COSC2391/2401 Further Programming

TuteLab 2 – Courier Management System (CMS)

The Scenario

A Courier company wants to build a system that schedules incoming delivery jobs to its Fleet of Vans and Trucks. Each job requires that a vehicle travel some distance to deliver a given item, and thus incurs a proportional cost. For simplicity, the cost of a Job is calculated as the wear and tear expense of the scheduled vehicle and a percentage profit margin (calculations described in *A.* below). A vehicle must be serviceable in order to be scheduled to a Job. The serviceability (i.e. "schedulability") of a vehicle is dependent on its usage and maintenance data, and is calculated based on the serviceability constraints discussed in *B)* below.

A) The Fleet

The company uses both vans and trucks. Both vehicle types have a
- Registration number (used to identify the vehicle);
- Year, make and model
- Usage information indicated by an odometer reading (in km)
- Maintenance data consisting of a
  - *Last service point:* the odometer reading at the time the vehicle was last serviced, and
  - *Service interval*: the maximum distance a vehicle can travel before it has to be serviced again. This information is specific to each vehicle, and should not be modifiable once the initial information is entered.

Trucks have an additional piece of information; *load capacity*.

The *wear and tear expense* for a job can be calculated as such:
- Vans: a flat rate of 60 cents per kilometre
- Trucks: a rate of 50 cents per tonne (load capacity) per kilometre *(for example: a truck with a load capacity of 2.5t, has a rate of $1.25 a kilometre.)*

The Profit Margin for a Job is a Fixed Rate of 50% of the wear and tear expense.

B) *Vehicle Serviceability/Schedulability Constraint:*

When a vehicle is to take a job, information about the job (i.e. the distance in kilometres) is provided to the system. The system must then check to see if the specified vehicle is serviceable (with regards to the specified Job). This is determined by checking if the vehicle would exceed its next maintenance point (next service point) if it were to travel the distance associated with the given Job; if so, the Job should be refused. This is to ensure that all Vehicles on duty remain roadworthy (i.e. within their maintenance schedules) for the duration of a Job. If however a vehicle is serviceable, its usage information should be updated and the approximate wear-and-tear cost of the job reported.

Example: A van with an odometer reading of 33,449km, a service interval of 20,000km and a last service point of 14,449km, can accept Jobs with a total distance of as much as 1000km, before it has to be serviced again. This is because the next service point of the vehicle is 34,449km.

C) The CourierManagementSystem:

At the top level, the CourierManagementSystem section of the system must allow *at least* the following core actions to take place:

- Add a vehicle to the system
- Display information about a vehicle

- Schedule a vehicle on a Job (If the vehicle is serviceable, the Job should be accepted, the usage information of the vehicle updated, and the wear-and-tear estimate returned)
- Service a vehicle

You may find that other functionality may be necessary, to help facilitate these actions.

Exercise

Implement the above system in two stages. 1) create a ***class diagram*** for your solution and then convert your class diagram to ***Java code***.

This will likely involve an ***iterative process*** of first doing design, then implementation, and then identifying areas where you need to go back and modify your original design. Repeat this process until you are happy with your final design and implementation.

If you have trouble with the implementation, discuss with your peers (and the teaching staff) the possible ways of overcoming the problems.

To simplify testing you can use a provided application driver class (`CMSTestHarness.java`) designed to test your system in a structured manner in order to produce the output in `OutputTrace.txt`. The source code for this class and trace are available on Canvas alongside this tutelab sheet. NOTE: This code deliberately places some constraints on your implementation in terms of the required classes and methods in order to ensure that the code written by separate authors can work together. Nevertheless, you still have freedom in how you choose to implement your solution; however, you must implement it in such a way that the `CourierManagementSystem` type naming (e.g. class names) is not modified (with one exception, see comments in the supplied source code for details!). Consider the following lines of code from the test harness:

```
final static CourierManagementSystem cms = new CourierManagementSystemImpl();
...
cms.addVehicle(new Van("v1", "Toyota", "Sienna", 1998, 0.0, 500.0));
```

Which specific constraints, in terms of the required types and associated constructors and methods, can be identified from the above code? **NOTE**: The `CMSTestHarness.java` provides some additional comments that will help regarding constructors etc.

NOTE: So far in the lectures we have only covered inheritance and polymorphism and are yet to cover abstract classes and interfaces so there is no need to use abstract classes and interfaces if you are not confident. That said if you want to work ahead you can consider how abstract classes and interfaces could make your code more extensible which will help next week since you will be extending your code with new functionality :)