CISC124 - Assignment 3

Description

GTransport Inc. writes innovative software for taxi, bus and truck companies. As one of their best software architects and developers, you have been commissioned to create a hierarchy of Java classes and interfaces to support the implementation of a prototype of a tool named **SmartDispatcher**. This tool will simulate the operation of some new vehicle-dispatching algorithms which have the potential of significantly improving service levels for the companies using it.

The specifications for this prototype of SmartDispatcher, which you assembled in agreement with the GTransport's business analysts, read as follows:

The prototype will independently model dispatching algorithms for fleets of taxis, fleets of buses, and fleets of trucks. Each modeling activity will start with initial information about the fleet of vehicles (called a **state**), and then apply a sequence of "transactions" in which individual vehicles are assigned to specific customer requests (a customer request might be a call for a taxi, or a request for a private bus for a group trip, or a request to ship a package to another city by truck).

The tool will read these input files:

fleetSizes.txt

- This specifies the maximum number of vehicles in each fleet (see the file for the format).
- In each state, the number of vehicles in the fleet must be $\leq$ the maximum number specified in this file.

busStates.txt

Each line gives information about a particular bus. Each line contains

- **a state identification number** (integer)
  - the state id is used to group the buses together into initial situations – it does not refer to US states!
  - all of the lines with equal state id numbers will be consecutive in the file
- **the bus identifier number** (integer)
- **the current mileage of the bus** (integer)
- **the number of seats on the bus** (integer)

taxiStates.txt
Each line gives information about a particular taxi. Each line contains

- **a state identification number** (integer)
  - the state id is used to group the taxis together into initial situations – it does not refer to US states!
  - all of the lines with equal state id numbers will be consecutive in the file
- **the taxi identifier number** (integer)
- **the taxi's coordinates on an x,y grid** (two integers)

truckStates.txt
Each line gives information about a particular truck. Each line contains

- **a state identification number** (integer)
  - the state id is used to group the taxis together into initial situations – it does not refer to US states!
  - all of the lines with equal state id numbers will be consecutive in the file
- **the truck identifier number** (integer)
- **the truck's assigned desitination city** (String)
- **the truck's maximum load** (integer)
- **the truck's current load** (integer)

As an example of what these files look like, here is a very small example of taxiStates.txt

```
1       23      7       8
1       12      4       -2
1       92      15      3
2       42      1       1
2       12      -9      -4
```

This file is giving us information about two states. In State 1 there are three taxis (23, 12 and 92), and each taxi has particular x and y coordinates. In State 2 there are two taxis (42 and 12), and each of them has coordinates.

Note that there is no connection between the two states shown. Each state represents a different initial condition, completely independent of the previous state and/or the next state. Any repeated taxi id numbers are coincidental.

The following three files describe the "transactions" - the customer requests for buses, taxis and trucks.

BusCalls.txt
Each line describes a customer request for a bus to travel to another city. Each line contains

- **a state identification number** (integer)
  - the state id is used to connect the request to one of the initial situations
  - all of the lines with equal state id numbers will be consecutive in the file
- **a destination city** (String)
- **the number of seats required** (integer)

TaxiCalls.txt
Each line describes a customer request for a taxi. Each line contains

- **a state identification number** (integer)
  - the state id is used to connect the request to one of the initial situations
  - all of the lines with equal state id numbers will be consecutive in the file
- **the caller's coordinates on an x, y grid** (two integers)

TruckCalls.txt
Each line describes a customer request for a truck. Each line contains

- **a state identification number** (integer)
  - the state id is used to connect the request to one of the initial situations
  - all of the lines with equal state id numbers will be consecutive in the file
- **a destination city** (String)
- **the mass, in kilograms, of the parcel to be shipped** (integer)

Here is a very small example of what TruckCalls.txt might look like

```
1    Edmonton  177
1    Twillingate    32
1    Edmonton  46
1    Moosonee  25
2    Churchill 100
2    Halifax   2
3    Revelstoke     9
3    Moosonee  120
3    Nanaimo   12
```

In this example, there are four transactions for State 1, two transactions for State 2, and three for State 3. (Remember, the initial conditions for each state are given by busesStates.txt, taxiStates.txt and truckStates.txt)

Resolving Transactions

For each state of the buses and taxis and trucks, your program must attempt to satisfy the transactions for that state. Each type of vehicle uses a different dispatching algorithm.

Resolving a Bus Transaction

For a request for a bus with n seats, SmartDispatcher will first determine which buses have enough seats to satisfy this request. Then it will choose the bus whose mileage is closest to the average mileage of the buses that have enough seats. The chosen bus is not available for other transactions relating to this state.

If there is no bus with enough seats to satisfy the request, SmartDispatcher should print a line stating that it was unable to satisfy the request.

Resolving a Taxi Transaction

For a request for a taxi at a particular location, SmartDispatcher will randomly assign one of the **two** taxis that are closest to the location (where distance is computed in the normal "straight-line" way). If the assigned taxi is not available, it will assign the second closest. If this second taxi is not available, SmartDispatch will randomly assign one of the next **four** closest taxis. If this taxi is unavailable, it randomly assigns one of the remaining three, and so on. If all of these are unavailable, then SmartDispatch attempts to assign one of the next **eight** closest taxis, repeating the same selection algorithm. If the search process does not find any available taxis, the SmartDispatcher reports an unsuccessful search. If a taxi is assigned to this customer request, the taxi is not available for other transactions relating to this state.

Resolving a Truck Transaction

For a given shipment request (city and load), SmartDispatcher will determine all trucks having the same destination as the request, and having sufficient load capacity (maximum load – current load) to carry the new parcel. The shipment will be assigned it to the truck in this group having the greatest "max load – current load" value. The current load of the truck is updated. If the new value of the current load is $\geq$ 90% of the maximum load, the truck is sent out and is unavailable for further transactions in this state.

If no truck can be assigned (which can result from no trucks going to the destination, or no truck going to the destination with sufficient room for the shipment), SmartDispatcher reports an unsuccessful search.

Output Files

For each of the three dispatching algorithms, SmartDispatcher will generate a file listing the vehicles selected as a response to processing the fleet states and transactions of the corresponding input files, that is:

BusSelections.txt
Each line of this file will either

- list a state number, a selected bus number, its mileage, and its number of seats.
  or
- describe a transaction that could not be satisfied

TaxiSelections.txt
Each line of this file will either

- list a state number, a selected taxi number, and the distance to the user who called the taxi.
  or
- describe a transaction that could not be satisfied

TruckSelections.txt
Each line of this file will either

- list a state number, a selected truck number, the weight of the assigned shipment, its unused weight before the shipment, its unused weight after including shipment.
  or
- describe a transaction that could not be satisfied

SmartDispatcher will also produce a report file named “finalUsageReport.txt”, showing the number of vehicles in use per fleet type at the end of processing the transactions for all states. An example of this file is:

```
Vehicle Type = Bus, State = 1, Fleet Size = 60, In Use = 25
Vehicle Type = Bus, State = 2, Fleet Size = 43, In Use = 5
Vehicle Type = Taxi, State = 1, Fleet Size = 92, In Use = 71
Vehicle Type = Taxi, State = 2, Fleet Size = 120, In Use = 110
Vehicle Type = Truck, State = 1, Fleet Size = 80, In Use = 55
```

Required Work

Your assignment consists of creating the java class hierarchy(s) and code required to implement the dispatching algorithms, processing all the input files for each dispatching algorithm, and generating the specified output files. Your code will also produce file “finalUsageReport.txt”, having the format described above, with the final vehicle-in-use values obtained after your program has completely processed all the input files provided to you.

Submit all your code (i.e., all the java classes), all the input files provided to you, and all the output files that you have generated) in one zip file named “12345678_Assignment3.zip”, where you replace 12345678 by your own student number.

The assignment is due on April 1, 2018, 11:00 PM.