TD 2 Design Patterns
composite, itérateur

Il existe de nombreuses façons de stocker des collections d'objets (listes, tableaux, piles, tables de hachages, etc.), chacune ayant ses avantages et ses inconvénients. Le plus souvent l'utilisateur final d'une bibliothèque n'a pas à connaitre la manière de stocker des éléments. Il peut par contre vouloir parcourir les éléments de cette collection. Il faut donc fréquemment fournir aux utilisateurs un moyen de parcourir une collection d'objets tout en masquant l'implémentation de cette collection. C'est ce principe qu'illustre l'exemple traité dans ce td.

Exercice 1 (Prise en main du code)

Nous considérons deux enseignes de restauration rapide « *Rapid Sandwich* » et le « *Le Bon Réveil* », le premier spécialisé dans la vente de sandwichs et le second dans des petits déjeuners à emporter. Les cartes de chacune de ses enseignes sont respectivement :

Rapid Sandwich		
Le végétarien	laitue, tomate, œufs, gruyère	2.50
Le parisien	jambon, emmental, cornichon, beurre	3.00
L'italien	jambon de Parme, mozzarella, tomate	3.00
L'américain	steak haché, frite	4.50
Le chaud	saucisse, béchamel, gruyère	4.50

Le Bon Réveil		
Réveil 1	1 Café ou 1 Chocolat + 1 croissant	2.00
Réveil2	1 Café ou 1 Chocolat + 1 pain au chocolat	2.50
Réveil 3	1 Café ou 1 Chocolat + 1 croissant + 1 tartine	3.00
Réveil 4	1 Café ou 1 Chocolat + 1 pain au chocolat + 1 tartine	3.50

Nous considérons deux codes Java pour implémenter ces deux cartes. Ces deux codes sont structurés en considérant des classes **Menu** et **MenuItem**. Attention la classe Menu représente une carte et la classe MenuItem un plat de la carte.

La classe MenuRapidSandwich implémentant la carte « Rapid Sandwich » utilise un **ArrayList** pour stocker les plats, tandis que la classe MenuBonReveil implémentant la carte « Le bon réveil » utilise un tableau Java classique. La classe **MenuItem** est partagé par les deux codes.

Suite à un rachat, ces deux enseignes ont fusionnées. Les serveurs et serveuses de ce nouveau restaurant doivent pouvoir renseigner les clients sur l'ensemble des cartes et répondre aux questions suivantes :

- Donnez-moi l'ensemble d'un menu ;
- Donnez-moi un item d'un menu ;
- Donnez-moi le prix d'un item ;
- Donnez-moi la composition d'un item.

1) *Observer le code Java fourni et construire le diagramme de classe UML associé.*

2) *Si jamais un troisième restaurant avec une autre carte était racheté par notre nouvelle enseigne, que se passerait-il pour notre classe* ***Waiter*** *?*

Exercice 2 (Faible couplage et itérateur)

Le propriétaire de notre enseigne ne compte pas s'arrêter à deux restaurants et envisage de racheter d'autres enseignes dans l'avenir. Cependant, il est bien conscient que la classe **Waiter** ne pourra pas être modifiée chaque fois qu'une nouvelle enseigne est rachetée. Le problème est qu'actuellement la classe **Waiter** est fortement couplée aux deux classes gérant les menus.

1) *Modifier le diagramme de classe de façon à proposer une solution qui diminue le couplage de la classe Waiter avec les classes MenuRapidSandwich et MenuBonReveil.*

Le problème de notre classe **Waiter** est que chaque carte stocke les instances de **MenuItem** d'une manière différente et qu'actuellement le serveur a connaissance de la façon dont les instances de **MenuItem** sont stockées. Pour remédier à ce problème, utilisez le motif de conception **Itérateur** de telle façon que la classe **Waiter** ne voit plus comment les éléments sont stockés dans chaque carte. Nos itérateurs devront disposer des opérations *hasNext():bool* et *next():Object*.

2) *Définir quelles classes ont la responsabilité de créer les itérateurs et modifier le diagramme de classe.*
3) *Coder le diagramme de classe final.*

Exercice 3 (Composite)

Ça y est, une troisième enseigne « *chez Antonio* » a finalement été rachetée ! En voici la carte :

Chez Antonio

Pizza 4 fromages	gruyères, emmental, bleu, parmesan	7.00
Pizza Regina	tomate, jambon, champignon, gruyère	7.00
Pizza Savoyarde	crème fraiche, lardon, oignons, pommes de terre, reblochon	8.50
Pizza orientale	tomate, gruyère, chorizo, merguez	8.50

Desserts

Glace 1 boule	2.00
Glace 2 boules	3.00
Tarte aux pommes	3.00

En fait, en rachetant cet enseigne, le propriétaire a décidé d'aller plus loin :

- Un client pourra choisir un plat parmi ceux des cartes « *Rapid Sandwich* », « *Le Bon Réveil* » et « Chez Antonio »,
- Un client pourra choisir un dessert s'il a pris un plat de la carte « *Rapid Sandwich* » ou du menu « *Chez Antonio* ».

Il faut donc maintenant prendre en compte le fait qu'une carte puisse être composée à la fois d'un ensemble de plats (instances de **MenuItem**) et également d'autres cartes. Par exemple, la carte « *Le Bon Réveil* » est composée de l'ensemble de ses plats et des desserts de « *Chez Antonio* ». En outre, n'oublions pas que notre propriétaire va probablement continuer à racheter d'autres enseignes !

1) *Sachant qu'un serveur doit pouvoir accéder à chacun des menus de la carte, construire un diagramme de classe présentant l'architecture globale de votre solution en considérant le motif de conception* ***Composite*** *; L'utilisation de ce motif va entraîner des modifications dans la façon de gérer les itérateurs.*
2) *Coder la solution proposée en Java et illustrer l'utilisation de votre solution dans le programme principal.*