Project #3: Google Trend Simulation

Points: 100

Key objectives of this project are:

- Working with different Data Structures
- Designing a new solution using effective Data Structures
- Considering Time Complexities in designing solution
- Practice Using Junit Version 4 Tester
- Practice using GIT and committing after every major change
- Review and adhere to CSE department Style guide

Submission Instructions

The GitLab link of this project must be turned-in electronically via Canvas.

Ensure your program compiles successfully, without any warnings or errors.

Ensure your class and methods are commented properly.

Ensure you have tested operations of your program as indicated.

Once you have tested your implementation and added all necessary files, do the last push and submit the **clone with HTTPS link of your project3 folder** on Canvas.

The program submitted for this homework **must compile and run in order to qualify for earning any score at all**. Programs that do not meet basic requirements or just skeleton code will be **assigned zero** score, even if 99.99% of the code is already finished!

- **NOTE:** Violating of CSE programming style guidelines is an error! Your program should not have any style errors.
- **Bonus points and Delayed submission:** The gamified grading scale will be used. Refer to the syllabus to learn more about gamified grading scale.
- **Formatting & Style Guide:** Full points are reserved for programs that have:
  - shorts methods, not longer than **35 lines**
  - the code is formatted nicely. You can use the Red Hot tool on VS Code to format your code. Here is a video that show you how you can use that tool:
    - **https://youtu.be/xAGbpoTYxXM?si=ZKPT_nYE7gOSIqef**
  - your class and all methods are commented properly

Set up your Project and Initiate a GIT repository

Open the VScode and add a project inside the **cse274** folder (which is already connected to your local repository) and call it **project3** (no space, all lowercase). Download the following four files from the project#3 folder on canvas:

1. Controller.java (DO NOT CHANGE)
2. Trends.java (DO NOT CHANGE)
3. StudentTrends.java
4. TrendsTest.java (DO NOT CHANGE)

and, add them to your project under the src folder. Here are some tips as you are initiating the project:

- If you haven't before, watch this video to learn how to create a default project in VSCode: **https://youtu.be/CK3C4KXVXdk**
- Ensure that your IDE is not automatically adding any packages. There should be no extra folder under the src folder, meaning all your files must be placed directly inside the src folder, with no extra subfolders.
- If you drag and drop your files under the src folder and you are getting errors regarding it can't see/find some of your files, you can do the following:
    - Remove all the files you dropped under the **src** folder.
    - Right-click on the **src** folder, select **"New" > "Java File" > "Class…"**.
    - Create a new class and call this one **Controller**.
    - Then, copy the content of the Controller.java that you downloaded from canvas, and paste it inside the new Controller class that you just added under the src folder.
    - The goal is to create the files manually first, then copy and paste **their content**, instead of dragging and dropping them under the src folder. So, repeat the same process for the rest of the files: first, add files manually. Then, copy and add the contents.
- Since you are using a JUnit tester, you need to ensure **JUnit library version 4** is also added to your project. Here is a video that shows you how to do so: **https://youtu.be/PZC5slRkyuc**
- You also need to work with provided text files. These text files provide all the data that you need to work with. Drag all **seven** text files and drop them directly into the project3 folder **(not inside the src folder nor any other subfolders). The text files must be placed directly in the project3 folder.**
- Now that all files are added to your project, this should be a great time to do your first: *git add, git commit -m "some comments",* and *git push* to save everything on your local and remote repositories.

What is Google Trends:

Google Trends is a tool provided by Google that analyzes the popularity of search queries over time. It allows users to see how often a particular term is searched in comparison to other terms and provides insights into trends based on geography, time, and search volume. Businesses, researchers, and marketers use Google Trends to identify patterns in public interest, monitor seasonal trends, and compare the relative popularity of different topics. The tool can also help detect emerging trends, as sudden spikes in search volume often indicate growing public interest in a topic.

The Trends.java interface is a simulation of a Google Trends:

The **Trends** interface provided mimics the functionality of Google Trends in a simplified way. It allows tracking the frequency of different strings, similar to how Google Trends tracks the frequency of search queries. The **increaseCount** method updates the count of a term, just as Google Trends records repeated searches. The **getCount** method retrieves how many times a term has been tracked, and **getNthPopular** helps rank terms by popularity, similar to how Google Trends ranks trending searches. Finally, **numEntries** counts the total number of unique terms being tracked, giving an overview of how many distinct queries exist in the dataset. This interface provides a foundation for implementing a data structure that efficiently tracks and ranks trending terms, just like Google Trends does on a much larger scale.

Implement StudentTrends.java

The **Controller.java** file tests overall functionality of your code (using an object from your **StudentTrends class**) by reading in a stream of terms (coming from a text file), feeding them to your object, and querying your object at various times to get certain information of the frequency of terms. Specifically, you will need to implement these options:

- void increaseCount(String s, int amount)
  This method is called when you see a string *s*, causing the count of the number of occurrences to increase by *amount*.
- int getCount(String s)
  This method will return the number of times string *s* has been seen. (That is, the number of times it's been an argument to increaseCount.)
- String getNthPopular(int n)
  This method returns the string that is the *n*th most popular term seen. That is, if *n=0* it should return the most popular string (the string with the highest count), if *n=1* it should return the *2nd* most popular string, etc... (Clearly, what string is the most popular might change over time as *increaseCount* is called more times.) Ties should be broken alphabetically. (So if "Marvin" and "Eddie" have each been seen once, getNthPopular(0)

should return "Eddie", because “Eddie” is smaller than “Marvin” the way strings are compared in Java.)

- int numEntries()
  The number of *unique* entries in the table. That is, the number of different strings we have seen. (So if "Prefect" has been entered twice and "Dent" has been offered four times, and nothing else has been entered, numEntries() should return 2.)

Project Rules & Considerations:

- What data structure will you use to store the strings that you are keeping track of? The only data structures you should use are **ArrayList**, **LinkedList**, and **HashSet** that are already imported for you inside the **StudentTrends.java** file. You should use the combination of those for the best result. Or, combine the techniques those data structures are using behind the scene to make your own data structure.
- You are also allowed to use **Arrays**, **String**, and **all primitive data types**.
- You are not allowed to import any other data structures in your code.
- Do not add any package to your project. PLEASE!
- When will you sort them? Will you keep them sorted on the fly or will you only sort them when you need them to be sorted? Or will you use some combination of both?
- You are not allowed to use the following static methods, and other methods similar to them: **System.arraycopy()**, **Arrays.copyOf(), Arrays.sort(), Collection.sort(), etc.** In other words, you are not allowed to **copy** elements or **sort** elements using built-in java library methods. Do your own search about what sort method could be more helpful to you, and implement it as a private method so you can only use it in your solution.
- Definitely review and study sorting algorithms. We are working with some very large files here.
- Consider the benefits of doing all of the sorting at once versus keeping everything sorted. On one hand, values might be moving up and down a lot, which is more work, but on the other hand, if everything is already sorted, it's easier to insert something into a fully sorted list.

- Feel free to add as many **classes\methods** as needed, and try to keep your methods short. No method should exceed **35 lines** including: comments, brackets, and empty lines.

- Where will you keep track of how many times the string has been found already? You could keep the values in separate data structures and try to keep track of both of them, you could keep them in a single struct, or you could come up with some other solution. You could even use regular arrays, if you can manage the book keeping properly.

- Keep in mind, speed does count in this Project. If your function runs properly, but moves at a glacial pace, it will get a few points off. That being said, if you are clever, you could get it to run very fast.

Test your code:

- Once you're done, use the **TrendsTest.java** to test your code with.
- You can also run the **Controller.java** file (has the main method in it) to check the timing of your code. If your code takes a long time to run, you will lose some points.

Bonus Points:

The first goal is to get this working. The second goal is to get this working *fast*.

There will be bonus points for fast implementations. Specifically, we will run a variant of the Controller on your code (It won't be exactly the same, but it will test for the same things). The first 3 people (the fastest / second fastest / third fastest) will get (5/3/1) bonus points. Those within **±50 ms** will be considered ties, and all ties will receive the bonus points.

You are only eligible for the bonus points if you pass all unit tests (TrendsTest.java).

Submission:

You will be submitting the **'Clone with HTTPS'** link of the **'project3'** folder on canvas. The only file that we grade and Must be in your GitLab in order to get full points is:

- **StudentTrends.java**

Rubric:

Description	**Points**
Code is pushed on Git and the Correct GitLab link is submitted	**10**
Your code passed all 17 Unit tests	**68**
You did not use any other data structures, nor did you use Sort or Copy methods from Java libraries. (If your code passed <= 15 tests, you will get 0 for this part)	**7**
Your code performs reasonably well in terms of speed. If your code takes a lot of time to get run, you will lose 10 points. (If your code passed <= 15 tests, you will get 0 for this part)	**10**
Your code is clean, and you have your name at the top, and you have short methods not more than 35 lines including: comments, brackets, and empty lines.	**5**
Bonus Points and Penalties	**0**