LOG2410 – TP4 Automne 2017

Conception à base de patrons I

1 - Objectifs

Ce laboratoire permettra aux étudiants de se familiariser avec l'implémentation des patrons de conception « Composite » et « Proxy ». 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)

L'application PolySonor permet de manipuler des fichiers en format binaire. 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 produit sur un système 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) Dans l'implémentation actuelle du système PolyScino, quel objet ou classe est responsable de la création de l'arbre des composantes.

3 - Patron Proxy (50 points)

Afin de rendre le système PolySonor, 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 très probable qu'il reste certains problèmes non détectés par les tests simples déjà implémentés. Nous vous encourageons donc à tester plus à fond cet ensemble de fichiers, et à proposer une version améliorée du code, en documentant les changements effectués.

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.
2) Proposez au moins un test supplémentaire des classes impliquées dans le patron, et expliquez quel aspect du patron est testé, qui ne l'était pas auparavant. Si des corrections sont nécessaires au code fourni afin que le code s'exécute correctement, documentez les changements que vous avez effectués.

4 – À remettre

1) Une archive `LOG2410_TP4_matricule1_matricule2.zip` qui contient les éléments suivants :
   a) Le fichier `ReponsesAuxQuestions.pdf` avec la réponse aux questions 2.1a), 2.2), 3.1a) et 3.2)
   b) Le fichier `DiagrammeDeClasses_Composite.pdf` pour le diagramme de classes des deux patrons composite de la question 2.1b)
   c) Le fichier `DiagrammeDeClasses_Proxy.pdf` pour le diagramme de classes des deux patrons composite de la question 2.1c)

d) Les trois fichiers C++ que vous avez modifiés pour le patron Composite, c'est-à-dire,
   i) RepeatTransform.cpp
   ii) InvertTransform.cpp, et
   iii) CompositeTransform.cpp
e) Les fichiers que vous aurez eu à modifier pour faire mieux fonctionner le patron Proxy.

.