Polytechnique Montréal

Laboratoire LOG 2410

Section : 02 (B2)

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

Par

Zhornytskiy, Roman

Ciliento, Kevin

Date de remise : 16 avril 2019

**2 - Patron Visiteur**

1. 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 ».

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 ».
3. En cours de conception, nous n’avons pas voulu ajouter de nouvelle sous-classe dérivée de AbsAudioFile.
4. 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.

Avantages : 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.

**3 – Patron Commande**

1. a)

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 » ».

Avantages

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

2.

a)

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

b)

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.

c)

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.

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.