ASSIGNMENT 5

COMP-202, Fall 2017, All Sections

Due: Wednesday, December 6^{th}, 11:59pm

Please read the entire PDF before starting. You must do this assignment individually.

Question 1:	100 points
	100 points total

It is very important that you follow the directions as closely as possible. The directions, while perhaps tedious, are designed to make it as easy as possible for the TAs to mark the assignments by letting them run your assignment, in some cases through automated tests. While these tests will never be used to determine your entire grade, they speed up the process significantly, which allows the TAs to provide better feedback and not waste time on administrative details. Plus, if the TA is in a good mood while he or she is grading, then that increases the chance of them giving out partial marks. :)

Up to 30% can be removed for bad indentation of your code as well as omitting comments, or poor coding structure. Marks will be removed as well if the class and method names are not respected.

To get full marks, you must:

- Follow all directions below
- Make sure that your code compiles
  - Non-compiling code will receive a very low mark
- Write your name and student ID as a comment in all .java files you hand in
- Indent your code properly
- Name your variables appropriately
  - The purpose of each variable should be obvious from the name
- Comment your work
  - A comment every line is not needed, but there should be enough comments to fully understand your program

Part 1 (0 points): Warm-up

Do **NOT** *submit this part, as it will not be graded. However, doing these exercises might help you to do the second part of the assignment, which will be graded. If you have difficulties with the questions of Part 1, then we suggest that you consult the TAs during their office hours; they can help you and work with you through the warm-up questions. You are responsible for knowing all of the material in these questions.*

Warm-up Question 1 (0 points)

Write a program that *opens* a **.txt**, *reads* the contents of the file line by line, and *prints* the content of each line. To do this, you should look up how to use the `BufferedReader` or `FileReader` class[1]. Remember to use the `try` and `catch` blocks to handle errors like trying to open an non-existent file. A sample file for testing file reading is found in the provided files as *dictionary.txt*.

Warm-up Question 2 (0 points)

Modify the previous program so that it stores every line in an `ArrayList` of `String` objects. You have to properly declare an `ArrayList` to store the results, and use `add` to store every line that your program reads in the `ArrayList`.

Warm-up Question 3 (0 points)

Modify your program so that, after reading all the content in the file, it prints how many words are inside the text file. To do this, you should use the `split` method of the `String` class. Assume the only character that separates words is whitespace `" "`.

Warm-up Question 4 (0 points)

Create a new method in your program which takes your `ArrayList` of `Strings`, and writes it to a file. Use the `FileWriter` and `BufferedWriter` classes in order to access the file and write the `Strings`. In the output file, there should be one `String` per line, just like the original file you loaded the `ArrayList` from.

Warm-up Question 5 (0 points)

Create a new method in your program which takes as input your `ArrayList` of `Strings`, and sort all the elements. The sorting criterion will be the length of the string. In other words, after calling this method, the shortest string must be located in the first position, the second shortest in the second position an so on.

Warm-up Question 6 (0 points)

Create a new method in your program which takes as input a sorted `ArrayList` (see the previous question for details about the sorting criterion) and two `ints`. The two `ints` will represent a range of values. This method should return an `ArrayList` with all the Strings whose length is inside that range. For example, if your original `ArrayList is equal to` {`"aa","aaa","aaaa","aaaaa"`} and the two `ints` are `3` and `4`, your method must return the `ArrayList` {`"aaa","aaaa"`} (because the length of the returned `Strings` is within `3` and `4`).

[1]The documentation on the `BufferedReader` class is available at `http://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html`. You can find an example on how to use it at `http://www.tutorialspoint.com/java/io/bufferedreader_readline.htm`

Part 2

The questions in this part of the assignment will be graded.

Question 1: Twitter (100 points)

For this question, you will write two classes to create a new data type that will model information extracted from twitter. Your code for this question will go in two `.java` files.

We *strongly recommend* that you complete all the warm-up questions before starting this problem.

Note that in addition to the required methods below, you are free to add as many other `private` methods as you want.

(a) Tweet Class

`Tweet.java` stores information and behaviours about a tweet. A tweet is a message sent using the social network Twitter. For this assignment, the tweets are defined as messages from 1 to 15 words (after omitting stop words). The `Tweet` class stores information about a tweet: the user who tweeted it, the date of the tweet, the time of the tweet, and the actual tweet message.

The *Tweet* class should contain the following `private` attributes:

- A `String` userAccount;
  This attribute stores the user id.
- A `String` date;
  This attribute stores the date on which the tweet was tweeted.
- A `String` time;
  This attribute stores the time on which the tweet was tweeted.
- A `String` message;
  This attribute stores the tweeted message

The *Tweet* class should also contain a `private static` attribute:

- An `HashSet<String>` stopWords.
  This attribute stores the set of stop-words. A stop word is a commonly used word that is ignored by search engines[2]

Regarding the behaviour of the class, `Tweet` must implement the following methods:

- A constructor that takes as inputs four `String`s. The `String`s correspond to: *i*) the user account *ii*) the date on which the tweet was posted. The format of the date is `YYYY-MM-DD`. *iii*) the time on which the tweet was posted. The format of `time` is `HH:MM:SS` and *iv*) the message of the tweet. Please notice that your arguments must respect the previous order. The constructor does not have to check the format of the inputs.
- A public non-static `checkMessage` method which takes no input. This method will check if the message of *this* `Tweet` is valid or not. The method returns `true` if if the message contains less than 16 words and more than 0 (excluding the stop-words). If the HashSet `stopWords` is equal to `null`, `checkMessage` must throw a `NullPointerException` indicating that the HashSet has not been initialized.
  Here a couple of hints.
  - You can use the `split` method of the `String` class to separate the words of the message. For this assignment, you can assume that the whitespace will be the only character that separates words [See warm-up question 3]).

[2] `https://en.wikipedia.org/wiki/Stop_words`

- You can use the `equalsIgnoreCase` method to make sure not to miss stopWords typed in either upper or lower case).
- You will also have to control cases where the stopWord might have a punctuation character appended to it [e.g., both "any" or "any," should be considered]. For this assignment, you can assume that the only punctuation characters appended to stop-words are ',', '.', ';' or ':'.
- You can create a `public static` method `isStopWord` that takes a `String` as input and returns `true` if it's a stop word, `false` otherwise. NOTE: if you do not create this method, then make the attribute `stopWords public`. This is the only additional `public` method you can write, all other helper methods should be `private`.

- `getDate`
  This returns the `date` instance variable.
- `getTime`
  This returns the `time` instance variable.
- `getMessage`
  This returns the `message` instance variable.
- `getUserAccount`
  This returns the `userAccount` instance variable.
- `toString`
  This method returns a new `String` that is the concatenation of the `userAccount`, a tab character, the `date`, a tab character, the `time`, a tab character, and the tweeted `message` in the end.
- `isBefore`
  This method takes as a parameter an instance of `Tweet` and it returns `true` when `this` tweet was posted at an earlier time than the input parameter. Otherwise the method returns `false`. Remember that the formats of the `date` and `time` attributes are `YYYY-MM-DD` and `HH:MM:SS`, respectively.
- `loadStopWords`
  This static procedure `loadStopWords` gets as input the name of a file that contains a set of stop words. The file has only one column containing a stop word in each line. This method should read the file and initialize the `static` attribute `stopWords`. This method has no `return` value.

(b) Twitter Class

`Twitter.java` stores information and behaviours generated by the twitter social network [3].

The *Twitter* class should contain the following private variable:

- `ArrayList<Tweet> tweets`
  This attribute stores a collection of valid tweets.

Regarding the behaviour of the class, `Twitter` must implement the following public methods:

- `Twitter`
  This constructor takes no arguments and it initializes `tweets` as an empty ArrayList.
- `loadDB`
  `loadDB` takes as argument the name of a file (i.e., its relative path) that contains a collection of tweets. Particularly, we provide to you the file called `tweets.txt`, which has four columns that are separated by tabs. The first column contains the `userAccount`, the second and third contain the `date` and `time` on which the post was made, respectively. Finally, the fourth column

[3] `https://en.wikipedia.org/wiki/Twitter`

corresponds to the posted message itself. `loadDB` must read the file line-by-line (please see the warm up questions), construct the corresponding `Tweet` object, if valid, the `Tweet` should be added to the ArrayList of `Tweets` (Hint: remember that you already coded a method to check if a tweet is valid or not). If the tweet is found to be not valid, your program should simply not put it into the ArrayList of `Tweets`. Once, all the valid tweets are stored in the ArrayList `tweets` `loadDB` must call the function `sortTwitter` to sort the list.

- `sortTwitter`
  `sortTwitter` takes no arguments and it sorts (in increasing order) the `Tweet` instances based on their date/time of publication. Here, you are free to select the sorting algorithm. This method has no `return` value.
- `getSizeTwitter`
  This returns the number of tweets in the data base.
- `getTweet`
  This method receives an index and returns the tweet stored at that index.
- `printDB` It returns a `String` that contains all the elements currently stored in the ArrayList `tweets`. The format of the output must be the same as the one used in the `tweets.txt` file (i.e., the first column contains the `userAccount`, the second and third contains the `date` and `time` the post was made, respectively. Finally, the fourth is the posted message). (Hint: use the toString method from the `Tweet` class)
- `rangeTweets`
  `rangeTweets` gets as parameters two instances of `Tweet` (i.e., `tweet1` and `tweet2`) and it returns a new ArrayList of `Tweets` containing those tweet posted between the date/time of `tweet1` and `tweet2` (inclusive). You can assume that `tweet1` and `tweet2` are elements of the ArrayList `tweets`. Please notice that there is not guarantee that `tweet1` was posted before `tweet2`. The method should verify which of those two tweets is the earliest. You can assume that the ArrayList `tweets` is sorted given that `sortTwitter()` was already called by `loadDB`.
- `saveDB`
  `saveDB` gets one parameters, the name of a file. `saveDB` must write in the file the list of `Tweet` following the same format of the `tweets.txt` file. (Hint: Use the `printDB` method to obtain the `String` to write).
- `trendingTopic`
  `trendingTopic` receives no parameters and it returns the word (that is not a stop word) that is the most frequent in the tweets from the data base. The most common word is chosen from the data base by counting in how many tweet messages the word appears (i.e., if a word appears more than once in a tweet message, it is counted only once).

(c) Main Method
Your main method will not be graded. However, we strongly advise that you write a main method in your class `Twitter` to test your code. You may include it in your submission as long as it compiles. In the following lines, we will report different versions of our `main` method. Like that, you will have some test cases to test your code.

- Example 1.

```
public static void main(String[] args){
   Twitter example = new Twitter();
   example.loadDB("tweets.txt");
}
```

The result of running Example 1 is:

```
Error checking the stopWords database: The file of stopWords has not been loaded yet
```

Comments: In this example, the function `loadDB` caught a `NullPointerException` thrown by `checkMessage`

- Example 2.

```
public static void main(String[] args){
   Twitter example = new Twitter();
   Tweet.loadStopWords("stopWords.txt");
   example.loadDB("tweets.txt");
   System.out.println("The number of tweets is: " +example.getSizeTwitter());
}
```

The result of running Example 2 is:

```
The number of tweets is: 58
```

Comments: The file `tweets.txt` contains initially 61 tweets. However, 3 tweets where discarded because their length was not greater than zero or less than sixteen. Particularly, the message "I can be MADE into a need." has length zero after subtracting the stop words (i.e., all the words in the message are stop words). On the other hand, the messages "USER_55cc3d0f i'm kidding. i'll die! lol jk **i** guess i'd **be** alright **but not for a** couple days **or** hours cause i'm **always a** happy spirit lol" and "RT USER_3a117437: **The** woman **at the** rental car spot tried 2 **give us a** Toyota! **No** ma'am lk **the old** spiritual **says** "aint **got** time 2 die!"" were discarded because they have 18 and 16 words (after subtracting the stop words [which are shown in bold]), respectively.

- Example 3.

```
public static void main(String[] args){
   Twitter example = new Twitter();
   Tweet.loadStopWords("stopWords.txt");
   example.loadDB("tweets.txt");
   System.out.println(example.printDB());
}
```

The result of running Example 3 is:

```
USER_a75657c2 2010-03-03 00:02:54 @USER_13e8a102 They reached a compromise
.............
.
.
.
.............
USER_a75657c2 2010-03-07 21:45:48 So SunChips made a bag that is 100%
biodegradeable. It is about damn time somebody did.
```

Comments: Here you should see the 58 elements sorted by date/time. We are just showing the first tweet (posted on 2010-03-03 00:02:54) and the last tweet (posted on 2010-03-07)

- Example 4.

```
public static void main(String[] args){
   Twitter example = new Twitter();
   Tweet.loadStopWords("stopWords.txt");
   example.loadDB("tweets.txt");
   System.out.println(example.rangeTweets(example.getTweet(4), example.getTweet(2)));
}
```

The result of running Example 4 is:

```
[USER_1e22f6a5 2010-03-03 00:20:45 RT @USER_561fe280: Nourish your .....,
```

```
USER_36607a99 2010-03-03 02:06:01 RT @USER_561fe280: Nourish your .....,
USER_c271e4ac 2010-03-03 02:07:37 I want a King Kong roll from Sushi .....]
```

Comments: Here you should obtain a sorted list of three tweets.

- Example 5.

```
public static void main(String[] args){
   Twitter example = new Twitter();
   Tweet.loadStopWords("stopWords.txt");
   example.loadDB("tweets.txt");
   System.out.println(example.trendingTopic());
}
```

The result of running Example 5 is:

```
spirit
```

Comments: The word "spirit" is the most common word and it appears in 26 tweets.

What To Submit

You have to submit one zip file that contains all your files to myCourses - Assignment 5. If you do not know how to zip files, please obtain that information from any search engine or friends. Google might be your best friend with this, and for a lot of different little problems as well.

These files should all be inside your zip. Do not submit any other files, especially .class files.

`Twitter.java`
`Tweet.java`
`Confession.txt` (optional) In this file, you can tell the TA about any issues you ran into doing this assignment. If you point out an error that you know occurs in your problem, it may lead the TA to give you more partial credit. On the other hand, it also may lead the TA to notice something that otherwise they would not.