Programmation orientée objets

Ensimag 2A, examen du lundi 15 décembre 2014

Durée : 2h (aucune sortie autorisée avant la fin de l'examen)

Documents : tous documents papiers autorisés

Le barème est indicatif (total sur 22 points).

Votre code Java devra contenir des **commentaires et des explications pertinentes**. Les réponses aux questions attendant une réponse textuelle seront aussi données sous forme de commentaires dans le code. Par contre la documentation javadoc n'est pas demandée.

Dans la mesure du possible, le code devra compiler et s'exécuter correctement. Si ce n'est pas le cas, en particulier pour les dernières questions, donnez en commentaire des explications et/ou du pseudo-code sur ce que vous cherchez à réaliser.

Vous disposez sur votre machine de deux liens vers :
- la documentation de l'API des classes Java (en html, naviguable)
- une copie locale du Kiosk du cours de POO. Seuls les documents sont accessibles, pas les liens externes.

N'oubliez pas :
- de remplir le fichier `QUI_SUIS_JE.txt` avec **votre nom, prénom et groupe** ;
- d'activer régulièrement le lien du bureau **"Sauvergader l'examen sans quitter"** ;
- d'activer **"Sauvegarder et terminer l'examen"** avant de quitter la salle.

Le but de ce sujet est de développer un ensemble de classes Java mettant en place un système simplifié de location de véhicules.

1 Hiérarchie de véhicules (6 points)

Les premières questions visent à mettre en place la hiérarchie de classes qui va nous permettre de décrire des véhicules de différents types.

La classe `Vehicule` est la classe mère de la hiérarchie. Elle comprend trois attributs : l'immatriculation du véhicule (une chaîne de caractères), un nombre de chevaux qui détermine la puissance du véhicule (un entier), et une consommation en litres pour 100 kilomètres (un réel [1]). La consommation doit toujours être supérieure à 2.0l/100km. Pour tenir compte du vieillissement du parc, la consommation d'un véhicule doit pouvoir être modifiée après instanciation de l'objet. Ce n'est pas le cas de tous les autres attributs.

La classe `Vehicule` a deux sous-classes distinctes `Voiture` et `Utilitaire`. La classe `Voiture` n'a pas d'attributs supplémentaires. La classe `Utilitaire` correspond à des véhicules de dimensions plus conséquentes et déclare donc un attribut supplémentaire : sa hauteur (un réel). La classe `Utilitaire` est spécialisée en deux sous-classes `Fourgonnette` et `Camion`. La classe `Fourgonnette` ne déclare pas d'attributs supplémentaires. La classe `Camion` déclare un attribut correspondant au poids du camion (un réel).

1. Il est conseillé pour cet examen de définir tous les réels en utilisant le type **`double`**.

On considère que deux objets `Vehicule` sont égaux si et seulement si ils ont le même numéro d'immatriculation.

Question 1 : Classes, attributs et constructeurs (2.5 points) Coder en Java les classes introduites précédemment. Chaque classe définit ses attributs avec les bons niveaux de visibilité et doit être équipée d'au moins un constructeur qui prend en paramètre une valeur pour chacun des attributs.

Question 2 : Définition de méthodes (2.5 points) Compléter votre code avec les accesseurs et mutateurs pertinents. Équiper également chaque classe de la hiérarchie d'une méthode qui renvoie une chaîne de caractères indiquant l'ensemble des caractéristiques de l'objet. Il est enfin demandé d'écrire une méthode qui vérifie si deux véhicules sont égaux et une méthode renvoyant une valeur de hachage ("*hash code*") permettant d'utiliser un véhicule comme clé d'une table de hachage Java. Les noms et signatures de toutes ces méthodes seront conformes aux usages en vigueur en Java.

Question 3 : Programme de test (1 point) Écrire un programme de test qui instancie quelques véhicules à partir des classes définies auparavant. Ces objets seront stockés dans un tableau de `Vehicule`. Enfin il est demandé de parcourir ce tableau et d'afficher les caractéristiques des véhicules référencés par le tableau et de tester aussi la méthode `equals`.

2 Calcul de coût (3 points)

Nous désirons maintenant compléter les classes afin de calculer le coût quotidien de la location pour les véhicules de la hiérarchie. Ce calcul se fait à partir d'une base qui est de 20 euros pour les véhicules de type `Voiture` et de 30 euros pour les véhicules de type `Utilitaire`. À cela il faut ajouter 2 euros par cheval. Enfin, pour les camions, le coût est aussi augmenté de 2 euros pour chaque tranche de 200 kilogrammes supplémentaire au delà de 2000 kilogrammes.

Question 4 : Calcul du coût (2.5 points) Ajouter la méthode `double calculerCout()` dans les classes de la hiérarchie.

Question 5 : Test du calcul de coût (0.5 point) Compléter le programme de test en parcourant le tableau de véhicules et en affichant pour chacun son immatriculation et son coût.

3 Location de véhicules (8 points)

Pour simplifier, nous considérons le loueur de véhicules comme un endroit où des clients peuvent se présenter pour louer un véhicule ou retourner un véhicule. La location et le retour sont immédiats et nous ne gérons pas non plus de notion temporelle, comme la durée de location ou la date de retour.

Soit la classe `Client` suivante qui définit un client en lui associant un nom et un prénom, que l'on suppose uniques dans la suite (il n'existera jamais deux clients avec le même prénom et le même nom).

```
public class Client {

    private String nom;
    private String prenom;
    // ...

    public Client (String nom, String prenom) {
        this.nom = nom;
        this.prenom = prenom;
        // ...
    }

    // ...
}
```

Complétons tout d'abord la classe `Client` afin qu'à chaque client soient associés les véhicules qu'il est en train de louer. Tous les structures de données et traitements associés dans la suite du sujet doivent être aussi optimaux que possible.

Question 6 : Classe `Client` (1.5 points) Ajouter un attribut à la classe `Client` permettant de stocker les véhicules loués par un client. Compléter le constructeur. Écrire une méthode `ajouterLocation` qui prend un véhicule et l'ajoute aux véhicules loués. Écrire une méthode `supprimerLocation` qui prend un véhicule et le supprime des véhicules loués.

D'autres méthodes pourront être ajoutées à la classe `Client` dans la suite du sujet si cela s'avère nécessaire pour répondre à d'autres questions.

Nous pouvons maintenant définir la classe `Loueur` qui permet de gérer les clients et un parc de véhicules. Nous proposons de stocker ces attributs en utilisant les structures abstraites ci-dessous. Pour les clients, la clé de l'association peut être obtenue en concaténant le prénom et le nom par exemple, puisque tout couple nom/prénom est considéré unique.

```
public class Loueur {

    private Collection<Vehicule> vlibres;   // vehicules libres
    private Map<String, Client> clients;    // clients

    // ...
}
```

Question 7 : Constructeur et affichage (2 point) Écrire un constructeur sans paramètre pour votre classe `Loueur`, qui initialise les différents attributs. Écrire également une méthode retournant une chaîne de caractères décrivant le loueur (ses véhicules libres, ses clients et les véhicules loués par chaque client).

Question 8 : Méthodes d'ajout (1.5 points) Écrire une méthode `ajouterVehicule` qui prend un véhicule en paramètre. Un nouveau véhicule est initialement libre. Assurez-vous qu'un véhicule ne puisse pas être ajouté deux fois malencontreusement.

Écrire ensuite une méthode `ajouterClient` qui prend le nom et le prénom d'un client en paramètre, crée un objet `Client` et l'ajoute aux clients connus par le loueur. Au départ, un

nouveau client ne loue aucun véhicule. La méthode retourne le client créé. Si le client est déjà connu du loueur (même nom, même prénom), la méthode retourne le client correspondant.

Question 9 : Méthodes de mise à jour (4 points) Écrire une méthode `louer` qui prend en paramètre un véhicule, le nom et le prénom d'un client, et qui met à jour les attributs de façon à ce que le véhicule ne soit plus libre et soit associé au client passé en paramètre. Cette méthode doit également vérifier la pertinence des paramètres et éventuellement lever des exceptions. Par exemple, on ne peut pas louer un véhicule qui n'est pas libre ou qui n'est pas connu du loueur, et on ne peut pas louer un véhicule à un client qui n'est pas connu du loueur.

Écrire une méthode `rendre` qui prend en paramètre un véhicule et qui met à jour les attributs de façon à ce que le véhicule redevienne disponible, et ne soit donc plus associé au client qui le louait. Lever des exceptions si nécessaire.

Dans des commentaires sans modifier votre code :

- Pour chacune de ces deux méthodes, donner la complexité au pire cas.
- Si vous le jugez souhaitable pour optimiser certaines méthodes, proposez des modifications des attributs de la classe `Loueur` et/ou des ajouts de nouveaux attributs dans la classe `Loueur` et/ou des modifications dans d'autres classes. Justifiez vos propositions. Indiquez quelles autres modifications deviendraient nécessaires. Indiquez comment la complexité des principales méthodes seraient impactées.

Question 10 : Test du loueur (1 point) Compléter votre programme de test en créant une instance de la classe `Loueur`, en y stockant quelques clients et quelques véhicules, puis en appelant les méthodes `louer` et `rendre` définies précédemment et en affichant l'état du loueur à chaque étape.

4 L'appât du gain (3 points)

Nous voulons maintenant tenter d'optimiser les gains du loueur. Pour cela, nous voulons que le loueur ait accès de façon efficace au véhicule libre qui soit le plus intéressant pour lui, c'est-à-dire celui dont le coût quotidien est le plus élevé.

Question 11 : Attribut et mise à jour des méthodes (2 points) Rajouter un attribut à la classe `Loueur` qui stocke les véhicules libres dans une structure de données additionnelle. Justifier votre choix. Mettre à jour le constructeur de la classe `Loueur` ainsi que les méthodes `ajouterVehicule`, `louer` et `rendre`. Indiquer le coût de ces trois méthodes. Si du code a besoin d'être ajouté au delà de ces modifications, l'indiquer.

Question 12 : Meilleur profit et test (1 point) Ajouter à la classe `Loueur` une méthode `getMeilleurProfit` qui renvoie le véhicule de coût quotidien le plus élevé. Donner le coût de cette méthode. Compléter le programme de test pour appeler cette méthode sur un exemple concret.