Université Paris Cité

Compléments en Programmation Orientée Objet
TP n° 5 : fabrique, adaptateurs

1 Adaptateur et Fabrique : Transport aérien

Exercice 1 : Adaptateur et Fabrique

Fichiers fournis : On considère ici le code fourni dans les fichiers `Plane.java`, `TransportRoutine.java`, `OutOfReachException.java`, `OverweightException.java`, `Chopper.java` et `LoggedPlane.java`.

Contexte : Dans le cadre d'un code permettant de modéliser du transport aérien, on s'intéresse à un code déjà produit permettant à partir d'un objet `Plane` de découper le transport d'une charge entre deux points en des segments de transport par cet avion. Le code de la classe `TransportRoutine` s'appuie pour cela sur le contrat fournit par l'interface Plane.

Objectifs : Écrire une implémentation de l'interface `Plane`, nommée `Pelican`, ainsi qu'un adaptateur de la classe `Chopper` pour pouvoir l'utiliser comme un `Plane`. Utiliser le patron "Factory Method" pour déléger la création de l'avion dans la classe `TransportRoutine`.

1. Programmez la classe `Pelican` qui implémente l'interface `Plane`. Elle vérifiera les choses suivantes :
- La charge maximale est de 5 tonnes (5000kg)
- La distance maximale est de 1000km
- On ne peut pas charger ou décharger en vol
- On ne peut pas décoller si la charge est supérieure à 5 tonnes
- On doit avoir atterri avant de parcourir 1000km

2. Un autre développeur à fourni une implémentation des hélicoptères. Malheureusement les hélicoptères ne se comportent pas comme des avions :
- Ils ne peuvent prendre une charge que lorsqu'ils sont déjà en vol
- Ils ne peuvent pas atterrir si ils transportent une charge

Nous voulons pouvoir utiliser un hélicoptère au lieu d'un avion sans avoir à reprogrammer le code de `TransportRoutine` (à part l'instanciation du `Pelican` à remplacer). Proposez un adaptateur `ChopperPlaneAdapter` de `Chopper` en un `Plane` et utilisez-le à la place du `Pelican`.

3. La création du `Plane` est actuellement effectuée dans la méthode `void transport(int freight, int distance)` (vous avez dû modifier une ligne du fichier `TransportRoutine` pour changer d'avion). On voudrait que cette méthode soit indépendante du type d'avion.

On va pour cela utiliser le patron de conception "FactoryMethod". Pour cela, transformer la classe `TransportRoutine` en une classe abstraite. Ajoutez une méthode abstraite `Plane createPlane()` et remplacez dans la méthode `void transport(int freight, int distance)` la création de l'avion par un appel à cette fonction abstraite.

Vous pouvez maintenant créer deux classes `PelicanTransportRoutine` et `ChopperTransportRoutine`, héritant chacune de `TransportRoutine` et implémentant seulement la méthode `createPlane` de la façon appropriée (respectivement création d'un `Pelican` et d'un `ChopperPlaneAdapter`).