LOG2410 – TP4 Hiver 2019

Conception à base de patrons I

1 - Objectifs

Ce laboratoire permettra aux étudiants de se familiariser avec l'implémentation des patrons de conception « Composite », « Proxy » et « Iterator ». Cette implémentation est effectuée à l'aide du logiciel Visual Studio et le langage C++ sera utilisé tout au long du processus de développement. Un cadriciel est fourni, qui doit être complété afin d'obtenir le résultat final souhaité.

2 - Patron Composite (50 points)

Afin de simuler le comportement de l'application *PolyVersion,* qui doit normalement permettre de manipuler des flux audio-numériques, dans le cadre du TP, les flux sont émulés grâce à des fichiers en format binaire. Ces fichiers peuvent être transformés de différentes façons, et normalement, ces manipulations devraient permettre de traduire un flux binaire d'une langue vers une autre. Dans le cadre du TP, des manipulations simplifiées seront appliquées aux données contenues dans les fichiers. Parmi les manipulations possibles, le logiciel permet d'appliquer certaines transformations aux sons enregistrés dans un fichier, afin de produire un nouveau fichier contenant le résultat de l'application des transformations à un fichier initial. Les transformations comprennent des transformations simples, telles que « dupliquer » (*RepeatTransform*) ou « inverser » (*InvertTransform*) des segments audios, et des transformations composées (*CompositeTransform*). Chaque transformation est appliquée à un segment audio appelé « Chunk » dans le code source du TP. Dans un fichier, tous les « Chunk » sont de taille identique, et cette taille est stockée comme premier élément dans le fichier binaire. Le résultat de la transformation produit un ou plusieurs nouveaux Chunk qui sont ensuite ajoutés à un fichier de sortie, passé en paramètre lors de l'application de la transformation.

Une classe de test est fournie (*Test_TP4*), qui génère un fichier de test initial appelé « audiof1.bin ». Les transformations sont testées séparément, puis ensemble à l'aide de la transformation composite. Le but du TP est d'analyser et de comprendre la structure du code fourni, et de compléter les classes afin de produire les différents fichiers de résultat attendus. Afin de vérifier les résultats, des fichiers de références sont fournis. Une fonction compare le fichier résultat produit par l'exécution du programme au fichier de référence correspondant. Il est à noter que tous les fichiers de référence ont été produits sur un système d'exploitation 64bits, ce qui influence la taille de l'information d'entête stockée dans les fichiers.

Implémentation

On vous demande de compléter les fichiers suivants :

- `RepeatTransform.cpp`
- `InvertTransform.cpp, et`
- `CompositeTransform.cpp`

afin que les tests programmés dans la méthode `Test_TP4::executeCompositeTest()` s'exécutent avec succès.

Questions à répondre

1) Identifiez les points suivants :
   a) L'intention du patron Composite.
   b) La structure des classes réelles qui participent au patron ainsi que leurs rôles. Faite un diagramme de classes avec Enterprise Architect pour l'instance du patron composite. Ajouter des notes en UML pour indiquer les rôles, et exportez le tout en pdf).
2) Identifiez toutes les abstractions présentent dans la conception du TP4, et pour chacune, identifiez les responsabilités spécifiques qui lui ont été assignées.
3) Dans l'implémentation actuelle du système *PolyVersion*, quel objet ou classe est responsable de la création de l'arbre des composantes.

3 - Patron Proxy (20 points)

Afin de rendre le système *PolyVersion*, plus efficace, des versions entièrement stockées en mémoire des fichiers d'entrée et de sortie peuvent être utilisées. Ceci

permet de limiter l'accès aux fichiers binaires uniquement aux phases de démarrage et de terminaison du programme. Afin d'intégrer l'approche de stockage en mémoire avec celle utilisant directement des fichiers, le patron de conception Proxy a été utilisé lors de la conception de cette partie du code.

On vous demande d'analyser et de comprendre le fichier suivant :

- `MemAudioFile.cpp`

Tel qu'il est fourni, ce fichier doit normalement permettre que les tests programmés dans la méthode `Test_TP4::executeProxyTest()` s'exécutent avec succès. La logique de ce fichier et son interaction avec le fichier « AudioFile.[h/cpp] » est cependant relativement complexe, il est donc fourni en entier, et il ne contient aucun code à compléter. Le but de cette partie vise donc à analyser le code fourni et en comprendre le fonctionnement.

Questions à répondre

1) Identifiez les points suivants :
   a) L'intention du patron Proxy.
   b) La structure des classes réelles qui participent au patron ainsi que leurs rôles. Faite un diagramme de classes avec Enterprise Architect pour l'instance du patron proxy. Ajouter des notes en UML pour indiquer les rôles, et exportez le tout en pdf.

4 – Conteneurs et Patron Iterator (30 points)

Afin de stocker les pointeurs sur les primitives enfant, la classe *CompositeTransform* utilise un conteneur de la STL et l'interface de la classe *AbsTransform* fournit une interface utilisant le patron Iterator pour donner accès aux enfants du Composite. Le code nécessaire pour implémenter cette fonctionnalité est complet et on vous demande de l'étudier afin d'en comprendre le fonctionnement. Pour cette partie, vous n'avez donc pas de code à ajouter, mais simplement des questions auxquelles vous devez répondre.

Questions à répondre

2) Identifiez les points suivants :
   a) L'intention du patron Iterator.

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.

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.
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? On vous demande de faire ce changement et d'indiquer toutes les modifications qui doivent être faites à l'ensemble du code suite au changement. Reliez la liste des changements à effectuer à la notion d'encapsulation mise de l'avant par la programmation orientée-objet. À votre avis, la conception proposée dans le TP4 respecte-t-elle le principe d'encapsulation ?
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.

5 – À remettre

Une archive `LOG2410_TP4_matricule1_matricule2.zip` qui contient les éléments suivants :

a) Un fichier en format pdf intégrant toutes vos réponses aux questions et les diagrammes produits avec Enterprise Architect.
b) Les trois fichiers C++ que vous avez modifiés pour le patron Composite, c'est-à-dire,
    a) `RepeatTransform.cpp`
    b) `InvertTransform.cpp, et`
    c) `CompositeTransform.cpp`