ECS 160 Assignment 3

DUE: *November 18, 2019, 8 p.m.*

Familiarize yourself with various overloadings of `Collectors.groupingsBy()`

You will create a class `TweetProc` in package `edu.ucdavis.ecs160.hw3` with several static methods that process streams of tweets and return `Map <T, R>` or `Map <T, Map <R, S>>` data structures. Use the seg3.java file as an example to process the tweets. We will give you a sample tweet data file. Each method takes as argument a file path containing tweets, according to the csv format given in the seg3.java example, and in the attached example tweet file. Each method will have a Stream pipeline, starting with the file, streaming each csv row as a separate line, and then processing it, and consuming it all a Terminal operation that `Collect`s everything into the required `Map` data structure. We encourage you to drop a "`parallel().`" early in your pipeline to ensure it can be parallelized safely, although we won't test for it.

Submit: You will need to create a Java file, namely TweetProc.java, inside the package `edu.ucdavis.ecs160.hw3`. You are free to have helper methods in addition to the static methods listed below, either in this file or in other files in the package. We will only test the static methods we ask for below. You will zip the 'hw3' package (the deep most folder) and submit that.

You must use Streams, and you may NOT use any for, do, for each or while loops. We'll check. If you don't follow these rules, you will get zero credit. Finally, please remember to test your code with Java 13. Streams have undergone some changes since they were introduced in Java 8, so your code may run differently on older versions of Java, and we will test your code on Java 13.

READ THIS BEFORE STARTING HOMEWORK:

We have done our best to provide a Tweet file that is relatively free of offensive language. We did want to give you guys access to Twitter data that was a) large and b) as realistic as possible. As you know with public social media data it's very difficult to remove all offensive language. In general, probably best to ignore the contents as much as you can, and just let your program process it. If you happen to see a tweet that you find hurtful, please let us know privately; we'll keep your notification confidential, and then we'll just update the data file on Canvas with that tweet removed. This file has currently about 11000 tweets. If you like, you can find much bigger Twitter datasets, that might be interesting for you (e.g., NBC has a collection of tweets from politically-oriented bots, relating to the recent indictments, where Twitter removed those bot accounts.). This is a really fascinating and quite large dataset; but the tweets therein may very well include language that may offend people. However, the larger the dataset, and the more cores in your computer, the more speedup you'll see (*effortlessly*) from running your

Stream in parallel by just adding “`parallel().`” early in your pipeline. 11000 tweets is probably too small to see any parallelization speedup.

REQUIRED METHODS:

These are the methods. Each takes a filename as input and returns a `Map` data structure. Specifically, the parameter “FullPathname” is the file path to a csv file we will provide that contains all of the tweet information. The questions are equally weighted.

```
1. public static Map <String, Long> getPerTweeterCount(String
FullPathname)
```
- A “Tweeter” is the account name of the person writing the tweet, found in the “name” column of the csv file. This method should return a map where the keys are the tweeters and the values are the number of tweets they have made.

```
2. public static Map <String, Long> getPerTaggeeCount(String
FullPathname)
```
- Consider a “taggee” in a tweet as any word of text that begins with an “@” symbol. For this method, return a map where the keys are the distinct taggees and the value is the count of the number of tweets the taggees appear in. If a taggee is marked more than once in a tweet, count the tweet only once. Taggees can be found in the “text” column of the provided csv file.

```
3. public static Map <String, Long> getTweeterURLTweetCount(String
FullPathname)
```
- Return a map where the keys are Tweeters and the values are the number of their tweets that contain any URL in the “text” column. For this problem we define a URL to be any word in the text that begins with “http://”. You do not need to consider other variations like “https://:” or “www.” for the purpose of this question.

```
4. public static Map <String, Long> getTweeterWordCount(String
FullPathname, String word)
```
- Return a map where the keys are Tweeters and the values are a count of the number of tweets that contain a specific word, where this word is passed in via the “word” parameter.

```
5. public static Map <String, Double>
getTweeterAverageVerbosity(String FullPathname)
```
- Return a map where the keys are Tweeters and the values are a count of the average length of each Tweeter’s tweets. The length of a tweet is defined in terms of the number

of words, where words are elements in the tweet divided by whitespace. For example, “Hi, I’m a tweet!” would have length 4.

```
6. public static Map <String, Map <String, Long>>
getTweeterTaggeeCount(String FullPathname)
```

- Return a map where the keys are Tweeters and the values are another map. This additional `Map <String, Long>` is keyed by taggees and map to the number of tweets in which these taggees occur. So the overall map counts the number of tweets by a Tweeter in which a specific tagge occurs.