Polytechnique Montréal

Laboratoire LOG 2410

Section : 02 (B2)

TP5 **–** Conception à base de patrons II

Par

Zhornytskiy, Roman

Ciliento, Kevin

Remis le 16 avril 2019

1. **- Patron Visiteur**

**2.1 – Identifiez L’intention et les avantages du patron Visiteur**

L’intention du patron visiteur est de « représenter une opération qui doit être appliquée sur les éléments d’une structure d’objets. Un Visitor permet de définir une nouvelle opération sans modification aux classes des objets sur lesquels l’opération va agir »[[1]](#footnote-1).

**Avantages**

1. Il y a plus de flexibilité, car les visiteurs et la structure d’objets sont indépendants.
2. La fonctionnalité est localisée, car « tout le code associé à une fonctionnalité se retrouve à un seul endroit bien identifié ».

**2.3 –** **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.**

En cours de conception, nous n’avons pas voulu ajouter de nouvelle sous-classe dérivée de AbsAudioFile.

**2.4 – 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.**

L’application des transformations aux fichiers audio peut être implémenté comme un visiteur, car on pourrait avoir une classe « visiteurTransformAbs » et les classes « InvertTransform », « RepeatTransform » et « CompositeTransform » dériverons de cette classe.

**Avantage :**

Il est plus facile d’ajouter de nouvelles transformations.

**Désavantages :**

Les opérations « transforms » ne sont pas distinctes. De plus, il n’y qu’une seule classe à visiter. Par les deux faits précèdent, l’applicabilité du patron n’est optimale.

1. **– Patron Commande**

**3.1 – Identifiez les points suivants :**

**3.1 a) – L’intention et les avantages du patron Commande**

L’intention du patron commande est « d’encapsuler une requête dans un objet de façon à permettre de supporter facilement plusieurs types de requêtes, de définir des queues de requêtes et de permettre des opérations « annuler » »[[2]](#footnote-2).

**Avantages**

1. Il est facile de créer de nouvelles commandes.
2. « Les commandes peuvent être assemblées en des commandes composites si nécessaires »[[3]](#footnote-3).
3. Les commandes sont encapsulées dans des objets et l’utilisation d’objets amène plus de flexibilité.

**3.2 – 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**

**3.2 a) – Quel sont les noms et les intentions de ces patrons de conception ?**

**Patron 1 : Singleton**

Son intention est de s’assurer qu’il ne soit possible de créer qu’une seule instance d’une classe, et fournir un point d’accès global à cette instance.

**Patron 2 : Composite**

Traiter les objets individuels et les objets multiples, composés récursivement, de façon uniforme.

**3.2 b) – Quels sont les éléments de la classe ExecuteurCommandes qui sont caractéristiques de ces patrons de conception ?**

Pour le singleton, la ligne static CommandExecutor& getInstance(void); et le constructeur en private représente bien le fait qu’il n’existera qu’une seule instance de cette classe

Pour le composite, la ligne void executeAllCommands(AbsAudioFile& f); permet de traiter tous les objets de commande de façon uniforme.

**3.2 c) – Pourquoi avoir utilisé ici ces patrons de conception ?**

Il ne serait pas logique dans notre contexte de mettre plusieurs exécuteur de commande puisque l’on veut seulement mettre toutes les commandes dans un seul conteneur d’un seul exécuteur de commande. Également, on aimerait pouvoir traiter toutes ces commandes simultanément, et ce, de façon uniforme.

**3.3 – 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, car s’il s’agit d’une transformation, il ne sera pas nécessaire de modifier les classes déjà existantes puisque l’on peut traiter les transformations par le composite. Également, s’il s’agit d’une nouvelle fonctionnalité sur les fichiers, il suffit de l’implémenter avec un patron visiteur. Donc les classes déjà existantes ne seraient pas changées.

Nota bene :

Toutes les diagrammes sont dans le même fichier dénommer : Diagrammes.pdf

1. Notes de cours [↑](#footnote-ref-1)
2. Notes de cours [↑](#footnote-ref-2)
3. Idem [↑](#footnote-ref-3)