Patron Visiteur

Question 1:
A) Intention et avantages

Ce patron permet de définir des nouvelles méthodes à certaines classes sans modifier ces classes et leur structure. Un des principaux avantages se définit par l'indépendance des opérations par rapport à une classe. Ceci est très utile afin d'éviter la duplication de méthodes dans plusieurs classes lorsqu'il y a de l'héritage entre les classes. La classe visiteur va donc regrouper ces méthodes/opérations.

C) Classes à modifier
Si on ajoute une nouvelle sous-classe dérivée de AbsObjet3D, il faudrait implémenter la classe en question, modifier :

- AbsObjet3DVisitor
- TP5_Tests (pour tester la classe)
- OutputVisitor
- OutputTransformVisitor
- TransformVisitor

Patron Singleton

Question 1:
A) Intention et avantages

Ce patron est utilisé lorsque l'on veut s'assurer qu'une seule instance d'une classe existe. Pour se faire, on emploie une interface afin d'effectuer les manipulations de cette classe. Cette classe singleton possède donc un mécanisme empêchant la déclaration de d'autres instances et possède une méthode qui crée une seule et unique instance de la classe.

B) Instanciation du patron Singleton
TransformStack diffère de la conception habituelle car ses méthodes sont statiques. Ceci étant dit, puisque la méthode getCurrent(void) est statique, aucune instance de la classe ne sera crée. On pourra accéder à l'état du Stack en appelant la méthode de la manière suivante: TransformStack::getCurrent(void). On devra appeler les autres méthodes de la classe avec la même syntaxe. Habituellement, les méthodes de la classe du patron Singleton ne sont pas statiques et nécessitent la création d'une classe afin de manipuler les méthodes.

C) Statefull ou Stateless

La classe TransformStack est Statefull puisque dépendamment du moment où on appelle la méthode TransformStack::pop(), on obtiendra pas le même Objet3DTransform en retour nécessairement. L'état du stack est donc important dans cette classe Singleton. Si la classe avait été Stateless, on obtiendrait le même Objet3DTransform indépendamment du moment où on aurait appelé la méthode TransformStack::pop().

D) Avantages et inconvénients attribut composite

Si on stockait la transformation d'un niveau de l'arbre comme un attribut d'objet composite, on aurait pas l'assurance qu'une seule instance est crée puisqu'il n'y aurait pas le mot-clé static attaché à cet attribut. Cependant, si on utilisait un attribut d'objet composite, on aurait un moins grand nombres de classes crées donc moins de dépendance (faible couplage).