TP : Hiérarchie

Compétences visées :

- Mise en pratique des connaissances acquises en modélisation UML.
- Mise en pratique des connaissances acquises en programmation Java.
- Mise en pratique des connaissances acquises en ingénierie logicielle (automatisation des tâches, utilisation d'un logger, tests unitaires…).
- Découverte du patron de conception (*design pattern*) *Composite*.

De nombreuses entreprises ont une organisation hiérarchique dans laquelle les employés sont répartis dans des équipes dirigées par des managers. Nous allons travailler sur un cas particulier d'une entreprise qui a des employés qui peuvent être soit des ouvriers soit des managers. Un manager peut encadrer des ouvriers et/ou des employés que l'on nomme des subordonnés. A la tête de la hiérarchie, on retrouve le patron qui est un manager qui n'a pas de supérieur hiérarchique. La Figure 1 illustre un exemple de hiérarchie.

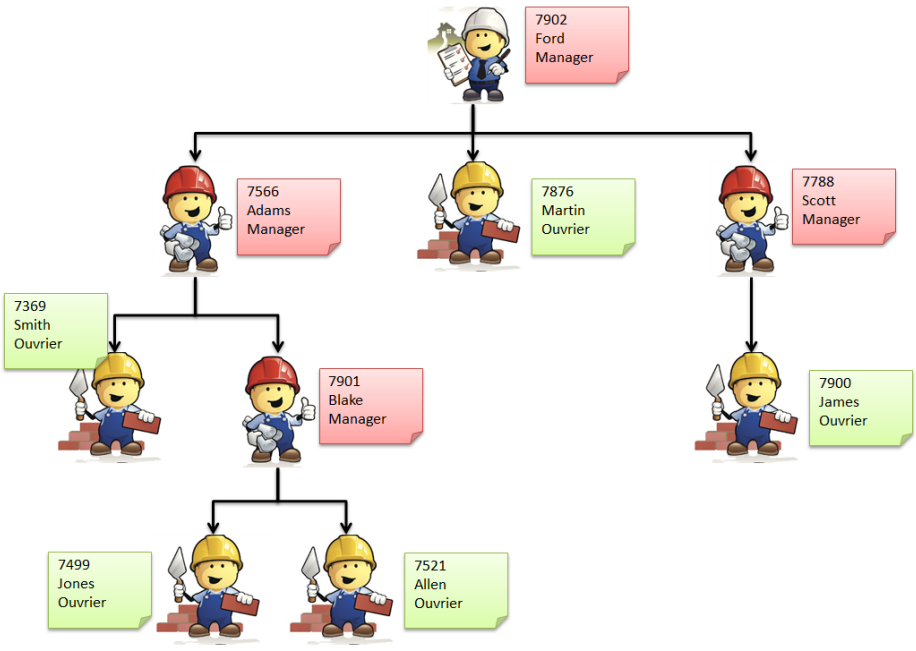

Figure 1 : Hiérarchie de l'entreprise.

L'objectif de l'application que vous allez développer, est de pouvoir compter le nombre d'employés sous les ordres d'un employé.

Pour modéliser la hiérarchie des employés d'une entreprise, nous utiliserons le patron de conception (en anglais, *design pattern*) Composite[1] qui permet d'implémenter des structures de données sous forme d'arbre.

[1] https://fr.wikipedia.org/wiki/Composite_(patron_de_conception)

Conception de l'application

Pendant la phase de conception de l'application, le patron de conception Composite a été adapté au problème de l'entreprise pour aboutir au modèle de la Figure 2.

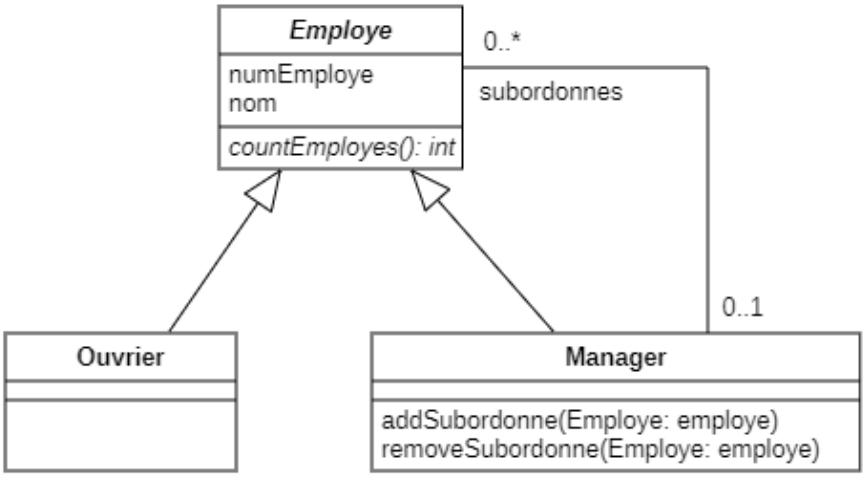

Figure 2 : Diagramme de classes pour la conception.

Question 1 : Est-il possible, avec le diagramme de classes de la Figure 2, de représenter la hiérarchie de l'entreprise donnée dans la Figure 1 ? Illustrez votre réponse avec un diagramme d'objets.

Conception avancée de l'application

Avant d'implémenter l'application, un certain nombre d'éléments doivent être spécifiés de manière plus précise :

- Un employé a un numéro (de type chaîne de caractères) et un nom qui sont définis, une fois pour toute, lors de son initialisation (constructeur).
- Les attributs de la classe Employé doivent être accessibles directement (c'est-à-dire sans getters/setters) depuis les classes Ouvrier et Manager mais accessible depuis n'importe quelle autre classe de l'application via une méthode spécifique (getter et/ou setter selon les besoins).
- Pour des raisons de simplification de l'implémentation, l'association entre Manager et Employé devra :
    - assurée l'unicité de ses subordonnés (pas de doublons) pour un manager sans notion d'ordre entre subordonnés
    - et être unidirectionnelle : « le manager connaît ses subordonnés, un subordonné ne connaît pas son manager ».

Question 2 : Réalisez un diagramme de classes pour l'implémentation de l'application. On doit retrouver tous les éléments utiles à l'implémentation. N'oubliez pas de vous poser les bonnes questions ! (voir, entre autres, les TD et TP précédents).

Implémentation de l'application

En complément d'information pour la partie algorithmique de l'application, la méthode countEmployes() :

- doit retourner la valeur 1 pour un ouvrier
- et, pour un manager, elle doit retourner le nombre d'employés sous ses ordres + 1. Le « +1 » correspond au manager. Par exemple, avec la hiérarchie de la Figure 1, countEmployes() retourne 3 pour le manager Blake et retourne 9 pour le manager Ford.

Question 3 : Implémentez l'application. Réalisez une classe EntrepriseTestDrive qui contient une méthode main() et qui permet de « tester » l'implémentation de la hiérarchie.

Tests unitaires de l'application

Pour cette application, seuls quelques tests vont être réalisés. Ils seront tous développés dans une classe HierarchieTest dans un répertoire de type « Source Folder », nommé « test ».

L'objectif de cette classe HierarchieTest est de tester quelques opérations possibles sur la hiérarchie donnée dans la Figure 1. Dans cette classe, on doit retrouver comme attribut l'ensemble des employés de la hiérarchie :

```
public class HierarchieTest {

	private final Ouvrier ouvrier1 = new Ouvrier("7369", "Smith");
	private final Ouvrier ouvrier2 = new Ouvrier("7499", "Jones");
	private final Ouvrier ouvrier3 = new Ouvrier("7521", "Allen");
	private final Ouvrier ouvrier4 = new Ouvrier("7900", "James");
	private final Ouvrier ouvrier5 = new Ouvrier("7876", "Martin");

	private final Manager chef = new Manager("7902", "Ford");
	private final Manager manager1 = new Manager("7566", "Adams");
	private final Manager manager2 = new Manager("7788", "Scott");
	private final Manager manager3 = new Manager("7901", "Blake");

	...

}
```

Pour que tous les tests s'appliquent sur cette hiérarchie, vous devez créer deux méthodes createHierarchie() et cleanHierarchie() qui doivent respectivement construire et détruire (c'est-à-dire défaire les liens de subordinations mais pas détruire les employés) la hiérarchie. Ces deux méthodes devront être appelées respectivement avant et après chaque test.

Question 4 : Implémentez une méthode de test qui permet de tester l'ajout et la suppression d'un ouvrier à un manager (manager1) de la hiérarchie. Les éléments à vérifier pour ce test avant et après chaque opération sont :

- le nombre subordonnés du manager,
- si le nouvel ouvrier fait partie des subordonnés
- et le nombre d'employés renvoyés par la méthode countEmployes().

Question 5 : Comme pour la question précédente, implémentez une méthode de test qui permet de tester l'ajout et la suppression d'un manager à un manager (manager2) de la hiérarchie.

Analyse de la conception et de l'implémentation

Question 6 : Est-il possible, avec votre implémentation, qu'un ouvrier ait plus qu'un manager ? Ecrivez une méthode de test pour argumenter votre réponse. Est-ce juste par rapport aux spécifications ?

Question 7 : Est-ce qu'en transformant l'association unidirectionnelle entre les classes Manager et Employé en une association bidirectionnelle, cela pourrait résoudre le problème rencontré précédemment ? Quelles sont les modifications à apporter à votre code ? Complétez votre code pour implémenter cette nouvelle contrainte.

Pour finir

Question 8 : Ajoutez un logger qui informe (level INFO) lorsqu'un employé a été ajouté ou supprimé de la hiérarchie et qui alerte (level WARN) lorsque l'on ajoute à un manager un employé ayant déjà un manager.

Question 9 : Automatiser les tâches de compilation, packaging, d'exécution du projet et de documentation.