Programmation Objet en Java - Feuille de TP 5

Exercice 1. On souhaite créer une classe permettant de représenter des voitures.

a) Une voiture est définie par un modèle (une chaîne de caractères `model`) ainsi qu'une année de fabrication `year`. De plus, elle peut être tout-terrain ou non (booléen `offRoad`). Créer une classe `Car` qui représente une voiture et écrire le constructeur permettant d'initialiser ses trois champs. Attention, les champs ne peuvent pas être vides.

b) Ajouter un constructeur (utilisant le précédent) ne prenant en paramètre que le modèle et l'année pour créer des voitures qui ne sont pas tout-terrain par défaut.

c) Ajouter de quoi afficher une voiture de la façon suivante :

```
Car mustang = new Car("Ford Mustang", 2014);
System.out.println(mustang); // Ford Mustang de 2014 (tout-terrain: non)
Car beetle = new Car("Coccinelle", 1985, true);
System.out.println(beetle); // Coccinelle de 1985 (tout-terrain: oui)
```

d) Écrire la classe de test et créer quelques voitures pour vérifier que les constructeurs et l'affichage fonctionnent.

Exercice 2. Le but de cet exercice est de créer et modifier les classes permettant de gérer une agence de location de voitures.

a) Écrire une classe `Rental` qui représente une agence de location de voitures et qui stocke l'ensemble des voitures qui peuvent être louées dans une `ArrayList`, ainsi que l'année courante (`currentYear`), par exemple 2017.

b) Ajouter ce qu'il faut pour pouvoir créer un `Rental` à partir de l'année courante.

c) Écrire une méthode `setToNextYear` qui augmente l'année courante de 1.

d) Écrire une méthode `add` qui permet d'ajouter une voiture à l'agence.

e) Ajouter de quoi afficher l'agence avec l'année courante suivie de toutes les voitures qu'elle peut louer, séparées par des petites lignes:

```
Agence (2017)
-----
Ford Mustang de 2014 (tout-terrain: non)
-----
Coccinelle de 1985 (tout-terrain: oui)
-----
```

f) Écrire une méthode `contains` permettant de tester si l'agence de location contient une voiture donnée et tester que le code suivant fonctionne:

```
Car mustang = new Car("Ford Mustang", 2014, false);
Car mustang2 = new Car("Ford Mustang", 2014, false);
Rental rental = new Rental(2017);
rental.add(mustang);
System.out.println(rental.contains(mustang2)); // true
```

g) Le coût d'entretien d'une voiture dépend de son année de fabrication. Plus elle est récente plus elle coûte cher:

- Le coût d'entretien de base est de 100 euros.
- Si la voiture a été fabriquée après 2000, on ajoute 10 euros de plus par année après 2000 (+ 10 euros si elle est fabriquée en 2001, + 20 euros en 2002, ...).
- De plus, si la voiture est tout-terrain, on ajoute 50 euros.

Par exemple, la `Ford mustang` a un coût d'entretien de 240 euros et la `Coccinelle`, un coût de 150 euros. Écrire un méthode `careCost` permettant de calculer le coût d'entretien d'une voiture.

h) Écrire une méthode `totalCareCost` permettant de calculer le coût total d'entretien des voitures que possède une agence de location. Êtes-vous sûr que cette méthode ne peut pas créer d'erreur ?

Exercice 3. Pour attirer plus de clients, une agence de location peut désormais proposer des limousines qui, en plus des autres caractéristiques des voitures, doivent indiquer un nombre de places.

a) Écrire une classe `Limousine` avec un constructeur qui prend en paramètre son modèle, son année de fabrication et son nombre de places (`nbSeats`). Par défaut, une limousine n'est pas tout-terrain. N'oubliez pas que l'on doit pouvoir ajouter une limousine dans la liste des véhicules d'une agence.

b) Ajouter de quoi afficher une limousine ainsi:

```
Limousine limo = new Limousine("Cadillac", 2001, 13);
System.out.println(limo);
// Limousine Cadillac de 2001 (tout-terrain: non), nombre de places: 13
```

c) Assurez-vous que toutes les méthodes du `Rental` continuent de fonctionner.

d) Les limousines ont un coût d'entretien de 500 euros supérieur aux voitures normales. Modifier votre code pour que l'on puisse encore calculer le coût d'entretien des voitures pour toute l'agence, en prenant en compte les limousines.

e) Toujours pour attirer les clients, l'agence souhaite éditer une brochure publicitaire où elle fera figurer ses plus chouettes véhicules:

- tous les véhicules tout-terrain,
- tous les véhicules de moins de 3 ans (par rapport à l'année courante de `Rental`),
- tous les véhicules dont le modèle commence par "Ford",
- toutes les limousines de plus de 12 places.

Écrire une méthode `allNiceVehicles` qui renvoie la liste de tous les véhicules que l'agence compte faire figurer dans sa brochure. Attention, pour cette question, vous ne devez pas utiliser `instanceof`.

Exercice 4.

L'agence de location se diversifie et propose désormais des vélos (classe `Bike`) qui ont une couleur et peuvent être tout-terrain ou non. Le coût d'entretien d'un vélo est de 10 euros, plus 50 euros s'il est tout-terrain. Seuls les vélos rouges figurent sur la brochure promotionnelle.

a) On souhaite pouvoir ajouter indifféremment des voitures, des limousines ou des vélos dans un `Rental`. Pour cela, on va écrire une interface `Vehicle` que devront implémenter ces 3 classes. Quelles méthodes doit déclarer l'interface pour que le `Rental` continue de fonctionner correctement?

b) Écrire l'interface `Vehicle` et faire en sorte que les classes `Car` et `Limousine` l'implémentent et que `Rental` l'utilise.

c) Écrire la classe `Bike` qui implémente également l'interface `Vehicle` et vérifier que l'on peut ajouter des vélos dans un `Rental` au même titre que les voitures et les limousines. Vous ne devez avoir qu'une seule méthode d'ajout.

d) Vérifier que toutes les méthodes de `Rental` fonctionnent, sachant qu'un `Rental` peut désormais contenir des vélos.

e) (Optionnel) Faire en sorte de ne pas avoir de code dupliqué entre les classes des différents véhicules.