

LOG2410 : Conception logicielle

Conception à base de patrons II

TP5

Présenté par :

Stéphanie Mansour (1935595)

Nanor Janjikian (1901777)

Groupe : B1

Remis le 26 mars 2019

Hiver 2019

**Patron Visiteur**

**Question 1 :**

L’intention et les avantages du patron visiteur :

L’intention du patron visiteur est d’effectuer des opérations distinctes sur les feuilles qui sont des opérations spécifiques du visiteur effectuées via « visit() ». Un visiteur permet de définir une nouvelle opération sans modifier les classes des objets sur lesquels l’opération indépendante va agir. Un patron visiteur est flexible puisque les visiteurs et la structure d’objets sont indépendants. Il a aussi une fonctionnalité localisée. En d’autres mots, tout le code associé au patron à une fonctionnalité qui se retrouve à un seul endroit bien identifié.

**Question 2 :**

Diagramme des classes du patron Visiteur :

Voir les fichiers *DiagrammeDeClasses\_FileChecksumCalculator.pdf* et *DiagrammeDeClasses\_FileStringFindReplace.pdf*.

**Question 3 :**

L’ajout d’une nouvelle sous-classe dérivée d’« AbsFileVisitor » ne changera pas significativement le code. L’ajout d’une sous-classe ne demande que de rajouter une nouvelle fonction « visit() » dans « AbsFileVisitor ». En effet, c’est un des avantages du patron visiteur puisqu’il peut accepter des visiteurs.

**Question 4 :**

Selon nous, il serait possible d’implémenter l’application des transformations aux fichiers audio comme un visiteur. Ces transformations peuvent être implémentées de la même façon que les sous-classes de « AbsFileVisitor » puisqu’elles agissent sur les fichiers audios dans lesquels les éléments sont conservés.

**Patron commande**

**Question 1 :**

1. L’intention et les avantages du patron commande :

L’intention du patron commande est d’encapsuler une requête dans un objet afin de permettre de supporter facilement plusieurs types de requêtes, de définir des queues de requêtes et de permettre des opérations « annuler ». Le patron commande découple l’objet qui invoque la requête de celui qui sait comment la satisfaire. Ces commandes sont encapsulées dans des objets qui peuvent être manipulées comme tout objet. Alors, l’utilisation d’objets amène également plus de flexibilité. Il est facile de créer de nouvelles commandes. Les commandes peuvent être assemblées en des commandes composites si nécessaires. Le patron Command peut être combiné au patron composite pour représenter des commandes qui sont un ensemble d’autres commandes.

1. Diagramme des classes du patron Commande :

Voir le fichier *DiagrammeDeClasses\_Command.pdf.*

**Question 2 :**

1. La classe *CommandExecutor* participe aux deux autres patrons : Template Method et Strategy.
2. Les éléments de la classe :

* Strategy : l’utilisation facilite l’implémentation des opérations, plutôt que d’écrire à chaque fois l’opération que l’on veut effectuer. Ce patron permet d’implémenter un algorithme de façon simple, et de créer des sous-classes pour chacune des variantes de cette algorithme. Le code commun est alors placé dans une classe abstraite pour réduire la quantité de code redondante.
* Template method : Le patron Template Method suggère la définition d’une interface, ce qui permet à ses sous-classes d’avoir un bon point de départ pour leur construction plus élaborée. Il donne avantage principalement à l’implantation d’un algorithme.

1. Utilisation des patrons ??

**Question 3 :**

Selon nous, d’autres classes ne doivent pas être modifiées pour ajouter les nouvelles commandes puisqu’avec le patron commande l’ajout de nouvelles commandes peut s’effectuer par l’ajout d’une nouvelle sous-classe de *AbsCommand*.