LOG2410: Conception à base de patrons I

Travail pratique # 4

Hiver 2019

Équipier 1: Mariam Sarwat (1901777)
Équipier 2 : Yasmina Abou-Nakkoul (1897266)

Date de remise :
26 mars 2019

2 - Patron Composite

1) Identifiez les points suivants :

a) L'intention du patron Composite.

Un patron Composite sert à uniformiser le traitement d'objets seuls et de composition d'objets (hiérarchie représentée par une structure en arbre).

Le concept de composition récursive est également inclus dans l'intention du patron composite. En effet, si nous imaginons le composite comme étant un dossier, celui-ci peut contenir soit des fichiers, soit d'autres dossiers. Ceci représente une récursivité qui dure tant et aussi longtemps que le composite contient d'autres composites.

b)

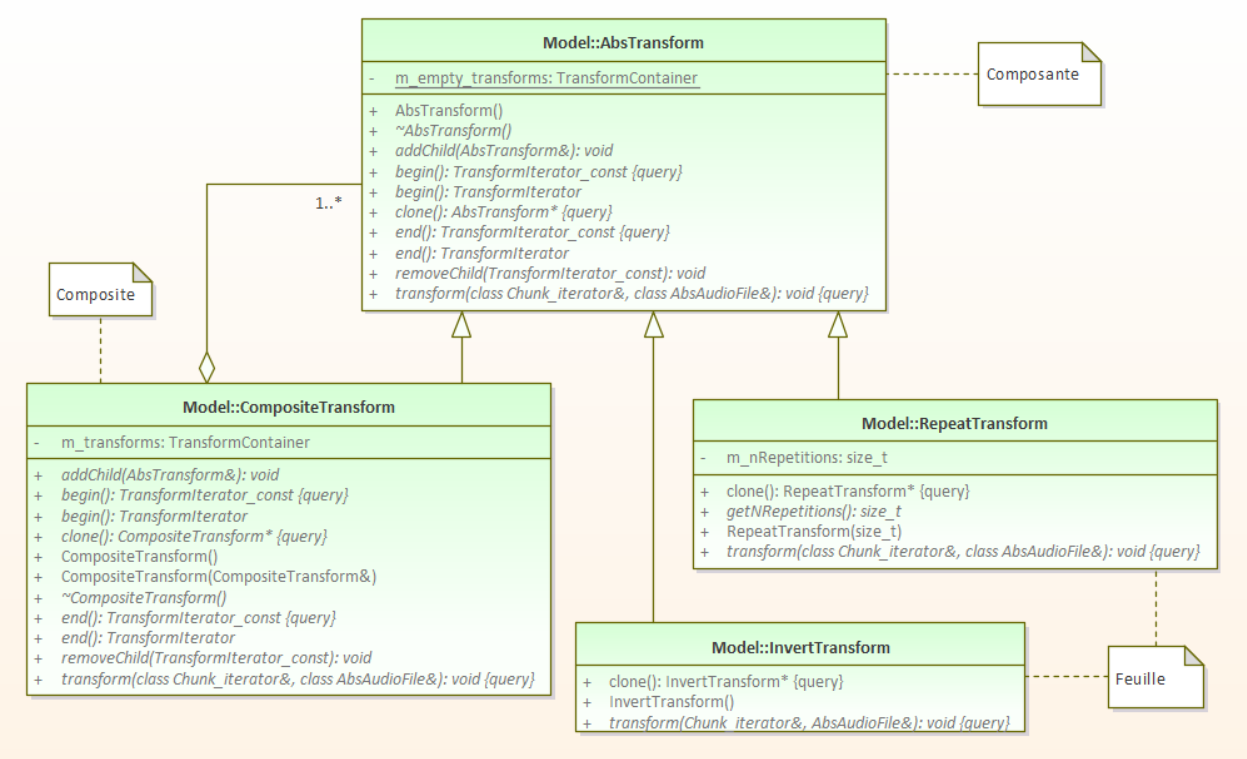

Figure 1: Diagramme de classe pour l'instance du patron composite

Ici, on peut voir le principe de récursivité grâce à la relation de composition entre **AbsTransform** et **CompositeTransform**. Cette dernière classe, peut autant être une transformation d'inversion qu'une de répétition.

- **AbsTransform** : Classe composante (abstraite), soit l'interface, qui gère les différentes transformations possibles à effectuer. Cette classe posède des méthodes virtuelles pures.

- **RepeatTransform** : Classe qui permet d'effectuer n-répétitions d'un Chunk et les ajouter à un fichier de sortie.
- **InvertTransform** : Classe qui permet d'inverser un Chunk et qui l'ajoute à un fichier de sortie.
- **CompositeTransform** : Classe composite qui est formée de RepeatTransform et de InvertTransform.

2) Identifiez toutes les abstractions présentent dans la conception du TP4, et pour chacune, identifiez les responsabilités spécifiques qui lui ont été assignées.

D'après les diapositives du cours, la forme la plus tangible des abstractions d'un design sont les classes abstraites qui le représentent. Il existe également une règle qui stipule que dans une hiérarchie de classes, seules les classes terminales (qui représenteraient les feuilles d'un arbre de hiérarchie) devraient être des classes concrètes.

Avec cette information et nos connaissances acquises en programmation orientée objet, nous savons qu'une classe abstraite contient au moins une fonction virtuelle pure, soit une fonction virtuelle initialisé à 0 dans le .h, devant être redéfinie dans une classe enfant. Le TP4 a donc deux abstractions, soit **AbsAudioFile** et **AbsTransform**. Ces abstractions ont comme responsabilité respective, la lecture et l'écriture des fichiers binaires grâce auxquels les flux audios sont émulés, et les transformations (duplication, inversion et transformations composées) sur les fichiers lus.

3) Dans l'implémentation actuelle du système PolyVersion, quel objet ou classe est responsable de la création de l'arbre des composantes.

D'après le code et le diagramme de classe présenté ci-dessus, c'est la classe **CompositeTransform** qui est responsable de la création de l'arbre des composantes, car elle contient les méthodes *addChild()* et *removeChild()* qui servent à ajouter et enlever soit des feuilles, soit des composites de transformation des fichiers.

3 – Patron Proxy

1) Identifiez les points suivants :

a) L'intention du patron Proxy.

Un patron Proxy sert de substitut à un objet, fournissant une méthode d'accès à celui-ci. Il ajoute un niveau d'indirection à l'objet qu'on veut partager, afin de rendre son accès plus contrôlé.

Un exemple de proxy serait d'utiliser un chèque à la place d'argent liquide lorsqu'on veut donner de l'argent à quelqu'un.

b)

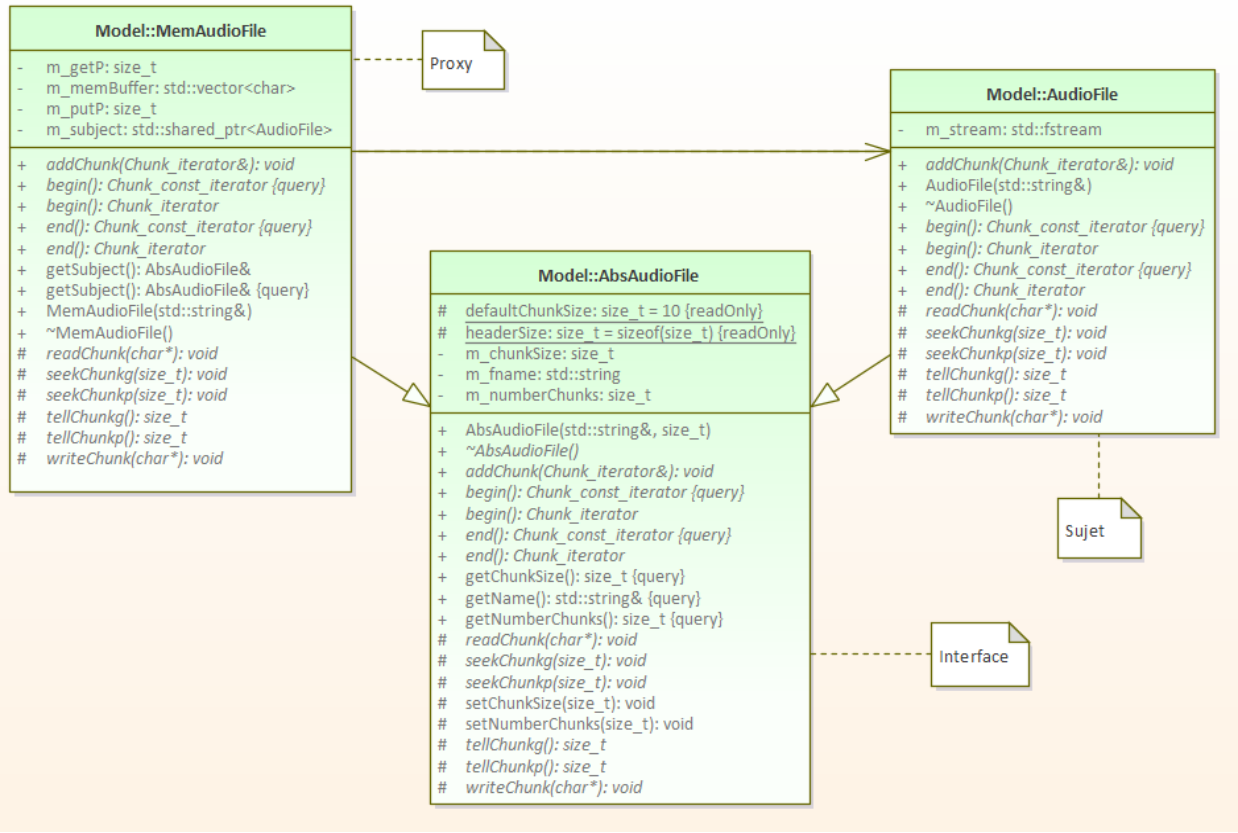

Figure 2 : Diagramme de classe de l'instance du patron de proxy

- **AbsAudioFile** : Interface qui gère les différents types de fichiers possibles (sujet abstrait).
- **AudioFile** : Cette classe s'agit du sujet du proxy (représente l'argent liquide dans notre exemple).

- **MemAudioFile** : cette classe s'agit du proxy. Elle accède AudioFile à l'aide de l'attribut *m_sujets* (représente le chèque dans notre exemple).

4 – Conteneurs et Patron Iterator

2) Identifiez les points suivants :

a) L'intention du patron Iterator.

Le patron Iterator offre un moyen d'accéder séquentiellement aux éléments d'un objet agrégat sans avoir à exposer sa structure interne.

b) La classe de conteneur de la STL utilisée pour stocker les enfants dans la classe Composite et les classes des Iterators utilisés dans la conception qui vous a été fournie.

La classe de conteneur de la STL utilisée pour stocker les enfants dans la classe Composite est la classe **TransformContainer**.

Les classes des Iterator utilisés dans la conception fournie sont **TransformIterator** et **TransformIterator_const**, les deux héritant de leur itérateur de base respective (**TransformBaseIterator** et **TransformBaseIterator_const**).

3) Expliquez le rôle de l'attribut statique m_empty_transforms défini dans la classe AbsTransform. Expliquez pourquoi, selon vous, cet attribut est déclaré comme un attribut statique et privé.

L'attribut statique ***m_empty_transforms*** a pour rôle d'exister afin que les classes héritant de la classe **AbsTransform** puissent appliquer sur l'attribut (hérité) les méthodes (héritées) de la classe. En effet, cet attribut permet de signaler la présence d'une transformation sur un Chunk.

Comme il s'agit d'une classe abstraite, il est impossible d'y instancier des objets, donc l'attribut est déclaré comme statique afin qu'il soit inchangé dans toutes les instances des classes enfants de **AbsTransform**. Celui-ci est privé car seules les classes héritant de **AbsTransform** l'utiliseront, donc il respecte le principe d'encapsulation.

4) Quelles seraient les conséquences sur l'ensemble du code si vous décidiez de changer la classe de conteneur utilisée pour stocker les enfants dans la classe Composite? À votre avis, la conception proposée dans le TP4 respecte-t-elle le principe d'encapsulation ?

Il n'y aurait pas vraiment de conséquences sur l'ensemble du code si nous changions la classe de conteneur utilisée pour stocker les enfants dans la classe Composite, puisque le changement sera contenu dans les différentes classes Iterator du code. Tout ce qui sera à modifier se trouve dans un seul fichier (le conteneur).

Ceci est une preuve que la conception proposée dans le TP4 respecte le principe d'encapsulation, puisque le changement de choix du conteneur n'expose pas ses éléments, et son processus d'accès reste inchangé.

5) Les classes dérivées TransformIterator et TransformIterator_const surchargent les opérateur « * » et « -> ». Cette décision de conception a des avantages et des inconvénients. Identifiez un avantage et un inconvénient de cette décision.

L'avantage est que ça simplifie l'accès à l'objet sur lequel pointe l'itérateur (système de double pointeur).

Tandis que le désavantage est définitivement l'impossibilité d'accès au pointeur de l'itérateur (soit le premier pointeur). On accède soit à l'itérateur (équivalent de * (déréférencement)), ou à l'objet pointé par l'itérateur (équivalent de ** (double déréférencement)).