LOG2410 – TP4 Hiver 2018

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 « Decorator ». 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 (40 points)

L'application PolyFusion3D permet de manipuler des objets 3D qui peuvent être constitués de plusieurs parties. Chaque partie regroupe un certain nombre de triangles. Le nombre de parties et le nombre de triangles dans chaque partie est arbitraire et peut être aussi grand que désiré. La structure des objets 3D est représentée en appliquant le patron Composite auquel participe principalement trois classes :

- *AbsObject3D* est une classe abstraite dont toutes les méthodes sont virtuelles.
- *Objet3DPart* est une classe concrète dérivée de la première, qui permet de représenter une partie d'objet composée d'un certain nombre de triangles.
- *Objet3DComposite* est une seconde classe concrète dérivée de la première classe qui permet de regrouper les différentes parties d'un objet selon une complexité arbitraire.

Les fichiers d'entête de chacune des classes vous sont fournis et sont complets. Les seuls fichiers à compléter sont les fichiers .cpp correspondant aux deux classes concrètes. Chaque fonction à compléter est clairement identifiée et sa fonctionnalité est décrite directement dans le fichier.

Une classe de test est fournie (*TP4_Test*), qui construit des objets simples et des objets composites et les imprime à l'écran. Le fichier de résultat attendu vous est également fourni afin de vous permettre de vérifier le bon fonctionnement de votre code.

Implémentation

On vous demande de compléter les fichiers suivants :

- `Objet3DPart.cpp, et`
- `Objet3DComposite.cpp.`

Les tests programmés dans la méthode `TP4_Test::testComposite()` doivent produir le même résultat que celui fourni.

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.

3 - Patron Proxy (20 points)

Afin de permettre d'appliquer des transformations aux triangles qui représentent une partie spécifique d'un objet, une classe d'itérateur « intelligent » a été développée. Cette classe peut être configurée afin d'appliquer dynamiquement une transformation ajoutée comme un décorateur à une partie d'objet (voir section suivante). La classe de l'itérateur est entièrement développée et incorpore le concept d'une cache pour stocker localement dans l'objet itérateur le résultat de l'application d'une transformation à un triangle. Afin d'éviter de charger la cache lorsque l'itérateur pointe à la fin du conteneur des triangles, un itérateur de fin doit toujours être spécifié lors de la construction de l'itérateur intelligent. Cette classe joue le rôle d'un Proxy pour accéder aux triangles et leur appliquer au besoin une transformation.

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

- `TriangleProxyIterator.h`

Tel qu'il est fourni, ce fichier doit normalement permettre qu'une fois le code du patron composite complété, que les tests programmés dans la méthode `TP4_Test::testComposite()` s'exécutent avec succès. La logique de ce fichier et son interaction avec le fichier « *Objet3DPart.[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, advenant que vous y détectiez des problèmes.

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 – Patron Decorator (40 points)

Pour illustrer l'application de transformations aux différentes parties des objets 3D, une classe de transformation simple est fournie, qui permet d'appliquer des translations dans l'espace aux sommets des triangles. La classe *Objet3DTransform* dérive, elle aussi, de la classe de base abstraite *AbsObjet3D*, selon le patron Decorator. Lorsque la classe *Objet3DTransform* est associée à une partie d'un objet, toutes les instances des itérateurs intelligents qui serviront à d'accéder aux triangles de cette partie doivent être informées qu'une transformation est active. Le mécanisme permettant d'informer les itérateurs intelligent de la présence de la transformation consiste à injecter le pointeur `this` de la transformation dans l'itérateur à l'aide de la méthode `setTransform` de l'itérateur. Il revient à la

classe *Objet3DTransform* la responsabilité de s'injecter dans l'itérateur lorsque les itérateurs sont créés.

Implémentation

On vous demande de compléter le fichier suivant :

- `Objet3DTransform.cpp`

afin que les tests programmés dans la méthode `Test_TP4::testDecorator()` produise le même résultat que celui fourni.

Questions à répondre

1) Identifiez les points suivants :
   a) L'intention du patron Decorator.
   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 Decorator. Ajouter des notes en UML pour indiquer les rôles, et exportez le tout en pdf).

5 – 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), 3.2) et 4.1a)
   b) Le fichier `DiagrammeDeClasses_Composite.pdf` pour le diagramme de classes du patron composite de la question 2.1b)
   c) Le fichier `DiagrammeDeClasses_Proxy.pdf` pour le diagramme de classes des du patron proxy de la question 3.1b)
   d) Le fichier `DiagrammeDeClasses_Decorator.pdf` pour le diagramme de classes des deux patrons composite de la question 4.1b)
   e) Les trois fichiers C++ que vous avez modifiés , c'est-à-dire,
      i) `Objet3DPart.cpp`
      ii) `Objet3DComposite.cpp, et`
      iii) `Objet3DTransform.cpp`
   f) Les fichiers que vous aurez eu à modifier pour tester le patron Proxy (par exemple le fichier `TP4_Tests.cpp`).