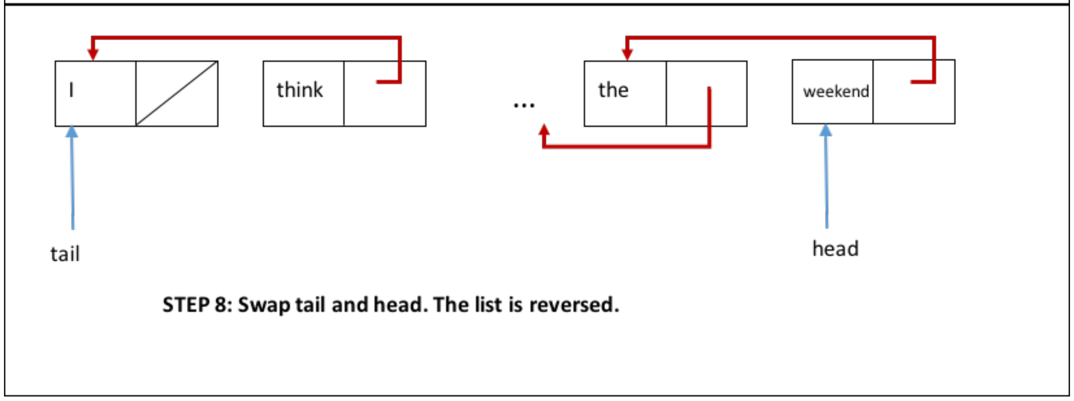
STEP 5: Assign currTail to thisNd (*but don't move tail!!!*)



STEP 6: Repeat Steps 2 through 5 until the currTail points to head. At this time each list node should point to the one behind it, except the head node.



STEP 7: Change head.next to point to null



STEP 8: Swap tail and head. The list is reversed.

PART 5 – Reversing the list of Strings by enqueuing them in a Queue and then pushing/popping them to/from a Stack

Instantiate a StringStack. As each string from the StringQueue instance is de-queued, push it onto the StringStack. Then, pop each element in turn from the StringStack and en-queue it into the StringQueue. (This effectively reverses the StringQueue.)

<u>OUTPUT</u>

Your program must report the following using standard System.out.println console output…

A. Number of words read from input file
B. Time (in nano-seconds) of executing *just* Part 3 (the array reversal)
C. Time (in nano-seconds) of executing *just* Part 4 (the linked list reversal)
D. Time (in nano-seconds) of executing *just* Part 5 (the queue/stack reversal)
E. Results of each of the reversals, using that data structure's `print()` method (see Part 1).

You can use this output to help answer the "QUESTIONS FOR README.TXT"

QUESTIONS FOR README.TXT

1) What was the largest number of words *W* you were able to test with? Why was this the largest (in other words, what behavior did you notice above this size)?
2) What is the runtime of the Array in-place reversal for *W words*?
3) What is the runtime of the Linked List reversal for *W words*?
4) What is the runtime of the Queue/Stack reversal for *W words*?

Now, try again for a file of size *W/2 words…*

5) What is the runtime of the Array in-place reversal for *W/2 words*?
6) What is the runtime of the Linked List reversal for *W/2 words*?
7) What is the runtime of the Queue/Stack reversal for *W/2 words*?

Now, try again for a file of size *W/4 words…*

8) What is the runtime of the Array in-place reversal for *W/4 words*?
9) What is the runtime of the Linked List reversal for *W/4 words*?
10) What is the runtime of the Queue/Stack reversal for *W/4 words*?

Finally…

11) Based on the answers from #1 to #10, what conclusions can you draw about the relative choice of data structures and strategy for reversing a list of Strings?

SUBMISSION

Prior to the deadline upload your file `Project1.zip` to Canvas. Do not submit any other documents!

GRADING CRITERIA

10%: Submission instructions followed to the letter (1 zip file named Project1.zip, containing appropriate Readme.txt and all required .java files: StringArray.java, StringLinkedList.java, StringStack.java, StringQueue.java, and Main.java)

20%: Readme.txt answers questions

30%: All source code compiles and executes as specified

40%: Source code looks proper and matches specification

A Word On Cheating: This project is an individual project. You can discuss this project with other students. You can explain what needs to be done and give suggestions on how to do it. You cannot share source code. If two projects are submitted which show significant similarity in source code then both students will receive an F in the course. Note a person who gives his code to another student also fails the class (you are facilitating the dishonest actions of another).