

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, do not copy code found online, and do not post questions about the assignment online. Please direct all questions to the instructor or TA: cmpt-213-help@sfu.ca; 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.
- ◆ Assignment must be done in **IntelliJ**.

Revision History

- ◆ Rev 2 (Jan 17):
 - Posted updated output files: replace " in student confidential comments with ', removed extra header data, sorted students correctly.
 - Added requirement for using a `FileFilter`.
 - Added details of input JSON files: some errors to check, how to know who submitted file.
 - Added details of errors during grouping.
 - Added details on output file format (data sorting, headers, cell formatting).
 - Clarified deliverables: require a JAR.
- ◆ Rev 3 (Jan18):
 - When exiting due to an error, use exit code -1:


```
final int FAILURE = -1;
System.exit(FAILURE);
```
 - If there are multiple different errors encountered during execution, it is fine to either:
 - Print one applicable error message and exit
 - Print multiple applicable error messages and exit
 - Assume program arguments contain no spaces.

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 and processes them recreate the student groups. It 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. You must have at *least* 3 non-trivial¹ classes by the time you are done. 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 for this course: <http://www.cs.ca/CourseCentral/213/bfraser/CodeStyleGuide.html>

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

1. Input Data

- ◆ Your main class must be named `PeerFeedbackProcessor` (i.e., it contains your `main()` function).
- ◆ Your main class accepts two command line arguments:
 1. Path to the directory containing the input .JSON files.
 2. Path 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 $(20 * \text{number of group members})$, with a tolerance of 0.1. i.e.: $\text{Math.abs}(\text{sum} - (20 * N)) < 0.1$, where N is the number of students in the “group”.
 - Any score is less than 0.
- ◆ If there are any errors with the input data then:
 - 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.
 - OK to display multiple errors before exiting (if multiple problems with input data detected).

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 student 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.
 - ▶ In the event of an error:
 - Display a message stating the error,
 - Display the filename and path of the problematic JSON file (when possible), and
 - Exit the program with exit code -1, without creating any output files.
 - OK to display multiple errors before exiting if multiple problems are found.
 - Two students are the same if they have the same email address.
 - ▶ Ignore spaces at the start or 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

Generate a comma separated value file (.csv) which contains the following columns.

- ◆ 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 lower-case email address).
 - ▶ For each student, output the feedback for that student in alphabetical order (sorted by

- lower-case email address of student who provided the feedback).
- 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 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

- ◆ In the event of any of the following errors, display an error message and exit the program with exit code -1:
 - Output file given by user cannot be created, such as already exists and is a directory (not a file), or invalid file name, or unable to create (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.

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).

If using GSON, you don't need to create the `gson/` folder and manually download the `gson.jar` file (contrary to my earlier email suggesting you do this to run the application from the command line). Instead, using the JAR simplifies this.

Please remember that all submissions will automatically be compared for unexplainable similarities.