2) Patron Visiteur -1ère partie

1) Précisez les points suivants :

a) Identifiez l'intention et les avantages du patron Visiteur.

Le patron Visiteur permet d'ajouter ou retirer des nouvelles classes sans toutefois apporter de modification aux classes des objets sur lesquelles l'opération est définie.

b) Tracer un diagramme de classes avec Enterprise Architect des différentes classes impliquées dans cette application du patron Visiteur, et ajouter des notes en UML pour indiquer les rôles, et exportez le tout en PDF.

À noter que pour la classe Objet3DIterator, seulement les définitions des méthodes agissant sur des Objets3D constants ont été ajoutées dans le diagramme bien que ces méthodes existent en plus pour des Objets3D non-constants.

c) Si en cours de conception vous constatiez que vous voudriez ajouter une nouvelle sous-classe dérivée de AbsObjet3D, établissez la liste de toutes les classes qui doivent être modifiées.

-OutputVisitor
-TransformVisitor
-OutputTransformVisitor

Il faudrait ajouter une méthode virtuelle *visit* prenant un objet de la nouvelle classe sous-dérivée pour toutes les classes mentionnées.

3) Patron Singleton

a) L'intention du patron Singleton

Le patron de Singleton est utilisé pour se restreindre à une instanciation de la classe à un objet et fournir un point d'accès global à cette instance.

b) La classe TransformStack n'est-pas une instanciation classique du patron Singleton. Expliquez en quoi cette mise en œuvre du Singleton diffère de l'approche habituelle.

Dans la structure classique d'un patron Singleton, le constructeur et destructeur sont protected au lieu d'être public. Alors nous ne pouvons pas le construire comme normal. Il y aura une fonction de "getInstance" static qui est appelé pour son "instanciation". Dans le code du travail, toutes les méthodes de TransformStack, alors ils sont tous accessible en dehors de la classe. Alors il n'y a pas eu d'appel pour son instanciation.

c) Quels sont les avantages et les inconvénients de cette approche pour la gestion des transformations ? En particulier, selon vous, cette approche est-elle « statefull » ou « stateless » tel que discuté en classe ?

Pour l'avantage, il y a le fait que celui-ci utilise moins de la mémoire comparer à la situation s'il y avait l'instanciation de la classe. Pour l'inconvénient, les données gardées en mémoire peuvent être accéder par n'importe quoi alors ça brise le principe de l'encapsulation des données privées.

Dans notre approche, celui-ci est une approche "statefull", car les informations mis dans le conteneur se retrouve dans le Stack, alors il n'y a pas la perte de mémoire à chaque fois.

d) Serait-il possible de stocker la transformation d'un niveau de l'arbre comme un attribut des objets composites ? Quels seraient les avantages et les inconvénients d'une telle approche ?

Il serait possible et plus avantageux de mettre les transformations comme attributs des objets composites, car nous travaillons avec l'objet à transformer. Le composite est avantageux, car nous utilisons plusieurs transformations sur un objet seulement. Ce qui sera désavantageux c'est qu'il faut ajouter la fonctionnalité des liens symboliques entre les feuilles du patron. Ainsi, le patron aura moins d'importance dans le programme et nous devons l'adapter avec le composite aussi.