CmpE 160 - Introduction to Object Oriented Programming

Project #2 - Cargo Delivery Simulation
Due Date: 30.04.2019 23:55 PM

1.Introduction

In this project you are going to implement a cargo delivery simulation. There are trains, train stations and cargo packages. The main aim of this project is to deliver each cargo package in each train to the right train station. However, while delivering, there are certain rules which must be taken into consideration.

Train consists of some number of carriages that are connected with each other in order. These carriages can store certain amount of cargo packages in the last in first out format.

When the train comes to a train station, it must unload all of the cargo packages within it to the queue of the train station, and after giving the packages which must be delivered to that station, the packages in the queue must be loaded to the train again with the first in first out order. The process of unloading of carriages must go through from the head of the train to the tail of the train. In other words, the first carriage in the train must be unloaded at first, and afterwards the second carriage must be unloaded and so on.

2.Class Diagram and Implementation Details

You are provided with 5 empty class files.
You have to implement `Train, Carriage, Station, Cargo` and `Main` classes.
`Main` class takes two arguments: input and output files.
Station index starts with 0.

Train.java:

`Train.java` represents the train objects. It holds the carriage capacity as `carCapacity`, length of the train as an integer `length`, and head and tail carriages of the train as `head` and `tail`.

This class has the constructor `Train(int length, int carCapacity)` and the methods `load(Queue<Cargo>)` and `unload(Queue<Cargo>)`.

`Train(int length, int carCapacity):`

Takes the length of the train (number of total carriages) and the capacity of each carriage in the train as integers and initializes the necessary fields.

`load(Queue<Cargo> cargos):`

This method will load all of the cargos into the train starting from the first carriage. If all the carriages in the train are full, add necessary amount of carriages to the train. If one or

more carriages are left empty after the loading train, you should leave those empty carriages at the current station before leaving (by removing them from the train).

`unload(Queue<Cargo> cargos):`

Train will unload all of its cargo to cargos starting with the first carriage until the train is completely empty.

Carriage.java:

`Carriage.java` represents the carriages of the train. Each carriage of the same train has the same capacity. Each carriage object will contain an integer capacity field as `emptySlot` and a stack of cargos as `cargos`. Also, there will be pointers for the next and previous carriages as `next` and `prev`.

This class will contain the methods (and the constructor) below:

`Carriage(int capacity):`

Initializes the necessary fields.

`isFull():`

Checks if the carriage is full, returns true if it is full and false otherwise.

`push(Cargo cargo):`

Pushes a cargo into the cargo stack of the carriage.

`Cargo pop():`

Pops a cargo from the cargo stack of the carriage.

Station.java:

`Station.java` holds `id` as the station id, `cargoqueue` as the queue of cargos and a print stream object called `printstream` to give the necessary output.
This class also contains the `process(Train)` method.

`process(Train train):`

Handles the events from the train's arrival to the leaving of the train.This method also takes care of all the output operations. In this method train unloads into `cargoqueue` and prints the cargos that have reached their destination without changing the order in the `cargoqueue`. Then it loads rest of the queue into the train also without changing the order. Lastly prints the length of the train.

Cargo.java:

`Cargo.java` has mainly 3 fields. These fields are `id, loadingStation` and `targetStation`.This class also overrides `toString()` method to print the output respective to the fields.

Main.java:

`Main.java` consists of two methods:
First, write a method named `readAndInitialize()`, reading the inputs and initializing necessary classes and fields, and also placing cargos to their initial station.
Second, write a method named `execute()` that will start the train from the first station, it will move the train and perform the necessary operations until train leaves the last station.

INPUT FORMAT:

First line of the input consists of three integer values:
Initial number of carriages the train has, the capacity of each carriage, and the number of stations the train will stop.

Rest of the lines have three integer for each line:
Id of the cargo itself, id of the station the cargo is waiting and id of the target station.

OUTPUT FORMAT:

There are 3 variables to be printed for each cargo:
Id of the cargo, id of cargo's initial station and id of cargo's target station.
Each cargo will be printed respective to the order of unloading.
Each output is separated with next line.
Also while the train is leaving each station, the final amount of carriages is printed.

3. Some Remarks

- Do not use the **LinkedList** class of Java explicitly, you are expected to write the necessary linked list operations yourself.
- Please keep in mind that providing the necessary accessibility and visibility is important: you should not implement everything as public, even though the necessary functionality is implemented. The usage of appropriate access modifiers and other Java keywords (super, final, static etc.) play an important role in this project since there will be a partial credit, specifically for the software design.
- There will be also a partial credit for the code documentation. You need to document your code in Javadoc style including the class implementations, method definitions (including the parameters, return if available etc.) and field declarations. You do not need to create and submit a documentation file generated by Javadoc as the software documentation.
- Please do not make any assumptions about the content or size of the scenarios defined by the input test files. Your project will be tested through different scenarios, so you need to consider the all the possible criteria and implement the code accordingly.