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

Due date: January 20, midnight.

Lab Assignment

Assignment Preparation

This is a pair programming lab. 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

In this class, a healthy dose of programming will be done in Java. We will also be working with a variety of input data, but a lot of the time, the data will be stored in JSON, a popular lightweight data format.

In this lab, you will practice the use of Java for handling JSON objects and collections of JSON objects.

For this part of the lab, just as is the case for Lab 1-1, you will perform two subtasks:

- **Data Generation.** You will write a series of programs that generate random data (in some pre-specified formats) and dump this data into files.
- **Data Consumption.** You will write a series of programs that consume the data produced by your data generators and compute certain infomration about the consumed data collections.

The dataset you are generating for Lab 1-2 is somewhat more complex and is described below.

Data Format for MovieSurvey dataset

A marketing company conducted a simple study of moive preferences among a group of respondents. They have stored the results of the movie preference study (survey) in a collection of JSON objects. Each JSON object in this collection corresponds to the preferences of one respondent. The company collected some demographic data about the respondents (including personal identification information), as well as asked the respondents to rate a few well-known movies from different genres. All respondents were asked to rate the same movies. Some respondents did not see all the movies; the survey responses reflect that.

A JSON object representing a single survey response looks as follows:

```
{
 RID: <respondentId>,
 respondent: { name: {first: <firstName>,
                      last: <lastName>
                      },
               gender: <gender>,
               age: <age>,
               state: <stateOfResidence>,
               education: <educationLevel>,
               income: <incomeLevel>
              },
 ratings: [ <ratings> ]
}
```

Below are the explanations of the various values in the object.

- `<respondentId` is a unique identifier for each respondent. It is an integer number.
- `<firstName>`, `<lastName>` are the first and the last name of the respondent. These are mandatory.
- `<gender>` is the gender of the respondent. The survey company provided only two options, "Male" and "Female" (labelled `"M"`, and `"F"` in the JSON document). However, a certain percentage of the respondents chose to either decline to state their gender, or provided a different answer. The survey company coded all responses that were neither `"M"`, nor `"F"` as `"N/A"`.
- `<age>`: the age of the respondent, and integer number. Ages range from 16 to 85.
- `<stateOfResidence>`: state of residence of the respondent. The survey was conducted among the US residents of the 50 states and Washington,DC. Therefore, 51 possible two-letter code combinations can be the value of the `state` key: `"AL", "AK", "AR", "CA","DC",....`

- <educationLevel>: the highest education level achieved by the respondent. Possible values are listed in the table below:

JSON Value	Explanation
0	Declined to state
1	Less than high school
2	High school graduate
3	Some college
4	BS degree
5	an advanced degree (MS, Ph.D., JD, MD, etc.)

- <incomeLevel>: a code for the individual income level of the respondent coded as follows

JSON Value	Explanation
0	No independent income
1	$\$0 - - - \$20,000$
2	$\$20,000 - - - \$40,000$
3	$\$40,000 - - - \$60,000$
4	$\$60,000 - - - \$100,000$
5	over $\$100,000$ per year

- <ratings> is an array of the movie ratings provided by the respondent. Each movie rating is a fixed point value between 1 (very poor) and 10 (best movie ever!) with one decimal place. E.g, 1.3, 4.7, 8.0, 9.3 are all possible values. A special value 0 is also a possible value and it means that the respondent has not seen a given movie. There are a total of 13 ratings. The ratings are provided in the following order of the movies:

Rating Position in Array	Movie
1	*Star Wars: A New Hope*
2	*Godfather*
3	*Memento*
4	*Saw*
5	*Rocky*
6	*Princess Bride*
7	*Sleepless in Seattle*
8	*Pretty Woman*
9	*Avatar*
10	*Dogma*
11	*Batman Begins*
12	*Suicide Squad*
13	*Beverly Hills Cop*

Here is a possible JSON object representing one respondent's information and ratings:

```
{RID: 25,
 respondent: { name: {first: "Elizabeth"
                      last: "Baker"
```

```
                    },
            gender: "F",
            age: 32,
            state: "NY",
            education: 4,
            income: 4
           },
 ratings: [ 5.3, 3.2, 7.6, 2.3, 6.5, 7.9, 6.0, 5.4, 8.2, 3.4, 5.4, 0, 2.1 ]
}
```

Data Generation

You will write a simple data generation program for MovieSurvey data called `surveyGen.java`. The primary inputs to the data generation program will be the name of the output JSON file, and the number of JSON objects to generate. The program will take the input parameters and will generate the appropriate number of correctly formatted (according to the data description provided above and the generation rules provided below) JSON objects. The objects will be saved into the file with the given file name.

The data generation program must abide by a special set of object generation rules designed to make your generation procedure provide reasonable approximations of real activity.

The generation rules are outlined below.

Please note, that this part of the assignment is essentially an exercise in the creative use of the random number generator. There is a number of different ways in which each JSON object generator can be constructed that will yield correct data. Most of the specifications below are declarative, that is, they explain, what a correctly generated sequence of objects should look like, and what sorts of objects violate correctness conditions. I do not provide algorithmic instructions here, but I am happy to discuss them with any interested team. The actual generation procedure is not really a secret I am hiding from everyone.

Data Generation for the MovieSurvey data

The following rules must be obeyed:

SR1. **Unique Ids.** On each independent run of the data generator, number your JSON objects from 1 to the total number of objects being generated.

SR2. **Names.** Roughly 50% generated names should be female names and roughly 50% of all generate names should be male names. The lists of last names, male and female first names (based on 2000 US Census) are provided to you on the Lab 1 web page.

SR3. **Gender.** As stated above, a small percentage (pick one) of respondents decline to identify their gender. All other respondents should have their stated gender align with their name (i.e., if you generated the first name from the list of female names, such a respondent should be assigned female gender and vice versa).

SR4. **Age.** Most participants should be ages 18 — 55. I am giving you more leeway here with how to set up the age distribution, but it should be realistic.

SR5. **State.** There are large states, and there are small states. Look up proportion of population by state, and create a simplified probability distribution to generate the state label (including DC). "Simplified" means that you are allowed to increase the probability of residence in smaller states (Hawaii, Wyoming, Idaho, Rhode Island, etc) at the expense of the probability of residence in larger states (New York, California), but there should be rough correspondence between the real population of a state and the probability or generating a resident of this state. You can do it layers: split all states into large, medium and small, assign probability to each layer, and use uniform distribution within each layer. Or you can assign a probability to each state individually. It is left up to you.

SR6. **Education level.** In general the distribution should be consistent with the distribution of education levels in the US (we won't worry about individual state differences). Persons who are younger than 18 can only have values 0 or 1; persons who are younger than 21 can only have values 0, 1, 2, or 3.

SR7. **Income level.** Persons younger than 18 should largely be in category 0, with some small percentage in category 1. Persons ages 18 through 21 should largely be distributed into categories 0 through 3. If you wanted to be fancy, you could make income level probability conditional on education level, but this is not strictly a requirement here.

SR8. **Movie ratings.** About 60% of all respondents should have provided all ratings. For other respondents, there should be a fixed, movie-specific probability that a respondent has not seen the movie. The probability should not be overly large (no more than 20%), and should not be the same for all movies (some movies are more popular and well-known than others).

SR9. **Movie rating dependencies.** When creating movie ratings, please incorporate the following soft rules in your generator:

(a) People who like romantic comedies (Princess Bride, Pretty Woman, Sleepless in Seattle) tend to not like horror movies (Saw).

(b) People who like Star Wars, also tend to like other big budget heroic movies (Batman Begins, Avatar, Suicide Squad).

(c) Princess Bride is more or less universally liked by middle-aged people (viewers ages 35 through 55).

(d) Dogma, Beverly Hills Cop and Saw are polarizing: people either like them (individually and independently of each other) quite a bit, or really dislike them.

(e) People who like Rocky also like Godfather and vice versa.

(f) People who like Batman Begins also tend to like Memento, although the inverse might not be true.

It is left up to you how you implement these soft rules in your generator, but over a course of a large data generation job, these trends should be observable with software.

You can also add your own rules that are not listed above, tying movie preferences to geography (Washington State residents and Maryland residents love Sleepless in Seattle; residents of Wyoming hate Mememnto), income levels or levels of education.

MovieSurvey JSON Reader

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

MR1. The program takes as input the name of a JSON file containing JSON objects created by your MovieSurvey JSON generator. Optionally, the program shall take a second filename, indicating the name of the file for the output. See Requirement for the description of what to do with the second parameter.

MR2. The program shall parse the JSON objects from the input file and scan them one-by-one.

MR3. The program shall collect a number of aggregate statistics based on the input file content. The specific statistics are outlined below.

MR4. Population Statistics. Your program shall report:

- total number of survey respondents
- gender distribution of survey respondents, both as numbers and percentages
- distribution of survey respondents by age groups: younger than 20, 20–29, 30–39, 40-49, 50-59, 60-69, over 70.
- distribution of survey respondents by income groups

- distribution of survey respondents by education levels
- geographic distribution of survey respondents into five categories: North-East, MidWest, South, West, Pacific West. State breakdowns will be provided to you separately.

MR5. Movie Ratings. For each movie specify the following information

- Total number of non-zero ratings
- Average rating of the movie
- Standard deviation of the rating.

MR6. Movie Ratings by Gender. For each movie compute and output

- total number of non-zero ratings for each gender value
- average movie rating for each gender value (output 0 if a rating cannot be computed)

MR7. Output. Your program shall produce two types of output. First, all the information collected in requirements **MR4–MR6** 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 surveyStats surveys.json
```

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

```
$ java surveysStats surveys.json stats.json
```

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

MR8. 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.

MR9. 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 MovieSurvey 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.

Submission

Submit both your programs, any supplemental files and a `README` file describing your program (and how to run it), and containing the names of all team members.

Use `handin` to submit as follows:

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

Good Luck!