LOG2410 – TP5 Hiver 2018

Conception à base de patrons II

1 - Objectifs

Ce laboratoire permettra aux étudiants de se familiariser avec l'implémentation des patrons de conception « Visiteur » et « Singleton ». 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 Visiteur – 1ère partie (30 points)

Comme vu lors du TP4, le système PolyFusion3D permet de manipuler des objets 3D simples et des objets 3D composés récursivement selon le patron Composite. Afin de poursuivre le développement de l'application PolyFusion3D, on veut permettre de définir un grand nombre d'opérations sur les objets 3D, qu'ils soient simples ou complexes. Ces opérations seront implémentées à l'aide du patron Visiteur.

Dans un premier temps, on veut remplacer la fonction d'impression d'un objet composite sur un *stream* par une approche plus flexible, capable de traiter chaque objet du composite selon son type spécifique. La fonction d'impression `outToStream` fournie dans le TP4 a donc été remplacée dans le TP5 par l'utilisation d'un visiteur implémenté dans la classe `OutputVisitor`, que vous devez compléter.

Une méthode pure virtuelle `accueillir` recevant en paramètre une référence à un visiteur abstrait a été ajoutée à l'interface de la classe `AbsObjet3D` afin de permettre aux visiteurs d'agir sur les objets 3D. Par ailleurs, l'interface de la classe `AbsObjet3D` a été grandement simplifiée suite à la migration de la fonction d'impression vers un visiteur. Il n'est en effet plus nécessaire que la classe abstraite inclue des méthodes d'itération sur les triangles, qui sont maintenant spécifiques à la classe `Objet3DPart`.

Implémentation

On vous demande de compléter le fichier suivant :

`1) OutputVisitor.cpp`

afin que les tests programmés dans la méthode `TP5_Test::testVisiteurOutput()` s'exécute correctement et produise le résultat contenu dans le fichier *`resultatVisiteurOutput.txt`*. Les parties à compléter ont été clairement identifiées par un commentaire fournissant une description des algorithmes à implémenter. À noter que l'indentation des sorties en fonction du niveau de parcours dans l'arbre composite doit être gérée dans les méthodes de traitement du visiteur.

Questions à répondre

1) Précisez les points suivants :
   a) Identifiez l'intention et les avantages du patron Visiteur.
   b) Tracer un diagramme de classes avec Enterprise Architect des différentes classes impliquées dans cette application du patron Visiteur, et ajouter des notes en UML pour indiquer les rôles, et exportez le tout en pdf.
   c) Si en cours de conception vous constatiez que vous voudriez ajouter une nouvelle sous-classe dérivée de `AbsObjet3D`, établissez la liste de toutes les classes qui doivent être modifiées.

3 – Patron Singleton (40 points)

Tel que développé dans le TP4, le décorateur implémenté par la classe `Objet3DTransform` permettait d'appliquer une transformation à un objet 3D simple représenté par un objet de la classe `Objet3DPart`, mais ne permettait pas d'appliquer de transformation aux objets composites implémentés par la classe `Objet3DComposite` et à tous leurs enfants. En fait, même si un décorateur était appliqué à un `Objet3DComposite`, ce décorateur restait sans effet puisque la transformation incluse dans le décorateur n'était pas utilisée pour construire les itérateurs sur les triangles des objets enfants du composite.

Une solution à ce problème consiste à rendre disponible une « transformation courante » accessible de façon globale au travers du patron Singleton. De cette façon, chaque fois qu'un itérateur de triangle est construit, le processus de

construction peut accéder à la transformation courante et l'utiliser. Le TP5 utilise cette approche pour permettre de transformer tous les objets 3D, qu'ils soient simples ou composites. Afin que le mécanisme de transformation soit complètement fonctionnel, il est nécessaire durant la traversée de l'arbre composite, de sauver la transformation associée à un niveau de l'arbre avant de passer au niveau des enfants. De cette façon, lorsqu'une transformation est spécifiée parmi les enfants d'un objet composite, cette transformation n'affecte que les enfants. Une fois les enfants traités, la transformation associée à leur niveau doit être éliminée, et celle associée au niveau supérieur doit être rétablie. Une façon naturelle de gérer les transformations selon les niveaux de l'arbre composite est donc de les placer sur une pile, qui est manipulée durant la traversée. À cette fin, la classe `TransformStack` implémente une pile de transformations accessible de façon globale. Chaque fois qu'un itérateur sur des triangles est construit, la transformation courante est utilisée automatiquement dans l'itérateur pour transformer les coordonnées des sommets.

Implémentation

On vous demande de compléter les fichiers suivants :

`2) TransformStack.cpp`

afin que les tests programmés dans la méthode `TP5_Tests::testSingleton()` s'exécutent correctement et produisent les résultats contenus dans le fichier *`resultatTestSingleton.txt`*. Les parties à compléter ont été clairement identifiées par un commentaire.

Questions à répondre

2) Précisez les points suivants :
   a) L'intention du patron Singleton.
   b) La classe `TransformStack` n'est-pas une instanciation classique du patron Singleton. Expliquez en quoi cette mise en œuvre du Singleton diffère de l'approche habituelle.
   c) Quels sont les avantages et les inconvénients de cette approche pour la gestion des transformations ? En particulier, selon vous, cette approche est-elle « statefull » ou « stateless » tel que discuté en classe ?

d) Serait-il possible de stocker la transformation d'un niveau de l'arbre comme un attribut des objets composites ? Quels seraient les avantages et les inconvénients d'une telle approche ?

4 - Patron Visiteur – 2ème partie (30 points)

La pile de transformation développée à l'aide du Singleton est prévue pour être utilisée lors de la traversée des objets composites afin de conserver, pour chaque niveau de l'arbre, la transformation qui s'applique aux objets d'un niveau donné. Comme cette fonctionnalité de gestion des transformations est étroitement liée à la fonctionnalité de traversée de l'arbre, et que les traversées sont implémentées à l'aide de visiteurs, il semble logique d'intégrer ces deux fonctionnalités dans une classe abstraite qui servira ensuite de base à l'implémentation de visiteurs concrets. À cette fin, une nouvelle classe abstraite `TransformVisitor` est incluse dans le projet. Cette classe est abstraite puisque sa fonction de traitement des `Objet3DPart` est pure virtuelle, mais les méthodes de traitement associées aux `Objet3DComposite` et `Objet3DTransform` doivent être implémentées afin de pouvoir être réutilisées par les classes dérivées.

Une fois la classe abstraite `TransformVisitor` implémentée, il devient beaucoup plus simple d'implémenter des visiteurs concrets permettant de réaliser différentes fonctionnalités sur un objet composite. Par exemple, on peut définir un visiteur de la classe `BoundingBoxCalculator` pour calculer les dimensions de la boîte 3D englobant tous les triangles contenus dans un objet simple ou complexe. On peut aussi définir un nouveau visiteur de sortie, de la classe `OutputTransformVisitor`, qui, plutôt que de sérialiser les `Objet3DTransform` sur un *stream*, gère les transformations et produit en sortie une représentation des triangles transformés par les différentes transformations contenues dans l'arbre.

Implémentation

On vous demande de compléter les fichiers suivants :

3) `TransformVisitor.cpp`
4) `OutputTransformVisitor.cpp, et`
5) `BoundingBoxCalculator.cpp`

afin que les méthodes

- `TP5_Tests::testVisiteurTransformOutput(), et`
- `TP5_Tests::testBoundingBoxCalculator()`

produisent des résultats correspondant à ceux contenus dans les fichiers `resultatOutputTransformVisitor.txt` et `resultatBoundingBoxCalculator.txt`. Les parties à compléter ont été clairement identifiées par un commentaire fournissant une description des algorithmes à implémenter.

5 - À remettre

Le TP5 est à remettre sur le site Moodle du cours au plus tard le **mardi 17 avril 2018 à 23h55**. Vous devez remettre une archive `LOG2410_TP5_matricule1_matricule2.zip` qui contient les éléments suivants :

a) Le fichier `ReponsesAuxQuestions.pdf` avec la réponse aux questions posées (sauf le diagramme de classes).
b) Le fichier `DiagrammeDeClasses_VisiteurOutput.pdf` pour le diagramme de classes du patrons Visiteur de la question 2.1b).
c) Les 5 fichiers C++ que vous avez modifiés, c'est-à-dire, `OutputVisitor.cpp, TransformStack.cpp, TransformVisitor.cpp OutputTransformVisitor.cpp` et `BoundingBoxCalculator.cpp`. Vous ne pouvez pas modifier les autres fichiers `.h` et `.cpp`. Le correcteur va insérer vos fichiers dans le code, et ça doit compiler et s'exécuter.