COMP 1020 Winter 2016 Assignment 4

Due Date: Monday, April 4, before midnight

1 MATERIAL COVERED

- Linked lists

2 NOTES AND INSTRUCTIONS—PLEASE FOLLOW *EXACTLY*

- Follow the programming standards posted on the course website to avoid losing marks.
- You must complete the "Blanket Honesty Declaration" checklist on the course website, before you can submit any assignment.
- To submit the assignment you will upload the required files, as specified for each question, to the Dropbox for Assignment 4 on the course website.
- The markers may compile and execute your code. To be eligible to earn full marks, your Java programs must compile and run as downloaded, without requiring any modifications.
- Hand in **only and exactly** the files specified in this assignment, in the specified format. Failure to do so will result in lost marks.

3 QUESTION 1: WORD LIST

3.1 LINKEDLIST CLASS

In this question, you will write a linked list class, and use it to store all of the words contained in a text file. You will write your linked list class in a file called `LinkedList.java`, which will contain BOTH the `LinkedList` class and the `Node` class (only the `LinkedList` class is public).

You have been provided with a main program file called `A4Q1Test.java` and a text file called `darwin.txt`.

First, create a `Node` class. It should contain, as usual, a pointer to another node, and a `String` reference. Write a constructor that sets the data being stored in the `Node`, and set its link to the next `Node` to `null`. Write appropriate getters and setters as well.

Next, create your `LinkedList` class. As usual, it should contain a `first` (`top`) pointer to the first node in the list. Additionally, create a `last` pointer to the last `Node` in the linked list (we'll see why later). Add an integer instance variable to keep track of the `size` (i.e. the number of words) in the linked list.

Create the following `public` methods:

- A null constructor that initializes the two `Node` pointers to null and the size to 0.

- `int size()` that returns the size of the list.
- `void append(String)`, which takes the given word and places it at the *end of the list*, not at the beginning. Instead of performing a costly traversal of the linked list from the beginning until the end is reached, use the `last` pointer instead, when appending a new item. Be mindful of the case when the list is empty.
- A second constructor that takes a `String` parameter, which is the name of a file. The constructor will attempt to read the file and add all the words in the file to the linked list (using `append`). Use whitespace to separate the lines into words. In Java, the String literal `"\\p{Space}+"` specifies one or more whitespace characters.
- `String format(int width)` that returns a String containing all the words in their original order, formatted in lines that are `width` characters long, fully justified (i.e. the lines are aligned on ***both*** the left and right sides). This is achieved by adding extra spaces between the words so that the right side is aligned at `width` characters. In order to make the justification look balanced, distribute the extra spaces throughout the line. Add the extra spaces left-to-right on each second line, and right-to-left on the alternating lines. You may find it helpful to use two private methods to perform these justification steps (though you are not required to do so). The last line is ***not*** justified, since it may contain only a few words.

3.2 EXAMPLE OUTPUT

Here is the output that should be obtained from running `A4Q1Test.java`:

```
This is A4Q1Test.

Let us now briefly consider the steps by which domestic races have been
produced, either from one or from several allied species. Some little effect
may, perhaps, be attributed to the direct action of the external conditions of
life, and some little to habit; but he would be a bold man who would account by
such agencies for the differences of a dray and race horse, a greyhound and
bloodhound, a carrier and tumbler pigeon. One of the most remarkable features in
our domesticated races is that we see in them adaptation, not indeed to the
animal's or plant's own good, but to man's use or fancy. Some variations useful
to him have probably arisen suddenly, or by one step; many botanists, for
instance, believe that the fuller's teazle, with its hooks, which cannot be
rivalled by any mechanical contrivance, is only a variety of the wild Dipsacus;
and this amount of change may have suddenly arisen in a seedling. So it has
probably been with the turnspit dog; and this is known to have been the case
with the ancon sheep. But when we compare the dray-horse and race-horse, the
dromedary and camel, the various breeds of sheep fitted either for cultivated
land or mountain pasture, with the wool of one breed good for one purpose, and
that of another breed for another purpose; when we compare the many breeds of
dogs, each good for man in very different ways; when we compare the game-cock,
so pertinacious in battle, with other breeds so little quarrelsome, with
"everlasting layers" which never desire to sit, and with the bantam so small and
elegant; when we compare the host of agricultural, culinary, orchard, and
flower-garden races of plants, most useful to man at different seasons and for
different purposes, or so beautiful in his eyes, we must, I think, look further
than to mere variability. We cannot suppose that all the breeds were suddenly
produced as perfect and as useful as we now see them; indeed, in several cases,
we know that this has not been their history. The key is man's power of
accumulative selection: nature gives successive variations; man adds them up in
certain directions useful to him. In this sense he may be said to make for
himself useful breeds.

darwin.txt contains 381 words.
```

3.3 HANDIN INSTRUCTIONS

Submit two files: your `LinkedList.java` file (containing your `LinkedList` and `Node` classes), and your output file. Your output file (just a record of your program's standard output) should be named `A4Q1-output.txt`.

4 QUESTION 2: WORDS IN ALPHABETICAL ORDER

For this question, you will need to make the following modifications to your `Node` and `LinkedList` classes. You do not need to create a separate linked list implementation for this question, since both Question 1 and Question 2 will work just fine with the following additions:

4.1 NODE MODIFICATIONS

Add another pointer to a Node. This will be used to create a linked list in alphabetical order. Add getter and setter methods for this pointer. Modify the constructor to initialize this pointer to `null`.

4.2 LINKEDLIST MODIFICATIONS

Add another pointer to a Node. This will be the pointer to our alphabetical list. Add another integer variable to keep track of the size of the alphabetical list. Modify the constructor to initialize these new variables appropriately.

Add the following methods to your `LinkedList` class:

- `int sizeAlpha()` that returns the size of the alphabetical list.
- `String toAlphaString()` that returns a String containing all the words in the alphabetical list. No justification of this String is performed.
- `String toAlpha(String word)`. Since we don't want words with different capitalization being entered as different words, and we don't want punctuation such as periods and commas included in our list, this method will return a new String such that all letters in `word` are changed to lowercase, and punctuation is removed (however we DO want to keep dashes (-) and apostrophes (')). The String methods `toLowerCase()` and `replaceAll()` may be used for this. The String literal `"[^\\p{Alpha}-']"` may be used to match any character that is NOT alphabetic, a dash, or apostrophe; you may find this useful when using `replaceAll()`.
- `void insertAlpha(Node keyNode)`. This method is used to build the alphabetical list. In your initialization constructor, once you have appended a new Node to the end of the word list, call this method to insert this Node in order into the alphabetical list (i.e. call `insertAlpha(last))`. This method will traverse the alphabetical list to find the correct spot for this word, or else determine that the word is already present in the list; in that case, the insertion is not done. Remember to use the `toAlpha()`method on words before comparing them.

You have been provided with a main program file called `A4Q2Test.java`.

4.3 Example Output

Here is the output that should be obtained from running `A4Q2Test.java`:

```
This is A4Q2Test.

a account accumulative action adaptation adds agencies agricultural all allied amount
ancon and animal's another any arisen as at attributed bantam battle be beautiful
been believe bloodhound bold botanists breed breeds briefly but by camel cannot
carrier case cases certain change compare conditions consider contrivance culinary
cultivated desire differences different dipsacus direct directions dog dogs domestic
domesticated dray dray-horse dromedary each effect either elegant everlasting
external eyes fancy features fitted flower-garden for from fuller's further game-cock
gives good greyhound habit has have he him himself his history hooks horse host i in
indeed instance is it its key know known land layers let life little look make man
man's many may mechanical mere most mountain must nature never not now of one only or
orchard other our own pasture perfect perhaps pertinacious pigeon plant's plants
power probably produced purpose purposes quarrelsome race race-horse races remarkable
rivalled said seasons see seedling selection sense several sheep sit small so some
species step steps successive such suddenly suppose teazle than that the their them
think this to tumbler turnspit up us use useful variability variations variety
various very ways we were when which who wild with wool would
darwin.txt contains 195 unique words.
```

4.4 Handin Instructions

Submit your `LinkedList.java` file, and your output file. Your output file (just a record of your program's standard output) should be named `A4Q2-output.txt`.

Note: You only need to hand in **one copy** of your `LinkedList.java` file, as long as they contain the modifications necessary to run both Q1 and Q2. **Thus, for this entire assignment, it is only necessary to hand in 3 files, total: your `LinkedList.java` file, and your two output files.**