Assignment 2: Process Peer Feedback

- Submit deliverables to CourSys: https://courses.cs.ca/
- Late penalty is 10% per calendar day (each 0 to 24 hour period past due).
  - Maximum 2 days late (20%)
- This assignment is to be done **individually**. Do not show another student your code to help them, and do not copy code found online. Post your assignment questions to Piazza (if posting code, make post private). You may use general ideas you find online and from others, but your solution must be your own.
- See the marking guide for details on how each part will be marked.

Revision History

- None yet

Assignment Overview

For many group projects I require students to submit feedback on their group members to help me evaluate the contribution of each student, and to provide feedback to the student from their group members. Students generate this feedback using the online Peer Feedback JSON Generator.

Your task is to create a program which finds these JSON files in a folder. It first processes them to recreate the student groups and then generates a .CSV file which the instructor can read to understand the performance of each student in the groups.

General Requirements

- You must create an object oriented system of your own design containing at *least* 3 non-trivial[1]. Each class should demonstrate strong cohesion: the class has one guiding purpose, and all methods and fields of the class fit well in the class.
- Put all of your classes in the package `ca.cmpt213.as2` (this helps with marking).
- Your code *must* conform to the programming style guide linked from the course website.

Error Handling

- The input JSON files often have errors. Your program should handle each of the error cases listed in this document.
- Some of the error cases partially or fully overlap. In such cases your program needs to generate ***a*** correct error message; however, *which* error you generate is not critical. So it is OK if there are some error types which your program never reports because it always detects it as a different (and 100% reasonable) error.
  - For example (unrelated to the assignment):
    If an elevator can hold 100kg or 100 adults, it may not be possible to ever test with 101 adults in the elevator because they will always weigh more than 100kg. So in this case the system might never say “too many adults” because it always says “too much weight” first.
- When an error is detected:
  - Display a message stating the error,
  - Display the filename and path of the problematic JSON file (if applicable), and
  - Exit the program with exit code -1 (without creating any output files when possible):
    `final int FAILURE = -1;`

1 Must encapsulate some reasonable functionality. For example, a 3 line `Exception` or `FileFilter` class is trivial.

```
System.exit(FAILURE);
```

- If multiple errors are encountered, you may either print one error message and exit, or print multiple applicable error messages and exit.

1. Input Data

- Your main class must be named **`PeerFeedbackProcessor`** (i.e., it contains your `main()` function) and be in the package `ca.cmpt213.as2`.
- Your main class must accept two command line arguments:
  1. Path (relative or absolute) to the input *directory* containing the input JSON files.
  2. Path (relative or absolute) to the output *directory* where the generated CSV will be placed.

  - If an invalid number of command line arguments is supplied (too many or too few), then print out what arguments are expected and exit the program.
  - If the input folder (for .JSON files) or the output folder (for the generated .csv file) do not exist then print an error message and exit.
  - You may assume that each argument contains no spaces.
- Search the input folder (recursively) for any JSON files and read them into your program.
  - You must use a `FileFilter` object for filtering the files found by `File.listFiles()`.
  - You may use any approach you like for reading JSON files. I recommend using the GSON library.
  - JSON files end in the .json extension, case insensitive. Other files must be ignored.
  - JSON input files are expected to have been generated by the online Peer Feedback JSON Generator.
  - The first student listed in the "group" in the JSON file is the student who submitted this file.
- Generate an error if any of the following errors occur:
  - Incorrect number of arguments provided to program.
  - Input folder does not exist.
  - No JSON files are found.
  - Bad JSON file format, or missing required fields. OK to not check if file has extra fields.
  - Sum of scores in the file is not (20 * number of group members), with a tolerance of 0.1. i.e.: `Math.abs(sum - (20 * N)) < 0.1`, where *N* is the number of students in the "group".
  - Any score is less than 0.

2. Grouping

- Analyze the data from each student's JSON file to recreate the groups (i.e., infer who is grouped with whom based on who they provide feedback on).
  - Note that each student's JSON input file has an entry for not only themselves, but also each other group mate.
  - Grouping algorithm:
    - Student *S* belongs to group *G* if group *G* already contains another student *R* such that *R* provided feedback for *S*.
    - If a students *S* belongs to no existing groups, place that student in a new group.
  - Errors:
    - It is an error if student *S* is already found in a group (i.e., *S* is not just mentioned in the feedback by another student, but *S* is actually found in the group already). This likely means we have a duplicate JSON file for that student.
    - It is an error if student *S* is mentioned in feedback by students who are in two different groups. (i.e., *S* can only be a member of one group).
    - It is an error if any student who is mentioned in the feedback of another student in the group fails to submit a JSON feedback file.
    - It is an error if any student in a group does not provide feedback about all other students in that group.
  - Two students are the same if they have the same email address.
    - Ignore spaces at the start of end of an email address (hint `String`'s `trim()` function)
    - Comparison is case insensitive (hint `String`'s `equalsIgnoreCase()` function)
    - Assume email addresses are all just user ID's and don't include "@sfu.ca".

3. Output Result

The second argument to the program is the path for the output folder. Into this output folder create a comma seperated value file (.csv) name `group_feedback.csv`. The file will contain:

- For each group, generate a one row header that just says "Group 1" or "Group 2",… (1 indexed)
- For each student *S* in the group:
  - Generate one row for each other student *R* in the group. This row should list the email of *R*, the email of *S*, the score *R* gave to *S*, and the feedback *R* submitted about *S*.
  - Generate one row showing the score *S* gave themselves, and the feedback *S* put for themselves.
  - Generate one row showing the average score S's group mates gave them (excludes the score they gave themselves), and showing S's private feedback to the instructor
- Output Format Constraints
  - Use the following for the column headings in your CSV file:
    `Group#,Source Student,Target Student,Score,Comment,,Private`
  - Sorting:
    - Inside the groups, output students in alphabetical order (sorted by email address, case insensitive).
    - For each student, output the feedback about that student in alphabetical order (sorted by email address of student who provided the feedback, case insensitive).
  - Don't add any additional header information (such as date generated, …)
  - Try to match the expected output file format as closely as possible as an automated file difference tool will likely be used during marking.

- See sample output on course website.

Column description

- Group Number
  - This column only has data for the "header" row for each group.
- Source Student
  - Email address of the "other" student *R*.
  - For the rows that display *S*'s score of themselves, and S's average score, display "`-->`".
- Target Student
  - Email address of student *S*.
- Score
  - Score assigned to the student.
  - Display as a floating point number with 1 digit after the decimal point.
  - For the row that displays S's average score, display "`avg <score> /<n>`" where:
    <*score*> is the user's average score rounded to 1 digit.
    <*n*> is the number of other group members who contributed scores to the average.
- Comment
  - Display the feedback associated with this row.
  - Any quote (") characters in the feedback must be replaced with apostrophes/single-quote(').
  - Feedback must be written to file surrounded by quotes, such as for the feedback:
    ```
    He said, "Hi!"
    ```
    write to the file:
    ```
    "He said, 'Hi!'"
    ```
- Blank Column
  - This column is blank.
  - Hint: Just print an extra "," in your CSV row.
- Private Comment
  - Private feedback entered by student.
  - Same formatting requirements as the Comment column with respect to being in quotes and replacing quotes with apostrophes.

Error Handling

- Generate an error when the output file cannot be created. This will happen in cases such as:
  - file already exists and is a directory (not a file)
  - invalid file or path name
  - unable to create file / folder (locked, …)

4. Deliverables

Submit a ZIP file of your full IntelliJ project to CourSys: https://courses.cs.sfu.ca/
See course website for directions on creating and testing your ZIP file for submission.
For using GSON, follow the directions for setting up your project in the video linked on the course website.

Your project must:

- Import external dependencies (such as GSON) via Maven, or else included them in your project.
- Generate a JAR file as part of the build process. (See video on course website).

Remember that all submissions will automatically be compared for unexplainable similarities.