­

Travail pratique # 4 :

LOG2410

Conception à base de patrons (1)

Département de génie informatique et génie logiciel

Fabrice ***NDUI***(1914377)

Serge **GNAVO** (1869985)

**Trimestre :** Hiver 2019

Date de remise (2019-03-25 00h)

# Patron Composite

## L’intention du patron Composite

L’intention du patron composite est de traiter les objets individuels et les l’objets multiples, composés récursivement, de façon uniforme.

La figure 2-1 montre le diagramme de classe représentant un patron Composite.



Figure 2‑1 UML Patron composite

## Responsabilités des Abstractions

Nous avons identifié deux abstractions dans le code, soit :

* **AbsTransform** : Applique une transformation ou une composition de transformations au chunks d’un objet de type AbsAudioFile.
* **AbsAudioFile** : Permet d’abstraire la composante AudioFile qui pourrait être l’audio dans la mémoire ou celle tampon, memAudioFile qu’on va utiliser comme proxy.

## Objet responsable de la création de l’arbre des composantes

Il s’agit de la classe CompositeTransform.

# Patron Proxy

## Intention du patron Proxy

Il s’agit ici d’un proxy virtuel car il permet des optimisations en accédant uniquement au fichier réel en mémoire en entrant et sortant de l’application, le reste du temps, le client utilise memAudioFile qui est un objet virtuel copié du vrai fichier en mémoire, et une fois terminé, les changements seront recopiés dans ce meme fichier.

## Graph du patron Proxy

La figure 3-1 montre l’uml du diagramme de classe qui forment le patron proxy.



Figure 3‑1 UML diagramme proxy

# Conteneurs et Patron Iterator

## Intention du patron Iterator

a)

L’intention de ce patron est de fournir une méthode d’accès séquentielle aux éléments d’un objet agrégat sans exposer sa structure interne.

b)

La classe de conteneur de la STL utilisée est vector<TransformPtr> et les iterator utilisés sont iterator et const\_iterator.

## Rôle de l’attribut m\_empty\_transforms

L’attribut m\_empty\_transforms a pour rôle d’indiquer la présence de chunk ayant subi une transformation.

Il est déclaré privé afin de respecter le principe d’encapsulation et il est déclaré statique afin d’exister indépendamment de l’instanciation de la classe AbsTransform.

## Principe d’encapsulation

En remplaçant notre conteneur vecteur par un conteneur List, il n’y a eu aucun changement à faire dans le code, à part cette ligne de code :

using TransformContainer = std::list<TransformPtr>;

Tout le reste du code compile et les tests passent tous, donc pour le client il n’y a eu aucun changement. Ce qui respect bien le principe d’encapsulation.

## Avantage et inconvénient surcharge operateur

**Avantages** :

Simplifie certaines opérations sur les itérateurs .

**Désavantage** :

L’avantage de la surcharge, peut devenir un désavantage pour une personne externe au code qui doit le lire.

On ne s’attend pas toujours au comportement qu’on obtient avec l’operateur qu’on utilise, ce qui peut porter à confusion pour une personne externe.