Corrigé de l'EXAMEN
Design Pattern 2010-2011

Lundi, 13 décembre 2010, 13h45
ING-2, Génie Informatique

Durée de l'examen : 2h
Conditions d'examen : accès à tous les documents personnels (papier ou numérique en local sur leur machine)

Cet examen comporte 4 parties :

1. Quelques questions de réflexion (5 pts)
2. Quelques petits exemples à décrypter (4,5 pts)
3. Un ravalement de façade (3,5 pts)
4. L'expression numérique sous toutes ses coutures (7 pts) + 4 points de Bonus

---

1 - Quelques questions de réflexion (5 pts)

Cet exercice est un ensemble de questions de réflexion sur le cours. Certaines sont assez proches du cours d'autres moins. Les réponses ne doivent pas être des "copier/coller" mais un propos personnel de votre part pour montrer que vous avez compris. Nous noterons en conséquence.

Question 1) (1pt)

Le savoir-faire d'une classe est définie par son interface : ensemble des prototypages de toutes les méthodes publiques. On peut avoir deux classes dont l'intersection de leur interface est non vide.

En s'inspirant de la remarque précédente, expliquer l'intérêt pour une classe de définir plusieurs sous-interfaces qui sont des sous ensembles de son interface.

Solution : Une classe qui implémente autant d'interfaces a autant d'identités différentes. Le fait que les interfaces soient multiples donne le choix à d'autres classes de n'implémenter que certaine et ainsi avoir seulement les identités qui les intéressent. Cette méthode permet aussi de ne pas implémenter une grosse interface dont la majorité des méthodes peuvent être inutiles. On peut aussi parler de plus de réutilisabilité !

Question 2) (1pt)

Les deux principaux patterns créateurs sont les patterns Méthode de Fabrique et Fabrique abstraite. On note souvent une confusion entre ces deux patterns.

Quel est le lien entre ces deux patterns ?

Solution : Le pattern Fabrique Abstraite est un ensemble de Méthodes de Fabrique de classes appartenant à la même famille…

Question 3) (1pt)

Dans un développement logiciel, on essaie le plus possible de séparer la création des objets et l'utilisation de ces objets.

Quel est l'intérêt de cette séparation ?

Solution : Cette séparation est nécessaire pour réduire le coût de la maintenance et modification du code. En effet, en séparant le code de construction de la classe (méthode de fabrique, par exemple) et en utilisation de l'objet construit au travers de son interface publique on s'assure de pouvoir modifier, sans incidence de part et d'autre, sa méthode de construction ou son interface utilisation.

Question 4) (1pt)

Dans la description d'un pattern comportemental, on utilisera plus fréquemment des diagrammes de séquences à contrario d'un pattern structurel où on se servira plus de diagrammes de classes. Pourquoi ?

Solution : Dans un Pattern comportemental, il est question de proposer une solution qui exprime les comportements et interactions entre objets décrits par des classes, le diagramme de séquence est donc un bon choix de formalisme. Cependant, un Pattern Structurel est une solution qui propose plutôt une structure précise et statique de classes avec héritage et/ou association pour pallier à un problème. Le meilleur formalisme UML pour ce faire, est le diagramme de classe.

Question 5) (1pt)

Quels sont les bonnes questions à se poser pour choisir entre définir une interface ou définir une classe abstraite ?

Solution :
1) Est-ce que j'ai un ou plusieurs attributs communs à mes classes filles ? Si oui, c'est une classe abstraite le meilleur choix

2) Est-ce que j'ai une ou plusieurs méthodes dont l'implémentation est commune à toutes les classes filles ? Si oui, c'est une classe abstraite le meilleur choix.

---

2 - Quelques petits exemples à décrypter (4,5 pts)

Cet exercice est un ensemble de questions indépendantes. Dans chaque question, il s'agit soit
- de découvrir l'utilisation d'un design pattern;
- de détailler l'utilisation de ce design pattern dans le cadre de la question.

Dans tous les cas de figures, toute réponse non justifiée et non commentée sera considérée comme nulle et notée comme telle.

Question 1) (1.5pt)

Dans un programme qui permet de faire des plans d'une pièce (cuisine, salon, ...) avec ses meubles, nous avons, suivant les cas, plusieurs modes d'affichage d'un même plan :
- le plan lui même avec l'emplacement des différents meubles : le plan de base;
- le plan de base avec les mesures des meubles et de la pièce;
- le plan de base avec un devis;
- le plan de base avec un devis et les mesures.

Quel Design Pattern identifiez-vous ?

Solution : Le pattern décorateur est une bonne solution. Effectivement, On pourra considérer le PlanDeBase, comme la classe minimale et nécessaire qui pourra être décorée par Mesures et/ou le devis.

Question 2) (1.5pt)

On récupère une image et on veut la traiter en lui appliquant successivement une série de filtres. D'un traitement à un autre, les filtres et le nombre de filtres peuvent varier.

Quel Design Pattern identifiez-vous ?

Solution : Le pattern Chaîne de responsabilité est une bonne solution. Effectivement, on pourra enchainer les différents filtres (Maillons) comme voulu et avoir au bout de la chaine, le résultat attendu sur l'Image qui sera associé au premier maillon de cette chaine.

Question 3) (1.5pt)

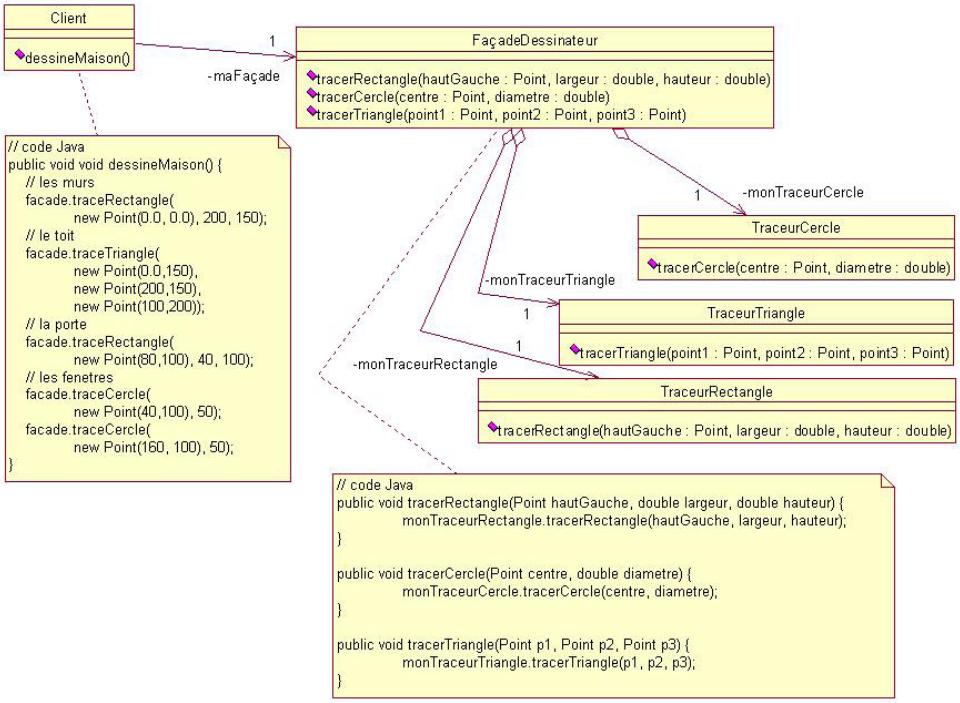
Un avatar est un être virtuel dans une animation vidéo. Suivant les bonus et malus qu'il a pu emmagasiner dans son parcours, l'avatar ne réagit pas de la même façon aux différents messages qu'on lui envoie.

Quel Design Pattern identifiez-vous ?

Solution : Le pattern Etat est la meilleure solution. Effectivement, l'avatar aura besoin de réagir différemment selon son état (décrit par nb malus ou bonus). Par conséquent, le comportement de l'avatar devra être décrit dans chq sous classe de l'Interface EtatAvatar, dont est composé l'Avatar.

---

3 - Un ravalement de façade (3,5 pts)



On donne le diagramme de classes avec des extraits de code Java du Client et de la Façade :
En respectant le patron de conception Façade, on remplace les classes TraceurCercle et TraceurRectangle par la classe SuperTraceur avec les méthodes suivantes :

- `void tracerRectangle(Point hautGauche, Point basDroite)`
- `void tracerCercle(Point centre, double rayon)`

Question 1) (1pt)

Le changement effectué a-t-il des répercussions sur le code de la classe Client ?

Solution : Non, il ne doit pas en avoir si on veut préserver le Pattern Façade. L'interface de la façade ne doit pas changer, à savoir le prototypage complet de la méthode.

Question 2) (1pt)

L'interface de la classe Façade est-elle modifiée ?

Solution : Non, Il ne doit pas en avoir si on veut préserver le Pattern Façade. (voir la réponse à la question précédente)

Question 3) (1,5 pt)

Donner le nouveau diagramme de classes en précisant en *note* les changements dans le code Java

Solution : Les changements sont les suivants :
1) la façade ne sera plus associée aux classes `TraceurCercle` *et* `TraceurRectangle` *mais plutôt à* `SuperTraceur`*.*
2) L'implémentation des méthodes `tracerCercle( centre,diamètre)` *et de* `tracerRectangle (point, largeur, hauteur)` *changera en conséquence pour appeler correctement les nouvelles méthodes de la nouvelle classe associée.*
Le code des méthodes :

```
Void tracerCercle( Point centre, Double diamètre)
{
      monSuperTraceur.tracerCercle(point, diamètre/2);
}

Void TracerRectangle (Point pointHGauche,
                      Double largeur, Double hauteur)
{
      MonSuperTraceur.tracerRectangle(pointHGauche,
                  new Point(pointHGauche.x + largeur,
                  pointHGauche.y + hauteur)) ;
}
```

---

4 - L'expression numérique sous toutes ses coutures (7 pts)

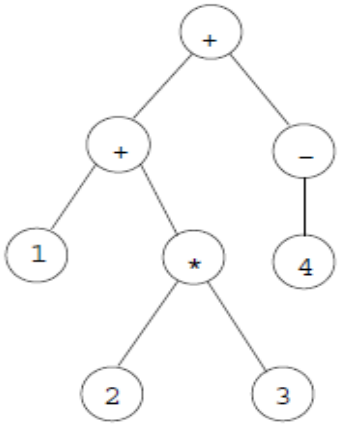
On souhaite développer un ensemble de classes permettant de calculer et d'afficher des expressions mathématiques. On se restreindra ici aux opérations habituelles de l'arithmétique sur les entiers à savoir l'addition, la multiplication, l'inversion du signe d'un entier.
Prenons un exemple d'expression mathématique :

1 + (2 * 3) + (- 4)

On remarque tout d'abord que les constituants de l'expression peuvent être :
- des opérateurs binaires (qui prennent deux opérandes), comme + et * ;

- des opérateurs unaires comme - ;
- des constantes qui sont des entiers.

Pour représenter une expression mathématique, on peut utiliser une structure d'arbre. Dans notre problématique de représentation d'expressions, chaque nœud de l'arbre est soit un entier, soit un opérateur. Chaque nœud représentant un opérateur possède un nombre de nœuds fils égal à son arité1. Par exemple, l'expression 1 + (2 * 3) + (- 4) peut être représentée par l'arbre représenté suivant :



Question 1) (1,5 pt)

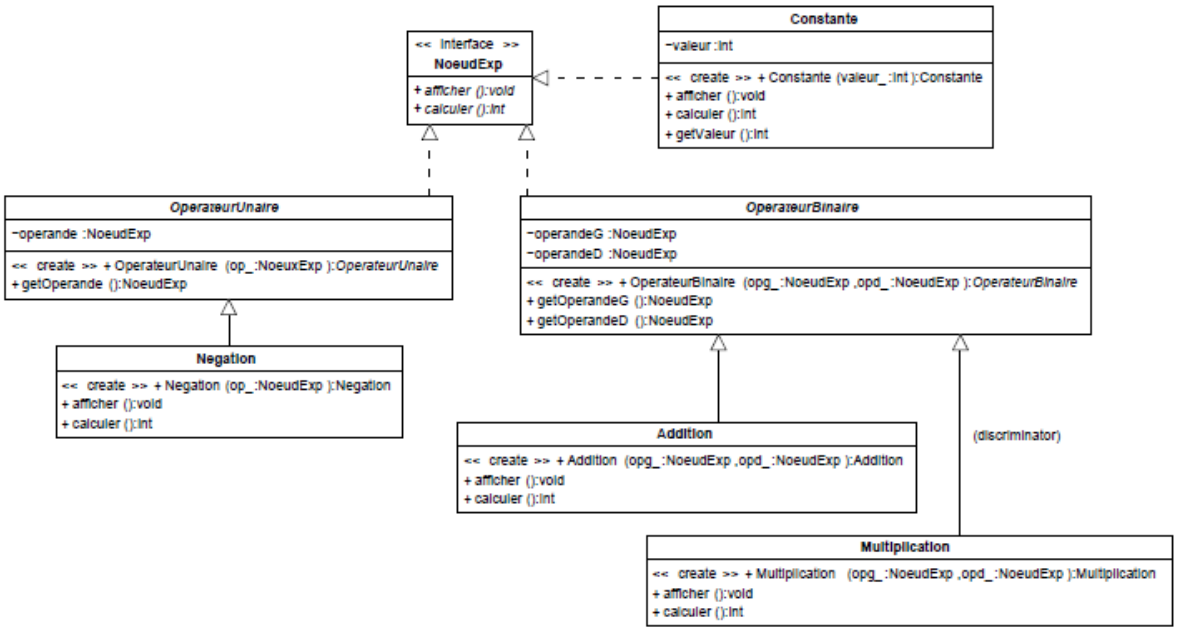
Proposer un diagramme de classes simple mais complet permettant de représenter les différents nœuds que l'on peut représenter. On introduira les opérations nécessaires à l'affichage et au calcul des expressions et on veillera à construire un diagramme suffisamment générique (via l'héritage ou la réalisation d'interfaces). On fera ainsi apparaître les différents types d'opérations (unaire ou binaire), l'opération de multiplication, d'addition et de « négation » et les constantes ainsi qu'un type de nœud générique.

Solution : Le design pattern utilisé sera le composite pour représenter la structure arborescente de la structure numérique. On aurait pu aussi utiliser interpréteur !

Attention, sur votre solution, on doit aussi voir les liens d'association entre chaque opérateur (Classe OpérateurBinaire) et ses opérandes (OperandeG, OpérandeD) de type NoeudExp. En effet, on ne devrait pas voir dans un diagramme UML des attributs dont le type existe sur le même diagramme de classe). Même remarque pour l'association entre OpérateurUnaire et NoeudExp.

De plus, il important de spécifier les classes abstraites avec « abstract » (Italique sur le diagramme) : OpérateurUnaire, OpérateurBinaire

[1] En mathématiques, l'**arité** d'une fonction, ou opération, est le nombre d'arguments ou d'opérandes qu'elle requiert (Wikipédia)



Question 2) (1pt)

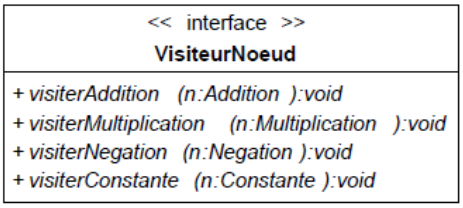
Quelles critiques peut-on émettre sur ce modèle ? Vous réfléchirez en particulier à l'ajout d'une nouvelle opération (autre que le calcul et l'affichage) sur les expressions ;

Solution : Effectivement, le modèle proposé est difficile à faire évoluer si on décide de réécrire une opération ou si on décide de l'ajouter. A chaque fois, il faudra revenir dans toutes les classes filles pour réécrire une méthode existante ou pour implémenter la nouvelle opération.

Question 3) (1pt)

Pour pallier les problèmes évoqués dans la question précédente, on décide d'appliquer un patron de conception particulier, le visiteur. L'idée de ce patron est d'encapsuler une opération dans un objet appelé visiteur et de passer cet objet aux nœuds de l'arbre. Lorsqu'un nœud accepte un visiteur pour une opération particulière, il appelle une méthode du visiteur correspondant à son type et se passe en paramètre de cette méthode.

On les nommera par convention `visiterMultiplication`, `visiterAddition`, `visiterNegation`, `visiterConstante`. On dispose ainsi des méthodes pour chaque type de nœud. Comme tous les visiteurs devront disposer de ces méthodes, on peut créer une interface `VisiteurNoeud` les possédant comme suit :

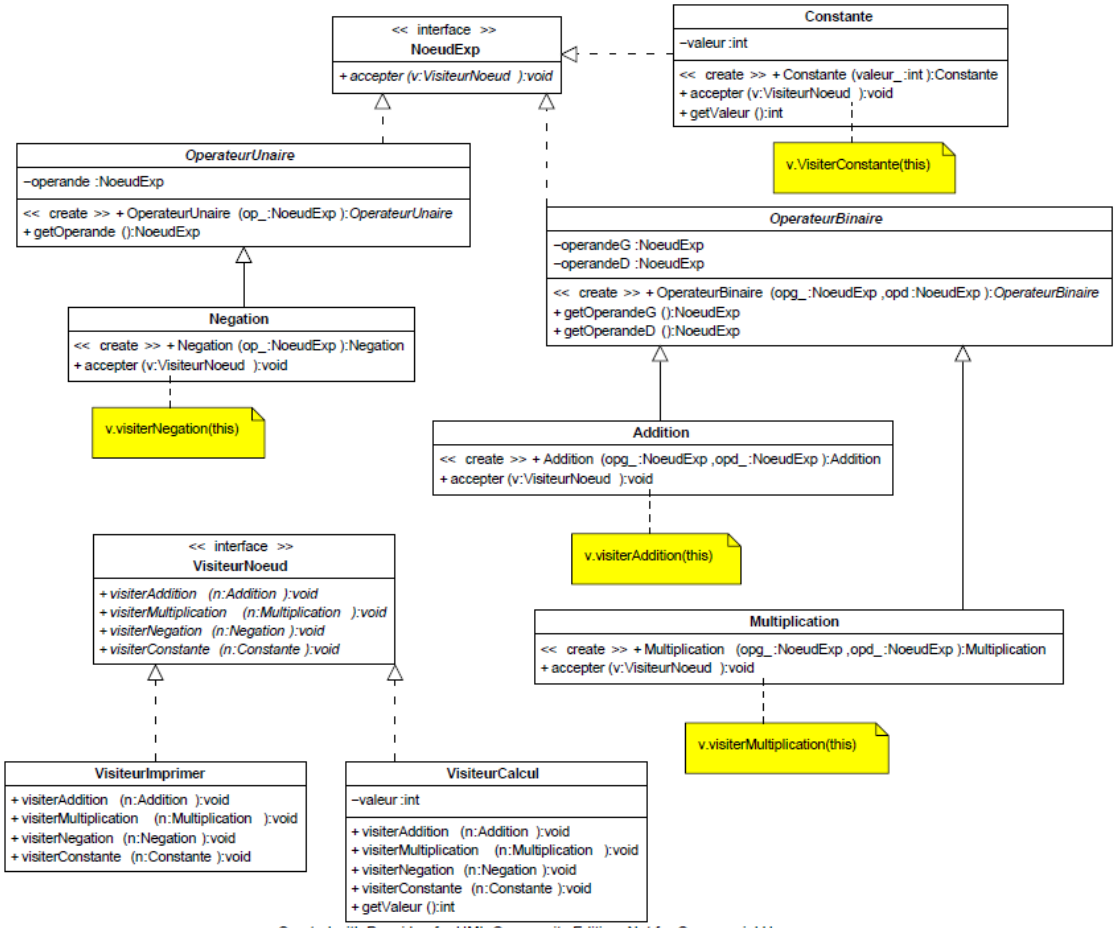


Les nœuds n'auront plus à proposer les différentes opérations qui peuvent s'effectuer sur eux, celles-ci seront « stockées » dans un visiteur particulier.

Modifiez les classes que vous aviez proposées en question 1 et ajoutez les deux classes de visiteurs nécessaires à la réalisation des opérations d'affichage et de calcul de l'expression (attention pour cette dernière, il faut trouver un moyen de conserver la valeur de l'expression) ;

Solution : Attention, sur votre solution, on doit aussi voir les liens d'association entre chaque opérateur (Classe OpérateurBinaire) et ses opérandes (OperandeG, OpérandeD) de type NoeudExp. En effet, on ne devrait pas voir dans un diagramme UML des attributs dont le type existe sur le même diagramme de classe). Même remarque pour l'association entre OpérateurUnaire et NoeudExp.

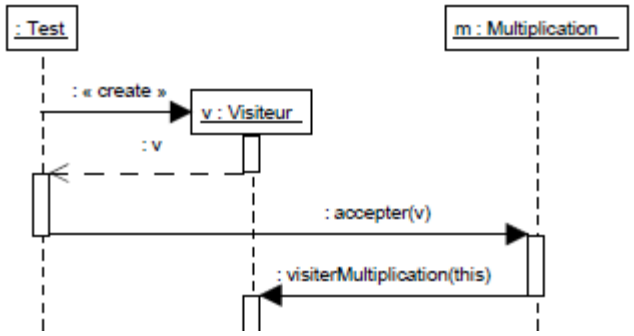
De plus, il important de spécifier les classes abstraites avec « abstract » (Italique sur le diagramme) : OpérateurUnaire, OpérateurBinaire



Question 4) (1pt)

Montrer à l'aide d'un diagramme de séquence l'utilisation du `VisiteurCalcul` sur un nœud de type Multiplication

Solution :



Question 5) (1pt)

Ecrire un programme de test construisant l'expression **1 + (2 * 3) + (- 4)**, la calculant, puis l'affichant ;

Solution :

```
public class TestVisiteur {
    public static void main(String[] args) {

        Addition exp = new Addition(new Addition(new Constante(1),
                                                 new Multiplication(
                                                                    new Constante(2),
                                                                    new Constante(3))),
                                    new Negation(new Constante(4)));

        VisiteurNoeud v1 = new VisiteurAfficher();
        exp.accepter(v1);

        VisiteurCalcul v2 = new VisiteurCalcul();
        exp.accepter(v2);
        System.out.println("Valeur de l'expression : " + v2.getValeur());
    } // end of main()
}
```

Question 6) (0,5 pt)

On souhaite introduire une opération qui peut lever une exception. Est-ce possible avec la solution développée précédemment ?

Solution : C'est possible seulement si on réécrit tous les prototypes des méthodes du Visiteur pour déclarer la levée de l'exception et pareillement pour toutes les méthodes qui accepte ces visiteurs. Donc ca ne sera pas une modification triviale. ☹.

Question 7) (1pt)

Pour une expression numérique sélectionnée, le logiciel propose de calculer plusieurs statistiques possibles.

- Le nombre de nœuds de l'arbre binaire ;
- Le nombre d'opérandes de type « variable » ;
- Le nombre d'opérandes de type « constante ».

L'utilisateur sélectionne les statistiques qu'ils désirent ou annulent celles qu'ils ne désirent plus. A chaque modification de l'expression, ces statistiques sélectionnées sont calculées dans un objet Statistique associé à l'expression numérique.

Que proposez-vous d'utiliser pour mettre en place la gestion de cet objet Statistique associé à une expression ? Vous devez envisager que d'autres statistiques puissent être créées à terme.

Solution : Le pattern le plus adéquat serait Observateur. Avec le NoeudExp comme sujet Observé et une Classe Statistique comme classe Observateur. On pourra ainsi observer n'importe quel nœud et y ajouter autant de types de statiques possibles (Statistique1, Statistique2) héritant de Statistique.

FIN DE L'EXAMEN