CS3114/5040 (Fall 2025)
PROGRAMMING ASSIGNMENT #1

Due Friday, September 19 @ 11:00 PM for 100 points

Due Thursday, September 18 @ 11:00 PM for 10 point bonus

Note: This project also has three intermediate milestones. See the Piazza forum for details.

[Last updated September 10 at 2:00pm]

Assignment:

Your assignment is to create a movie rating database and analysis system. For a given movie and reviewer (each simply identified by an integer index value), there is a rating from 1 to 10. The primary issues here are how to find ratings with various properties (find all ratings for a given reviewer, or for a given movie), or how to find a movie that is most similar to a given movie, or reviewer that is most similar to a given reviewer. The data structures problem is that each movie is rated by relatively few reviewers, and each reviewer has rated relatively few movies.

One approach might be to store a two-dimensional array where each row corresponds to information about a reviewer, and each column corresponds to information about a movie. This would let us process various rows and columns of the table on demand. Unfortunately, with any significant number of movies and reviewers, only a tiny fraction of movies will be rated by any particular reviewer, and only a tiny fraction of reviewers will rate any particular movie. This means that most entries in such a table will be empty. This is a natural situation to implement what is known as a *sparse matrix*. In particular, you will implement the *orthogonal list* representation shown in the figure in Module 15.1 of the OpenDSA course textbook.

One of the features that your project will support is finding the reviewer "most similar" to a given reviewer, or movie "most similar" to a given movie. The definition that you will use for the "most similar" reviewer to reviewer X is to look at each of the other reviewers in the database. For each reviewer Y, look at each movie. If reviewers X and Y both rated a given movie, then add the absolute value of the difference to a sum. Once you sum up the difference for all movies that are rated by both reviewers, divide by the total number of movies sharing a review. This is the similarity score for the two reviewers. The "most similar" reviewer is the one with the lowest similarity score. If there are no movies that both rated, then define the similarity score to be -1. Computing the "most similar" movie is computed in a similar way.

Invocation and I/O Files:

The name of the program is `MovieRaterProj`. Your program will implement an interface called `MovieRater`, with this interface implemented by a class named `MovieRaterDB`. Graded reference tests for the project will work by making calls to the `MovieRater` interface. Note that this means none of the reference tests will read from or write to `System.out`, so reference tests will not conflict with whatever debug print statements you happen to implement. You may add other test files, but all tests that you add to the file `MovieRaterTest.java` should not call any internal methods of your project that were not specified here. One service that Web-CAT will provide for this project is to run any tests that you place in `MovieRaterTest.java` against the project reference implementation. If any of your tests fail against the reference implementation, that will indicate some mistake that you have made in your understanding of the project requirements.

The interface, example tests, and other starter code will be made available in the Project 1 Starter Code available from Web-CAT.

Design Considerations:

Your main design concern for this project will be how to implement the Sparse Matrix class to be as general as possible. While the similarity measure implementation will be fairly specific to this project, that does not mean that the sparse matrix needs to know about things like movies and reviewers.

Your project will need to have at least the following classes: `MovieRaterProj` to include the main routine, the `MovieRater` interface that we give to you, `MovieRaterDB` to implement the `MovieRater` interface, your test classes (including `MovieRaterTest`) and one or more classes to implement the Sparse Matrix.

Programming Standards:

You must conform to good programming/documentation standards. Web-CAT will provide feedback on its evaluation of your coding style, and be used for style grading. Beyond meeting Web-CAT's checkstyle requirements, here are some additional requirements regarding programming standards.

- You should include a comment explaining the purpose of every variable or named constant you use in your program.
- You should use meaningful identifier names that suggest the meaning or purpose of the constant, variable, function, etc. Use a consistent convention for how identifier names appear, such as "camel casing".
- Always use named constants or enumerated types instead of literal constants in the code.
- Source files should each be under 600 lines. Test files may be longer if needed.
- There should be a single class in each source file. You can make an exception for small inner classes (less than 100 lines including comments) if the total file length is less than 600 lines.

We can't help you with your code unless we can understand it. Therefore, you should no bring your code to the GTAs or the instructors for debugging help unless it is properly documented and exhibits good programming style. Be sure to begin your internal documentation right from the start.

You may only use code you have written, either specifically for this project or for earlier programs, or code provided by the instructor. Note that the OpenDSA code is not designed for the specific purpose of this assignment, and is therefore likely to require modification. It might, however, provide a useful starting point.

Java Standard Data Structures Classes:

You are not permitted to use Java classes that implement complex data structures. This includes `ArrayList`, `HashMap`, `Vector`, or any other classes that implement lists, hash tables, or extensible arrays. (You may of course use the standard array operators.) You may use typical classes for string processing, byte array manipulation, parsing, etc.

If in doubt about which classes are permitted and which are not, you should ask. There will be penalties for using classes that are considered off limits.

Deliverables:

You will implement your project using Eclipse (be sure that you have updated to a 2025 distribution) that has been set to use the Java 11 compiler (see the instructions in Module 2.4 of the

OpenDSA course textbook) and you will submit your project using the Eclipse plugin to Web-CAT. Links to Web-CAT client are posted at the class website. If you make multiple submissions, only your last submission will be evaluated unless you arrange otherwise with the GTA. There is no limit to the number of submissions that you may make.

You are required to submit your own test cases with your program, and part of your grade will be determined by how well your test cases test your program, as defined by Web-CAT's evaluation of mutation testing coverage. Of course, your program must pass your own test cases. Part of your grade will also be determined by reference test cases that are provided executed by Web-CAT. Web-CAT will report to you which test files have passed correctly, and which have not. Note that you will **not** be given a copy of these test files, only a brief description of what each accomplished in order to guide your own testing process in case you did not pass one of our tests.

When structuring the source files of your project, use a flat directory structure; that is, your source files will all be contained in the project "src" directory. Any subdirectories in the project will be ignored.

You must include in your submission (at the top level of the project) a file named `transcript.txt` that contains the transcript of any use of an LLM for the purpose of writing or debugging this project. If you did not use an LLM, then `transcript.txt` should include a statement to that effect. Projects will not be graded without this. It is considered an honor code violation to use an LLM for this project without providing the transcript file.

You must also submit a short video presentation about the project, to help the grader insure your conceptual understanding of the project implementation. Detailed instructions for this video requirement will be provided separately.

You are permitted to work with a partner on this project. While the partner need not be the same as who you worked with on any other projects this semester, you may only work with a single partner during the course of one project unless you get special permission from the course instructor. When you work with a partner, then **only one member of the pair** need make a Web-CAT submission. Be sure both names are included in the javadoc documentation. Whatever is the final submission from either of the pair members is what we will grade unless you arrange otherwise with the GTA. Note that if you work with a partner, each individual is required to submit their own video.

Pledge and Honor Code:

Your project submission must include a statement pledging your conformance to the Honor Code requirements for this course. Specifically, you must include the pledge statement that is provided with the project starter code. The text of the pledge will also be posted online. Programs that do not contain the pledge will not be graded. Use of an LLM without submitting the transcript as required is considered an honor code violation.