Code Review 2 – Rating Analyzer

The due date is Friday, 11/10/2023 at 7am PT. Submission details later in this document.

Overview

We have provided a specification for **JAMS** 1.2 – the **Java API for Mathematical Statistics**. The spec consists of an interface definition and API documentation for package `org.stats`. You are in the role of ***provider***, which means you are to implement this specification. (In technical terms, this simply means you will write an implementation class for the `org.stats.RatingAnalyzer` interface.)

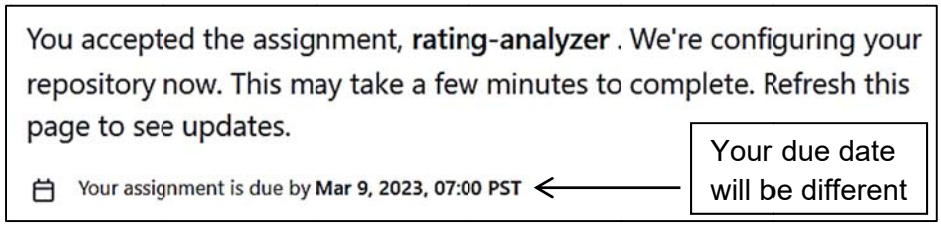
Your submission should include a `RatingAnalyzer` implementation class, along with any supporting classes (if any). Your class(es) should (of course) be in a package, e.g., `com.yourname.stats`. **Your solution should also include unit tests.** This is likely going to involve only two classes: your implementation class and a test class.

Getting Started

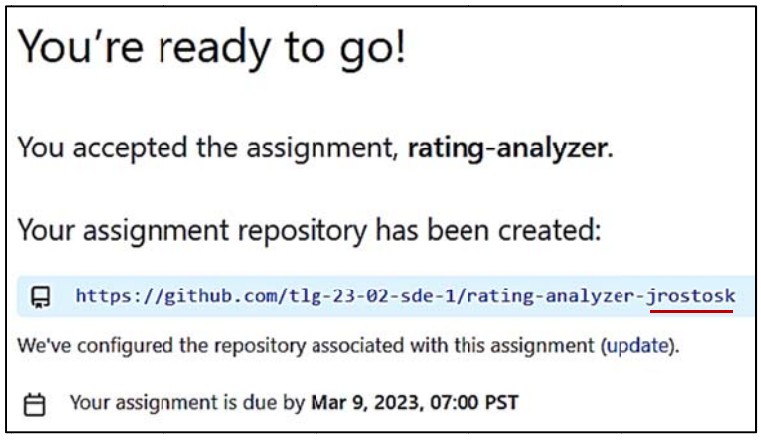
We've created a GitHub Classroom assignment for this work, located at the following URL:

https://classroom.github.com/a/xahC-LgY

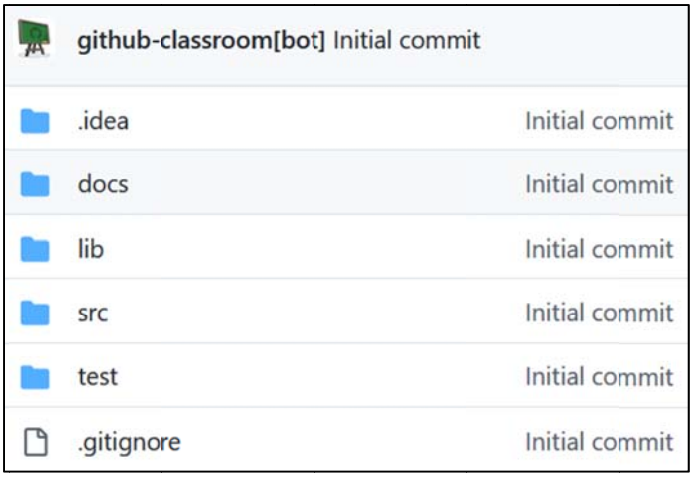
Log in if necessary, and then accept the assignment, at which point GitHub Classroom will create a repository for you – this process takes about 10-30 seconds.



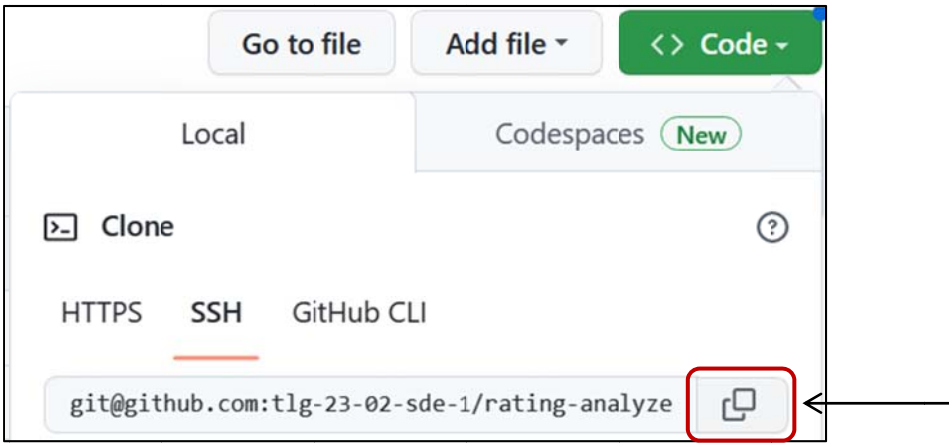
Refresh the page, and you'll see the URL to your personal repository for this assignment.
Note that your personal repository is named in the pattern "rating-analyzer-<github-username>".



Click on the URL above, which takes you to the repo – partial screenshot shown below.



Click the green **Code** button and copy the repo's SSH URL to the clipboard.

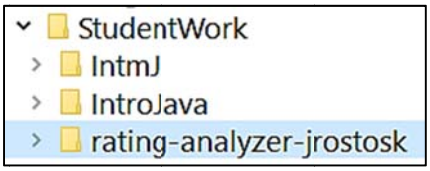


Open a Windows Command Prompt, macOS Terminal, or git-bash shell to your ***StudentWork*** directory, the same one in which your *IntroJava* and *IntmJ* filesets are located.

Clone the repo to your local machine. The `$` sign below is a generic representation of a prompt. You do not type the `$` sign in the command. Your URL will, of course, be different than this one.

```
$ git clone git@github.com:tlg-23-02-sde-1/rating-analyzer-jrostosk.git
```

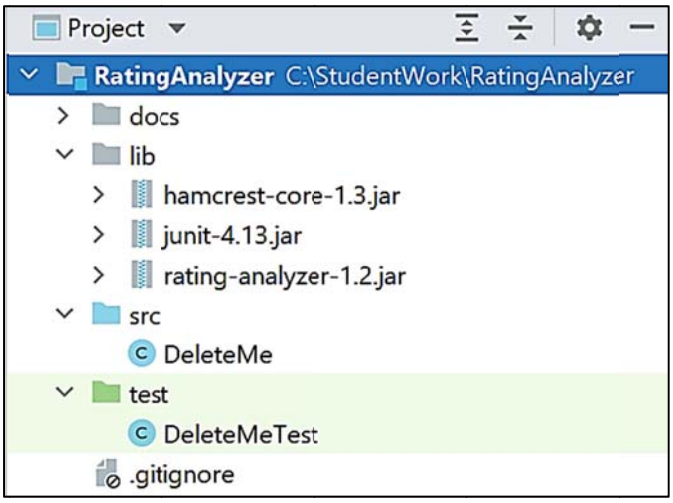
If prompted, provide your SSH key passphrase. The repo should be cloned to your machine, into a subdirectory of your *StudentWork* directory (the directory in which you issued the command).



NOTE: this local repo contains a fully-configured IntelliJ project, including the JARs for both JUnit 4 and the JAMS 1.2 specification, the JAMS API docs, pre-configured *src* and *test* folders, a *README.md* file, and this document itself.

OPTIONAL (but recommended)**:** you can rename this *rating-analyzer-yourname* directory to just *RatingAnalyzer* if you'd like. This repo is a single-module IntelliJ project, and its sole module is named **RatingAnalyzer**. IntelliJ does not require that the module name match the underlying directory name, but if you prefer that it does, simply rename this directory on the filesystem to *RatingAnalyzer*. This will not change your GitHub repo URL, nor affect the ability to push to the remote repo on GitHub.com.

Launch IntelliJ, close any open project, press the **Open** button, navigate to the repo directory, and **OK** to open the project – partial screenshot shown below. Note that the optional step above is reflected in this screenshot.



Delete the `DeleteMe` and the `DeleteMeTest` classes – these were provided as placeholders for the *src* and *test* directories.

Implementing the Specification

Read the API docs! Start at *docs/index.html* (if that isn't obvious). Alternatively, the *README.md* file has a link to an online copy of the API docs. You probably won't find it overwhelming, but you **absolutely must read it**. Then proceed.

Your implementation approach is up to you, i.e., you can use arrays, collections, the `java.util.stream` package, whatever you want. However, **it must be your own work** – this is an **individual assignment** to verify that you can solve basic algorithm problems and write clean, well-formatted, well-structured Java code that adheres to standard naming conventions and fulfills basic OO principles.

Avoid any temptation to copy code from *stackoverflow* or other websites. While we don't want to spend time tracking down plagiarized solutions, we do have a code copy detection tool that we can use to readily determine if your code is actually not your own. Besides, this is an **excellent** opportunity to practice the very same things you'll be doing at work, while getting the satisfaction that, "Hey, I did it!"

Implementation Notes

There are several websites that will calculate these statistics, so you can quickly enter in a sample dataset, get the results, and then use them as the expected values for assertions in your unit testing. This is a good one:

http://www.calculator.net/mean-median-mode-range-calculator.html

Your test class should **only** be coupled to the `RatingAnalyzer` interface, i.e., it should not have any knowledge of your exact implementation class. The static factory method **`RatingAnalyzer.newInstance()`** reads the *rating-analyzer.properties* file and instantiates your implementation class using a process called Reflection (the details of which you don't need to be concerned with). Therefore, in your test class, you can simply call this method to get an instance of your analyzer for testing. **Repeat:** this static factory method is already implemented in the interface itself, you **do not** need to code it. **However**, you should read the API docs for it.

Example from a test method:

```
int[] ratings = { 3, 5, 2, 3, 4, 1, 3, 4, 3 };
RatingAnalyzer analyzer = RatingAnalyzer.newInstance(ratings);
```

In IntelliJ, you need to ensure that the *rating-analyzer.properties* file can be located by the `RatingAnalyzer.newInstance()` method, which is looking for it in the base directory of that module. Run a unit test at least once, which creates a "run configuration" for that module.

Then set the "working directory" to be the module base directory, as follows: on the top menu bar in IntelliJ, choose Run → Edit Configurations, and in that dialog, set the "Working directory" text field to `$MODULE_WORKING_DIR$`. (Most likely, it is already set to this value.)

Evaluation Criteria

We will run our tests against your code, and they will need to pass. We have been very thorough in our test cases, as you should be in your own testing. We will also take a brief tour of your implementation class(es), to verify that your code meets the criteria set forth in "Implementing the Specification" above. This assignment was inspired by the Weekend Ratings poll ("How was your weekend?"), and our test cases will use ratings in the range [1-5], but your implementation should of course be able to handle any set of integer values.

Submission Instructions

You shall do your final commit(s) and push by the deadline set forth at the top of this document. Note that you can push committed changes after the deadline, but such changes will not be considered in your final submission. Don't wait until the last minute for your final push.

Getting Assistance

The instructor will help you with any structural issues you have getting started, but cannot give any implementation advice. We believe this assignment is on the right level of challenging, yet very attainable, **if** you take the time to follow the instructions, read the API docs, and get started earlier than later.

Good luck!