LOG2410

CONCEPTION LOGICIELLE

TP5

Trimestre : HIVER 2019

Antoine Martin (1929587)

Jaafar Kaoussarani (1932805)

Présenté à : François Guibault

École Polytechnique de Montréal
Le mardi 16 avril 2019

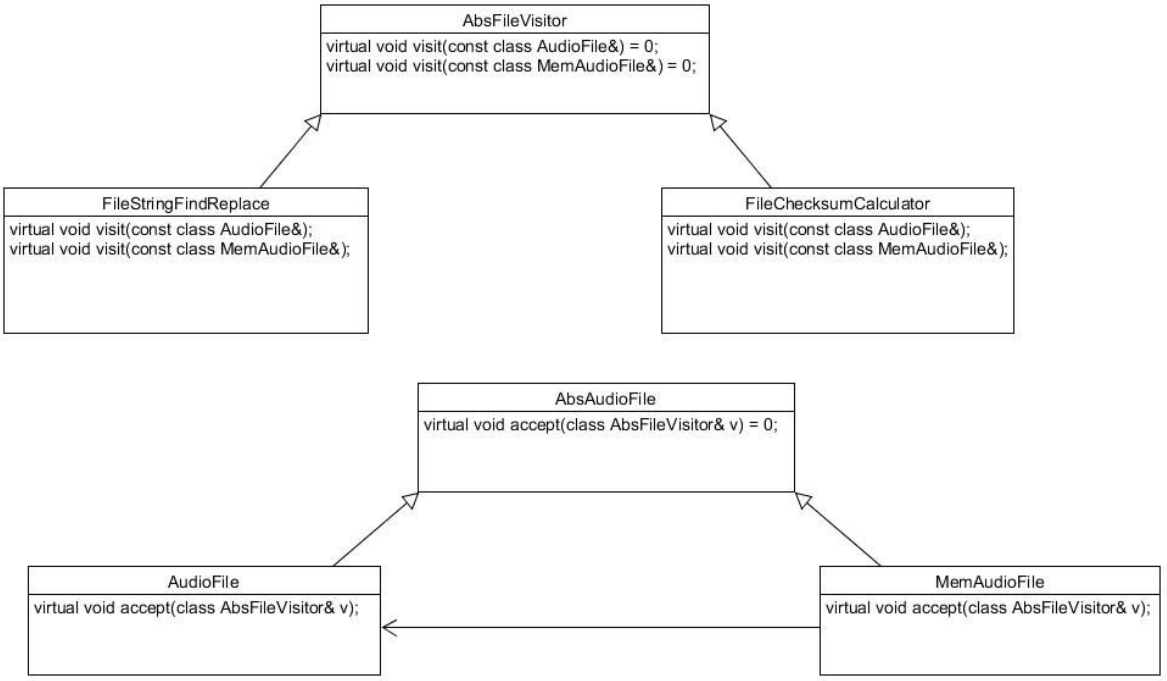
Patron Visiteur

1. Identifiez L'intention et les avantages du patron Visiteur.

Le patron Visiteur a pour objectif d'implémenter une méthode modifiant un objet de la classe X sans toutefois modifier la structure de celle-ci. L'opération que l'on veut effectuer avec notre méthode est représenté avec par une nouvelle classe.

Ce patron permet de rester flexible dans l'implémentation de nouvelles méthodes sans surcharger nos classes déjà existantes. De plus, tout le code participant à la nouvelle méthode se retrouve au même endroit.

2. Tracer un diagramme de classes avec Enterprise Architect pour chacune des deux instances du patron Visiteur (calcul du checksum et find & replace), et ajouter des notes en UML pour indiquer les rôles, et exportez le tout en pdf.



Ici les visiteurs sont *FileChecksumCalculator* et *FileStringFindReplace*. Ils représentent une opération que l'on veut effectuer sur un *AbsAudioFile*. Dans les deux classes concrètes *AudioFile* et *MemAudioFile*, on voit qu'une méthode *accep()* est implémenté. Elle appel la méthode *visit()* des visiteurs.

Les opérations *FileChecksumCalculator* et *FileStringFindReplace* possède 4 fonctions chacune qui modifient un *AbsAudioFile* et particulièrement les chunk qui le compose. Toutes ces opérations ne sont inclue dans un *AbsAudioFile* et cela allège le code grandement. De plus, si on veut vérifier une modalité de l'opération, on sait exactement où regarder.

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.

On devrait ajouter une méthode *visit()* pour tous les visiteurs concernés par la nouvelle sous-classe de *AbsAudioFile*. Cette méthode devra être adapté à la nouvelle structure de la sous-classe

.

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.

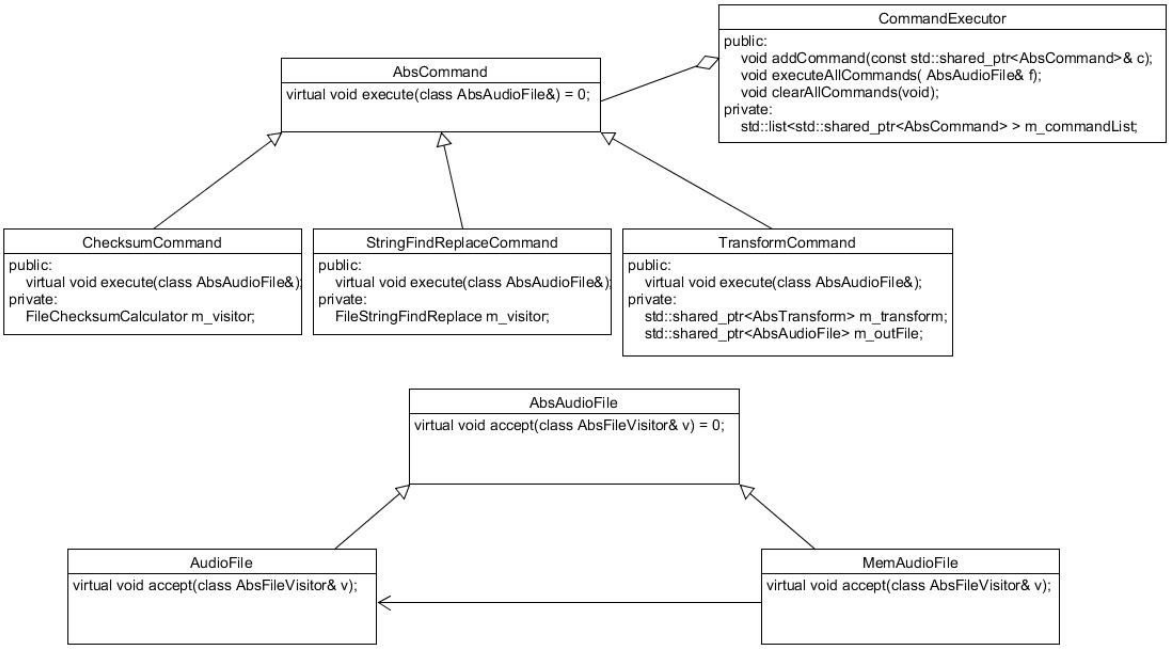
Non, car les transformations nécessitent un fichier audio de sortie ainsi qu'un itérateur. Dans le patron Visiteur, la méthode *accept()* est implémenté en utilisant le *this*, c'est-à-dire l'objet *AbsAudioFIle* lui-même. Il n'existe présentement pas d'objet que l'on pourrait passer en paramètre à *visit()* dans PolyVersion qui possède l'information d'un itérateur et d'un fichier de sortie.

Patron Command

1. a. L'intention et les avantages du patron Command.

Le patron Command est utilisé lorsqu'on veut conserver plusieurs opérations à un endroit spécifique(un objet). Ainsi on peut les appeler comme une liste et revenir en arrière dans le besoin.

b.La structure des classes réelles qui participent au patron Commande ainsi que leurs rôles (faite un diagramme de classes avec Enterprise Architect, ajouter des notes en UML pour indiquer les rôles, et exportez le tout en pdf).



Les commandes dans PolyVersion sont *Checksum, StringFindReplace* et *Transform.* Ils dérivent de *AbsCommand*. L'exécuteur des commandes est *CommandExecutor*. Il contient

toutes les commandes et c'est avec cette classe que le client interagit pour exécuter les commandes.

Les receveurs sont les *AbsAudioFile*. Lorsque l'exécuteur appel les opérations à effectuer, ce sont les receveurs qui sont appelés.

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.

   a. Quel sont les noms et les intentions de ces patrons de conception ?

CommandExecutor utilise le patron Composite. Ce patron permet d'interagir avec plusieurs classes sans devoir implémenter des méthodes différentes pour chacunes d'entres elles.

Il participe aussi au patron Visiteur vu plus haut.

   b. Quels sont les éléments de la classe ExecuteurCommandes qui sont caractéristiques de ces patrons de conception ?

Tous les patrons utilisent aussi les classes dérivées de *AbsAudioFile.*

Visiteur : *ChecksumCommand* et *StringFindReplaceCommand*

Composite : *InverseTransform, RepeatTransform, CompositeTransform* et *TransformCommand*

   c. Pourquoi avoir utilisé ici ces patrons de conception ?

Visiteur: *expliqué en partie 1*

Composite: Ce patron est utilisé pour traiter les opérations de *InverseTransform* et *RepeatTransform* de manière uniforme en interagissant avec la classe *CompositeTransform*.

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.

Cela dépend. Il n'est pas nécessaire d'implémenter de nouvelle classe si on possède déjà une opération sous la forme d'une classe. On peut simplement implémenter des nouvelles sous-classes de *AbsCommand* avec les attributs nécessaire pour effectuer les opérations relié à la commande. Sinon, on devrait implémenter l'opération sous forme de classe, afin de la stocker comme attribut dans la sous-classe dérivé de *AbsCommand*.

Dans notre situation les commandes possédait déjà un objet et une implémentation donc les commandes étaient une seule ligne..