	Université de Corse - Pasquale PAOLI	
	Diplôme : MASTER1 DFS	***2018-2019***
	Cours Patterns **TD N°3 : Design Patterns Gof structurels** **Enseignant : Evelyne VITTORI**	

Les exercices sont indépendants les uns des autres.

Exercice 1 - Design Pattern Adapter

On considère à nouveau l'exemple du simulateur de jeu d'aventure évoqué dans les TD précédents. Le simulateur permet de gérer différents personnages de plusieurs types (humains, trolls, orcs, etc.). Chaque personnage peut se battre mais il doit aussi savoir marcher, courir et sauter.

On dispose d'une interface Deplacable (cf. figure 1) pour regrouper l'ensemble des savoir-faire liés aux déplacements.

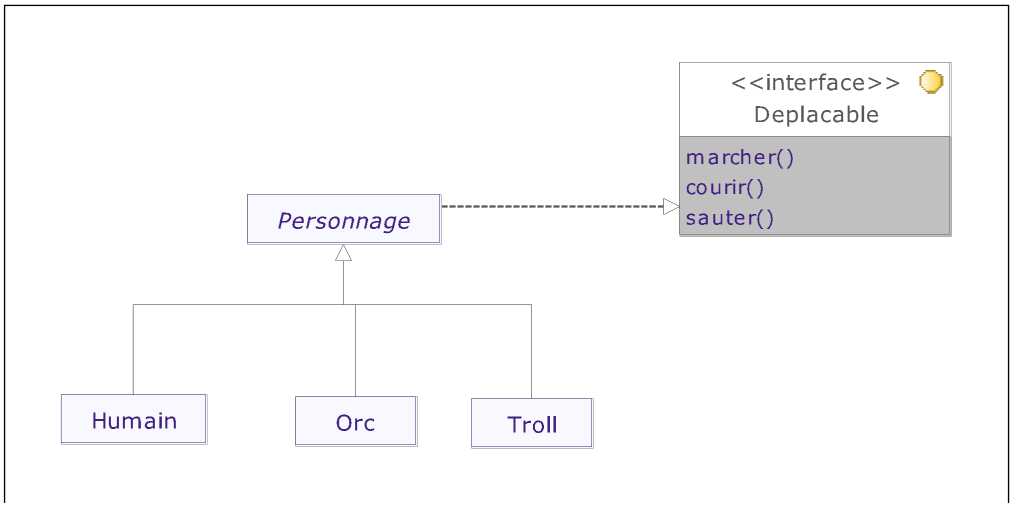

Figure 1 : Interface Deplacable

Comme le montre la figure 1, chaque type de personnage est modélisé par une classe concrète qui doit implémenter l'interface Deplacable .
On suppose à présent que l'on doit ajouter un nouveau type de Personnage, les taurens.
Une autre équipe de concepteurs, nous propose de nous fournir une classe déjà développée et possédant des fonctionnalités similaires. Il s'agit de la classe TaurenEtranger (cf. figure 2).
Les méthodes avancer() et trotter() correspondent respectivement à des implémentations des méthodes marcher() et courir() de l'interface Deplaçable.

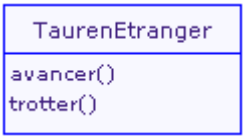

Figure 2 : Classe Tauren Etranger

Pour éviter de recoder inutilement des méthodes qui existent déjà, on souhaite réutiliser les méthodes cette classe TaurenEtranger. Malheureusement, nous ne pouvons pas l'intégrer directement dans notre hiérachie et nous ne pouvons pas non plus la modifier. Le pattern Adapter nous propose deux solutions.

Question 1.1 : Utiliser le pattern Adapter dans sa version « Class Adapter » pour résoudre ce problème. Définissez un diagramme de classe illustrant votre solution.

Question 1.2 : Utiliser le pattern Adapter dans sa version « Object Adapter » pour résoudre ce problème. Définissez un diagramme de classe illustrant votre solution.

Question 1.3 : Coder en Java votre solution de la question 1.2 en utilisant la classe simplifiée TaurenEtranger suivante :

```
public class TaurenEtranger {
	public void avancer(){
		System.out.println("Le tauren se met à avancer!");
	}
	public void trotter(){
		System.out.println("Le tauren avance au trot!");
	}
}
```

Vous définirez un programme de test permettant de créer un personnage de type Tauren, de le faire marcher puis courir.
L'exécution du programme doit donner lieu à un affichage console similaire à l'affichage suivant :

```
Le tauren se met à avancer!
Le tauren avance au trot!
```

Question 1.4 : Représentez sur deux diagrammes de séquence l'enchaînement des opérations mises en œuvre pour réaliser le programme de test de la question 1.3, en supposant :

i. Que vous avez choisi la solution de la question 1.1
ii. Que vous avez choisi la solution de la question 1.2

Exercice 2 - Design Pattern Decorator

On considère un simulateur de jeu d'aventures disposant de deux catégories de personnages : les humains et les taurens et l'on s'intéresse à la modélisation de leurs déplacements.
Une première modélisation a conduit à la définition du diagramme de classe de la figure 3.

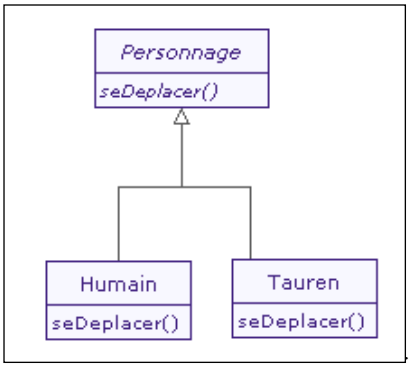

Figure 3 : Diagramme de classe Personnage

La méthode seDeplacer() définit le comportement des personnages lors de leur déplacement. Chaque type de personnage (humain ou tauren) possède un comportement de déplacement spécifique.
On souhaite à présent permettre à ce comportement d'être complété par de nouvelles actions en fonction de l'acquisition d'un ou plusieurs pouvoirs par le personnage.
Ainsi selon les pouvoirs qu'il aura acquis, lorsqu'on lui demandera de se déplacer, un personnage pourra non seulement se déplacer mais en plus devenir invisible et/ou déclencher un orage.
N'importe quel personnage pourra acquérir n'importe quel pouvoir. On peut ainsi imaginer toutes les combinaisons possibles de pouvoirs.
Pour l'instant, on se limite aux deux pouvoirs évoqués ci-dessus mais la conception doit permettre d'enrichir le jeu par la définition de pouvoirs supplémentaires.

Question 2.1 Proposer un diagramme de classe permettant de modéliser ce problème en utilisant le design **pattern Decorator**.

Question 2.2 Coder en Java les classes de votre diagramme.
Définissez un programme de test implémentant le scénario suivant :

- définition d'un personnage de type humain ayant pour nom Titi et possédant les pouvoirs de devenir invisible et de déclencher des orages
- déplacement du personnage (ce déplacement doit se traduire par le fait de se déplacer et devenir invisible et déclencher un orage).

L'exécution de ce programme doit avoir pour résultat un affichage console tel que :

```
L'humain de nom Titi commence à
avancer.
Titi devient invisible!!
Titi déclenche un orage!!
```

Question 2.3 Illustrer le fonctionnement de votre programme de test en définissant un diagramme de séquence modélisant son exécution.

Exercice 3 - Design Pattern Composite

Dans le cadre de la modélisation d'un jeu d'aventures, on souhaite modéliser le contenu de certaines pièces. Une pièce peut en effet contenir des boîtes susceptibles de déclencher une action lors de leur ouverture.
Certaines boîtes contiennent des bombes et provoquent une explosion. L'explosion entraîne la mort du personnage qui a déclenché l'ouverture de la boîte. D'autres boîtes contiennent des bonus (sommes d'argent). Lorsqu'un personnage ouvre une telle boîte, il s'enrichit du montant du bonus. Les boîtes peuvent également être vides ou contenir d'autres boîtes de différentes natures. Les boîtes contenant des bombes ou des bonus ne peuvent pas contenir d'autres boîtes.

L'ouverture d'une boîte doit déclencher l'ouverture en cascade de chacune des boîtes qu'elle contient. Attention, si le personnage ouvre une boite Bombe, il meurt et il n'est donc plus en mesure d'ouvrir les autres boites non encore ouvertes.
Dans une pièce, on doit pouvoir déclencher l'ouverture de toutes les boîtes qu'elle contient mais aussi déclencher l'ouverture d'une seule boîte en spécifiant son numéro d'ordre dans la pièce.

Question 3.1 Utilisez le pattern Composite pour résoudre ce problème.
Définissez un diagramme de classe illustrant votre solution.

Question 3.2 Codez en Java une version simplifiée de votre solution.
Vous définirez un programme de test permettant d'implémenter le scénario suivant :

- Créer successivement trois boites : une boite contenant un bonus de 10000 euros, une boite Bombe et une boite contenant elle-même deux boites (une boite Bonus de 5000 euros et une boite vide).
- Créer une pièce de nom « Cuisine » contenant les trois boites précédentes,
- Créer deux Personnages ayant respectivement comme nom, « Diablo » et « Torino »
- Faire entrer Diablo dans la pièce « Cuisine » et le faire ouvrir toutes les boites,
- Faire entrer Torino dans la pièce « Cuisine » et lui faire ouvrir la boite Complexe (boite Numéro 3),
- Afficher l'état des deux personnages.

L'exécution du programme doit donner lieu à un affichage console similaire à l'affichage suivant :

```
CREATION DE LA PIECE Cuisine contenant 3 boites
- boite N°1 : boite bonus
- boite N°2 : boite bombe
- boite N°3 : boite complexe

MANIPULATION PERSONNAGE DIABLO
Diablo entre dans la pièce Cuisine qui contient 3 boites
Il décide d'ouvrir toutes les boites
-Ouverture de la boite N° 1
Diablo a ouvert une boite Bonus : il gagne 10000 euros!!
-Ouverture de la boite N° 2
```

```
Diablo a malheureusement ouvert une boite BOMBE : il meurt !!

MANIPULATION PERSONNAGE TORINO
Torino entre dans la pièce Cuisine qui contient 3 boites
Il décide d'ouvrir la boite numéro 3
Torino ouvre une boite contenant 2 sous-boites
Torino a ouvert une boite vide!!!!
Torino a ouvert une boite Bonus : il gagne 5000 euros!!

ETAT FINAL DES PERSONNAGES
Diablo est  mort
Torino est  en vie et possède 5000 euros
```

Exercice 4 - Design Pattern Proxy

On considère la modélisation d'une application de vente en ligne de véhicules.
L'application comporte notamment une classe FicheVehicule représentant le composant graphique chargé d'afficher la fiche d'un véhicule et les informations associées. Cette fiche offre également la possibilité pour chaque véhicule de visualiser un film qui présente le véhicule.

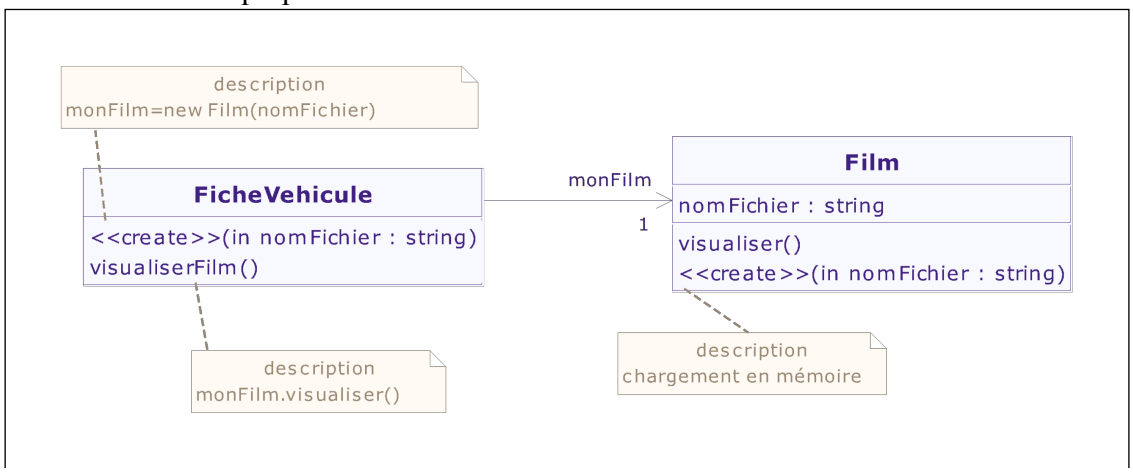

Figure4 : Diag Classe Vehicule-Film

Le diagramme de la figure 4 propose une solution mais on vous demande de l'améliorer sans modifier le corps des méthodes de la classe Film.
En effet, dans la mesure où un film est un objet très lourd à gérer en mémoire (place occupée + temps de chargement), l'objet Film ne devra être créé (et donc chargé en mémoire) que lorsque l'utilisateur demande effectivement sa visualisation en invoquant la méthode visualiserFilm.
Une solution consiste à afficher une photo dans la fiche du véhicule à la place du film lors de la création de la fiche et de déclencher la création et la visualisation du film uniquement lorsque la méthode visualiserFilm est invoquée pour la première fois.
Le pattern Proxy nous donne une solution pour définir cette conception sans modifier la classe Film.

Question 4.1 Définissez un diagramme de classe représentant la solution.

Question 4.2 Coder en java votre solution et définissez un programme de test réalisant les actions suivantes :

1. Création d'une fiche véhicule avec le film contenu dans le fichier « film.mpg » et la photo associée dans le fichier « photo.jpg ».
2. Visualisation du film.

L'exécution du programme doit donner lieu à un affichage console similaire à l'affichage suivant :

```
AFFICHAGE DE LA FICHE VEHICULE
Affichage de la photo : photo.jpg
DEMANDE DE VISUALISATION DU FILM
Visualisation réelle du film (fichier film.mpg)
```

Question 4.3 Définissez un diagramme de séquence représentant l'enchainement d'actions mises en œuvre dans votre programme de test.