CIT 5940 - Module 7 Programming Assignment
CSV Slicer

Contents

Assignment Overview

In this assignment you will read files in a format known as "comma separated values" (CSV), interpret the formatting and output the content in the structure represented by the file.

Learning Objectives

- Implement a method to extract content from an input CSV
- Read and understand a formal specification for the CSV format
- Use a "state machine"

Advice

We highly encourage you to utilize the course discussion forum to discuss the project with classmates and TAs. Before asking the question, please search in the forum to see if there are any similar questions asked before. Please check *all* suggestions listed there.

For this and all other assignments in this course, we recommend that you work on your local computer using an integrated development environment (IDE) such as Eclipse, IntelliJ, Visual Studio Code, or whatever IDE you used for any prior Java courses that you are familiar with. Although you will need to upload your solution to Codio for grading (see the Submission Section below) and could develop your solution on that platform, we recommend using industry standard tools such as Eclipse or IntelliJ so that you become more used to them and can take advantage of their features.

If you have trouble setting up your IDE or cannot get the starter code to compile, please post a note in the discussion forum and a member of the course staff will try to help you get started.

We expect that you will create your own JUnit tests to validate your solution before submission.

1 Setup

1. Begin by downloading the `CIT594-csv.zip` archive from Canvas, then extract the contents. The main file you will work in is `CSVReader.java`. This contains the unimplemented methods for the code that you will write in this assignment.
2. Compile this code before making any changes. This will ensure your local environment is configured correctly.
3. Review the current implementation. You'll want to understand the various parts of the code and how they work together before making changes!

2 Requirements

2.1 Structure and Compiling

1. You MUST use a JDK version of 17 or higher.
2. You MAY use a JDK version higher than 17, but you MUST set the Java language level to 17 for compatibility with Codio.
3. You MUST NOT change any of method signatures for any of the provided methods. This includes the parameter lists, names, and return value types, etc.
4. You MUST leave all classes in the default package.
5. You MUST NOT create additional `.java` files for your solution.
6. You SHOULD create JUnit tests to help validate your code. These tests SHOULD NOT be a part of your solution. We do expect you to test your code thoroughly with JUnit tests before submission.
7. You MUST follow our directives in the starter code comments. This means if we ask you to not change a variable's value, you MUST NOT change that variable's value.
8. You MAY create additional helper functions, as long as they meet the other requirements of the assignment (e.g. doesn't crash with invalid inputs, etc.).
9. You MUST fill out the required Academic Integrity signature in the comment block at the top of your submission files.

2.2 Functionality Specifications

2.2.1 General Requirements

1. Your method MUST NOT crash if a user enters an invalid input. Remember you do not have control over what a user inputs as arguments. For example, all methods should handle `null` inputs gracefully.

2. You MAY add supporting fields and helper methods to the `CSVReader` object as needed.

2.2.2 CSV Format

CSV is a common delimited text format for storing and transmitting tables. The name comes from the fact that it separates fields with the comma character (rows of the table are separated by line breaks).

To support values in the fields that include delimiter characters (for example, if a comma is part of the data, not just a field separator) requires some added complexity. However, there is not a single specification for the CSV format, so we will use one specific format for this assignment, RFC 4180. To simplify the assignment, we will relax some of the rules.

1. You MUST follow the specification in section 2 of RFC 4180, unless otherwise specified here.

2. **RFC 4180 Rule Clarification**:

   Formatting characters describe structure; they are not part of the field content. For example, the comma that separates two fields MUST NOT be included in either field:

   ```
   "example of using "" in a field",1
   ```

   MUST result in a single row that is equivalent to an array constructed with the Java expression

   ```
   new String[] { "example of using \" in a field", "1" };
   ```

3. **RFC 4180 Rule Clarification**:

   Escape characters in an escaped field follows the above rule. Two double quotes in the middle of an escaped field count as a single double quote character in the content.

4. **RFC 4180 Rule Clarification**:

   Regarding RFC 4180, Section 2, Rule 2, there is no additional record if the file ends at the start of a line. This includes, but is not limited to, the example in RFC 4180, Section 2, Rule 1, where the file ends with `CRLF EOF`.

5. **RFC 4180 Rule Adjustment**:

   You MUST use these additional rules that modify RFC 4180, Section 2, Rule 4.

   (a) Do not apply special treatment to the first row. You MUST NOT treat a header row differently than a record row.

   (b) You do not need to check if all rows have the same number of fields.

   (c) Commas at the end of a line signify empty fields. For example

   ```
   a,b,c,
   ```

   results in a row with four fields[1]:

   ```
   [ "a", "b", "c", "" ]
   ```

   (d) An empty line terminated by a line break is a valid row if it is outside an escaped field. Inside an escaped field, it is just part of the content of that field

6. **RFC 4180 ANBF Grammar Adjustment**:

   `CRLF = [CR] LF`

   This change makes carriage return optional everywhere `CRLF` is used.

7. **RFC 4180 ANBF Grammar Adjustment**:

   `TEXTDATA = %x00-09 / %x0B-0C / %x0E-21 / %x23-2B / %x2D-7f`

   This change expands `TEXTDATA`[2] to all characters that are not comma, double quote, carriage return, and line feed.

8. For the purposes of the assignment there are five classes of characters for you to consider:

Common Name	RFC Name	RFC Code (hex)	Decimal	Java Character
Line Feed	LF	`%x0A`	10	`\n`
Carriage Return	CR	`%x0D`	13	`\r`
Double Quote	DQUOTE	`%x22`	34	`"`
Comma	COMMA	`%x2C`	44	`,`
Anything Else	TEXTDATA			`[^\r\n,"]`[3]

[1]The line in rule 4 in RFC about "The last field in the record must not be followed by a comma" is referring to their rule that each line should contain the same number of fields. A comma at the end of a line would insert an additional, empty field. The statement indicates that an extra comma that would increase the number of fields beyond the expected limit should not be dropped or ignored.

[2]We will not test with characters above `%x7F` (127), but you are welcome to include `%xFF-7FFFFFFF` (`Integer.MAX_VALUE`) in `TEXTDATA` if you wish. That will cover many more special characters and allow you to process Unicode values.

[3]This is just a regular expression to write any other character aside from the four listed.

2.2.3 `readRow` method

Your implementation will all be in this method.

1. Each call to `readRow` MUST return one row of data from `reader` until the input is exhausted
2. If there is a format error in the input, you MUST raise a `CSVFormatException`. You MAY use the optional fields to report informative error messages for debugging purposes, but we will not evaluate them.
3. The runtime MUST be `O(n)`, where `n` is the number of characters in the input. Choose your data structures carefully: seemingly convenient operations and data structures may not be appropriate for this assignment.
4. You MUST process the input one character at time (hence the provided `CharacterReader` which is more restricted than the standard `Java.io.Reader`. You may wish to organize your code in a "state machine", detailed in the supplemental reading.

2.3 Writing Test Cases

Upon completion of each functional requirement, always conduct thorough testing of the corresponding functions. Consult the guidelines on composing JUnit tests and creating test cases in Module 1. You MUST develop JUnit tests to cover all corner cases you can think of for each of your implemented function in order to get full credit. This testing component carries a weight of approximately 5% towards the total assignment score.

3 Submission

3.1 Pre-submission Check

Before you submit, please double check that you followed all the instructions, especially the items in the Structure and Compiling section above.

3.2 Codio Submission

1. When you are ready to submit the assignment, go to the Module 7 Programming Assignment Submission item and click the button to go to the Codio platform.
2. Once you are logged into Codio, read the submission instructions in the `README` file. This should be the first thing that appears in the window.
3. Upload your solution and any JUnit test files to the "submit" folder.

4. In the menu bar, select Run JUnit Tests. This will run any unit test files that you have uploaded with your submission. Note that there are no pre-submission checks provided. However you can use this feature as a basic validation check to be sure your code compiles.
5. When you're ready to submit, go to Education and select Mark As Completed. Confirm at the prompt.
6. You will see quite a bit of output, even if all the tests pass. At the bottom of the output, starting at `YOUR AUTOGRADING RESULTS BELOW` you will see the number of successful and failed test cases.
7. If you want to update your code and resubmit, for example if you did not pass all of the tests, un-mark the assignment as complete. You will then be able to edit your code, run your JUnit tests, and resubmit.

4 Grading

4.1 Grading Overview

1. There is no peer review for this assignment. You will not be marked on code style. However, we do expect you to use best practices (comments, indentation, etc), especially if you want to review with a TA.
2. There are no hidden tests. You will be able to see the name of each test, if it passed or failed, and the associated point value.
3. Failed tests will have brief feedback about the specifics of the failure.
4. We will not provide the exact test cases.
5. You have unlimited submissions, until the due date of the assignment as noted in the Syllabus (or your approved extension).
6. The score at the due date is final. Scores will sync to Canvas relatively immediately.

4.2 Rubric Items

This assignment will be marked on 5 rubric items.

- **Trivial CSV Inputs** : 2 points
- **Advanced (Valid) CSV Inputs** : 20 points
- **Line Endings** : 14 points
- **Invalid Inputs** : 24 points
- **Speed Tests** : 24 points

Total : 84 points (+ 5 points for JUnit test cases)

5 Additional Resources

RFC 4180 (CSV Format Specification):
https://datatracker.ietf.org/doc/html/rfc4180
ANBF Grammar Rules:
https://datatracker.ietf.org/doc/html/rfc2234
`Java.lang.StringBuilder`:
https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/StringBuilder.html
`Java.io.Reader`:
https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/Reader.html

Good Luck!