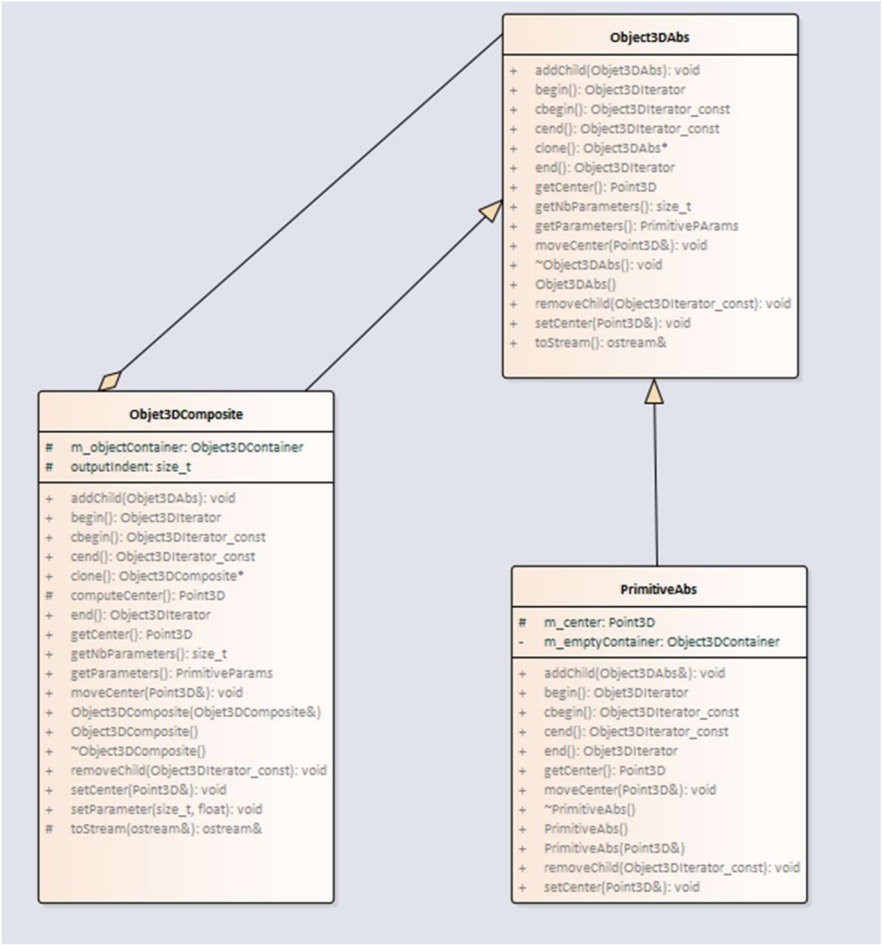
Patrons composites

1) Identifiez les points suivants :

a) L'intention du patron Composite.

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

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 présentes dans la conception du TP4, et pour chacune, identifiez les responsabilités spécifiques qui lui ont été assignées.

Les abstractions présentes dans la conception du TP4 sont les classes Objet3DAbs et Objet3DComposite. La classe Objet3DAbs est une classe virtuelle pure (interface) et permet d'utiliser le type de primitive voulu grâce à l'héritage à travers la classe PrimitiveAbs. La classe Objet3DComposite hérite de la classe Objet3DAbs et peut aussi être un Objet3DAbs, ce qui permet de faire de la composition récursive et donc d'utiliser le patron composite grâce à ces deux classes.

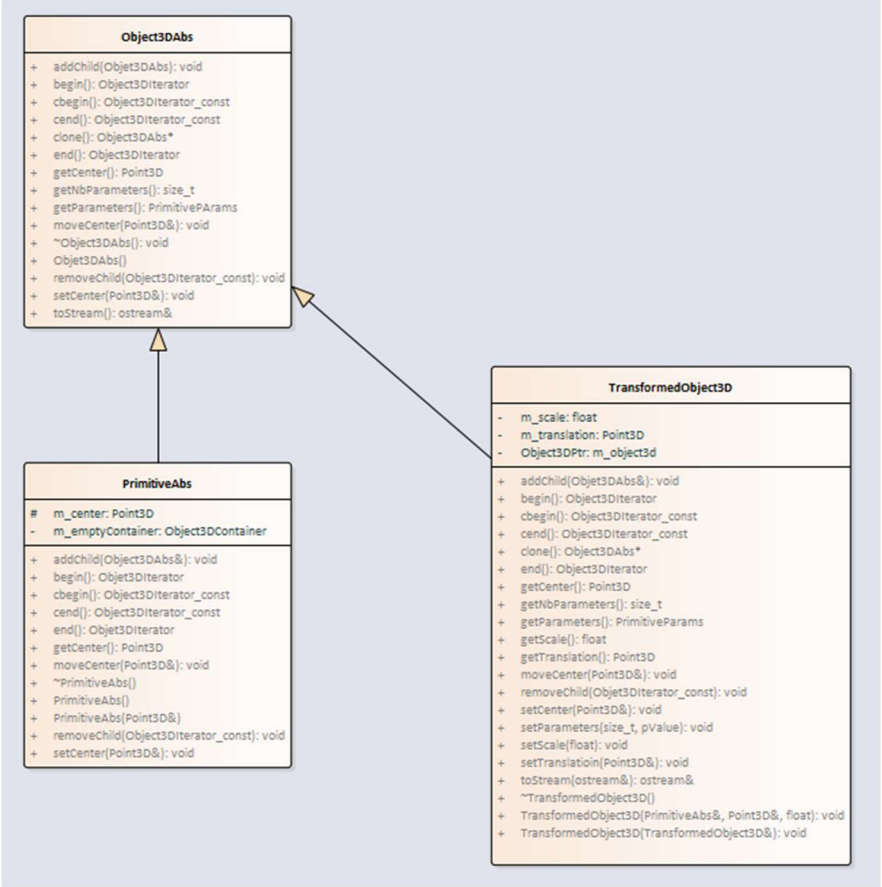
Patron Decorator

1) Identifiez les points suivants :

a) L’intention du patron Decorator.

L'intention du patron Decorator est d'attacher dynamiquement des responsabilités additionnelles à un objet en fournissant une alternative flexible à la dérivation pour étendre la fonctionnalité d’une classe.

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.

Les responsabilités des classes primitives ne changent pas. La classe TransformedObjt3D qui applique le patron Decorator sur la classe PrimitiveAbs permet justement d'ajouter des responsabilités aux objets de la classe PrimitiveAbs qui peuvent ensuite être enlevées. Ces responsabilités ajoutées sont la translation et l'agrandissement de la primitive.

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 ?

Parce que la classe Objet3DAbs est une classe virtuelle pure (interface) alors que la classe PrimitiveAbs ne l'est pas en plus qu'elle accède plus directement aux différentes primitives au niveau de l'héritage. Il serait possible d'appliquer le Decorator à tous les objets 3D (Objet3DAbs), mais cela aurait comme conséquence de réduire la flexibilité qu'apporte le patron Decorator en remplaçant l'héritage statique.

Conteneurs et Patron Iterator

1) Identifiez les points suivants :

a) L'intention du patron Iterator.

L'intention du patron Iterator est de fournir une méthode d'accès séquentielle aux éléments d'un conteneur comme une liste ou un vecteur sans exposer sa structure interne.

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.

La classe du conteneur est Objet3DContainer qui est en fait un vecteur de unique_ptr et les classes des Iterators sont Objet3DBaseIterator et Objet3DBaseIterator_const.

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.

Le rôle de l'attribut statique m_emptyContainer est de permettre à un objet de la classe PrimitiveAbs de pouvoir y stocker les différents solides qui le compose. Cet attribut est statique et privé, car on veut que cet attribut appartienne uniquement à la classe PrimitiveAbs, donc pas aux classes des différents solides.

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?

Si le nouveau conteneur utilisé est un conteneur permettant d'accéder à ses éléments grâce aux [] comme un deque, les seules modifications à faire sont de changer les déclarations du conteneur dans la classe Objet3DContainer. Sinon, si par exemple le nouveau conteneur est une list, il faut changer les appels aux éléments de list dans les tests puisqu'il n'est pas possible d'utiliser les [] pour aller chercher un élément de list. Dans mon cas, j'ai simplement changé les appels de vector par des appels de deque. La conception proposée dans le TP4 repecte le principe d'encapsulation,

car le conteneur est uniquement utilisable par la classe Objet3DContainer et que les attributs de type Objet3DContainer utilisés dans les autres classes sont de visibilité protected.

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.

L'avantage de la surcharge de ces opérateurs est l'accès à l'objet 3D sur lequel pointe l'itérateur est simplifié. Le désavantage de la surcharge de ces opérateurs est que leur utilisation devient ainsi moins flexible ue s'ils ne seraient pas surchargés.