BBM104 MAKEUP ASSİGNMENT

Subject: Inheritance, Encapsulation, Access Modifiers
Due Date: 25.06.2021 23:59
Programming Language: Java

1. EXPERIMENT

In this experiment you are expected to develop a ***Trip Company System*** by using Java programming language. The system will process an *input file* and will print some log messages to an *output file*. Input file will be error free *only syntactically*. The program will be executed with two command line arguments:

<input_file><output_file>

The requirements and rules for the system are given below:

1. There are two types of ***Vehicle*** in the system. These are ***Train*** and ***Bus***.
   a. System Manager can add a ***Train*** to the system.
      i. Each Train has *name, age, speed, type, capacity for compartments with bed*
      ii. There cannot be two *Trains* with same *name*.
      iii. There are two *types* of trains; *express* or *regional*.
      iv. *Compartments with bed capacity* the limits for train trip reservations.
   b. System manager can add a ***Bus*** to the system.
      i. Each Bus has *plate, age, speed, seat capacity*.
      ii. There cannot be two Buses with same *plate*.
      iii. *Seat capacity* determines the limit for the bus trip reservations.
2. There are two types of ***Person*** in the system. These are ***Passenger*** and ***Captain.***
   a. System manager can add a ***Passenger*** to the system
      i. Each Passenger has *id, name, gender, age, phone number.*
      ii. There cannot be two Passengers with same *id*.
   b. System manager can delete a ***Passenger*** from the system.
      i. A Passenger can be deleted with *passenger id.*
      ii. Passenger is depended on *reservations*. If a passenger has reservations for any trip those reservations should be canceled before the passenger is deleted.
   c. System manager can lists all passenger by sorting passenger_id in descending order.
3. There are two types of ***Captain*** in the system. These are ***Machinist*** and ***Driver.***
   a. System manager can add a ***Machinist*** to the system**.**
      i. Each Machinist has *corporate id, name, gender, age, years worked.*
      ii. There cannot be two Machinists with same *corporate id*.

    b. System manager can sort Machinists by their years_worked in ascending order.
    c. System manager can add a ***Driver*** to the system**.**
        i. Each Driver has *corporate id, name, gender, age, license number, years worked.*
        ii. There cannot be two Drivers with same *corporate id*.

4. There are two types of ***Trip*** in the system. These are ***Bus Trip*** and ***Train Trip***.
    a. System manager can add a ***Bus Trip*** to the system**.**
        i. Each Bus Trip has *trip number, departure station name, arrival station name, departure time, driver corporate id, bus plate.*
        ii. There cannot be two Bus Trips with same *trip number.*
        iii. Bus Trip conflicts should be checked during the registration of a Bus Trip.
        iv. There could not be two bus trips with same *bus plate* on the same *departure time.*
    b. System manager can delete a ***Bus Trip*** from the system**.**
        i. A bus trip can be deleted with *bus trip number*.
    c. System Manager can add a ***Train Trip*** to the system**.**
        i. Each train trip has *trip number, departure station name, arrival station name, departure time, machinist corporate id, train name.*
        ii. There cannot be two Train Trips with same *trip number.*
        iii. Train trip conflicts should be checked during the registration of a train trip.
        iv. There cannot be two train trips with same *train name* on the same *departure time.*
    d. System manager can delete a ***Train Trip*** from the system**.**
        i. A train trip can be deleted with *train trip number.*
    e. System manager can add ***Station*** to the system.
        i. A Station has *name.*
        ii. Departure stations and arrival stations must be assigned to a trip. *Departure stations* must be different from *arrival stations*.
    f. System Manager can make ***Bus Trip Reservation***.
        i. A Bus Trip Reservation has *trip number* and *passenger id*.
        ii. System manager can cancel a bus trip reservation with *trip number* and *passenger id*.
    g. System Manager can make ***Train Trip Reservation***.
        i. A train trip reservation has *trip number* and *passenger id*.
        ii. System manager can cancel a train trip reservation with *trip number* and *passenger id*.

2. FILE FORMAT

Input File Format

Input file can include any number of command/data lines. Formats of possible commands are given below.

- ***ADDPASSENGER*** *<ID><NAME><GENDER><AGE><PHONE_NUMBER>*

- ***ADDTRAIN**<NAME><TYPE><AGE><SPEED><COMPARTMENTS_WITH_BED_CAPACITY>*
- ***ADDBUS** <PLATE><AGE><SPEED><SEAT_CAPACITY>*
- ***ADDSTATION** <NAME>*
- ***ADDMACHINIST** <CORPORATE_ID><NAME><GENDER><AGE><YEARS_WORKED>*
- ***SORTMACHINISTS**: This command sorts the machinists by their years_worked in ascending order.*
- ***ADDDRIVER**<CORPORATE_ID><NAME><GENDER><AGE><YEARS_WORKED> <LICENSE_NUMBER>*
- ***ADDTRAINTRIP** <TRAIN_TRIP_NUMBER><DEPARTURE_STATION_NAME> <ARRIVAL_STATION_NAME><DEPARTURE_TIME><MACHINIST_CORPORATE_ID> <TRAIN_NAME>*
- ***SEARCHTRAINTRIP**<TRAIN_TRIP_NUMBER>*
- ***ADDBUSTRIP** <BUS_TRIP_NUMBER><DEPARTURE_STATION_NAME> <ARRIVAL_STATION_NAME><DEPARTURE_TIME><DRIVER_CORPORATE_ID> <BUS_PLATE>*
- ***SEARCHBUSTRIP**<BUS_TRIP_NUMBER>*
- ***TRAINTRIPRESERVATION**<TRIP_NUMBER><PASSENGER_ID><COMPARTMENT_TYPE>*
- ***BUSTRIPRESERVATION** <TRIP_NUMBER><PASSENGER_ID>*
- ***CANCELTRAINTRIPRESERVATION** <TRIP_NUMBER><PASSENGER_ID>*
- ***CANCELBUSTRIPRESERVATION** <TRIP_NUMBER><PASSENGER_ID>*
- ***DELETEPASSENGER** <PASSENGER_ID>*
- ***LISTALLPASSENGERS:** This command lısts all passenger by sorting passenger_id in descending order.*
- ***DELETETRAINTRIP** <TRAIN_TRIP_NUMBER>*
- ***DELETEBUSTRIP** <BUS_TRIP_NUMBER>*

Output File Format

Output file will include some log messages and error messages if any for the commands processed. The general format of a single command related output entry is given below:

```
<COMMAND>
<RESULT>
----------------------------------------------------------------
```

3. ERROR HANDLING

The input file *can (and will) have semantic errors*, like referencing to a no existing object, trying to make a reservation for a trip which reached its capacity, etc. *There are also some rules which are defined above.* All these should be considered carefully and they'll have an important role for evaluating your work.
An important point for handling errors is explained below with a sample scenario: Consider an input file entry like this:

```
...
ADDBUSTRIP BT-2311 Ankara Istanbul 2006/03/28-22:00 2390 06KS345
...
```

There are several possible errors that can occur while processing this command. Here is the right order of validations for this command:
Check if there exists any station named *Ankara* (for availability).
Skip this command, printing an error message, if no such station exists.
Check if there exists any station named *Istanbul* (for availability).
Skip this command, printing an error message, if no such station exists.
Check if there exists any driver with corporate id *2390* (for availability).
Skip this command, printing an error message, if no such driver exists.
Check if there exists any bus with plate *06KS345* (for availability).
Skip this command, printing an error message, if no such bus exists.
Check if there exists any trip with number *BT-2311* (for duplication).
Skip this command, printing an error message, *if such a trip exists*.
Here is another sample command sequnce and its error handling procedure:

```
...
BUSTRIPRESERVATION BT-2311 132292
...
```

Check if there exists any Bus Trip with number *BT-2311* (for availability).
Skip this command, printing an error message, if no such Bus Trip exists.
Check if there exists any Passenger with id *132292* (for availability).
Skip this command, printing an error message, if no such Passenger exists.
Check if there is any available seat on the bus for the trip with number *BT-2311*.
Skip this command, printing an error message, if the trip (bus) reached to its *capacity* of reservations.

4. DESIGN NOTES

- Class that contains your main method should be named “Main.java”
- The files specified in the pdf are going to be given as program arguments. There will be text files (input.txt, output.txt) in your working directory. The input and Output files will be given as program arguments from the command line.
- In order to test your program, you should follow the following steps:
  - Upload your java files to your server account (dev.cs.hacettepe.edu.tr)
  - Compile your code (javac *.java)
  - Run your program (java Main input.txt output.txt)
  - Control your output data (output.txt)
- Samples of input and output files can be found on Piazza. Examine them carefully during coding.

5. SUBMIT FORMAT

- Do not submit any file via e-mail. You should upload your files via “Online Experiment Submission System” which is at http://submit.cs.hacettepe.edu.tr.
- File hierarchy must be zipped before submitted (Not .rar, only .zip files are supported by the system)

- student id.zip
  <src (Main.java, *.java)>
  <report.pdf, *.jpg(optional)>.

6. GRADING POLICY

Task	**Point**
Compiled	10
Output	60
Report	30
Total	100

7. IMPORTANT NOTES

- The assignment must be original, individual work. Downloaded or modified source codes will be considered as cheating.
- Also the students who share their works will be punished in the same way. We will be using the Measure of Software Similarity (MOSS) to identify cases of possible plagiarism. Our department takes the act of plagiarism very seriously. Those caught plagiarizing (both originators and copiers) will be sanctioned.
- You can ask your questions through course’s piazza group and you are supposed to be aware of everything discussed in the piazza group. General discussion of the problem is allowed, but DO NOT SHARE answers, algorithms, source codes and reports.
- Ignore the cases which are not stated in this assignment and do not ask questions on Piazza for such extreme cases.
- You are responsible for a correct model design. Your design should be accurate. It is important to draw a class diagram to show the whole of the system that you have created.
- In your REPORT, it is important giving explanation about problem definition and steps of algorithms that you followed. Also, you must add class diagram of your code in the report. Do not give so much unnecessary details in your report (totally max 2 pages written with Times New Roman 12)
- Don’t forget to write comments of your codes when necessary.
- The names of classes’, attributes’ and methods’ should obey to Java naming convention.
- Do not submit your project without first compiling it on dev machine.
- Save all work until the assignment is graded.
- Do not miss the deadline. Submission will be end at 25/06/2021 23:59, the system will be open until 23:59:59. The problem about submission after 23:59 will not be considered.