Assignment #1
CSCI 201 Spring 2018
2.5% of course grade

Title
Creating a Grading Tool

Topics Covered
Java Classes
File I/O
Sorting Algorithms
Basic Java Topics

Introduction
Much of your computer programming experience is likely using C++. In this course, Java is the language we will primarily use. To help transition your knowledge from C++ to Java, you will be creating a grading tool with simple functionalities.

This assignment requires that you parse a file containing a roster of students along with some additional student data. To ensure accurate parsing, you will then provide a command line interface to allow a user to query the parsed data.

Data Persistence
Many software projects need to store persistent data, and there are a few ways to do this. Databases, which we will cover later in the class, are primarily used for most web applications, but storing data in files is also very common for many types of applications (i.e. mobile, standalone, web). This assignment will require you to parse a file that contains data about academic courses.

The data in the file is going to be stored as a JavaScript Object Notation (JSON) file. JSON is a lightweight data-interchange format. In other words, it is a syntax that allows for easy storage and organization of data. It is commonly used to exchange information between client and server, and it is popular because of its language independence and human readability. JSON is built upon two basic data structures that you should already be familiar with: dictionaries (maps) and ordered lists. An object in JSON is represented by an unordered set of name/value pairs (i.e. dictionary). Objects are contained by braces, { }, inside of which will list the object's attributes (with the following syntax — name : value), using a comma as the separator.

There are quite a few Java JSON parsing APIs out there (unfortunately the JDK does not currently have built-in support for JSON), but some of the more popular ones include GSON, Jackson, and JSON.simple. Here are a couple of blogs discussing the merits of choosing one of these APIs over another if you are interested:

http://blog.takipi.com/the-ultimate-json-library-json-simple-vs-gson-vs-jackson-vs-json/
http://javarevisited.blogspot.com/2016/09/top-5-json-library-in-java-JEE.html

In short, it seems GSON is known for its ease and flexibility of converting Java objects into JSON objects (and vice versa), and it is simple and straightforward to use. You may want to start there. No matter which API you choose (and you are not limited to choosing from the ones mentioned on these instructions), you will need to download a JAR file and add it to your Eclipse project. Below are links to the JAR file download for the APIs mentioned above:

GSON: https://mvnrepository.com/artifact/com.google.code.gson/gson/2.8.0 (under 'Files', click download JAR)
Jackson: https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-core/2.2.3 (under 'Files', click download JAR)
JSON.simple: https://mvnrepository.com/artifact/com.googlecode.json-simple/json-simple/1.1.1 (under 'Files', click download BUNDLE)

You will need to add the JAR file to your Java Build Path in Eclipse in order to use the library. First, move the JAR file to the top directory of your Eclipse project. Next perform the following steps in Eclipse:

1. Right click on your Eclipse project
2. Choose 'Properties'.
3. Select 'Java Build Path'.
4. Click the 'Libraries' tab.
5. Click 'Add External JARs'.
6. Find the JAR in your project directory and add it.
7. Click 'Okay'.

Assignment

First, you will need to prompt the user to enter the name of a JSON file. After reading the filename from the user, you will attempt to parse the file. If there are errors in the file, you will display an error message with as much description as you can as to the problem. This will most likely be provided to you by your chosen parser, so don't worry about trying to make it more descriptive than that. After displaying the error message, you will loop back to allow the user to enter another JSON file.

If the file is a valid JSON file, you will parse it and store it in variables in your program. I recommend that you create classes to store all of the data, and use good object oriented principles.

After all the data is stored in your Java objects, you will give the user a menu to allow them to query the data.

Here is a sample JSON file. A JSON file that can be used for testing purposes is posted on the course website. I recommend that you create your own test files, since the file we will use for testing will be different form the sample one provided.

```
{
	"students": [{
		"name": {
			"fname": "Tommy",
			"lname": "Trojan"
		},
		"average" : "86",
		"numGrades" : "4",
		},
		{
		"name": {
			"fname": "George",
			"lname": "Tirebiter"
		},
		"average" : "74",
		"numGrades" : "6",
		},
		{
		"name": {
			"fname": "Billy",
			"lname": "Bruin"
		},
		"average" : "63",
		"numGrades" : "1",
		},
		{
		"name": {
			"fname": "Jeffrey",
			"lname": "Miller"
		},
		"average" : "92",
		"numGrades" : "5",
		},
		{
		"name": {
			"fname": "Michael",
			"lname": "Shindler"
		},
		"average" : "93",
		"numGrades" : "4",
		}
	]
}
```

The data contained with each student includes their first and last names, their current grade average, and the number of grades they have received. The number of grades received is necessary for inserting more grades and recalculating the average. All grades are weighted equally, so there is no need to worry about weighted averages.

Once the data is correctly parsed by the program, the menu of options will give the user the following functionalities: Display Roster, Add Student, Remove Student, Add Grade, Sort Roster, Write File, and Exit.

The "Display Roster" option should print to the user the names of all the students as well as their current averages.

The "Add Student" and "Remove Student" options are self-explanatory, but when adding a new student, the average and number of grades should be set to 0.

The "Add Grade" option uses the number of grades, the current average, and the entered score to recalculate the average. All scores should be weighted equally.

The "Sort Roster" option gives the user the option to sort the roster in A-Z alphabetical order or by highest-to-lowest average. If two students have the same average, then sort them alphabetically. If two students have the same name, then sort them by highest-to-lowest average.

The "Write File" option gives the user the option to write the current roster out to a file to save the data. Save the results in the same file that is currently opened.

The "Exit" option displays a message indicating the end of the program before terminating said program. If a change has been made to the file since the last time the file was saved, prompt the user to ask if he would like to save the file before exiting.

Sample Execution

Here is a sample execution of the program.

```
What is the name of the input file? invalid.json

That file is not a well-formed JSON file.


What is the name of the input file? notfound.json

That file could not be found.


What is the name of the input file? test.json

     1) Display Roster
     2) Add Student
     3) Remove Student
     4) Add Grade
     5) Sort Roster
     6) Write File
     7) Exit
```

```
What would you like to do? 8


That is not a valid option


      1) Display Roster
      2) Add Student
      3) Remove Student
      4) Add Grade
      5) Sort Roster
      6) Write File
      7) Exit

What would you like to do? 1


Trojan, Tommy 86

Tirebiter, George 74

Bruin, Billy 63


      1) Display Roster
      2) Add Student
      3) Remove Student
      4) Add Grade
      5) Sort Roster
      6) Write File
      7) Exit

What would you like to do? 4

   1) Trojan, Tommy
   2) Tirebiter, George,
   3) Bruin, Billy


Please choose a student to grade: 1

Please enter a new score: 100


Trojan, Tommy 88.8
```

```
      1) Display Roster
      2) Add Student
      3) Remove Student
      4) Add Grade
      5) Sort Roster
      6) Write File
      7) Exit

What would you like to do? 6

File has been saved


      1) Display Roster
      2) Add Student
      3) Remove Student
      4) Add Grade
      5) Sort Roster
      6) Write File
      7) Exit

What would you like to do? 2

What is the student’s name? Traveler

Invalid, must have first and last name


What is the student’s name? Traveler Horse


      1) Display Roster
      2) Add Student
      3) Remove Student
      4) Add Grade
      5) Sort Roster
      6) Write File
      7) Exit

What would you like to do? 1


Trojan, Tommy 86

Tirebiter, George 74

Bruin, Billy 63

Horse, Traveler 0
```

```
      1) Display Roster
      2) Add Student
      3) Remove Student
      4) Add Grade
      5) Sort Roster
      6) Write File
      7) Exit

What would you like to do? 5


      1) Alphabetically
      2) GPA

How would you like to sort the roster? 1

Bruin, Billy 63

Horse, Traveler 0

Tirebiter, George 74

Trojan, Tommy 86


      1) Display Roster
      2) Add Student
      3) Remove Student
      4) Add Grade
      5) Sort Roster
      6) Write File
      7) Exit

What would you like to do? 7

Changes have been made since the file was last saved.

      1) Yes
      2) No

Would you like to save the file before exiting? 2

File was not saved.

Thank you for using my program!
```

Grading Criteria

The manner in which you go about implementing the solution is not specifically graded, but make sure you parse the JSON file properly and produce output as similar as possible to what is displayed above

File Parsing (0.8%)
0.2% - the filename is read from the user
0.3% - if the file cannot be parsed, an appropriate error message is displayed
0.3% - if the file cannot be found, an appropriate error message is displayed

Output (1.7%)
0.2% - The roster is displayed properly
0.1% - An invalid menu entry gives an error message and returns to the same menu
0.2% - Adding a student requires a first and last name and saves new student to roster
0.2% - Removing a student properly removes them from the roster
0.2% - Adding a grade correctly calculates a new average
0.3% - Sorting the roster correctly sorts based on option chosen by the user
0.2% - Write File option correctly outputs roster and all changes to the same file that is currently open
0.1% - Exiting prints a message before terminating the program
0.2% - Exit prompts the user to save the file if a change has been made before it was saved and saves properly if necessary