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

Due date: January 13, 11:59pm.

Note: Lab 1-2 will be assigned on January 13 in class.

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, and for Lab 1-2 you will remain in the same pair.

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 lab, you will perform two subtasks:

1. **Data Generation.** You will write a series of programs that generate random data (in some pre-specified formats) and dump this data into files.
2. **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.

   You will generate and consume data for two different putative applications: one requiring a relatively simple data format, and one requiring a more complex data format.

Lab 1-1 is the data generation and data consumption for the simple data format.

Lab 1-2, to be assigned on Friday, January 13, is data generation and data consumption for a more complex data format.

Java Support for JSON

JSON[1] (JavaScript Object Notation) is a lightweight text-based format for representing structured and semi-structured objects and their collections. It is a somewhat less bulky (albeit not as powerful) format than XML, but JSON documents and XML documents share a lot in common and can be translated into each other.

Java has ample support for JSON. http://www.json.org lists 26 different Java libraries for parsing and generating JSON objects. In this lab you can select any of the JSON libraries for Java.

In preparation for this lab I used one one of the libraries, `org.json`. I have made the `.jar` file for the library available from the course web page. I have also made a couple of small programs for parsing JSON and generating JSON objects available.

Data Generation: Data Formats

In this assignment, you will "spoof" JSON objects generated by a(n, I hope, fictional) text messaging service `Thghtshre`.

Data Format for Thghtshre Messaging Application

Thghtshre messaging application is a simple mobile messaging application that allows its users to send messages to other users subscribed to their messaging feeds. For this particular application, we concentrate on generation of messages themselves.

A single message is encoded as a JSON object in the following way:

```
{ "messageId": <messageId>,
  "user": <userId>,
  "status": <messageStatus>,
  "recepient": <recepient>,
  "text": <messageText>
}
```

Optionally, the message may be made in response to another message, in which case, one more attribute:

```
 "in-response": <originalMessageId>
```

[1] http://www.json.org/

is added to the JSON object (typically, after the "recepient" attribute, but since JSON is not order-sensitive, it does not matter where it is placed).

Here,

- <messageId> is a unique Id of the message. It must be an integer.
- <userId> is the unique id of the user sending the message. User ids have the same format in Thghtshre app as they do in the Befuddled app. We also assume that Thghtshre has 10,000 users with the same user ids.
- <messageStatus> is either "public", "protected" or "private".
- <recepient> is either "all", "self", "subscribers" or any single user id.
- <messageText> is a short message consisting of a number of words. (See the rules for message generaion in the description of the generator).
- <originalMessageId> is an id of a message. It is an integer number that must be smaller than the the <messageId> number.

Here is a simple JSON object representing a message:

```
{
 "messageId": 204532,
 "user": "u4053",
 "status": "public",
 "recepient": "subscribers",
 "in-response": 204302,
 "text": "I do not think so!"
}
```

Data Generation

You will write a simple data generation program for Thtshre data called thghtShreGen.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 Thghtshre Messenger

The following rules must be obeyed:

TR1. In a single run of your program, message Ids for individual messages must in increasing order. There is no need to go in a row, but a message Id can be generated only if no other message with a higher id has been generated.

TR2. As mentioned above, if a message object has a `"in-response"` component, the message Id of the `"in-response"` component must be smaller than the message If of the current message object. At the same time, *it is not necessary for you to have generated an actual message with the give `"in response"` message id.*

TR3. Messages with the status `"private"` can have either `"self"` or a user Id as a `recepient`.

TR4. Messages with the status `"protected"` can have either `"self"`, or `"subscribers"`, or a user Id as a recepient.

TR5. Messages with the status `"public"` can have any syntactically legal `"recepient"` value.

In addition to these rules, we adopt the following conventions and provide the following suggestions.

1. There are 10,000 users of the messaging platform. You are not required to generate messages for all of them.
2. Most messages are `"public"`.
3. Most `"protected"` messages are addressed to `"subscribers"`.
4. Most `"private"` messages are addressed to an individual user Id (usually not equal to the user Id of the author of the message).
5. `"Public"` messages are roughly evently distributed between `"all"` and `"subscribers"`, with other options (`"self"`, or a user Id) taking no more than about 20% of the total number of `"public"` messages.

Text Generation for the Messages

Basically, any text generation procedure that uses legible words will suffice.

Here I describe a default procedure you can use. For this lab, I am making available a list of unique English terms used in Jane Austen's *"Sense and Sensibility"* novel. Please note, that this list may include some punctuation, proper names, and ALLCAPS words.

You are not required to generate any meaningful text, beyond generating text that is made out of English words. A simple generation procedure is to generate randomly (using either a normal or a uniform distribution) a number of words a specific message shall contain, and then select the generated number of words at random from the provided list. I am not looking for anything that is significantly fancier than this.

You can use your own word lists if you want to (submit them with your program). Please, no swearing/cursing in this case.

Your messages can be of size 2 to 40 words. You can pick any reasonable distribution of message lengths you want.

ThghtShre JSON Reader

You will write a `thghtShreStats.java` program that reads in a JSON file containing Thghtshre message JSON objects, and provides a variety of statistical information about the observed data.

In what follows a "histogram" refers to the output that display frequency of occurrences of different features in the data. For example, in an input document has 10 JSON objects, with five representing "public" messages, two representing "private" messages and three representing "protected" messages, the histogram of message statuses can look as follows:

```
Message Status Histogram:

      Public: 5
   Protected: 3
     Private: 2
```

I am looking for a simple text output, not for some complex graphical visualization here.

TR1. The program takes as input the name of a JSON file containing JSON objects created by your ThghtShre JSON generator. Optionally, the program shall take a second filename, indicating the name of the file for the output. See Requirement **TR7** 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[2]:

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.

[2]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`.

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–TR6** 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

You will create two programs, `thghtShreGen.java` and `thghtShreStats.java` (these, obviously, do not have to be the only files you submit, rather, these are the Java programs that should be compiled and run).

For simplicity, organize your submission as follows. Create a directory `thghtShre` and place all code for both the generator and the statistics reporting program in it. Submit your archive so that when unpacked, the `thghtShre` directory was in the directory in which your submitted archive resided. Call your archive `lab1.zip` or `lab1.tar.gz`.

In addition, you must submit a `README` file directly (i.e., NOT as part of the archive). Put the following information in it:

- Names and email adresses of all students on the team.
- Any `.jar` files needed to run the program. If you used any `.jar` files that are not part of Java Standard Libraries, or that were not provided on the course web site, it is best to include them in your submission.
- Any compilation instructions and/or run instructions.
- Description of any command-line parameters (see below).
- Description of any specific generation assumptions made. For example, if you decided to use a normal distribution with the mean of 11 and standard deviation of 5 to generate the number of words in a text message, specify this.

I advise you to start building the README file early and to record all necessary information in it as you go.

ThghtShre generator. For the `thghtShreGen.java`, the two expected input parameters are the name of hte output file and the number of JSON objects to generate. A possible optional parameter is the word file for message generation (I welcome using it to give your program the flexibility of building messages from a variety of vocabularies). If you are using any word files that are alternative to the one(s) provided by the instructor, submit them. If your word file format is different than the one provided by the instructor, (a) you must submit at least one properly formatted file, and (b) describe the format in the README file[3].

ThghtShre statistics. For this program no additional command-line parameters, beyond the two described above (input file name, and optional output file name) are needed.

Use `handin` to submit as follows:

[3] As an example, you may want to add frequency of each word/probability of word selection to the file, to be more fancy.

```
$ handin dekhtyar lab01-1 <FILES>
```

Good Luck!