LOG2410 – TP4 Automne 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 », « Decorator » 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)

L'application PolyIcone3D permet de manipuler des objets 3D qui peuvent être constitués de plusieurs primitives. Chaque primitive est décrite par un certain nombre de paramètres. Le nombre de primitives 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 :

- *Objet3DAbs* est une classe abstraite dont toutes les méthodes sont virtuelles.
- *PrimitiveAbs* est une classe abstraite dérivée de la première, dont dérivent chacune des primitives concrètes. Trois primitives concrètes sont fournies, soit *Cube*, *Cylinder* et *Sphere*.
- *Objet3DComposite* est une classe concrète dérivée de *Objet3DAbs* qui permet de regrouper les différentes primitives pour former des objets d'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 différentes classes décrites ci-dessus, sauf pour la classe *Objet3Dabs*, pour laquelle il n'y a aucune méthode à implémenter. 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 :

- `PrimitiveAbs.cpp,`
- `Cube.cpp,`
- `Cylinder.cpp,`
- `Sphere.cpp,`
- `Objet3DComposite.cpp.`

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

Une fois le code complété et les test réussis, on vous demande d'implémenter une nouvelle primitive *Torus* (https://fr.wikipedia.org/wiki/Tore) en ajoutant vous-même à la solution les fichiers *Torus.h* et *Torus.cpp*, et de tester votre nouvelle primitive en remplaçant les énoncés créant ou utilisant la primitive *Cylinder* par des énoncés équivalents utilisant la primitive *Torus* dans la fonction `TP4_Test::testComposite()`.

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 intégrer votre diagramme sous forme d'image dans votre fichier de réponses.
2) Identifiez la ou les abstractions présentent dans la conception du TP4, et pour chacune, identifiez les responsabilités spécifiques qui lui ont été assignées.

3 – Patron Decorator (30 points)

Pour illustrer l'application de transformations aux différentes primitives, une classe de transformation simple est fournie, qui permet d'appliquer des translations et des mises à l'échelle uniformes sur les différentes primitives. La classe *TransformedObjet3D* dérive, elle aussi, de la classe de base abstraite *Objet3DAbs*, selon le patron Decorator. Lorsque la classe *TransformedObjet3D* est associée à une primitive, certaines responsabilités des classes de Primitives sont réinterprétées en fonction du Decorator de façon à tenir compte des transformations appliquées.

Implémentation

On vous demande de compléter le fichier suivant :

- `TransformedObjet3D.cpp`

afin que les tests programmés dans la méthode `Test_TP4::testDecorator()` produise le même résultat que celui fourni. Assurez-vous que les tests fonctionnent également pour votre primitive *Tore* en remplaçant les primitives *Cylinder* par des primitives *Torus* et en réexécutant les tests.

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 intégrer votre diagramme sous forme d'image dans votre fichier de réponses.
2) Identifiez les responsabilités des classes primitives qui sont réinterprétées lorsque le Decorator est utilisé.
3) Selon vous, pourquoi dans la conception actuelle, un Decorator s'applique aux primitives (classe *PrimitiveAbs*) et non à tous les objets 3D (*Objet3Dabs*) ? Serait-il possible d'appliquer le Decorator à tous les objets et quelle en serait les conséquences ?

4 – Conteneurs et Patron Iterator (20 points)

Afin de stocker les pointeurs sur les primitives enfant, la classe *Objet3DComposite* utilise un conteneur de la STL et l'interface de la classe *Objet3DAbs* 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

1) 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.
2) Expliquez le rôle de l'attribut statique m_emptyContainer défini dans la classe PrimitiveAbs. Expliquez pourquoi, selon vous, cet attribut est déclaré comme un attribut statique et privé.
3) 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 à ce 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 ?
4) Les classes dérivées Objet3DIterator et Objet3DIterator_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

1) Une archive `LOG2410_TP4_matricule1_matricule2.zip` qui contient les éléments suivants :
   a) Le fichier `ReponsesAuxQuestions.pdf` avec les réponses aux questions posées dans les sections 2, 3 et 4
   b) Les fichiers C++ que vous avez ajoutés ou complétés, c'est-à-dire,
      a) `PrimitiveAbs.cpp`
      b) `Objet3DComposite.cpp`
      c) `Cube.cpp`
      d) `Cylinder.cpp`
      e) `Sphere.cpp`
      f) `Torus.h et Torus.cpp`
      g) `TransformedObjet3D.cpp`
   c) Le nouveau fichier de test que vous aurez eu à modifier pour tester la primitive Torus et le nouveau conteneur pour stocker les enfants du Composite (`TP4_Tests.cpp`).