LOG2410 -- Conception logicielle

TP4:
Conception à base de patrons I

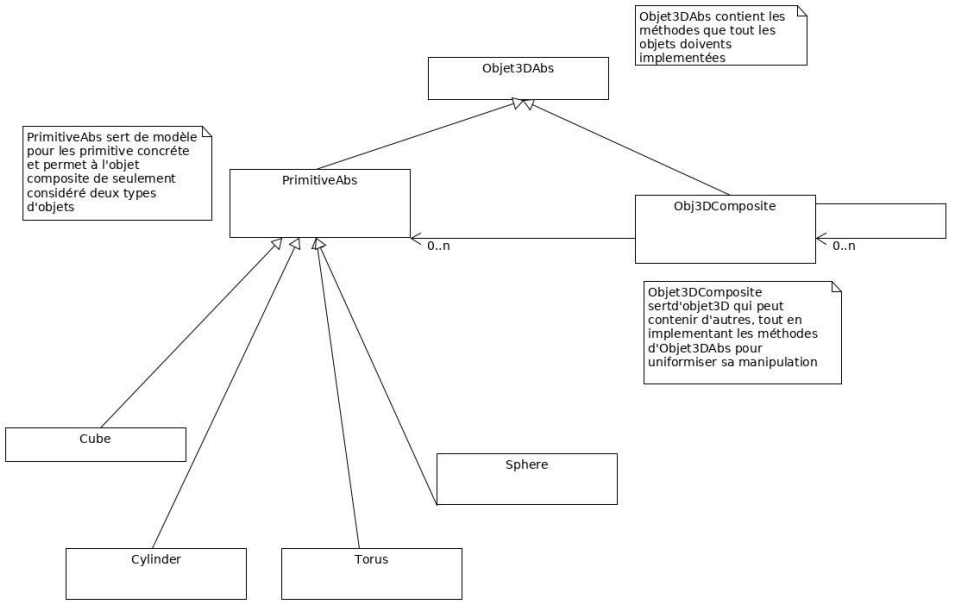
Par

Evan Kirby McGregor 1896985
Alexandre Falardeau 1899696

École Polytechnique de Montréal
18 novembre 2018

Patron Composite

1. Sous-questions répondues ci-dessous
    a. L'intention du patron composite est de remplacer la logique conditionnelle dans les programmes par du polymorphisme (ce qui a aussi comme bénéfice d'enlever le besoin de faire de l'introspection sur les objets avant de les manipuler). L'idée est de pouvoir manipuler des objets ou des structures (composites) d'objets de la même façon, réduisant ainsi la complexité du code.
    b.

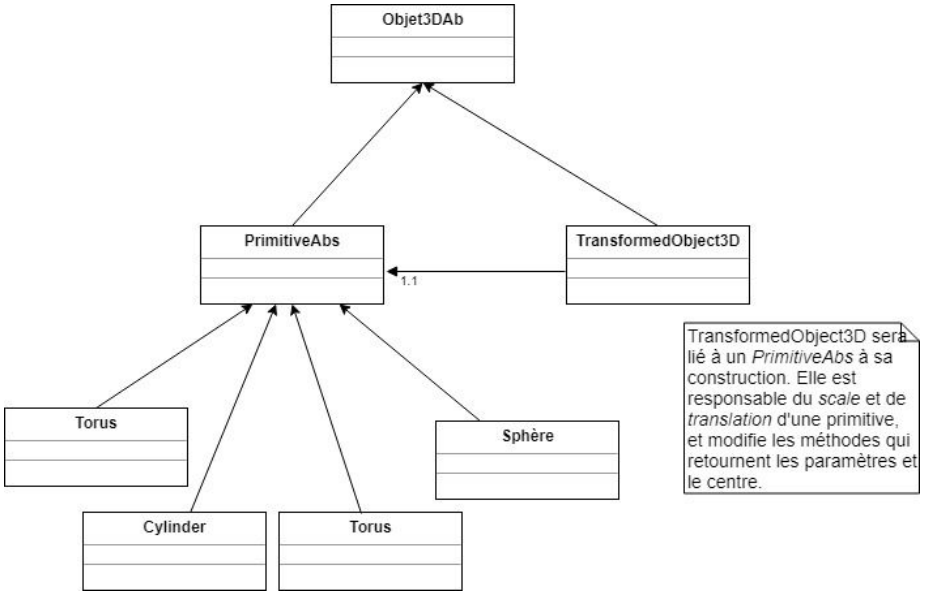


2. Du point de vue du main, la manipulation d'objets se fait par la manipulation *d'Objet3DAbs*. Cette classe virtuelle sert de modèle pour les *Objet3DComposites* et les *PrimitiveAbs* pour que les deux puissent être manipulées de façon uniforme. Une instance de la classe *Objet3DComposite* fait abstraction de la façon dont les objets qu'elle contient sont modifiés en implémentant les méthodes uniformes *d'objet3DAbs*. La classe *PrimitiveAbs* est une classe qui modélise les objets 3D de base qui seront utilisés pour composer un *Objet3DComposite*. Ceci permet à un objet composite de considérer seulement deux types d'objets lors de l'implémentation des méthodes qui affectent ses "enfants".

Patron Décorateur

1. Questions répondues ci-dessous
    a. Le patron décorateur sert à ajouter des fonctionnalités supplémentaires à un objet sans avoir à modifier la classe qui modélise l'objet. Le patron accomplit ceci en créant des "emballages" pour les objets qui doivent être modifiés après l'instanciation.

b. À la structure déjà représentée dans la question 1.b) du patron *Composite*, on ajoute la classe *TransformedObject3D*, qui est construit avec une instance d'une *PrimitiveAbs.*



2. La plupart des méthodes qui sont implémentées dans la classe *TransformedObject3D* ne font que réutiliser les méthodes de la classe *PrimitiveAbs*. Toutefois, la fonction *getCenter* retourne le *Point3D* du centre plus la translation à effectuer (*m_translation),* et la fonction *getParameters* retourne les paramètres avec le facteur de multiplication (*m_scale*) appliqué à chacun.
3. La classe Decorator que nous avons implémenté s'applique aux primitives et non à tous les objets 3D puisque les responsabilités ajoutés sont spécifiques aux primitives, et ne seront donc pas tous utiles pour les autres classes qui dérivent de la classe *Objet3Dabs*. Il ne serait donc pas possible d'appliquer le *Decorator* à tous les objets sans répéter du code trouvé dans *PrimitiveAbs*. En effet, les méthodes qui sont redéfinies dans *PrimitiveAbs* devront être redéfinies dans *TransformedObject3D,* plutôt que de simplement les réutiliser.

Conteneurs et Patron Iterator

1. Le patron iterator
    a. Le patron iterator a comme but d'uniformiser la façon qu'on itère à travers différents contenants d'objets (agrégat), et ce, sans utiliser ou révéler l'implémentation interne de l'objet agrégat. Ce patron permet aussi d'avoir plusieurs itérateurs qui traversent un agrégat en même temps.
    b. Le conteneur STL est un un vecteur (*std::vector*) de pointeurs uniques (*std::unique_ptr*) qui pointent à des *Objet3DAbs*. On y accède avec la classe *Objet3DContainer*. Les itérateurs utilisés sont *Objet3DBaseIterator* et *Objet3DBaseIterator_const,* et sont implémentés selon le *std::vector* de la STL.

2. Nous voulons pouvoir itérer à travers de n'importe quel *Objet3DAbs* de façon uniforme. Cette itération est seulement pertinente si l'objet manipulé est un *Objet3DComposite*. Toutefois, afin de maintenir une manipulation uniforme pour toute instance *d'Objet3DAbs*, même les primitives doivent avoir accès à un conteneur pour qu'ils puissent retourner un itérateur lorsque demandé. Puisque le conteneur ne sera pas utilisé, il peut être statique (pour réduire la mémoire du programme) et privé, puisqu'il ne devrait pas être utilisé sauf pour obtenir un itérateur.
3. Un changement possible serait simplement de modifier la ligne 18 de *Objet3DContainer.h* et de remplacer *std::vector* par *std:list*. Bien sur, il faut aussi rajouter *#include <list>* au fichier. Puisque list et vector utilisent tous les deux une implémentation fonctionnelle de l'itérateur, ceci est le seul changement qui est nécessaire, et le code affiche les mêmes résultats de tests.
4. La surcharge d'opérateurs, tels que "*" et "->", a l'avantage de simplifier la lisibilité et la facilité d'écrire du code. En effet, la surcharge effectue essentiellement la tâche d'une fonction, mais avec une syntaxe qui est plus intuitive à lire. Toutefois, un inconvénient est que l'implémentation de cet opérateur n'est pas nécessairement constante dans tout le code. Lorsque l'implémentation d'un certain opérateur varie selon où il est utilisé (et des paramètres précédents et suivants), cela peut entraîner de la confusion et rendre le code plus difficile à tester et à déboguer.