[CS209A] Assignment 1

Deadline: 23:59pm at March 31 (No late submission).

TextProcessor

In this assignment, you will design a `TextProcessor` class, which could read an English text file and perform various useful analyses. You'll have a chance to use techniques such as `Collections` , `Lambda` , and `Streams` , which we've covered in the lectures.

The `TextProcessor` class has one constructor as below, which reads a file from a given path. The class also has 5 other methods, which are described below. You'll implement these methods in the `TextProcessor` class, and also a utility class `Position` , which refers to a position that uniquely points to a certain character. See below for details.

Constructor (0.5 points)

```
public TextProcessor(String path)
```

1. Finding the top N most frequent words (1.5 points)

```
public TreeMap<String, Integer> getCommonWords(int n, String[] stopwords)
```

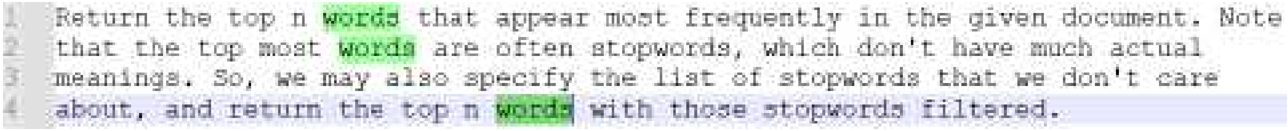
Given `n` , return the top `n` words that appear most frequently in the given document. Note that the top most words are often stopwords (e.g., the, a, not, and, but), which don't have much actual meanings. So, we may also specify the list of stopwords that we don't care about, and return the top `n` words with those stopwords filtered.

The results are `<word, count>` pairs with **descending order by `count`** (i.e., from the most frequent to the least frequent). If multiple words have the same `count` , these words should be ordered by the natural order of `String` (lexicographic order). The case of words should be ignored when computing frequency; please use lowercased words for the results (e.g., if one "Test" and one "test" appeared in the document, then we have `<"test", 2>`).

2. Highlighting all appearances of a word by its position (2 points)

```
public ArrayList<Position> highlightWord(Position pos)
```

When we put our mouse cursor at a word in our document, a smart text processor will highlight all appearances of that word, as shown in the figure.



```
1 Return the top n words that appear most frequently in the given document. Note
2 that the top most words are often stopwords, which don't have much actual
3 meanings. So, we may also specify the list of stopwords that we don't care
4 about, and return the top n words with those stopwords filtered.
```

To describe the cursor's position, define a `Position` class with `Position.line` representing the line number and `Position.col` representing the index of a character of that line. Note that both `line` and `col` are integers and **start from 1**. For example, in the above example, `Position(line=1, col=18)` , `Position(line=2, col=21)` , `Position(line=4, col=32)` all point to "word".

Implement the above method which takes a `Position pos` . If `pos` points to a word, you'll find all the occurrences of this word, representing by their *starting positions*. Again, we'll ignore the case of words (e.g., "Test" and "test" are considered as the same word).

The *starting position* of a word refers to the position of the **first character** of that word. Note that, if `pos` points to a non-first character in a word (e.g., it points to a second or a third letter of a word), the word itself should also be highlighted, and its position should be the position of its first character.

The returned list should be **sorted** first by `line` and then by `col` (i.e., sorted by the reading order). If the input `pos` points to non-words (e.g., a whitespace), an empty list should be returned.

3. Fixing lowercased first letters (2 points)

```
public List<Position> fixLowercaseFirstLetter()
```

An English sentence should start with a captial letter. Please find all the sentences that don't follow this rule, i.e., they start with a lowercase letter. For simplicity, sentences could be divided according to ".", "!" or "?".

Return all such instances indicated by their *starting positions*, again by the order of their appearance.

4. Text encryption (2 points)

```
public String encrypt()
```

We want to protect our file from malicious users. For this purpose, we come up with an encryption schema, which manipulates **each word** in the file with the following 3 steps:

- Step 1: Reverse the characters of the word. For example, "Java" becomes "avaJ" and "course" becomes "esruoc". Note that words are case-sensitive now (e.g., "Java" and "java" are different).
- Step 2: Now, if the first character is "a" or "e" or "s" (lowercase or uppercase), add its ASCII code after it (e.g., "esruoc" becomes "e101sruoc").
- Step 3: Add the length of the **original** word to the end of it (e.g., "e101sruoc" becomes "e101sruoc6")

During encrytion, punctuations like ".", "," and "?" remain the same. Return a string of the entire file content, with each word being encrypted based on the above steps.

5. N-gram similarity (2 points)

```
public HashSet<List<String>> ngramSim(int ngram, String pathForAnotherFile)
```

An n-gram is a continuous `n` words, where `n` could be any positive number. For example, all 3-grams for "This is a good Java course" are ["This is a", "is a good", "a good Java", "good Java course"], while all 4-grams for the same sentence are ["This is a good", "is a good Java", "a good Java course"].

We'll use n-grams to check the similarity between documents. Given `n` and `pathForAnotherFile`, find all n-grams that exist both in this file and in another file. Return a set of these n-grams (i.e., the intersection of the n-grams in both files), with n-grams being represented as a list of case-insensitive lowercase words (e.g., a 3-gram "This is Java" is stored as a list ["this","is","java"]).

Submission

- Please submit two files, `TextProcessor.java` and `Position.java`, to the OJ system. Instructions on how to access the OJ system will be announced soon. You could use our sample test cases to test your code first.
- We provide an `A1_sample.zip`, which specifies the project structure and provides sample test code. Please go through the `readme.txt` in this zip carefully before you get started with the assignment.

Notes

- Try using `Streams` and `Lambda` whenever possible. **DO NOT** use any 3rd-party libraries for this assignment.
- Please ensure that your code compiles; **DO NOT** use Chinese in your code comments.