Lab 1, Part 2: Java and JSON and Aggregation?

Due date: January 12, beginning of lab period.

Lab Assignment

Assignment Preparation

This is a pair programming lab. You are responsible for finding your teammates for this assignment, and you need to do it fast. I will assign partners for everyone who has not been able to find one.

For this lab, you retain the same partners as for Lab 1-1. If your partner dropped the course please let me know ASAP and I will help find a new pair/team.

The Task

For Lab 1-1 you are building two JSON generators, one for a logging actions in a putative[1] puzzle game BeFuddled, and one for generating text messages for a fictional text messaging application ThghtShre.

In Lab 1-2 you will develop parsers that

- Read back the JSON collections you created
- Compute some aggregate statisitcs and output them

BeFuddled JSON Reader

You will write a Java program `beFuddledStats.java` that performs the following actions.

[1]I cannot believe there actually **is** a game with this name

BR1. The program takes as input the name of a JSON file containing JSON objects created by your Lab 1-1 BeFuddled JSON generator. Optionally, the program shall take a second filename, indicating the name of the file for the output. See Requirement **BR10** for the description of what to do with the second parameter.

BR2. The program shall parse the JSON objects from the input file and scan them one-by-one.

BR3. The program shall collect a number of aggregate statistics based on the input file content. Remember that your JSON files for BeFuddles represent an activity log of users playing BeFuddled games. The aggregate statistics you need to collect are described in individual requirements below. You are responsible for setting up the appropriate Java data structures to store the statistics. Your program shall finish computation of all the statistics *after a single pass* over all input JSON objects.

(Essentially, your goal is to write a program that reads a JSON object, updates the statistics, and keeps doing it until it runs out of objects. You need to be careful however, as some information requires careful setup and processing).

BR4. **Games Stats.** Your program shall report total number of games seen in the input file and total number of completed games. Additionally, your program shall report total numbers of wins and losses.

BR5. **Points.** For the set of **completed games** in the log your program shall compute and report the averages and standard deviations for

- Total number of points in the game.
- Number of points in games won by the users.
- Number of points in games lost by the users.

BR6. **Moves.** For the set of **completed games** in the log your program shall compute and report the averages and standard deviations for

- Number of moves in a game.
- Number of moves in games won by the users.
- Number of moves in games lost by the users.

Additionally, your program shall compute and report a histogram detailing the frequency of occurrence of games where number of moves falls in the following ranges:

$[0, 15)$	$[15, 30)$
$[30, 40)$	$[40, 50)$
$[50, 60)$	$[60, 80)$
$[80, \infty)$	

The histogram shall be based on **completed games** only.

BR7. Users. Your program shall compute and report the following information about the users who played the games (only the users who played at least one complete game shall count):

1. Total number of users who started at least one game.
2. Total number of users who completed at least one game.
3. The largest number of games a user started.
4. The largest number of games a user compelted.
5. The largest number of wins a single user had (for completed games only).
6. The largest number of losses a single user had (for completed games only).

For items (4)–(7) from the list, also report the userIds of all users who reached that number.

Additionally, report the users/user who played the longest game(s)[2].

BR8. Histogram of Moves. Your program shall identify 10 most popular locations on the board that are used as the targets for regular moves in the games. This shall be computed over **all** moves from **all** games (completed and incomplete).

BR9. Special Moves. Your program shall compute and report the histogram of frequencies of the four types of special moves.

BR10. Output. Your program shall produce two types of output. First, all the information collected in requirements **BR4–BR9** shall be printed out to terminal in a nice, well-formatted, readable display. Second, as an option, your program shall create a JSON object that stores all the collected statistics in a human-readable and easily-parseable format. If `beFuddledStats` program is run with a second input parameter, the name of the output file, your program shall generate the JSON statistics object and write it to the given output file.

[2]The longest game is the game with the largest number of moves. It is possible that there is a tie and multiple games reached that number. Report all users who played such a game. This is **not** the same as ”report users who played top X longest games”...

This way, if your program is invoked as

```
$ java beFuddledStats myBeFuddledLog.json
```

the program will output the statistics to terminal. If the program invoked as

```
$ java beFuddledStats myBefuddledLog.json stats.json
```

the program will *both* output the statistics to terminal, *and* write the statistics JSON object to `stats.json`.

BR11. The program shall perform exactly one pass over the JSON objects in the input file. That is, your program is allowed to access each JSON object (its Java representation) exactly once during its work.

BR12. Error-checking. Minimally acceptable graceful error-checking includes:

- Check for the number of command-line arguments. If the program is run without command-line arguments, output a help prompt.
- Check for presense of input files.
- Check that the JSON objects contained in the file are indeed BeFuddled log records. If the JSON format is incorrect (e.g., the program is run on the ThghtShre JSON object collection), the program shall report the format error and gracefully exit.

ThghtShre JSON Reader

You will write a `thghtShreStats.java` program that performs the following actions.

TR1. The program takes as input the name of a JSON file containing JSON objects created by your Lab 1-1 ThghtShre JSON generator. Optionally, the program shall take a second filename, indicating the name of the file for the output. See Requirement **TRxx** for the description of what to do with the second parameter.

TR2. The program shall parse the JSON objects from the input file and scan them one-by-one.

TR3. Basic stats. The program shall compute and report the following information[3]:

[3]Here and everywhere, as a "word" please count **any** token that has been generated by your `thghtShreGen` program, including any punctuation - such as present, for example in `sense.txt`.

1. Total number of messages reported.
2. Total number of unique users who authored the messages.
3. Average length (in terms of number of words, and in terms of number of characters) of a message, as well as the standard deviations for each computed average.

TR4. Message Type Histrograms. The program shall compute and report the following histograms.

1. Distribution of number of messags by status.
2. Distribution of number of messages by recepient.
3. Distribution of number of message by whether or not a message is in response to another message.
4. Distribution of number of messages by the number of words in a message. (I.e., for each possible length of a message expressed as number of words/tokens, find the number of messages that had that length).

TR5. Stats for subsets of messages. The program shall compute and report the following information:

1. For messages of each status, report the average length of the message (both as number of words and number of characters), and the standard deviation.
2. For messages of each type of recepient, report the average length of the message (both as number of words and number of characters), and the standard deviation.
3. For messages written in response to another message, and messages NOT written in response to another message, report the average length of the message (both as number of words and number of characters), and the standard deviation.

TR6. Conditional Histograms. The program shall compute and report the following conditional histograms (i.e., histograms documenting the frequency of occurrences of certain properties over subsets of the message collection).

1. For each status value, report the frequency histogram of recepient values, and (a separate) frequency histogram of presense/absense of `in-response` flag.
2. For each recepient value, report the frequency histogram of presence/absense of `in-response flag`.

TR7. Output. Your program shall produce two types of output. First, all the information collected in requirements **TR3–BR6** shall be printed out to terminal in a nice, well-formatted, readable display. Second, as an option, your program shall create a JSON object that stores all the collected statistics in a human-readable and easily-parseable format. If `beFuddledStats` program is run with a second input parameter, the name of the output file, your program shall generate the JSON statistics object and write it to the given output file.

This way, if your program is invoked as

```
$ java thghtShreStats myMessageList.json
```

the program will output the statistics to terminal. If the program invoked as

```
$ java thghtShreStats myMessageList.json stats.json
```

the program will *both* output the statistics to terminal, *and* write the statistics JSON object to `stats.json`.

TR8. The program shall perform exactly one pass over the JSON objects in the input file. That is, your program is allowed to access each JSON object (its Java representation) exactly once during its work.

TR9. Error-checking. Minimally acceptable graceful error-checking includes:

- Check for the number of command-line arguments. If the program is run without command-line arguments, output a help prompt.
- Check for presense of input files.
- Check that the JSON objects contained in the file are indeed ThghtShre log records. If the JSON format is incorrect (e.g., the program is run on the BeFuddled JSON object collection), the program shall report the format error and gracefully exit.

Program Design and Submission

Just as for Lab 1-1, organize your submission as follows. Create two directories, `beFuddl ed` and `thghtShre` and place all code for each of the parsers in the respective directories. You can either submit a single archived directory at a time (but it must be archived from its parent directory, so that a `be Fuddled/` or `thghtShre/` directory is proprely created when unpacked), or a single archive that, when unpacked create s two directories.

In addition, you must submit a `README` file (either directly, or as part of an archive that unpacks to the current directory) that describes the submission and lists all members of the student team.

Submission

Use `handin` to submit as follows:

Section 01:

```
$ handin dekhtyar lab01-2-01 <FILES>
```

Section 03:

```
$ handin dekhtyar lab01-2-03 <FILES>
```

Good Luck!