LOG2410-TP5

Hiver 2019

Paul Clas 1846912

Mazigh Ouanes 1721035

TP5

Conception à base de patrons 2

1- **Objectifs**
2- **Patron Visiteur**

1. **Identifiez L'intention et les avantages du patron Visiteur.**

L'intention du patron Visiteur et de séparer un algorithme d'une structure de données. Le patron Visiteur permet d'ajouter des fonctionnalités sur des éléments d'une structure d'objets sans modifier la structure des classes. Ce patron permet beaucoup de flexibilité en donnant le pouvoir d'ajouter et d'enlever des méthodes de traitement avec beaucoup de simplicité. Les opérations deviennent indépendantes d'une classe spécifique.

2. **Tracer un diagramme de classes avec Enterprise Architect pour chacune des deux instances du patron Visiteur (calcul du checksum et find & replace), et ajouter des notes en UML pour indiquer les rôles, et exportez le tout en pdf.**

VOIR DOCUMENTS ANNEXÉS À LA REMISE.

Les diagrammes se décomposent en deux parties. D'une première part, on retrouve les visiteurs, et de l'autre on retrouve les éléments sur lesquels les visiteurs agissent. Les visiteurs possèdent une classe abstraite commune qui est *AbsFileVisitor*. S'en suit des deux sous-classes *FileStringReplace* et *FileChecksumCalculator*. Ces deux classes permettent la visitent des éléments, qui eux accepteront la visite. Les éléments possèdent une classe abstraite commune qui est *AbsAudioFile*. S'en suit des deux sous-classes *AudioFile* et *MemAudioFile*.

3. **Si en cours de conception vous constatiez que vous voudriez ajouter une nouvelle sous-classe dérivée de AbsAudioFile, établissez la liste de toutes les classes qui doivent être modifiées.**

Il sera très simple d'ajouter de nouvelles sous-classes dérivées de *AbsAudioFile*, sans pour autant devoir effectuer de nombreux changements aux classes qui y sont reliées. En effet, c'est l'avantage que permet le patron Visiteur. Puisque que chaque sous-classe de *AbsAudioFile* représente un élément qui peut accepter des visiteurs, l'ajout d'une sous-classe ne demandera qu'à ajouter une fonction de visite dans chacun des visiteurs.

4. **Selon vous, l'application des transformations aux fichiers audio pourrait-elle être implémenté comme un visiteur ? Si oui, discuter des avantages et inconvénients d'utiliser le patron visiteur pour cette fonction et sinon expliquez pourquoi le patron n'est pas applicable.**

Oui, l'application des transformations aux fichiers audio pourrait être implémenté comme un visiteur car la transformation doit pouvoir agir sur les chunks conservés. Ainsi, les transformations seraient implémentées dans des Visiteurs, ce qui les séparerait de la structure des objets, augmentant la flexibilité du système : ajouter des transformations ne changerait pas la hiérarchie des classes de structure. Cependant cette flexibilité viendrait avec un coût car il y aura une duplication du code pour chaque implémentation des transformations. Les transformations pourraient fonctionner de la même façon que les sous-classes *FileStringReplace* et *FileChecksumCalculator*. Puisque les transformations agissent sur les fichiers audios, on conserve les éléments, et on met visiteur les transformations, et on obtient un diagramme semblable à celui de la question 2.

3- Patron Commande

1. **Identifiez les points suivants :**
   i. **L'intention et les avantages du patron Commande.**

L'utilisation du patron visiteur permet d'ajouter, de supprimer et, ainsi, de gérer aisément de nouvelles opérations sur une classe sans avoir à modifier celle-ci. Les opérations sont indépendantes de cette classe.

   ii. **La structure des classes réelles qui participent au patron Commande ainsi que leurs rôles (faite un diagramme de classes avec Enterprise Architect, ajouter des notes en UML pour indiquer les rôles, et exportez le tout en pdf).**

VOIR DOCUMENTS ANNEXÉS À LA REMISE.

2. **Observez attentivement la classe CommandExecutor qui permet de gérer la relation entre les commandes et les différents fichiers audio. En plus de participer au patron Commande, cette classe participe à deux autres patrons de conception vu en cours.**

   i. **Quel sont les noms et les intentions de ces patrons de conception ?**

La classe *CommandExecutor* participe également aux patrons *Template Method* et *Strategy*.

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.

Strategy : On utilise une classe pour faciliter l'implémentation d'opérations, plutôt que d'écrire au long à chaque fois l'opération que l'on veut effectuer. Cela 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.

ii. Quels sont les éléments de la classe ExecuteurCommandes qui sont caractéristiques de ces patrons de conception ?

Template Method : Offre des interfaces pour les différentes classes, on peut associer cela aux classes abstraites telles que AbsCommand et AbsAudioFile.

Strategy : La classe abstraite AbsCommand est la base des commandes ayant chacune leur variante.

iii. Pourquoi avoir utilisé ici ces patrons de conception ?

Le patron de conception permet la génération de façon simple une série de commande que l'on pourra utiliser pour du traitement sur une classe. L'utilisation du patron permettra de grouper et exécuter toutes ces commandes au même endroit. Pour ajouter une commande durant le développement, il suffit d'ajouter une sous-classe à la classe abstraite des commandes.

3. Pour compléter la fonctionnalité de PolyVersion, il faudrait ajouter de nouvelles sous-classes de la classe AbsCommand. Selon vous, est-ce que d'autres classes doivent être modifiées pour ajouter les nouvelles commandes? Justifiez votre réponse.

Non, il n'y a pas de besoin de devoir modifier une classe pour ajouter une nouvelle commande autre que la nouvelle désirée. C'est l'intention du patron Commande. La simplicité et facilité de l'ajout est primordiale pour ne pas à repasser tout le code pour y ajouter des nouvelles méthodes. On ajoute une nouvelle commande en créant une nouvelle sous-classe AbsCommand.