1.1) Brainstorming on your own, what errors do you need to handle? What do you think it means to “handle errors *gracefully*”?

- Error for if the reader is empty
- Using a try-catch block to throw a custom error
- Line-specific error reporting
- Custom errors for better clarity of errors

- In the lecture, we discussed defensive programming the difference between loud and silent errors, and how we should design our code to protect against human error while handling errors in a way that will help us find our bugs by catching errors as loud errors.

1.2) What edge cases did the LLM miss? It is unlikely that the LLM will suggest all edge cases (although it could) because it lacks the full context of your application. So please remember to ensure that your error handling will meet expectations, even if it means handling things the LLM did not suggest.

1. **Handling `Reader` Initialization**
2. **Exception Handling in Reading Process**
3. **Robust Parsing with Regular Expressions**
4. **Graceful Closure of Resources**
5. **Custom Exceptions for Better Clarity**
6. **Validation of Parsed Data**
7. **Documenting Error Handling**
   - **Update Documentation**: Clearly document all exceptions that the parser might throw and under what circumstances. This documentation will help developers using your library understand how to handle these exceptions properly.
- It got everything I was thinking of….

1.3) Write a citation for the LLM following this format and include a short note about how you used the LLM. In future assignments, if you use an LLM, you are expected to cite it similarly. Include your citation both as a response to this question and in the README.

OpenAI. (2024). ChatGPT (Version March 14, 2023) [Large language model]. https://chat.openai.com/chat/

2. Correctness

For the CSV parser to be considered correct it needs the following:

- The parser must correctly process well-formatted CSV files, converting each row into appropriate objects according to the structure specified in the CSV.
- The parser should detect rows that deviate from the expected format (e.g., incorrect column counts, invalid data types) and throw appropriate exceptions, such as `CSVParserException` or `FactoryFailureException`, with meaningful error messages.
- The parser should handle edge cases like empty rows, rows with missing data, and extra white spaces.
- It should be robust against malformed inputs without causing crashes.
- The parser should consistently return the same results for the same input every time it runs.
- Any special characters, such as commas inside quoted fields, must be handled correctly according to CSV conventions.
- The parser should ensure that any `Reader` objects (such as files) are properly closed after parsing to avoid resource leaks.
- The parser should be able to handle larger files without performance degradation, and it should not run into memory issues for moderately large datasets.
- Errors in individual rows should not cause the entire parsing process to fail if designed to skip invalid rows, but should instead throw meaningful exceptions where expected.

3. Random, On-Demand Generation

1. Stress Testing:
   - Random data generation could simulate large datasets to stress test the parser.

2. Exploration of Edge Cases:
   - random data might uncover edge cases that were not considered

3. Fuzz Testing:
   - Randomly generated inputs can expose unusual bugs by providing inputs that a developer would not have thought of.

4. Testing Robustness of Exception Handling:
   - You could use this random CSV generator to test how well the parser handles unexpected scenarios, such as invalid rows, missing columns, or mismatched data types.

4. Was this experience new? What bugs did you encounter and resolve?

1. Increased Focus on Testing:
   - This sprint had a stronger emphasis on ensuring robustness through test coverage. This made the assignment more about defensive programming and exception handling than just functionality.

2. Challenge of Handling Edge Cases:
   - One of the key challenges was dealing with various malformed CSV inputs (e.g., incorrect column counts, and invalid data types). These cases required more careful thought about how to structure error handling without overcomplicating the code.

3. Bug Encountered and Resolved:
   - A bug occurred when handling invalid star names, where exceptions weren't showing as expected. Initially, the wrong exception type was expected in the test, but the actual exception was different as I forgot that I wrapped it with another exception which was the output. This was resolved by ensuring the parser throws the appropriate exceptions with detailed error messages.

4. New Experience:
   - The use of custom exception handling (`CSVParserException`, `FactoryFailureException`) and thorough error messaging to support debugging was also more emphasized in this project than in other assignments.