Programming in Java

Assignment 2

Tweet Cleaner and Generator

Marks available: 100

Date issued: 18/10/19

Deadline: 25/10/19, 17:00

This assignment is worth 12.5% of your final module mark.

The functionality component of the following exercises is worth 80% of the marks for each exercise. This means, the extent to which your methods provide the required behaviour as determined by the JUnit tests.

The remaining 20% for each exercise will be awarded for the 'quality' of your code. For this exercise that means only your layout and naming. You should refer to the lecture notes on this topic (lecture 3a) and also the style guide document on Canvas called 'style.pdf'.

Therefore, the maximum functionality mark for Exercise 1 is $70 \times 0.8 = 56$ marks and the maximum functionality mark for Exercise 2 is $30 \times 0.8 = 24$ marks. The remaining marks are for quality.

Submission instructions

Submit your work by the deadline above as a zip archive. Use only 'zip' format and no other. Submit only .java source files in your archive. Do NOT submit .class files or any type of file. Please do not submit entire Eclipse (or other IDE) projects.

To submit, gather all of your Java files into a folder, zip that folder, and then submit the zip.

In your programs, do not output anything to the console unless explicitly asked to.

Each Java file must have a package declaration at the topic of it (these declarations must be on the very first line of the file). The package declaration for all of the exercises below is:

com.bham.pij.assignments.twit;

Do not forget to add the semicolon at the end of this line. All characters in a package name should be lower case.

In each of these exercises, you are not restricted to only creating the *required* methods. If you want to, you can create others. However, you must create **at least** the required methods.

Do not make the methods required by these exercises `static`.

You **can** use the Java collections for these exercises, e.g. `ArrayList`.

You must **not** use *regular expressions* in your solution. Use of regular expressions in your solution will result in a mark of zero. You may use the `String split()` method if you wish but do not pass it anything other than a single character as a delimiter.

If you use regular expressions in your solution your mark for this assignment will be zero.

Do not put any code that gets user input in the required methods below. If you do, your work can't be tested. This means that you should not use `Scanner` in any required method or in any method called by a required method, etc.

Please follow these instructions. If you do not you could get a lower mark than you would have done for following them or a mark of zero. Please ask if you are unsure.

Exercise 1: Tweet Cleaner [70 marks]

Download the 'Assignment2Files.zip' file from Canvas. You do not need `Word.java` for Exercise 1. Unzip these files to some convenient location. Make sure the file 'donald.txt' is in the folder that you intend to run your program from.

For this exercise you will write a method that recieves raw Twitter tweet data as input and returns that input with particular elements removed. The elements to be removed are listed below. The point of this method is to allow only words to be retained from tweets with all other content removed. The tweets that you will work upon are contained in the file 'donald.txt' which contains some tweets by President Donald Trump.

Tweets contain elements such as @names, hashtags, URLs, punctuation, numbers, etc. Your task will be to remove most of this and return a list that contains words only. The list below specifies the elements in each tweet that must be removed:

- @names, e.g. @donaldjtrump, and any others that are mentioned.
- URLs.
- hashtags, e.g. `#Hillary`, etc..
- The strings "RT" or 'rt (RT means 'retweet').
- Any input element that contains digits.
- Ellipsis (i.e. three/four dots: ...). These appear in the text where a post is split into multiple tweets. Just remove the ellipsis.
- All punctuation marks BUT see the exceptions below.

- Remove hyphens from hyphenated words. This will render some words 'wrong' but don't worry about that.

Exceptions to the removal of punctuation marks:

- Do not remove exclamation marks that are at the end of a word. The word "great!" is to be treated as a single word and is distinct from the word "great". You don't need to consider there being more than one exclamation mark at the end of a word.
- Do not remove question marks that are at the end of a word. The word "really?" is to be treated as a single word and is distinct from the word "really". You don't need to consider there being more than one question mark at the end of a word.
- Do not remove possessive inverted commas, e.g. in the words"Biden's", "America's", "James' ".

Note that you don't need to worry about letter case. You can simply ignore it.

In the class `TweetCleaner` you will see there is a method called:

```
public String clean(String input)
```

You need to complete this method. All of the other required code is already in the file. You don't need to add anything else to the code for this exercise. Note, however, that, in the file, this method simply has `return null;` as its solitary line of code. You will need to remove that or incorporate it into your own code.

The `clean` method will receive a section of the file 'donald.txt' to clean each time it is called. You don't need to worry about how much of the file that is. Just clean what you are given. The input will not necessarily contain an entire tweet or it may contain multiple tweets. It doesn't matter.

When you have cleaned the input, you should return it as a single string of words with a space character between each word. If there is nothing left after you have cleaned the string, i.e. every part of it is invalid, then return `null` in that case.

Therefore, the return value of this method will be a sequence of words from the input, concatenated into a single string, that 'pass' all of the rules identifed above, each one separated by a space from the adjacent words, or it will be `null` if no words were found that pass the rules.

When the program exits, you will see that it has created a file called 'cleaned.txt'. This will contain all of the words that your method decided were valid. You can examine that file to see the output of your method and to consider if it is meeting the requirements.

There is a method you can use called `addClean()` that receives a `String` and adds the string to the 'cleaned' list. (or you can simply use `cleaned.add()` passing your string as a parameter).

Exercise 2: Tweet Generator [30 marks]

For this exercise you also need the file `Word.java`.

For this exercise you will take the output from Exercise 1 and use that to generate tweets in the style of Donald Trump. These tweets will be ungrammatical, however.

The overall idea of the class `TweetGenerator` is as follows. `TweetGenerator` uses the cleaned list of words loaded from 'donald.txt' to generate tweets. It does this by picking a word and then looking for the words that followed that word in the original tweets. It picks **one** of these subsequent words and adds that to the tweet. It then looks for the followers of this latest word, and so on. This process continues until a certain number of words have been selected. This is a form of auto-complete. The tweet generator chooses a 'likely' next word for any given word.

The `TweetGenerator.java` file provided has a main method so you don't need to write that. It also loads the words from the file 'cleaned.txt' that was generated by `TweetCleaner` and passes those words to the method that you will write below.

You will write a method that looks through the list of words from the file and creates another `ArrayList` of `Word`s. For each particular word in the 'cleaned.txt' file it creates one `Word` object. For example, the word "the" will appear many times but is represented by only one `Word` object. The number of times that the word appears in the file is also stored in the `Word` object as the *frequency.*

Your method will need to find the 'followers' of each word. The followers of a word are the immediately subsequent words each time that word appears in the text.

Consider the following sequence of words:

"From the day I announced I was running for President I have never had a good Poll."

We only look at two words at a time. The first word in this sequence is "from". We then look at the one word that follows it. That word is "the". This means we create a `Word` object for the word "from" (if it's the first time it has occurred). We then add the word "the" as a follower of the word "from".

Next we look at the word "the". In this case it is followed by the word "day". So we create a `Word` for "the" and add "day" as a follower. We continue along the whole sequence of words in this manner. In the above example, the word "I" appears three times. Its `Word` object (of which there will be only one) will have a list of followers comprising: "announced", "was" and "have".

If a word already exists in our list of words, then we don't create a new `Word` object the next time we encounter it, we simply increment the frequency of the existing `Word`. After the above small sample sequence of words, the word "I" would have a frequency of 3. The frequency of all of the other words in that sequence will be 1.

The class `Word` is provided for you complete and ready to use. You do not need to change this file. Read the `Word.java` file to see the comments above each data member and each method to see how the class works.

Once the list of `Word`s has been created, tweets can be generated. Tweets are generated as follows. A random start `Word` is selected. That will be the first word of the tweet. The next word is selected from the followers of that `Word`. The followers of the word will be a list of words (as `String`s). For this exercise you need only select a random word from that list. A method in the `Word` class is already provided for you to do that. The randomly selected word is the second word in the tweet. We then select a random word from the follower list of the *second* word, and so on, until a limit is reached.

You must add the following methods to the `TweetGenerator` class.

`public ArrayList<Word> findWords(ArrayList<String> cleanedWords)`

This method looks through the provided `ArrayList` looking for the words. When it finds a word it has never found before, it should create a `Word` object and add it to another `ArrayList` (of `Words`). When it finds a word it has found before it should increment the frequency count of that word. This method also needs to add the followers of each word to the relevant `Word` object. It should then return the `ArrayList` of `Words`.

Make sure you fully read the comments in the file `Word.java` to get all of the information needed for this exercise.

`public String createTweet(int numWords)`

This method creates a tweet, as described above, and returns it as a `String`. The parameter `numWords` specifies how many words should be in the tweet.

Extension activity: Better Tweet Generator [0 marks]

This is an extension activity. There are no marks for this and no tests, so it will not be marked.

Consider how the tweets could be made more realistic. For example, in Exercise 2, the words in the tweet are selected at random from lists of followers. However, this is not particularly realistic. It would be better if words were selected at a rate proportional to the likelihood that they follow a word. For example, if the word "fake" is followed 100 times by the word "news" and 2 times by the word "smile" it seems wrong that "news" and "smile" should have an equal chance of being selected. It is not the case that, in this example, "news" should always be selected but it should have more chance of being selected.

You can create what is called a 'Markov Chain' to represent the relationships between words, and the associated probabilities.