Laboratoire LOG 2410

TP4 **–** Patrons de conception I

Par

Ciliento, Kevin

Zhornytskiy, Roman

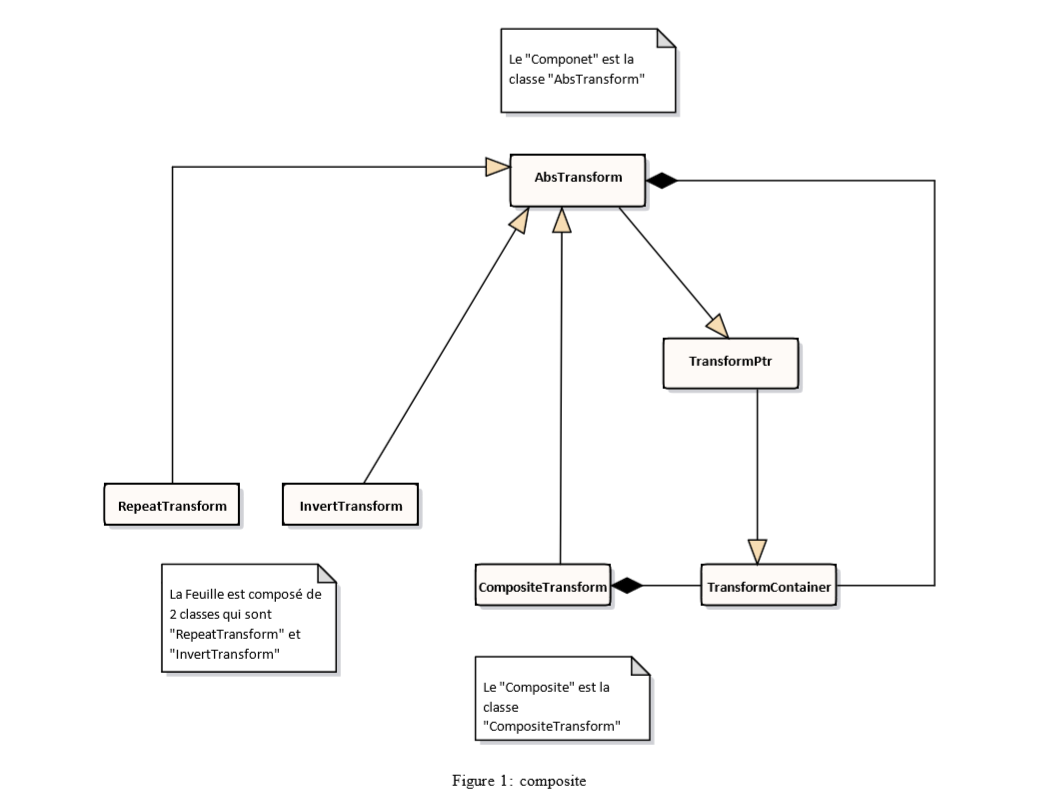
Section : 02 (B2)

Date de remise : 26 mars 2019

Polytechnique Montréal

Patron Composite

1) a) Généralement l’intention du patron composite est de « traiter les objets individuels et les objets multiples, composés récursivement, de façon uniforme ». Alors, dans ce cas l’intention de ce patron est de traiter les fichiers initiaux sans les transformations et les fichiers transformés à l’aide des différentes manipulations (RepeatTransform, InvertTransform et CompositeTransform) de façon uniforme.

b)

2) La classe AbsTransform est une classe abstraite et sa responsabilité est de fournir une « interface » commune à toutes les transformations possibles. En d’autres mots, elle fournit les méthodes nécessaires pour faire les autres transformations.

La classe AbsAudioFile est une classe abstraite et sa responsabilité est de créer l’interface de la classe AudioFile………à voir les notes de cours

3) La classe CompositeTransform est responsable de la création de l’arbre des composantes, car c’est cette classe

Patron Proxy

1) a) Généralement l’intention du patron Proxy est de « fournir un remplaçant ou une doublure pour un autre objet afin de contrôler l’accès à ce dernier ». Alors, dans ce cas on offre une doublure à la classe AudioFile avec la classe MemAudioFile.

b)

Conteneurs et Patron Iterator

2) a) Généralement l’intention du patron Iterator est de « fournir une méthode d’accès séquentielle aux éléments d’un objet agrégat (liste, vecteur, etc.) sans exposer sa structure interne ». Alors, dans ce cas la classe TransformContainer fournit une méthode d’accès aux éléments sans exposer le fait que la structure interne est un vecteur avec des pointeurs uniques.

b) La classe de conteneur de la STL utilisée pour stocker les enfants dans la classe Composite est la classe « TransformContainer ».

Les classes des Iterators utilisés dans la conception sont « TransformIterator », « TransformBaseIterator », « TransformIterator\_const » et « TransformBaseIterator\_const » pour ce qui concerne la classe CompositeTransform.

3) Le rôle de cet attribut est de permettre aux fonctions non virtuelles pures (comme begin et end) de retourner des itérateurs valides.

L’attribut est déclaré comme privé, car on ne veut pas que les enfants qui hérite de cette classe aille accès à ce conteneur et il est statique, car on veut que cet attribut soit pareil pour toutes ces classes.

4) Si on change la classe TransformContainer par une autre classe avec un nom différent comme « Container », alors, il faudrait changer aussi toutes les instances de la classe « TransformContainer » par la classe « Container » dans le fichier « TransformContainer.h ». De plus, il faut aussi changer le nom du conteneur de la classe « AbsTransform ». Finalement, il faut aussi modifier l’instance de la classe « TransformContainer » par « Container » dans le fichier « AbsTransform.cpp ».

À cause du fait qu’il y a beaucoup de changement à effectuer après le changement du nom du conteneur, alors cela veut dire que la conception proposée ne respecte pas le principe d’encapsulation, car s’il le respectait on aurait un seul changement à effectuer (le nom de la classe).

5) Un avantage de ces surcharges est que ces opérateurs sont très intuitifs aux programmeurs, alors les gens qui vont utiliser ces classes vont comprendre la signification de ces opérateurs très facilement, car ces opérateurs sont les opérateurs de base sur les vrais conteneurs.

L’inconvénient est le fait qu’on doit implémenter manuellement ces opérateurs pour rendre l’utilisation de cette classe facile à utiliser.