Decorator and Adapter Patterns

Exercice 1

1 Partie UML

Au cours de cet exercice, nous allons construire un diagramme de classe incrémentalement. Chacune des questions enrichira donc celui-ci. Pour chaque classe, indiquer ses attributs (privés), ses opérations (incluant la redéfinition de la méthode `toString()`), et son ou ses constructeurs.

On souhaite gérer un banc de test pour étudier les performances de divers composants de voitures. Une voiture est constituée d'un châssis, d'un ou plusieurs moteurs, d'un ou plusieurs types de freins,...

Question 1 • Définir l'interface **Voiture**, comprenant des accesseurs pour une accélération (`float`), un freinage (`float`), une masse (`float`) et pour un prix (`float`).

Question 2 • Définir une implantation `<<vide>>` de cette voiture nommée **Chassis**.

Question 3 • Créer le diagramme de classes comprenant l'interface **Voiture** et la classe **Chassis**.

Comme l'objectif du banc est de tester les diverses combinaisons d'options, vous allez utiliser le pattern *Décorateur* et monter la voiture comme un méccano.

1.1 Le décorateur

La **Voiture** possédant de nombreuses méthodes, définissez un **DecorateurVoiture** abstrait, qui délèguera toutes ses méthodes à la voiture enveloppée.

Question 4 • Compléter le diagramme avec la classe **DecorateurVoiture**.

Question 5 • Donnez le code Java de la méthode `getMasse()` de **DecorateurVoiture**.

Question 6 • Définissez maintenant les décorateurs **MoteurEssence** et **MoteurDiesel** qui modifient les méthodes :
`getAcceleration()`, `getMasse()` et `getPrix()`.

Question 7 • Donnez le code Java de la méthode `getMasse()` de **MoteurEssence** en supposant que le moteur pèse 300 Kg.

Question 8 • De manière similaire, ajoutez les décorateurs **FreinsDisque** et **FreinsFoucault** qui modifieront la force de freinage.

Question 9 • Donnez le code permettant d'ajouter une voiture hybride possédant un moteur essence, un moteur diesel et des freins à disque.

1.2 L'adaptateur

Le **BancDeTest** remplit une fiche de renseignements pour chaque voiture, et sa méthode `lancerTests()` retourne la liste des fiches remplies.

Question 10 • Ajouter au diagramme une classe **Fiche** contenant des données telles que la voiture concernée, sa vitesse maximale ou sa distance de freinage. On ne notera pas les accesseurs/mutateurs par souci d'économie.

On aimerait maintenant trier notre liste de fiches selon divers critères, choisis à l'exécution. En java, la méthode statique `Collections.sort` permet le tri d'une liste, à condition que les éléments soient `Comparable`.

Question 11 • Ajouter au diagramme UML une classe **TriFicheVitesse** qui servira d'*adaptateur* entre une **Fiche** et l'interface `Comparable <TriFicheVitesse>`. Elle devra en particulier définir la méthode `compareTo(TriFicheVitesse autre)` qui retourne un entier -1, 0 ou +1 selon l'ordre des fiches comparées.

2 Partie Java

L'objectif de cette partie est d'implanter et de tester les différents diagrammes que vous avez réalisés dans la première partie du TD. Comme toujours, n'attendez pas d'avoir tout écrit pour tester !

Vous trouverez sur `github` l'interface `Voiture`, qui est un peu plus riche que celle vue dans la partie UML, et les classes `Chassis`, `FreinDisque`, `FreinFoucault`, , `MoteurDiesel`, `MoteurEssence`, `Fiche` et `DynamiqueVoiture`. Cette dernière s'occupe de tous les calculs du banc de test.

Question 12 • Ajoutez la classe abstraite `DecorateurVoiture` qui implémente l'interface `Voiture`.

Question 13 • Connectez les sous-classes `FreinDisque`, `FreinFoucault`, `MoteurDiesel`, `MoteurEssence` à la super-classe `DecorateurVoiture` (héritage). Sachant que la masse, la puissance et le coefficient de freinage sont cumulables, n'oubliez pas de redéfinir les méthodes au niveau des sous-classes.

Question 14 • Dans la fonction `main` de la classe `BancDeTest`, construisez et ajoutez une voiture dotée d'un châssis, d'un moteur, et d'un système de freins. Ajoutez la voiture au banc de test et lancez les tests

Question 15 • Décorer maintenant avec différent composants vos voitures et admirez la force du pattern.

Question 16 • Ajoutez votre adaptateur `TriFicheVitesse`.

Vous pouvez ainsi trier votre liste en utilisant :

```
Collections.sort(resultats, (ficheResultat, autre) ->
    ficheResultat.compareAcceleration(autre));
```

Ou encore, avec une référence de méthode (Java 8+) :

```
Collections.sort(resultats, FicheResultat::compareAcceleration);
```