LOG2410 – TP5 Automne 2018

Conception à base de patrons II

1 - Objectifs

Ce laboratoire permettra aux étudiants de se familiariser principalement avec l'implémentation des patrons de conception « Visiteur » et « Commande ». 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 (50 points)

Comme vu lors du TP4, le système PolyIcone3D 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 PolyIcone3D, 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 a remplacé 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, et de permettre de faire varier facilement le format de sortie. À l'opérateur de sortie sur un *stream* et aux fonctions d'impression `toStream` fournis dans le TP4, on a donc ajouté dans le TP5 un visiteur implémenté dans la classe `VisitorXMLWriter`, qui fonctionne déjà et que vous n'avez pas à modifier. Cette classe vous est fournie comme exemple pour vous permettre de compléter la classe `VisitorPrimitiveSelector`, qui permet de parcourir un objet composite et de sélectionner des primitives d'un type spécifié contenues dans le composite. Le type de primitive à sélectionner est fourni en paramètre lors de la construction du visiteur. Lors du parcours d'un objet composite, une référence à chaque primitive dont le type correspond au type recherché est stockée dans un conteneur d'objets sélectionnés. La référence est un itérateur pointant sur la primitive sélectionnée. Les primitives « décorées » par une transformation peuvent être sélectionnées. L'itérateur pointe alors sur l'objet transformé (le décorateur).

Implémentation

On vous demande de compléter le fichier suivant :

- `VisitorPrimitiveSelector.cpp`

afin que les tests programmés dans la méthode `TP5_Test::testVisitor()` s'exécutent avec succès. Les parties à compléter ont été clairement identifiées par un commentaire fournissant une description des algorithmes à implémenter.

Questions à répondre

1) Identifiez l'intention du patron Visiteur.
2) Tracez un diagramme de classes avec Enterprise Architect illustrant les deux instances du patron Visiteur (impression en format XML et sélection des primitives) et les classes sur lesquelles agissent les visiteurs. Ajoutez des notes en UML pour indiquer les rôles de chaque classe, et incorporez le diagramme à votre document de réponses.
3) Deux approches différentes pour implémenter une fonction d'impression sur un *stream* sont démontrées dans le TP. La première, utilisant l'opérateur de sortie `std::ostream& operator<<(std::ostream& o, const Object3DAbs& obj3d);` est basée sur le patron Template Method. La seconde, utilisant la classe `VisitorXMLWriter` est basée sur le patron visiteur. Identifiez l'intention du patron Template Method.
4) Établissez la liste des classes et méthodes impliquées dans l'implémentation de chaque approche et discutez, en 250 mots ou moins, des avantages et inconvénients de chaque approche .
5) Si en cours de conception vous constatiez que vous voudriez ajouter une nouvelle sous-classe dérivée de `PrimitiveAbs`, établissez la liste de toutes les classes qui doivent être modifiées à cause de l'utilisation du patron Visitor.
6) Selon vous, l'application des transformations aux primitives pourrait-elle être implémentée comme un visiteur ? Si oui, discutez en 250 mots ou moins, des avantages et inconvénients d'utiliser le patron visiteur pour cette fonction et sinon expliquez pourquoi le patron n'est pas applicable.

3 - Patron Commande (50 points)

Afin de manipuler plus efficacement les icones3D, il pourrait être très intéressant pour un usager de pouvoir définir une séquence d'opérations qui s'appliquent à

une icône, et de pouvoir exécuter ces opérations, les annuler ou les réexécuter. Le patron de conception Command permet cette flexibilité en permettant d'exécuter différents types de manipulations sur les icônes en les invoquant grâce à la méthode `Invoker::execute(CmdPtr&)`.

Implémentation

On vous demande de compléter les fichiers suivants :

- `Invoker.cpp`
- `SelectPrimitiveCmd.cpp`
- `XMLPrintCmd.cpp`
- `TransformCmd.cpp`

afin que les tests programmés dans la méthode `TP5_Test::testCommand()` s'exécutent avec succès. Les parties à compléter ont été clairement identifiées par un commentaire.

La classe `TP5_Test` inclut une méthode interactive de test des commandes afin de faciliter le développement des commandes une à la fois. En activant la méthode `RESULTAT TP5_Test::interactivTestCommand()`, (en supprimant les commentaires devant l'énoncé '`#define INTERACTIF`' dans l'entête du fichier `PolyIcone3D.cpp`) vous pourrez invoquer chaque commande en saisissant interactivement le caractère qui lui est associé :

- 'p' : permet de déclencher l'impression soit de l'icône soit des primitives sélectionnées en format XML.
- 'q' : permet de terminer la saisie interactive de commande et l'exécution de la méthode.
- 'r' : permet de réexécuter une commande qui a été annulée.
- 's' : permet de sélectionner toutes les primitives d'un type donné en saisissant un second caractères pour identifier le type : 'c' pour les cubes, 'y' pour les cylindres, 's' pour les sphères.
- 't' : permet de transformer les primitives sélectionnées en leur ajoutant un décorateur. Le facteur d'échelle de la transformation doit être saisi au clavier et la translation est fixe.
- 'u' : permet d'annuler une commande.
- 'x' : permet d'annuler la sélection courante des primitives.

Questions à répondre

1) Identifiez les points suivants :
   a) L'intention du patron Command.
   b) La structure des classes réelles qui participent au patron Command ainsi que leurs rôles. Faite un diagramme de classes avec Enterprise Architect, ajoutez des notes en UML pour indiquer les rôles, et incorporez le tout à votre document de réponses.
2) Observez attentivement la classe `Invoker` qui permet de gérer la relation entre les commandes et les icônes 3D. En plus de participer au patron Commande, cette classe agit comme Mediator entre les commandes et les icônes. Puisque la classe `Invoker` agit comme Mediator, il pourrait sembler logique d'utiliser le patron Singleton lors de la définition de cette classe.
   a) Quel sont les intentions des patrons de conception Mediator et Singleton?
   b) Quels sont les avantages de définir la classe `Invoker` comme Mediator ?
   c) Discuter en 250 mots ou moins, des avantages et des inconvénients de définir la classe `Invoker` comme Singleton ?
3) Pour compléter la fonctionnalité de PolyIcone3D, il faudrait ajouter de nouvelles sous-classes de la classe `CommandAbs`. Selon vous, est-ce que d'autres classes doivent être modifiées pour ajouter les nouvelles commandes? Justifiez votre réponse.

4 - À remettre

Le TP5 est à remettre sur le site Moodle du cours au plus tard le **mardi 4 décembre 2018 à 23h55**. Vous devez remettre une archive `LOG2410_TP5_matricule1_matricule2.zip` qui contient les éléments suivants :

   a) Le fichier `ReponsesAuxQuestions.pdf` contenant vos réponses aux questions posées et les diagrammes de classes intégrés au document.
   b) Les fichiers C++ que vous avez modifiés, c'est-à-dire, `VisitorPrimitiveSelector.cpp, Invoker.cpp, XMLPrintCmd.cpp, SelectPrimitiveCmd.cpp` et `TransformCmd.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.