Le patron de conception Monteur

Source « https://www.math.univ-paris13.fr/~chaussar/ »

Vous venez d’être employé dans l’équipe de développement d’un nouveau jeux vidéo, et plus précisément, vous devez vous occuper de la partie de gestion des armes.

Exercice 1

Le code fournit permet de gérer trois types d’armes (pistolet, tronçonneuse et bicycle). Si l'on souhaite construire un objet de type `Gun`, il faut le créer (avec un `new`) puis exécuter la liste d'ordres programmée dans le `main`. Cette liste n’est pas sous votre contrôle, et permet de charger correctement dans la mémoire un objet 3D. Elle dépend de l'architecture utilisée, et peut varier. Si un ordre non programmé dans l'objet est demandé, on ignore l’ordre.

Changer le code du main pour charger correctement (en exécutant tous les ordres programmés dans le `main`) un objet de type `Saw`.

Exercice 2

Implémentez le patron de conception Fabrique afin de rendre la création et l'exécution des ordres programmés plus transparente au client. Vous devrez certainement construire une interface de `Weapon`.

Exercice 3

Un premier défaut de cette architecture vient du fait qu’on a mal placé certaines fonctions. Par exemple, dans `Gun`, tout ce qui n’est pas le constructeur devrait être dans une classe permettant de précharger un `Gun` dans la mémoire. Appelons cette classe `GunBuilder`.

Créez `GunBuilder`, qui doit posséder une variable interne de type `Gun`, une fonction `createNewWeapon()`, et une fonction `getWeapon()` (ainsi que toutes les fonctions de `Gun`).

Créez aussi `SawBuilder` et `BikeBuilder`.

Exercice 4

Créez une classe `Director` dont le but sera de construire un `Gun` à l’aide de `GunBuilder`, un `Saw` à l’aide de `SawBuilder`, ou un `Bike` à l'aide de `BikeBuilder`.

Exercice 5

On voudrait maintenant simplifier le `Director` pour que ce dernier n'ait pas du code spécifique pour le `Gun`, le `Saw` ou le `Bike`. L'idée serait de définir un `AbstractWeaponBuilder`, qui implémenterait l'ensemble de toutes les fonctions des `Builder` (implémentation simple : rien ne se produit). Chaque `Builder` spécialisé hériterait de `AbstractWeaponBuilder` et surchargerait seulement les fonctions où l'on souhaite avoir un comportement spécifique.

Exercice 6

Enfin, simplifiez au mieux le code du `Director`. Si tout s'est bien passé, vous devriez obtenir le patron Monteur : on délègue la construction complexe d'objets à des sous-classes spécialisées, et on appelle un `Director` qui s'occupe de faire l'assemblage.

Exercice 7

Ajoutez un autre arme : avez-vous dû modifier beaucoup de code existant ? Ajoutez, à cette arme, le fait qu'elle devra, dans son préchargement, posséder une toute nouvelle fonction (du type `Preload3dEffects`).