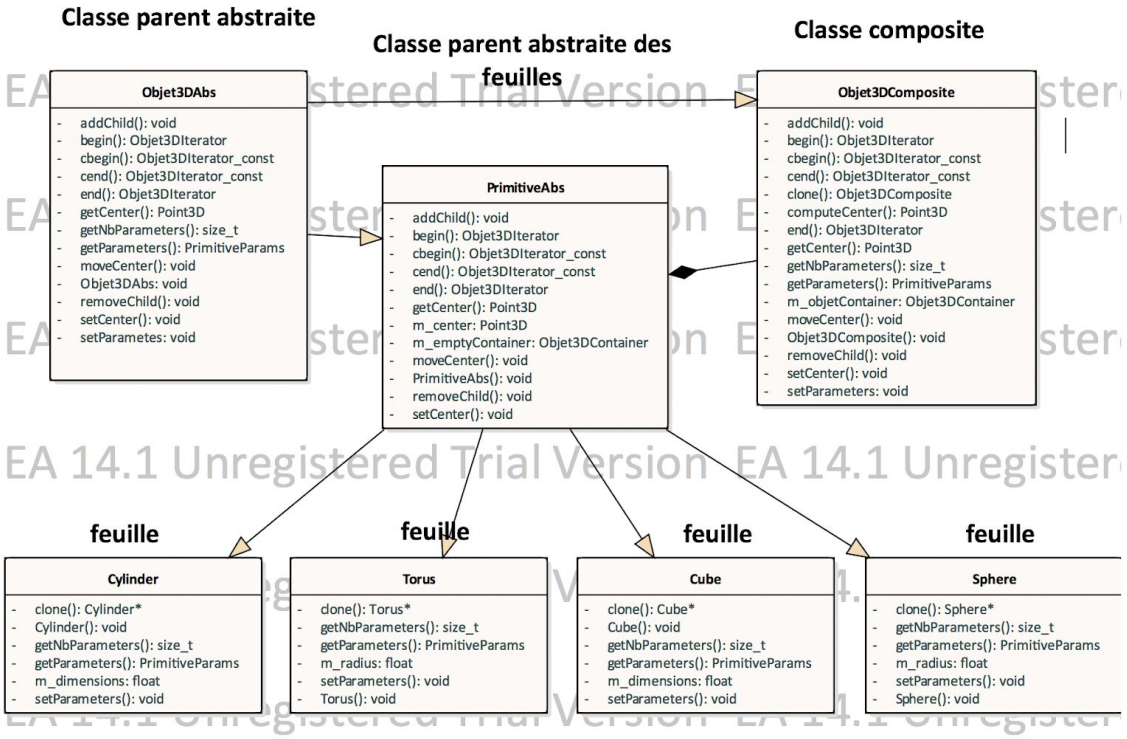
2 - Patron Composite

1.

a. L'intention du patron Composite est de traiter les objets individuels et multiples, composés récursivement, de façon uniforme.

b.



2.

PrimitiveAbs : Cette classe abstraite permet au moyen du polymorphisme, d'appeler les fonctions propres de chaque type de primitive. Elle sert, entre autres, à distribuer les appels aux méthodes, à leur forme respective.

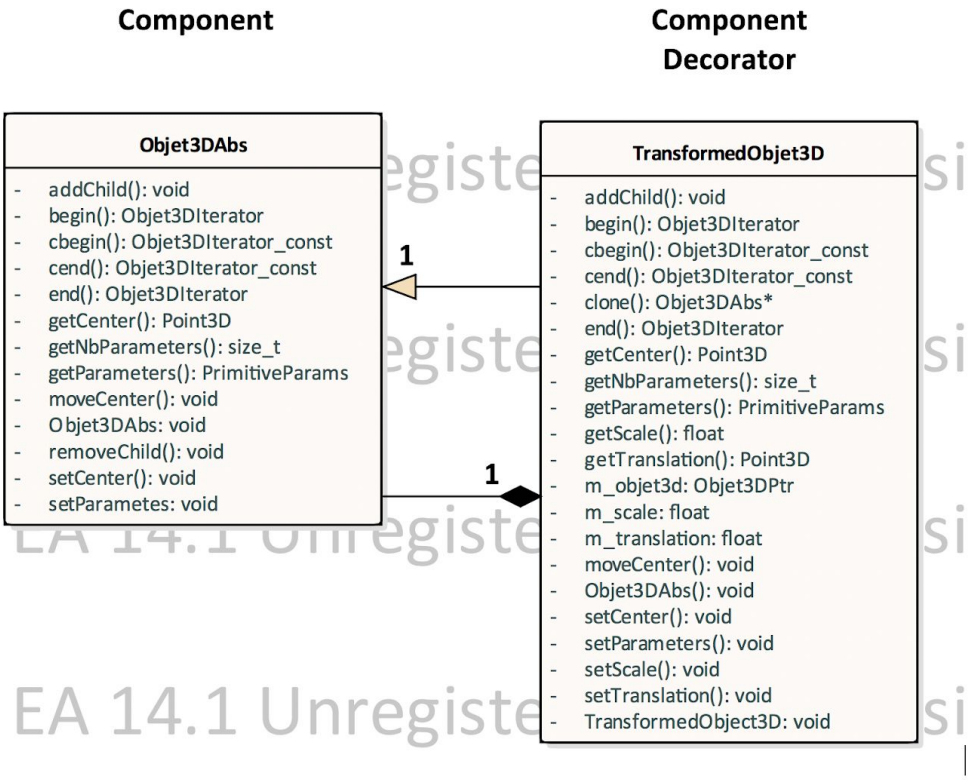
Objet3DAbs : L'objectif principal de cette abstraction est de faciliter l'ajout et la suppression d'un regroupement de Primitives dans une objet composite. Tous les objets héritent de cette classe, car le logiciel peut créer des regroupement de primitives, ou tout simplement une primitive simple.

3 - Patron Décorateur

1.

a. Le patron décorateur ajoute dynamiquement des responsabilités à un objet. Il fournit une alternative à la dérivation pour permettre d'étendre la fonctionnalité d'une classe.

b.



2. Lorsque la classe TransformedObjet3D est implémenté, ceci ajoute les responsabilités de redimensionner et de translater un objet. Puisqu'elle dérive aussi de la classe Objet3DAbs, elle hérite des mêmes fonctionnalités de cette class, et ajoute la possibilité de transformer un objet. Comme le patron décorateur explique, la classe étend les responsabilités de base d'un objet.
3. Selon nous, la classe décorateur ne s'applique qu'aux primitives car une primitive demeure simple à modifier. Tandis que si le décorateur s'appliquait à toutes les classes, il devra implémenter les fonctions de transformation à un vecteur d'objets, dans le cas de la classe Objet3DComposite. Il serait possible d'appliquer cette classe décorative à toutes les Objet3DAbs, mais ceci deviendrait plus compliqué. Les

conséquences de cette extension serait qu'il faudrait implémenter plus de méthodes capables de gérer le vecteur d'objets dans le cas de la classe Objet3DComposite.

5 - Conteneurs et Patron Iterator

1.
    a. Le patron Iterator sert à fournir des méthodes d'accès pour des conteneurs (objet d'éléments liés), sans dévoiler la structure externe de ceux-ci.
    b.
        i. Le conteneur utilisé pour stocker les enfants de la classe Composite est le `vector`.
        ii. Les itérateurs utilisés pour parcourir le conteneur sont Objet3DBaseIterator, qui est un objet de type `iterator`, et Objet3DBaseIterator_const qui est un objet de type `const_iterator`.
2. La classe PrimitiveAbs dérive de la classe Objet3DAbs qui, elle, contient des méthodes virtuelles pures d'accès à un conteneur. Cependant, la classe PrimitiveAbs ne représente pas un ensemble d'objets. PrimitiveAbs est obligé de surcharger toutes les méthodes virtuelles pures de Objet3DAbs, alors il faut implémenter un conteneur vide afin de pouvoir retourner des valeurs non-erronées lors des appels aux méthodes begin(), cbegin() etc. Le conteneur vide est représenté par l'attribut m_emptyContainer. Cet attribut doit rester vide en tout temps, pour maintenir la principale fonctionnalité de l'objet, il est donc déclaré statique et privé pour éviter qu'il soit modifié.
3. Si nous décidons de changer le type du conteneur utilisé pour la classe Composite, il faudra tout simplement modifier le type de Objet3DContainer de std::vector à un autre type de conteneur. Puisque l'accès au contenu se fait avec des itérateurs, il n'y a rien à changer dans l'implémentation des méthodes. Dans notre cas, l'encapsulation serait inchangé. La différence de conteneur ne changera pas la visibilité de son contenu. Puisque nous stockons des pointeurs et les méthodes de déréférencement resteraient inchangées. La conception dans ce TP respecte bien l'encapsulation. À cause du fait que le conteneur utilisé a un niveau d'accessibilité de type privé, son contenu est uniquement accessible via les classes qui dérivent de

Objet3DComposite. Ceci dit, les fonctions externes à ces classes ne pourront changer le contenu de m_objetContainer. Protégeant ainsi la structure interne du conteneur.

4.

a. Avantage : La surcharge des opérateurs permet un accès facilité aux méthodes des objets primitives. Elle permet d'appliquer les mêmes notions de déréférencement sur des classes non-standards.
b. Inconvénient : Il devient moins évident de déterminer sur quels objets les méthodes sont appelés.