COMP 2150 - Winter 2020 - Assignment 1

Due January 31, 2020 at 11:59 PM

You have started a new service that helps students find tutoring for courses they are taking. You've hyped it as "the uber of student help" and a "modern gig economy for studying" and a bunch more buzz phrases that you are not sure make sense. You are still working on a name like "tutrfindr" or something like that, but now you have to actually implement the system.

In your system you have two groups of people: **students** and **tutors**. Tutors are experts in a set of topics, and students want to be tutored in these topics. Tutors are only able to tutor for so many hours per term -- these hours will be sold to students that need tutoring. (You won't need to track scheduling of these hours -- that will be the responsibility of the tutor and student after they agree to some number of hours of tutoring.) When filling a request, a student can be tutored by any number of different tutors, as long as the student gets the total number of hours of tutoring they would like.

You will implement your solution to this system in Java. It will consist of code for tutors, students and requests, among other things. Here is an overview of the major components of the assignment, followed by a description of the commands for the system.

Tutors

When tutors register with your website, they register with a userid and the number of hours per term that they are able to tutor. After registering, each tutor can specify a list of topics that they are capable of tutoring, with their hourly rate for each topic. After providing the topics, tutors will be assigned students by the system.

Students

When students register for the website, they register with a userid. No student is also a tutor for any topic. After registering for the system, students can make requests for tutoring.

Requests

Once a student has registered, they can make a request for tutoring. A request consists of a topic and a number of hours required. Your system should fill all requests immediately for as low a price as possible.

To assign students to tutors, you must use a system similar to some stock markets. If a student makes a request for a given number of hours of tutoring in a topic, the following algorithm is used:

1. All tutors that have available hours for that topic are considered.

2. Among eligible tutors, the tutor with the lowest price is chosen for as many hours as possible. If several tutors have the lowest price, the tutor with the most hours of tutoring to offer should be used first. If a tie still exists then the tutor with the first userid in alphabetical order should be used.
3. If more hours are needed, the next remaining tutor is selected for as many hours as possible, using the same rules as in #2 (lowest price first, etc.). This is continued as long as the student still needs hours.
4. If all requested hours can be provided to the student in this way, then the request is successful.
5. If the requested hours cannot be provided to the student by any number of tutors, then the request fails. All hours that may have been given to that student are not used.

Requests are completely processed (resulting in either success or failure) before another request (or other command) is considered.

Commands

Your program will work by processing a text file that consists of commands. Each command is on its own line in the file. You should read the text files line by line to process command. The name of the text file **must not be hardcoded** into your program -- the user must be prompted for the name of the file (either through a text prompt in the program or through a command line parameter).

1. TUTOR [userid] [hours]
   - This action creates a new tutor.
   - Example: TUTOR mike 5
   - Outcomes: CONFIRMED, DUPLICATE
   - A tutor is a duplicate if there is another tutor with the same userid. Duplicates are ignored.
   - The userid is a sequence of at most 80 non-whitespace characters (uppercase and lowercase letters and numbers).
   - The number of hours will be between 1 and 1000.
2. STUDENT [userid]
   - This action creates a new student.
   - Example: STUDENT studyr
   - Outcomes: CONFIRMED, DUPLICATE
   - A student is a duplicate if there is another student with the same userid. Duplicates are ignored.
   - The userid is a sequence of at most 80 non-whitespace characters (uppercase and lowercase letters and numbers)
3. TOPIC [topic name] [tutor id] [price]
   - This command specifies that a tutor can tutor in a specified topic for a given price per hour.
   - Example: TOPIC OOProgramming mdomarat 10

- Outcomes: CONFIRMED, DUPLICATE, NOT FOUND
- A topic is a duplicate if has already been specified for a given tutor. Duplicates are ignored, even if it is a different price (so only the first price is ever used).
- The name of a topic is a sequence of at most 80 non-whitespace characters.
- The command reports not found if the tutor does not exist in the system. No further processing is done in this case.
- The price is an integer greater than zero and less than 1000, representing the price per hour for the tutor in that subject.

4. REQUEST [student id] [topic] [num hours]
    - Specifies that a student has made a request for tutoring in a given topic.
    - Example: REQUEST studyr OOprogramming 5
    - Outcomes: NOT FOUND, FAIL, SUCCESS
    - A request returns not found if the student is not found. No further processing occurs in this case.
    - A request fails if there is no way to schedule the hours with suitable tutors (this includes the situation where no tutors have expertise in the required topic).
    - A successful request prints out the details of the tutors assigned to a student (see below for the format).
    - The number of hours is an integer between 0 and 2000
5. STUDENTREPORT [student id]
    - Writes a report for the student including all appointments, total number of hours tutored and total cost.
    - Example: STUDENTREPORT studyr
    - Outcomes: NOT FOUND, REPORT
    - A report returns not found if the student is not found. No further processing occurs in this case.
    - A successful report prints the details of all appointments, number of hours and total cost (see below for possible format).
6. TUTORREPORT [tutor id]
    - Writes a report for the tutor including all appointments, total number of hours of tutoring and total revenue (amount of money taken in).
    - Example: TUTORREPORT mike
    - Outcomes: NOT FOUND, REPORT
    - A report returns not found if the tutor is not found. No further processing occurs in this case.
    - A successful report prints the details of all appointments, total hours of tutoring and total money taken in (see below for possible format).
7. QUIT
    - Ends the program. No other commands are read.
    - The message "BYE" should be printed and the program should terminate.
    - If the end of the program is encountered without a "QUIT" command, the program should report that the "QUIT" command was missing and then terminate.
8. COMMENTS

- Any line that starts with the character # is a comment.
- The comment is ignored by the system.
- Please note: **students have failed assignments in previous years because the assignments couldn't process comments**. Please don't forget about comments.

In all commands, whitespace (tabs and spaces) is ignored (i.e., there can be leading whitespace on any lines, or multiple whitespace between tokens in a line). However, each command is guaranteed to appear on a single line.

For error checking, you should ignore commands that are note one of the previous eight kinds. You can assume that integer values are valid in the input and don't need to do error checking.

Object Oriented Programming

Your assignment should use OO programming. In particular:

1. You should have code reuse as much as possible. If you have duplicated code for the same tasks (i.e., inserting an item into a data structure), you will lose marks.
2. You should have as little static code as possible. You should aim to have ten lines or less of code in static methods in your entire assignments. More than this will result in lost marks.

Data Structures

There are some restrictions on data structures you must use for the assignment. **Do not use ANY built-in Java collection classes** (e.g., ArrayLists, Hashes, ...) **or Java Generics in this assignment**: any linked structures must be your own, built from scratch. You should not use arrays other than for temporary operations (e.g., split() for strings).
IF YOU USE GENERICS OR COLLECTION DATA STRUCTURES, YOU WILL LOSE MARKS.

For this assignment, you do not need to concern yourself with data structure efficiency. Any searching through data structures can be exhaustive search and you do not to maintain any data structures in sorted order.

When dealing with hierarchies and data structures, you must use safe casting practices.

Unit Testing

Construct a set of unit tests for your code. To do unit testing in Java, you should:

1. Install junit jar files (either through your IDE or from https://junit.org/junit5/).
2. Create a new class for the tests.

3. Imports: add the following import statements.

```
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;
```

4. Create public void methods in your class to test conditions about your code.
5. Annotate each test method with the line "@Test" before the method.

Your tests should focus on the data structures you create for your project. You should include at least ten meaningful tests for your code, including a minimum of five for your data structures.

You will be graded on your tests, so write useful tests. The markers will be running your tests, as well, so ensure that they pass. If tests do not pass or your code does not compile, you will lose a substantial number of marks.

Hand-in

Submit all your source code for all classes. Each class should be declared as public in its own file.

You **MUST** submit all of your files in a zip file. Additionally, you **MUST** follow these rules:

- All of the files required for your project should be in a single directory (this probably means that you should not have any package statements in your code.)
- Include a README.TXT file that describes **exactly how to compile and run your code** from the command line. The markers will be using these instructions exactly and if your code does not compile directly, you will lose marks. Code will be run on a standard linux environment (aviary). Your instructions must include details on how to read the text files.
- You should also submit a text document giving the output on the official test data. (Official test data will not be released until just before the due date.)

The easier it is for your assignment to mark, the more marks you are likely to get. Do yourself a favour.

Submit all files on umlearn. Remember to submit your electronic honesty declaration on umlearn before the assignment deadline.

Appendix: suggested formats

The format of prompts and print statements is **not strict,** but all prompts and output should provide enough information for the marker to identify correctness quickly. It is suggested that some output be provided for every command in the input file.

Here are some examples of formats that can be used.

1. Student report:

```
Report for Student student1
---------------------------
Appointment: Tutor: tutor1, topic: OO, hours: 5, total cost: 100
Appointment: Tutor: tutor2, topic: OO, hours: 10, total cost: 150
Total number of hours of tutoring: 15
Total cost of tutoring: 250
---------------------------
```

2. Tutor Report:

```
Report for Tutor tutor2
-----------------------
Appointment: Student: student1, topic: OO, hours: 10, total revenue:  150
Total number of hours of tutoring: 10
Total revenue from tutoring: 150
-----------------------
```

3. Other responses:

- `Tutor with userid tutor2 successfully created.`
- `Student with userid student1 successfully created.`
- `Duplicate Student with userid student1.`
- `Duplicate Tutor with userid tutor1.`
- `Topic OO added to Tutor tutor2 with price 15.`
- `Duplicate topic OO for Tutor tutor1`
- `Tutor tutor10 not found.`
- `Student student10 not found.`
- `Attempting to fulfil request for student1 to receive 15 hours of tutoring in topic OO`
- `Tutor tutor1 will tutor student student1 for 5 hours in topic OO at a rate of 20.`
- `No tutors available for Student student1 for 1000 hours in topic OO.`