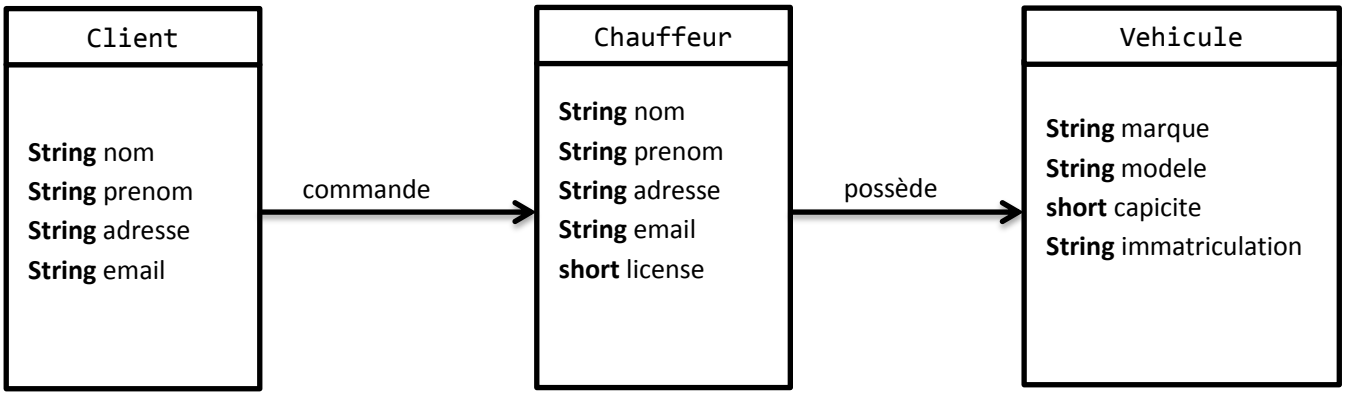
EPSI – JAVA – 3ème année –Comprendre et maitriser les objets

La société de taxi française Hubert souhaite se doter d'une application pour concurrencer un nouveau venu sur le marché au nom éponyme. Cette application permettra de mettre facilement en relation des chauffeurs de taxis d'une part, et des clients d'autre part. La société Hubert se rémunérant grâce à une commission perçue sur chaque course.

Vous avez été sélectionné par Hubert pour réaliser cette application, après un premier entretien avec les dirigeants de la société vous avez dégagé un premier modèle de la future application:



Par soucis de simplification, la notion de position sera implémentée à l'aide du code postale. On pourra par exemple trouver un taxi à proximité de Lyon 1er en recherchant le code postal 69001.

Exercice 1 : Premières classes

1. Implémenter les classes `Chauffeur`, `Client` et `Voiture` dans le package `fr.hubert.model`.
    a. Chaque classe doit posséder un constructeur par défaut et un constructeur recevant l'ensemble des attributs
    b. N'oubliez pas les getter/setter
    c. Redéfinissez la méthode `toString` sur l'ensemble des classes
    d. Redéfinissez la méthode `equals` et `hashCode` sur la classe Voiture
    e. Utilisez les possibilités de génération de code de votre IDE pour les autres classes
2. Testez votre implémentation à l'aide d'un programme Console.
3. Ajouter un compteur sur les classes `Client` et `Chauffeur` afin de connaitre le nombre de client ayant été instanciés par l'application à la fin du programme.

Exercice 2 : Un peu de logique…

1. Mettez à jour votre programme pour qu'il soit possible de :
    a. Crée un compte client
    b. Chercher les taxis disponibles en fonction d'une position
2. Votre programme doit lever une exception si :
    a. Aucun taxi n'est disponible
    b. L'email du compte client est déjà utilisé
3. Si ce n'est pas déjà le cas, géré les clients et les taxis en utilisant des listes de l'API collection.

Note :

- La levée d'une exception ne doit pas provoquer l'arrêt du programme
- Les exceptions doivent être « rangées » dans le package `fr.hubert.exception`

Votre client est satisfait de cette première version et vous avez pu discuter avec lui des futures ambitions de son entreprise. Vous avez notamment pu dégager les notions suivantes :

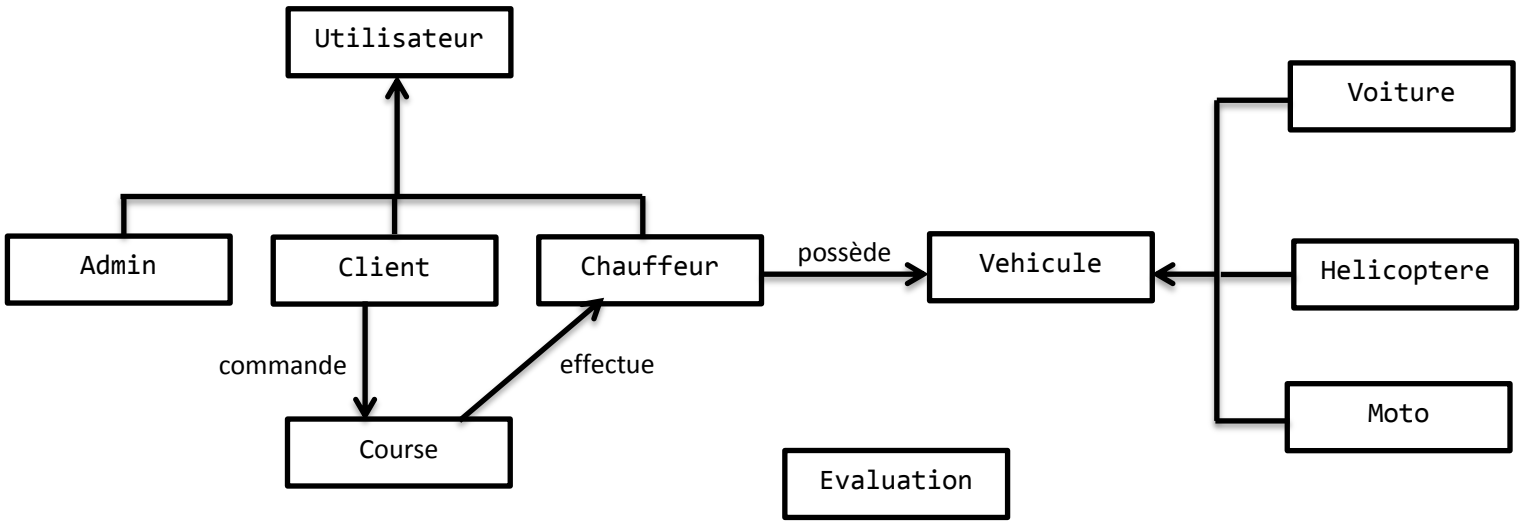
L'application devra gérée plusieurs types d'utilisateur qui ne pourront pas effectuer les mêmes actions :

- Un client pourra rechercher et réserver un chauffeur
- Un chauffeur pourra enregistrer son ou ses véhicules et se déclarer disponible
- Un administrateur pourra consulter les statistiques d'utilisation de l'application
    - Nombre et type d'utilisateurs inscrit
    - Montant moyen de courses
    - Type de véhicules possédés par les chauffeurs
    - Etc…

Tous les comptes utilisateurs doivent être protégés par login et un mot de passe.

Les chauffeurs pourront offrir des prestations avec différents types de véhicule (voiture, moto et même hélicoptère), bien évidement le prix des prestations ne sera pas le même…

A l'issue de cet entretien vous êtes parvenu à un modèle de donnée plus abouti :



Exercice 3 : Mise en œuvre de l'héritage

1. Implémentez et testez les hiérarchies des classes `Utilisateur` et `Vehicule`
2. Implémentez et testez la logique d'authentification
    a. Cette logique doit être encapsulée dans la classe `Utilisateur`
3. Ajoutez une fonctionnalité permettant à un administrateur de connaitre le nombre de véhicule de chaque type disponible dans l'application

Note :

- Les classes `Utilisateur` et `Vehicule` doivent être abstraites

L'application Hubert doit être une application sociale qui doit mettre en œuvre le principe de réputation. Les clients doivent ainsi pouvoir évaluer les chauffeurs, les véhicules et les courses. Les chauffeurs peuvent quant à eux évaluer les clients. La notion d'évaluation inclue une note allant de 0 à 5 et éventuellement un commentaire.

Exercice 4 : Utilisation des interfaces

Implémentez la logique d'évaluation décrite précédemment à l'aide des interfaces `IEvaluable` et `IEvaluateur` :

- L'interface `IEvaluable` doit comporter au moins deux méthodes :
    - `void setEvaluation(short note)`
    - `List<Short> getEvaluations()`
    - `void short getEvaluationMoyenne()`
- L'interface `IEvaluateur` doit comporter au moins une méthode :
    - `void evaluer(IEvaluable element)`

Exercice 5 : Persistance des objets

Mettez à jour votre programme pour que :

- Les objets de l'application soient persistés après chaque opération de création/modification
- Les objets de l'application soient chargés au démarrage de votre programme

Exercice 6 : Pour aller plus loin

Implémentez la classe `Course`, cette classe doit être instanciée à chaque fois qu'une course est commandée par un client à un chauffeur. Elle doit notamment comporter les attributs suivants :

- L'origine de la course
- La destination de la course
- Le client
- Le chauffeur
- L'état de la course : en cours, commandée, terminée
- Si la course est terminée le prix facturé en fonction du type de véhicule

Chaque utilisateur doit pouvoir consulter l'historique de courses effectué.