CSci 1933 Assignment 4

This assignment is due on **Friday, December 2nd, 2015** at **11:55 PM**

General:

1. **You are supposed to work in group of 2 (unless exempted by the instructor), otherwise you will lose 20% of your grade.**
2. **When an assignment description tells you to give something a particular name, you must give it exactly that name. Failure to do so may result in a very low or zero score due to inability to properly run your code.**
3. **Your program must output its results in exactly the format we describe, with no additional text being written. If you insert additional output statements to aid in debugging, make sure they are removed before you submit.**

   You have the options 1) to work with the previous teammate or 2) to find a new teammate. If you have difficulties finding a teammate, please post to Student Forum on Moodle site **no later than Nov. 20th**.

1. Overview

You will use maps to create a *rough* relational database of tweets, and run some simple statistics on it.

- In Part 1, you will load all of the twitter data place it into Maps.
- In Part 2, you will query this loaded data.
- In Part 3, you will implement your own version of the Map interface and compare its performance with Java's implementations for performing the queries.

For each part you complete, print out the corresponding part before it using the format "-- Part <number> --". Make sure you match the spacing and capitalization exactly. So if you complete all three parts, you main method should print:

```
-- Part 1 --
...
-- Part 2 --
...
-- Part 3 --
```

2. Getting started

Create a new project in IntelliJ and import following files:

- build.gradle (import this into your project root folder)
- uofmtweets.dat (import this into your dat/ folder)

- TweetReader.java (For reading in twitter data; import this into your src/ folder)
- Tweet.java (Wrapper object for a tweet; import this into your src/ folder)
- TwitterUser.java (Wrapper object for a twitter user; import this into your src/ folder)

3. Part 1

3.1. First you will need to create a class called TwitterDatabase. This class will contain your constructor, main method, and several other non-static methods.

3.2. Instance Variables

3.2.1. Your TwitterDatabase will need four maps in order to run our queries.

- Map<String,TwitterUser> name2User - This map will allow us to look up a user by name.
- Map<Tweet,TwitterUser> tweet2User - This map will allow us to look up who made any given tweet.
- Map<String,Set<Tweet>> word2Tweet - This map will allow us to look up which tweets contain a given word. Note that we don't need a tweet2Word Map because the Tweet class contains the getWords() method.
- Map<TwitterUser,Set<Tweet>> user2Tweet - This map will allow us to look the tweets made by any given user.

3.3. Data Methods

You will implement several public methods to make interacting with the maps easier.

3.3.1. TwitterUser addOrGetUser(String name) - This method should check to see whether name2User contains the key name. If it does, it should return the corresponding user. Otherwise it should create a new TwitterUser and add it to the name2User map using name as a key. In this case it should also add a new HashSet to user2Tweet using the new TwitterUser as a key before returning it.

3.3.2. int addWord(String word, Tweet tweet) - This method should check to see if word2Tweet contains the key word.

- If word2Tweet does contain word, it should add the given tweet to the Set corresponding to word. It should then return the size of the Set.
- If word2Tweet doesn't contain word, a new Set should be created (using the HashSet implementation), and tweet should be added to

it. It should then add the newly created Set toword2Tweet using word as a key, and return 1 (the size of the new Set).

3.3.3. Tweet addTweet(String msg, TwitterUser user) - This method should first create a new Tweet with the content of msg and add it to the tweet2User map using user as the value. Next it should retrieve the Set for user from the user2Tweet map, and add the new Tweet to it. It should then call addWord() for each word in the Set returned by the getWords() method in Tweet, before returning the new Tweet. Note that getWords() converts the words to lowercase before returning them, so you do not need to do any conversion before passing them addWord().

3.3.4. Map<String,TwitterUser> getNameTable() - This method should return the name2User map.

3.3.5. Map<Tweet,TwitterUser> getTweetTable() - This method should return the tweet2User map.

3.3.6. Map<String,Set<Tweet>> getWordTable() - This method should return the word2Tweet map.

3.3.7. Map<TwitterUser,Set<Tweet>> getUserTable() - This method should return the user2Tweet map.

3.4. The Constructor

This method will have the signature:

```
public TwitterDatabase(String datfile)
```

The constructor for TwitterDatabase should take in the name of tweet data file (a String) as a parameter, and use it to create a TweetReader. It should then read the tweet data file, and place the contents of the file into a series of maps. More specifically:

- The constructor should take in a String and create a TweetReader from it. TweetReader works the same as it did in the assignments and labs.
- The constructor should instantiate the four maps mentioned above. For the Part 1, you will use the HashMap implementation for these maps.
- The constructor should populate the four maps with data from the TweetReader. You will do this using a while() loop that calls the advance() method in TweetReader. Inside the loop, call addOrGetUser() with the username returned by getTweeterID() in TweetReader, and call addTweet() with the TwitterUser returned by addOrGetUser() and the Tweet returned by getTweet() in TweetReader.

3.5. Main

Your main() method will also go inside of TwitterDatabase. Remeber that for Java to recognize main(), it will need the following signature:

```
public static void main(String[]args)
```

Be sure to print out "-- Part 1 --" at the top of main().

Your main() method should create a new TwitterDatabase and print out the size of the tables. Your output should look like this:

```
name table size = 966
tweet table size = 1121
word table size = 5003
user table size = 966
```

These are not the actual values. If you wish to verify the correctness of your code at this point, you will need to come up with your own tests. Also note that based on the structure of the data (each line contains a user id and tweet), we know these values should have the following relationship:

```
name table size == user table size <= tweet table size
```

4. Part 2

For the Part 2 of the assignment you will need to add the following methods to TwitterDatabase. The methods will perform queries on the data. You will also need to create the class ItemCount, which we will use to sort the results.

4.1. ItemCount

Your will need to do the following for ItemCount class.

- Create a constructor that takes an Object and an int as parameters and stores their values inside class variables. The int will be used as the counter, and the Object will be used as the key.
- Create the methods getCount and getObject to access the two instance variables which were set in the constructor.
- Have ItemCount implement the Comparable<ItemCount> interface. The compareTo() method should sort the ItemCount in descending order based on the count (If this ItemCount has a higher count than the ItemCount passed in, this ItemCount should come first). If you've forgotten how the compareTo() method works, see the Java Documentation.
- Override the toString() method inherited from Object. The overriding version of this method should return a String composed of the String returned by the Object variable's toString() method followed by a tab character ("\t") and the corresponding count.

4.2. Query Methods

You will now create a series of methods for analysing this Twitter data set in TwitterDatabase. They will all return Lists of ItemCounts. This will allow us to sort the results in descending order. Each method should sort the List before it's returned using Collections.sort().

- List<ItemCount> getTweetCounts() - This method should calculate the number of tweets made by each user.
- List<ItemCount> getWordCounts() - This method should calculate the number of times each word is used. Remember that the words have already been converted to lowercase before being placed in the map, so you do not need to make any calls to toLower() or toUpper(). You may also note that because Sets do not contain duplicates, so we are only counting each word once per tweet, which is fine for our purposes.
- List<ItemCount> getWordUsage(String word) - This method should calculate the number of times each user uses the given word. This will require you to loop through multiple maps (Hint: The user table is a good place to start). You should convert word to lowercase

4.3. Display the Queries

Be sure to print out "-- Part 2 --".

Once you have the query methods implemented, we will want to see the results. After the output from the Part 1, print the top 10 results from both getTweetCounts and getWordCounts. For getWordUsage, print out the top 10 results for "minnesota" and "university". The specific format will look like the following:

```
Top 10 results of getTweetCounts
@Brent          100
@Ken   78
@Patrick        55
@Saurav         23
...
Top 10 results of getWordCounts
from    2015
Paris   11
with    15
love    0
…
Top 10 results of getWordUsage("minnesota")
```

```
@Brent          100
@Ken   78
@Patrick        55
@Henry          23
…
Top 10 results of getWordUsage("university")
@Brent          100
@Ken   78
@Patrick        55
@Ryan           23
…
```

5. Part 3

For the Part 3, we're going to implement a map using the same algorithms we've used before, and see how they stacks up against Java's HashMap and TreeMap implementations.

5.1. Timing

First, we are going to need a metric by which to compare the implementations. We will compare the time it takes for each query to execute. Java provides us with a simple way to do this: the System.currentTimeMillis() method, which returns the current system time. We can use this to time the execution of code by getting the system time at the start of the execution, and subtracting it from the system time at the end of the execution. It returns a long primitive, which is like an int, but can hold a larger number.

```
long start = System.currentTimeMillis();
...
long duration = System.currentTimeMillis() - start;
```

In each of the three methods we implemented in the Part 2: getTweetCounts(), getWordCounts(), and getWordUsage(), print out the time it takes to execute method just before the return statement. So for example when getWordCounts() is called, it should print out:

```
getWordCounts() took 54ms to execute.
```

5.2. Linear Scan Map

Create a new class called LinearScanMap. The Map interface has 14 methods that we would need to complete if we implemented the interface directly. Fortunately,

Java provides an abstract class called AbstractMap, which provides much of the functionality for us. LinearScanMap should extend AbstractMap using generics, so your class declaration should look like:

```
public class LinearScanMap<K,V> extends AbstractMap<K,V> {
   ...
}
```

You will need to override the following public methods to complete the implementation:

- LinearScanMap() - In the constructor you will need to instantiate a Set (declared as Set<Entry<K,V>>) for internal use. This Set will store the entries for this map. You may use whatever implementation of Set you wish.
- Set<Entry<K,V>> entrySet() - This method should return the internal Set which holds all of the entries for this map.
- V put(K key, V value) - This method should instantiate a new AbstractMap.SimpleEntry using key and value, and add it to the internal Set. It should return the previous value for this key, or null if no mapping exists for the key.
- V get(Object key) - This method should iterate through the internal Set and return the value stored in the Entry that contains the given key.

5.3. TreeMap

Simply use the default TreeMap in Java.

5.4. Overloaded Constructor

Next we need our TwitterDatabase to use whatever Map implementation we want. To do this add another constructor that takes in four Maps as arguments and assigns them to the corresponding class variables. It will also need to read in the twitter data like the previously defined constructor.

```
public TwitterDatabase (  String datfile,
                          Map<String,TwitterUser> name2User,
                          Map<Tweet, TwitterUser>  tweet2User,
                          Map<String,Set<Tweet>> word2Tweet,
                          Map<TwitterUser,
                          Set<Tweet>> user2Tweet)
```

5.5. Display the Results

Be sure to print out "-- Part 3 --".

Next, after the Part 2 output, instantiate two new TwitterDatabases, one using the TreeMap implementation, and the other using our LinearScanMap implementation.

We want to compare all three implementations (including the HashMap implementation we used for the 1st and 2nd parts). Rerun the four queries defined above for each implementation without printing the results, so only the execution time is displayed. Print out a header before each implementation is run (use the headers and order specified below).

```
HashMap
...
TreeMap
...
LinearScanMap
...
```

6. Submission

Before you submit your solution, **you must create a text file called `group.txt` in your `src/` directory**. In this file, put the names and x.500 ID's of the members of your group.

Put `build.gradle` file in the project root and run `gradle run` to make sure that you codes get compiled. Next, run command `gradle tar`. This will create a tar.gz file for submission. Please make sure the `src/` and `dat/` files are in the created tar.gz file.

Submit the tar.gz to the **Lecture Moodle Site.**

---End of Assignment 3---