TD - Design Pattern Composite

Cet énoncé s'étend sur 4 pages.

1. Design Patterns : Elements of Reusable Object-Oriented Software

Un Design Pattern (DP) ou patron de conception est une description d'une solution éprouvée à un problème récurrent dans un contexte de génie logiciel. C'est un modèle de solution (template) décrivant comment résoudre un problème (architectural) de développement logiciel (souvent orienté-objet). Il décrit une structure d'objets et les relations entre eux. Chaque implémentation de patron ne peut pas être identique puisqu'appliquée dans des contextes différents.

Le livre *Design Patterns : Elements of Reusable Object-Oriented Software* (1995) de Erich Gamma, Richard Helm, Ralph Johnson et John Visslides a popularisé l'utilisation des design patterns.

Un pattern n'est ni une règle, ni une méthode. Les DP permettent d'élaborer des architectures de meilleure qualité que si on *réinventait en permanence ses propres solutions ad hoc*. Ils permettent donc de réduire la complexité des solutions retenues et facilitent la maintenance.

Il existe différents types de pattern :

— **Créateurs** : Abstract Factory, Builder, Prototype, Singleton
— **Structuraux** : Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy
— **Comportementaux** : Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, Visitor

2. Design Pattern Composite

C'est un patron structurel, décrivant une relation hiérarchique entre des composants terminaux et non terminaux. Le composant non terminal est un composite. Le composant terminal est une feuille. Tous deux dérivent d'un composant, qui fournit une interface de service à un client.

En plus de décrire une hiérarchie entre objets qui sont reliés par une relation structurelle de composition, ce patron permet au client de manipuler une interface uniforme, que l'instance soit une feuille ou un noeud non terminal.

Exercice 2.1. Système de fichiers

L'objet de cet exercice est de concevoir une application `FileSystem` permettant de créer une arborescence de fichiers, contenus dans des répertoires.

Dans cette application, il existe deux types de fichiers : les fichiers ordinaires et les répertoires. Un répertoire peut contenir des fichiers et d'autres répertoires. Chaque fichier possède un nom (non nécessairement unique). Chaque fichier a un propriétaire. Un propriétaire possède un nom, sunique.

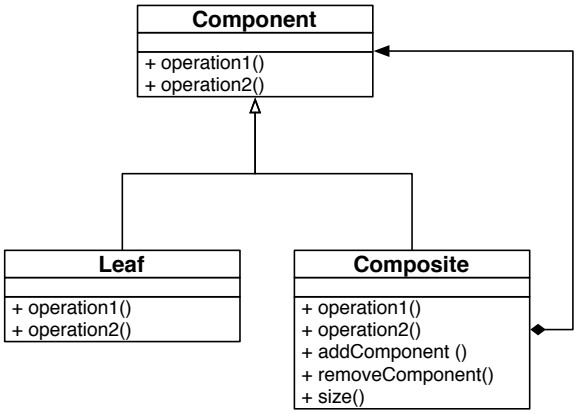

FIGURE 1 – Une description architecturale du DP Composite

Dans cette architecture, un composant offre les services suivants :

— getName : renvoie le nom du composant ;
— getContent : renvoie le contenu du composant ;
— getOwner : renvoie le propriétaire du composant ;
— getSize : renvoie la taille du composant ;
— appendContent : ajoute du contenu au composant ;
— setOwner : positionne le propriétaire du composant.

Un composite offre les services suivants :

— addChild : ajoute un nouveau composant à ce composite ;
— getChildren : renvoie les composants contenus dans ce composite ;
— removeChild : supprime un composant de ce composite ;
— removeChildren : supprime une liste de composants de ce composite ;

Une factory (FSFactory) propose un service de création de composant :

— createComponent : à partir du type de composant, son nom et son propriétaire renvoie un nouveau composant ayant ces propriétés précédemment citées.

Question 1. Proposez l'architecture d'une telle application en utilisant le patron composite. Entre quelles classes se trouve la relation de généralisation ? la relation de composition ? l'association simple ?

Question 2. Proposez, en Java, une implémentation pour cette application.

Le code suivants propose un aperçu du programme principal permettant d'instancier les objets de l'application FileSystem.

```
1  public static void main(String[] args) {
     FSFactory fsFactory = FSFactory.getInstance();
3    Owner owner1 = new Owner("Sarah");
     Composant f1 = fsFactory.createComposant(ComposantType.FICHIER, "Fichier 1", owner1);
5    Composant r1 = fsFactory.createComposant(ComposantType.REPERTOIRE, "Repertoire 1", owner1);
     ((Composite<Composant>)r1).addChild(f1);
7    f1.appendContent("Hello World\n");
     f1.appendContent("How are you doing?");
9    System.out.println(r1);
     System.out.println(f1);
11 }
```

Exercice 2.2. Système de forums

On souhaite écrire un ensemble de classes permettant de gérer les messages d'un forum :

— le forum est structuré en thèmes ; un thème a un sujet ;

— chaque thème possède un ou plusieurs salons de discussion ; un salon à un sujet ;

— chaque salon de discussion contient un ou plusieurs messages ; un message a un sujet et un contenu ;

On décide de définir **au moins** quatre classes modélisant le forum, les thèmes, les salons et les messages : `Forum`, `Theme`, `Salon` et `Message`. Les classes `Forum`, `Theme` et `Salon` possèdent une opération `add` (signature à préciser) qui permet d'ajouter, respectivement, un thème au forum, un salon de discussion à un thème et un message à un salon de discussion. Ces trois classes possèdent aussi les deux méthodes :

— `void remove(int i)` qui enlève le i^{e}élément ; s'il n'existe pas de i^{e}élément, la méthode ne fait rien ;

— `int size()` qui retourne le nombre d'éléments (le nombre de thèmes pour le forum, le nombre de salons de discussion pour les thèmes et le nombre de messages pour les salons de discussion).

Question 3. Proposez l'architecture d'une telle application en utilisant le patron composite.

Question 4. Proposez, en Java, une implémentation pour cette application.

Définir autant de classes que nécessaires de telle sorte que la série d'instructions suivantes :

```
1  Forum forum = new Forum("Programmation orientée objet");
   Theme theme1 = new Theme("Constructeurs");
3  Theme theme2 = new Theme("Design patterns");
   Salon salon1 = new Salon("Héritage");
5  Salon salon2 = new Salon("Composition");
   Salon salon3 = new Salon("Pattern Composite");
7  Message message1 = new Message("J'ai un problème", "voila, j'ai essayé ...");
   Message message2 = new Message("Au secours","ça marche pas ...");
9  Message message3 = new Message("Différence ?", "quelle est la
   différence entre l'aggregation et la composition ");
11 Message message4 = new Message("Question","quand utilise-t-on le pattern composite ?");
   forum.add(theme1);
13 forum.add(theme2);
   theme1.add(salon1);
15 theme1.add(salon2);
   theme2.add(salon3);
17 salon1.add(message1);
   salon1.add(message2);
19 salon2.add(message3);
   salon3.add(message4);
21 System.out.println(forum.toString());
```

affiche dans la console

```
Forum:Programmation orientée objet
Theme: Constructeurs
Salon: Héritage
Message: J'ai un problème
voila, j'ai essayé ...
Message: Au secours
```

```
ça marche pas ...
Salon: Composition
Message: Différence ?
quelle est la différence entre l'aggregation et le composition ?
Theme: Design patterns
Salon: Pattern Composite
Message: Question
quand utilise-t-on le pattern composite ?
```