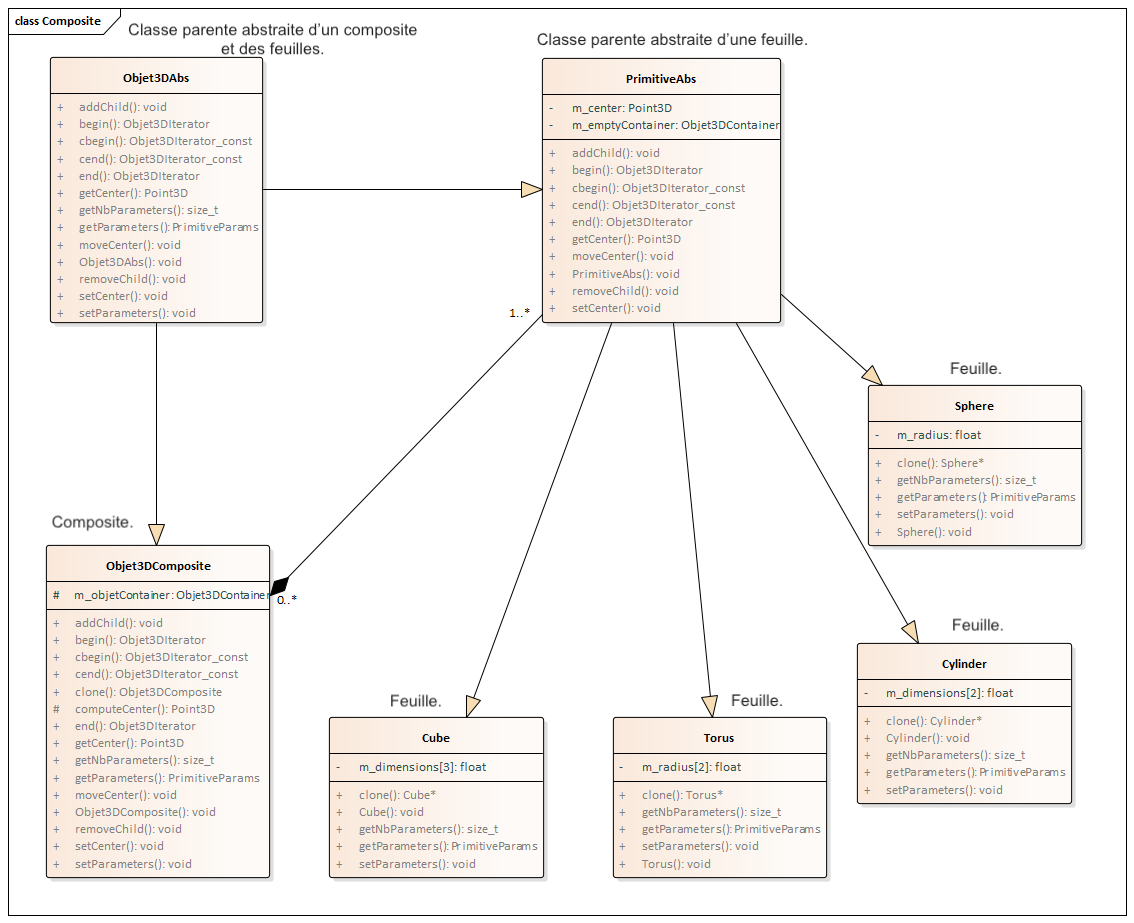
**Patron Composite**

Question #1-

1. L’intention du patron Composite est de traiter les objets individuels, les « leafs » et les objets multiples , les « composite », composés récursivement à partir des objets individuels, de manière uniforme.



Question #2-

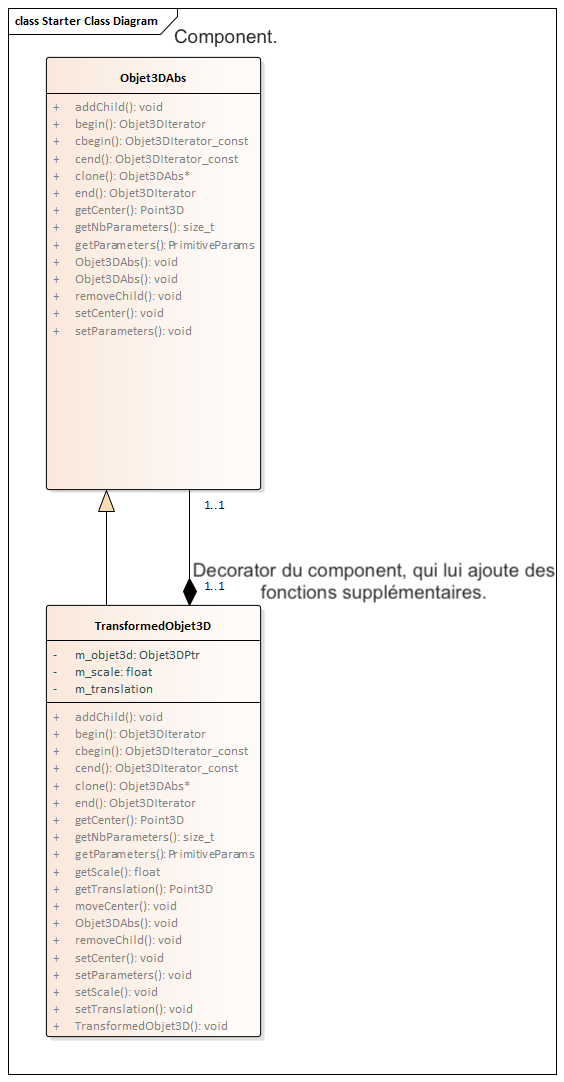
La première abstraction est la présence d’une classe abstraite, Objet3DAbs, qui n’a que des méthodes virtuelles pures, et de laquelle dérivent deux autres classes, soit Objet3DComposite et PrimitiveAbs. La responsabilité de cette abstraction est de permettre à un Objet3DComposite d’être composé et de Primitives, et d’autres Objet3DComposite. Ainsi, lorsqu’on veut poser une action sur un Objet3D, par exemple, modifier le point central de l’objet, il est possible d’appeler la méthode appropriée sur tous les objets dans son conteneur, soit les objets qui le composent, sans qu’il n’y aille d’erreur d’exécution dû au fait qu’une méthode soit appelée sur le mauvais objet.

La deuxième abstraction, est la classe abstraite PrimitiveAbs, de laquelle dérivent les trois classes concrètes Cube, Sphere et Cylinder. Cette abstraction permet d’implémenter de manière homogène les méthodes qui sont communes à toutes les primitives, mais de laisser les classes concrètes implémenter les méthodes qui sont spécifiques à leurs attributs. De cette façon, lorsqu’un Objet3DComposite appelle des méthodes sur les objets qui le composent, il n’a pas à se soucier de la classe concrète de ses primitives, la bonne méthode sera toujours appelée.

**Patron Decorator**

Question #1-

1. L’intention de ce patron est d’ajouter de façon dynamique des responsabilités et des attributs à un objet existant. Ainsi, il offre une alternative plus flexible que l’héritage afin d’étendre les fonctionnalités d’une classe.



Question #2-

La responsabilité des primitives est de fournir les bonnes valeurs de paramètres au TransformedObjet3D afin que ce dernier puisse appliquer les modifications, par exemple, les translations, aux bonne données.

Question #3-

Un patron Decorator s’applique seulement aux primitives, puisque les méthodes qu’ajoutent la classe TransformedObjet3D, telles que la translation et la mise à l’échelle, ne s’applique qu’aux primitives. En effet, lorsqu’on translate le centre d’un objet, c’est le centre des primitives qu’on déplace. Il serait toutefois possible d’appliquer le patron à tous les Objet3D sans conséquence notable, puisque les Objets3D appellent les méthodes des primitives qui les composent, lorsqu’on tente d’accéder à des paramètres qu’ils n’ont pas, par exemple, le centre.

**Patron Iterator**

Question #1-

1. L’intention de ce patron est de permettre d’accéder à tous les éléments d’un objet d’assemblage, telle qu’une liste, de manière séquentielle, sans que la structure interne de cet objet ne soit exposée.
2. La classe du conteneur des objets Composite est std::vector. Les classes des itérateurs sont de classe vector::iterator et vector::const\_iterator.

Question #2-

Comme la classe PrimitiveAbs dérive de la classe Objet3Dabs, elle hérite des méthodes qui permette de parcourir le conteneur des Objet3D, cependant, comme les primitives ne sont pas composées d’autres objets, elles n’ont pas de conteneur. Le conteneur statique et privé permet donc d’implémenter les méthodes héritées, sans que ces méthodes ne puissent réellement faire quoi que ce soit. Cet attribut est statique, afin que toutes les primitives se partagent le même attribut, puisqu’il est vide pour toutes les primitives. Il est privé afin qu’aucune méthode externe ne puisse le modifier.

Question #3-

Il n’y a aucune conséquence, et la seule modification à faire est de changer la ligne :

using Objet3DContainer = std::vector<Objet3DPtr>;

dans la classe Objet3DContainer, pour la ligne :

using Objet3DContainer = std::list<Objet3DPtr>;

Cela est dû au fait que les types ont été redéfinis dans la classe Objet3DContainer et donc, la classe du conteneur a été encapsulée dans cette redéfinition, et on ne la retrouve plus nulle part dans le code, il n’est donc pas nécessaire de changer d’autres lignes. La conception du TP4 respecte donc le principe d’encapsulation.

Question #4-

Un avantage de cette surcharge est qu’il est possible d’accéder directement aux méthodes de l’objet sur lequel l’itérateur Objet3DIterator pointe sans avoir à déréférencer le pointeur. Un désavantage est qu’il devient moins clair sur quel objet exactement les méthodes sont appelées.