EXAMEN MASTER 1 INFORMATIQUE

Première session

MODULE ACO

Lundi 12 décembre 2011
Durée : 2 heures
Tous documents autorisés

Description du sujet

Le sujet comporte deux parties :

1. un questionnaire, comportant des questions à choix multiples, où vous devez indiquer une ou plusieurs bonnes réponses ;
2. un exercice de conception et de développement à objet à réaliser en utilisant la notation UML et le langage Java.

Le barème est donné à titre indicatif et n'est pas définitif. Il est conseillé de commencer par le questionnaire.

Questionnaire (7 points)

Vous indiquerez vos réponses sur votre copie anonymée sous la forme n° de question- n° du choix correct. Exemple : si pour la question *l* la possibilité correcte est la réponse *a* vous indiquerez *1–a* sur votre copie d'examen. Certaines questions comportant plusieurs possibilités correctes, il faut les indiquer *toutes* pour que la réponse à la question soit considérée globalement correcte, par exemple : *2–cd*. Barème indicatif :

– une réponse globale correcte à une question vaut 1 point ;

– une réponse incorrecte à une question vaut –0,5 points ;

– l'absence de réponse vaut 0 points.

Questions

1. On peut réaliser les tests d'intégration
   a. *avant de commencer les tests unitaires*
   b. *après avoir fini tous les test unitaires*
   c. *au fur et à mesure de l'achèvement des tests unitaires*
   d. *seulement lorsque l'application sous test est installée et intégrée chez le client*

2. Parmi les fragments de code Java ci-après, indiquer ceux qui sont interdits par le langage Java, I1, I2 et I3 étant des interfaces, et C1 et C2 des classes concrètes
   a. *interface I4 extends I1, I2*
   b. *class C3 implements I4*
   c. *class C4 extends C3 implements I3*
   d. *class C5 extends C3*

3. Les méthodes au sens UML sont mises en œuvre dans
   a. *les interfaces*
   b. *les classes partiellement abstraites*
   c. *les classes concrètes*
   d. *les instances*

4. Indiquer la mise en œuvre la plus sûre et générale pour une séquence de chaîne de caractères
   a. *Collection l = new ArrayList();*
   b. *List<String> l = new List<String>();*
   c. *ArrayList<String> l = new ArrayList<String>();*
   d. *List<String> l = new ArrayList<String>();*

5. Dans un diagramme de conception, quand doit-on rendre une association entre A et B navigable de A vers B ?
   a. *on doit toujours le faire*
   b. *on doit le faire seulement si A peut envoyer des messages à B*
   c. *on doit le faire seulement si B peut envoyer des messages à A*
   d. *on doit le faire seulement si B est une composition de plusieurs A*

6. Lors de l'application d'un patron de conception, un type (classe ou interface) peut en théorie occuper plusieurs rôles dans un diagramme de classe
   a. *oui*
   b. *non*

7. Un invariant de classe s'applique à une instance de cette classe
   a. *seulement durant la phase d'initialisation de l'instance*
   b. *en permanence sauf pendant la phase d'initialisation de l'instance*
   c. *seulement pendant l'exécution d'une méthode de l'instance*
   d. *en permanence àprès initialisation, sauf pendant l'exécution d'une méthode de l'instance*

Conception (13 points)

On considère un système permettant de connaître à tout instant la configuration matérielle d'un ordinateur. Cette configuration est accessible au moyen d'un opération Java prédéfinie. Elle est représentée par un arbre dont les éléments sont typés. Toute configuration obéit à la grammaire abstraite donnée ci-dessous, exprimée dans un métalangage vu en travaux dirigés.

Grammaire abstraite d'une configuration

Configuration ::= procs : Processeur; memoire : Memoire*; graphique : CarteGraphique* ; réseau : CarteRéseau* ; alim : Alimentation*

Processeur ::= coeurs : CoeurProc; id : String*

Memoire ::= BancMemoire | Disque

BancMemoire ::= id : String; capacité : Integer

Disque ::= id : String; capacité : Integer; puissance : Integer

CarteGraphique ::= puissance : Integer; capacitéMémoire : Integer

CarteRéseau ::= débit : Integer

Alimentation ::= puissance : Integer

CoeurProc : puissance : Integer; mips : Integer

On rappelle que dans ce type de grammaire, l'astérisque (*) derrière un nom de type indique qu'il peut y avoir un nombre quelconque d'éléments de ce type.

L'accès à la configuration courante est possible grâce à la classe suivante :

```
class PrimitivesSysteme {
static Configuration getConfigurationActuelle(); // Prédéfinie
}
```

Question 1. Définir le modèle UML représentant les mêmes informations que la grammaire abstraite ci-dessus.

Impression d'une configuration

On veut pouvoir imprimer une configuration sous forme d'un texte encodé selon une syntaxe concrète fondée sur des balises XML selon l'exemple ci-dessous :

```
<configuration>
<processeur id="1">
<coeurproc puissance="10" mips="30" />
<coeurproc puissance="20" mips="40" />
</processeur>
<bancmemoire id="1" capacité="1000" />
...
</configuration>
```

Question 2. Modifier le diagramme UML représentant la structure d'une configuration, pour y insérer le patron de conception Visitor, en respectant les conventions vues en cours et en définissant en UML un visiteur concret ImprimerConfiguration.

Question 3. Donner le code Java de la mise en œuvre de l'ensemble des classes employées dans l'application de Visitor définie à la question précédente, en n'oubliant pas de donner le code Java de ImprimerConfiguration et des interfaces et autres rôles de l'application du patron.

Contrôle de validité

On veut maintenant mettre en œuvre un système de vérification des contraintes sur la configuration matérielle. Pour cela on va utiliser le patron de conception Visitor déjà appliqué pour l'impression.

Pour chaque contrainte sur la configuration on va définir un visiteur concret qui devra rendre une valeur booléenne, en rendant vrai si et seulement si la contrainte qu'il est chargé de vérifier est satisfaite par la configuration parcourue passée en paramètre. Pour cela on définit une interface Vérificateur étendant le visiteur et définissant une opération nommée Vérificateur::contrainteSatisfaite(c:Configuration):Boolean.

Question 4. Modifier le diagramme UML pour y intégrer Vérificateur et un visiteur concret VérifierPuissance dont l'opération contrainteSatisfaite(c:Configuration):Boolean rend vrai si et seulement la puissance de l'alimentation est supérieure ou égale à 120 % de la puissance totale requise par la configuration. La puissance électrique de chaque élément de configuration est donnée par le sous-élément puissance (voir la syntaxe abstraite et diagramme UML qui en découle). Si un type d'élément n'a pas de sous-élément puissance alors la puissance qu'il consomme est considérée comme nulle.

Question 5. Donner le code Java correspondant au système de vérification de puissance mentionnée ci-dessus.

Exécution automatique

On veut maintenant mettre en place une exécution automatique de la vérification des contraintes, lorsque la configuration change.

Question 6. Indiquer quel patron de conception est approprié, et pourquoi. Préciser l'attribution des rôles.

Question 7. Donner la mise en œuvre en UML de l'application de ce patron de conception à l'architecture définie dans les questions précédentes, par des diagrammes statiques et dynamiques, en précisant bien quand et comment la visite est déclenchée.

Question 8. Donner une mise en œuvre en Java de l'application de ce patron.