Lab Assignment 3: **Mass Transit Optimizer Tool**

Aim – **To implement a small and simple prototype tool, named MTOptimizer, to explore rough fleet size optimizations in the hourly deployment of the vehicles used in the different public transportation modalities (i.e., buses, subways, trains) at the city of Toronto. The tool takes as input: 1) a representative sample of a typical workday passenger ridership (approximately 10% of the true number of daily passengers) for all means of transportation, 2) fleet descriptions and vehicle properties for each means of transportation. The tool outputs hourly estimates of the optimal number of vehicles, for each means of transportation, that should be in operation, in a 24-hour period, to satisfy the projected demand indicated by the processed sample.**

MTOptimizer reads passenger and vehicle data from text files containing comma-separated values. It writes results generated from the processing of its input data to a text file. The problem description has been simplified in many ways to keep a very small number of scenarios to consider and produce rough estimates of fleet size allocations. For example, each means of transportation is represented just by one operation route (i.e., one line of subway, or GO-train, or streetcar, or bus, or GO-bus).

The two main learning objectives in this assignment are: 1) to design and implement small class hierarchies, and possibly use interfaces to delineate those classes, which could efficiently support the programming required in a given application domain (e.g., public transportation), and 2) to give you practice with allocating attributes and behaviours (i.e., methods) in class hierarchies that facilitate the reusing of programming code and the instantiation of objects that effectively encapsulate data and functionality in support of applications involving multiple actors.

To attain these goals, it is advisable that, before writing any code, you clearly identify and describe to yourself in plain language the nature, core attributes and minimum behaviours of the root class of the hierarchy(ies) that you will use in programming the MTOptimizer tool. It is a good idea to jot down a stylized class diagram, as shown in class. Before diagraming and analyzing your classes, manually inspect the input data for potential patterns, nature of values and unusual or extreme values. This will greatly help you shape the class hierarchy(ies) you will create to support your programming and handle exceptions.

General Description

As a software developer with the Toronto Transit Commission (TTC), you have been tasked to write a software prototype, in Java, named MTOptimizer. This tool would allow the analysts at the TTC to process daily passenger ridership data and, as a result, create in-advance hourly-based "in-operation-vehicle fleet sizes" for the different means of transportation supervised by this organization.

MTOptimizer reads in the following text files:

1) File "**ridership.txt**" contains about 65,000 lines, which represent a 24-hour sample (00:00 hours to 24:00 hours) of the ridership for all means of transportation in the city of Toronto, during a typical working day. Each line represents the recorded data for one passenger,

riding on one of the means of transportation, timestamped at a given hour of the day (i.e., hours 1-24). The list of comma-separated values included on each line is the following:

- **Person Identification** – A 7-digit card identification number for a rider using a monthly pass, or a 16-character ticket code (whose format is: Tyyyymmddnnnnnnn, where yyyy=year, mm=month, dd=day, nnnnnnn=unique 7-digit number) for a rider using a 10-trip discount ticket, or a "*" for a casual rider using tokens or paying cash fares.
- **Transportation Modality** – A one-character field indicating the means of transportation used by the rider, with values as follows: (S) Subway, (G) GO-train, (X) Streetcars, (C) City bus, (D) GO-Bus.
- **Age Group** – A 1-character field indicating whether the rider is a child (C), an adult (A), or a senior citizen (S).
- **Hour of the day** – A 1- or 2-digit number (1 – 24) indicating the hour of the day when the rider used the specified transportation modality.
- **Date** – The date when the rider used the specified transportation modality in the format **yyyymmdd**, where **yyyy** is 2018 or 2019, **mm** is in the range 01-12, and **dd** is in the range 00-31.

2) File "**subways.txt**" describes the fleet of trains on the Toronto subway system. Each line represents a single train. The comma-separated values for each line are:

- **Unit Number** – A unique 4-digit number that identifies a vehicle in the TTC's Computerized Information System.
- **Train Identification** – A 3-digit number in the range 100-999.
- **Number of Cars**– A 1-digit indicating the number of cars per train.
- **Passengers per Car** – A 2-digit number indicating a car's maximum passenger capacity.
- **Operational Status** – A 1-character field indicating whether train is available or not on the date indicated in the Operation Date field: Available (A), Unavailable (U), Unknown (*).
- **Operational Date** – Date when the train operational status is valid, in the format yyyymmdd.

3) File "**gotrains.txt**" describes the fleet of GO-trains to Toronto's suburban areas. Each line represents a single train. The comma-separated values for each line are:

- **Unit Number** – A unique 4-digit number that identifies a vehicle in the TTC's Computerized Information System.
- **Train Identification** – A 4-character string with the format Gnnn, where nnn is in the range 100-999.
- **Train Capacity**– A 2- to 3-digit number indicating the train's maximum passenger capacity.

4) File "**streetcars.txt**" describes the fleet of streetcars in Toronto's downtown area. Each line represents a single streetcar. The comma-separated values for each line are:
   - **Unit Number** – A unique 4-digit number that identifies a vehicle in the TTC's Computerized Information System.
   - **Streetcar Identification** – A 5-character string with the format SCnnn, where nnn is in the range 100-999.
   - **Type** – A 1-character field indicating a single (S) or double (D) section streetcar. (Note: each section of the street car can accommodate a maximum of 40 passengers).
5) File "**buses.txt**" describes the fleet of buses in Toronto's metropolitan area. Each line represents a single bus. The comma-separated values for each line are:
   - **Unit Number** – A unique 4-digit number that identifies a vehicle in the TTC's Computerized Information System.
   - **Bus Identification** – A 5-character string with the format MBnnn, where nnn is in the range 100-999.
   - **Capacity** – A 2-digit number indicating the maximum number of passengers that the bus can accommodate.
6) File "**gobuses.txt**" describes the fleet of buses to Toronto's suburban areas. Each line represents a single bus. The comma-separated values for each line are:
   - **Unit Number** – A unique 4-digit number that identifies a vehicle in the TTC's Computerized Information System.
   - **Bus Identification** – A 5-character string with the format GBnnn, where nnn is in the range 100-999.
   - **Capacity** – A 2-digit number indicating the maximum number of passengers that the bus can accommodate.

Class Hierarchies Design, Program Implementation and Others

I – Class Hierarchies Design

The analysis of the ridership and fleet description files will give many clues to design two or more class hierarchies, which will greatly expedite the programming of the MTOptimizer prototype. It is highly advisable that you consider at least one hierarchy centered on passengers, and another one centered on types of transportation vehicles. The root of each hierarchy must be a non-instantiable class (i.e., an abstract class) that captures essential hierarchy-wide attributes and methods. The derived classes should clearly help you with making easy the programming required to process the input files and produce optimal hourly operational fleet sizes for an entire day.

I – Program Implementation

(a) Assumptions

Your program will use the following assumptions:

- Files "subways.txt", "gotrains.txt", "streetcars.txt, "buses.txt" and "gobuses.txt" contain fleet sizes that will always be sufficient to satisfy the passenger ridership specified in file "ridership.txt". These files do not contain invalid or missing data.
- Because file "ridership.txt" is assembled with data from highly heterogeneous sources (e.g., turnstile readers, on-board readers, manual input, etc.), it is known to contain some lines with missing or faulty values (approximately 0.01% of the total number of lines). These bad lines will be skipped and reported in an "errorlog.txt" files.
- The **optimal fleet size** for each hour of the day and every means of transportation is the minimum number of vehicles (i.e., trains, buses, etc.) that will accommodate the number of passengers specified in the "ridership.txt" file for the respective hour of the day. When computing the total numbers of riders for a given hour assume that an adult is one rider, a child is a 0.75 rider, and a senior citizen is a 1.25 rider.

(b) Properties and behaviours

Your program will have the properties and behaviours described below:

- Its main class will be named **MTOptimizer**.
- It will create object instances of the classes in the hierarchies that you have designed ir order to generate of the lines of the "**InOperationFleets.txt**" output file.
- The contents of file "**InOperationFleets.txt**" is laid out in sections. Each section starts with one of the strings **[Trains]**, **[GoTrains]**, **[Streetcars]**, **[Buses]**, **[GoBuses]**. Each of these section starting lines will be followed by 24 subsections, each one starting with a line **[Hour = nn]**, where nn = 1 – 24. Each [Hour=nn] line will be followed by one or more lines showing the vehicles in use during that hour. The format of these lines will be **train:** *description*, or **gotrain:** *description*, or **streetcar:** description, or bus: *description*, or **gobus:** *description*. Each section will be finished with a **[Count=nn]** line, where nn=number of vehicles in the in-operation fleet for the current hour. The *description* field will match the fields provided in the input files for the different means of transportation.
- It will insert a blank line after each section, except after the last one.
- It will select the vehicles, within a fleet type, that will be part of the fleet representing the **optimal fleet size** for every hour of the day. It will use the riders' age group to compute, as explained before, the **total hourly ridership** for each means of transportation.
- It will produce a "**errorlog.txt**" file listing any bad lines in the "ridership.txt" file, with an associated error message clearly describing the nature of the error.

II – Submission

Compress the source code for class **MTOptimizer**, the classes in the hierarchies that you have designed, files **InOperationFleets.txt** and **errorlog.txt**, into a single zip archive file named:

[your student number]_Assignment3.zip

For example, if your student number is 12345678 then you will name your submission file as 12345678_Assignment2.zip

Submit file **[your student number]_Assignment3.zip** via onQ by **March 28, 2019, 11:00 p.m.**

III – Marking Scheme

• Implementation of class **MTOptimizer**	8 points
• Implementation of the class hierarchies	4 points
• Generation of **InOperationFleets.txt** file	4 points
• Generation of **errorlog.txt** file	2 points
• Code style and documentation	2 points
TOTAL:	20 points