EXAMEN MASTER 1 INFORMATIQUE

Première session

UNITÉ D'ENSEIGNEMENT ACO

Lundi 16 décembre 2013
Durée : 2 heures
Tous documents autorisés

ENGLISH TRANSLATION STARTS PAGE 4

Description du sujet

Le sujet comporte deux parties :

1. un questionnaire, comportant des questions à réponse ouvertes courtes ;
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 (6 points)

Vous indiquerez vos réponses sur votre copie anonymée sous la forme n° de question suivie d'une réponse courte *sur 3 lignes maximum*.

Barème indicatif :

– une réponse globale correcte à une question vaut 2 point ;
– une réponse incorrecte à une question vaut -1 points ;
– l'absence de réponse vaut 0 points.

Questions

A. En quoi les classe immuables (en anglais : immutable) contribuent-elles au principe d'encapsulation ?

B. Quels sont les désavantages de l'héritage de méthode ?

C. Comment gérer les préconditions d'opération en Java ?

Conception (14 points)

On considère la mise en œuvre d'une application de type tableur, selon les définitions suivantes :

– une feuille de calcul est représentée par une interface Feuille et une classe d'implémentation FeuilleImpl ;
– une feuille est considérée comme une matrice à deux dimensions, la dimension des lignes et celle des colonnes ;
– une feuille comporte des éléments de la classe Cellule ;
– chaque cellule comporte une valeur et une expression, la valeur étant obtenue par l'évaluation de l'expression ;
– une expression peut être soit une constante de type nombre à virgule flottante (classe Constante), soit une opération d'addition comportant deux expressions (classe Addition), soit une référence à une autre cellule (classe Référence) ;
– les classes Constante, Addition et Référence implémentent l'interface Expression.

Déclarations Java (incomplètes)

```
public interface Feuille {
        public Cellule getCellule(int numLigne, int numColonne);
        public int getNbLignes();
        public int getNbColonnes();
        public void recalculer(); // Provoque le recalcul
                // de la valeur de chaque cellule de la feuille
}

public class Cellule {
        private float valeur;
        public float getValeur() { ???? }
        public void setValeur(float f) { ???? }
        private Expression expression;
        public Expression getExpression();
        public void setExpression(Expression e);

public interface Expression {
        // Evaluation de l'expression dans le contexte de la                    //
cellule c , retourne la valeur de l'expression
        public float évaluer(Cellule c);
}
public class Constante implements Expression {
        public Constante(float valeur) { ???? }
        ????
}
public class Addition implements Expression {
        public Addition (Expression opérandeGauche,
                Expression opérandeDroit) { ???? }
        ????
```

```
}
public class Référence implements Expression {
        public Référence(Cellule c) { ???? }
        ????
}
```

Question 1. Donner un diagramme UML élémentaire signifiant la même chose que les déclarations Java ci-dessus.

Mécanisme d'évaluation

Pour calculer la valeur de chaque cellule de la feuille de calcul, il faut réaliser l'évaluation des expressions pour chacune des cellules de la feuille.

Question 2. À partir des éléments Java ci-dessus et des définitions données au début de la section de conception, donner les déclarations et le code Java complets, et remplacer les ???? par du code Java approprié. Donner en particulier les méthodes de l'opération évaluer pour les classes Constante, Addition, Référence et Feuille. On supposera qu'il n'y a pas de cycle dans le graphe des références.

Question 3. Construire un diagramme d'objets représentant une feuille de calcul F constituée de quatre cellules. La cellule de coordonnées (1,1) contient la constante 10, la cellule de coordonnées (1,2) la constante 20, la cellule de coordonnées (2,1) l'addition des constantes 30 et 40 et la cellule de coordonnées (2,2) la référence à la cellule de coordonnées (2,1).

Question 4. Construire un diagramme de séquence montrant le déroulement du recalcul pour la feuille de calcul F donnée ci-dessus.

Emploi de Observer

La technique adoptée jusqu'ici recalcule les valeurs des cellules même lorsque leur valeur ne dépend pas d'autres cellules. Pour diminuer le coût du recalcul vous allez mettre en œuvre le patron de conception *Observer*. Une cellule c1 sera sujet d'une autre cellule c2 si l'expression indiquée dans c2 fait référence à c1. Par exemple, dans la feuille F donnée plus haut, la cellule de coordonnées (2,1) est sujet de la cellule de coordonnées (2,2) car l'expression dans cette dernière fait référence à la cellule (2,1). Grâce à ce principe, la modification de (2,1) provoque son recalcul et le changement de sa valeur, puis l'appel d'une opération update sur la cellule (2,2). Celle-ci se recalcule alors. On limite ainsi le nombre d'opérations de recalcul.

Question 5. Compléter le diagramme de classe donné à la question 1 en montrant l'application du patron de conception *Observer*. Vous pouvez compléter directement votre diagramme de la question 1, en utilisant une autre couleur (sauf le rouge) pour montrer les ajouts.

Question 6. En utilisant comme exemple la feuille F donnée plus haut, donner un diagramme de séquence montrant comment l'opération update() du *subject* est appelée et quelle est la cascade des appels entraînant le recalcul.

Question 7. Donner le code Java de mise en œuvre de *Observer* et de la nouvelle technique d'évaluation de la feuille de calcul.

Application de Visitor

Il faut maintenant rajouter d'autres traitements sur la feuille de calcul, comme par exemple l'impression des valeurs d'une feuille dans un fichier.

Question 8. Indiquer en quoi le patron *Visitor* peut être utile.

Question 9. Proposer une modification de l'architecture vue jusqu'ici pour y insérer le patron de conception *Visitor*, avec une classe *Evaluateur* jouant le rôle de *ConcreteVisitor*.

Question 10. Donner le code Java d'une mise en œuvre de Visitor dans le cas de l'application tableur de ce sujet, en traitant le cas du recalcul (operation recompute()).

ENGLISH TRANSLATION

Introduction

This exam paper has two parts:

1. a quizz, with short answer questions;
2. an exercise on object-oriented design and programming, which uses the UML and Java.

The point count is not final. You should probably start with the quizz.

Quizz (6 points)

You will answer by refering to the question letter below, and you will use a maximum of three lines to answer each question of the quizz.

Marking system:

– a correct answer for one quizz question gives 2 points;

– an incorrect answer for one quizz question removes 1 point;

– no answer, 0 points.

Questions

A. How do immutable classes contribute to encapsulation?

B. What are the disadvantages of method inheritance?

C. What measures will you use to manage operation preconditions in your code?

Design and code (14 points)

You will design a very simple spreadshhet software, using the following requirements:

– a spreadsheet is described by a Sheet interface and a SheetImpl concrete class;

– a spreadsheet contains a matrix of cells, each cell being described by a Cell class ;

– each cell of a sheet has a row number and a column number;

– each cell has a value and an expression, the value being defined by the computation of the expression;

– an expression is defined by a Expression interface, and there are three possible implementations for an expression: a floating point constant defined by a Constant class; an

add operation made of two expressions, defined by an Addition class; a reference expression, which refers to another cell, defined by a Reference class.

Partial Java definitions

```
public interface Sheet {
        public Cell getCell(int lineNumber, int columnNumber);
        public int getLineCount();
        public int getColumnCount();
        // Triggers the recomputation of each cell
        public void recompute();

public class Cell {
        private float value;
        public float getValue() { ???? }
        public void setValue(float f) { ???? }
        private Expression expression;
        public Expression getExpression();
        public void setExpression(Expression e);

public interface Expression {
        // Evaluate the expression in the context of cell c
        public float evaluate(Cell c);
}
public class Constant implements Expression {
        public Constant(float value) { ???? }
        ????
}
public class Addition implements Expression {
        public Addition (Expression leftOp,
                Expression rightOp) { ???? }
        ????
}
public class Reference implements Expression {
        public Reference(Cell c) { ???? }
        ????
}
```

Question 1. Provide a UML class diagram corresponding to the Java declarations given above.

First algorithm for sheet computation

In this very simple implementation, the computation of the sheet is made by evaluating each cell's expression and setting the cell value accordingly.

Question 2. Provide an implementation of this simple algorithm by provide additional code for the partial Java declarations given above, and replacing the ???? with appropriate code. Do not forget to provide methods for the evaluate() operation in classes Constant, Addition and Reference. You will assume that there are no cycles in the cells' dependency graph.

Question 3. Provide a UML object diagram for a sheet F made of four cells. The cell at coordinates (1,1) holds constant expression 10, the cell at coordinates (1,2) holds constant 20, the cell at coordinates (2,1) holds an addition of constants 30 et 40, finally the cell at coordinates (2,2) holds a reference expression refering to the cell at coordinates (2,1).

Question 4. Provide a sequence diagram showing the recompute process for sheet F.

Using Observer

The approach taken until now is to recalculate the values of all cells, including those who do not depend on any other cell value. To avoid the cost of these unnecessary recalculations, you will apply the Observer Design Pattern. Cell c1 will be the subject of cell c2 if and only if the expression for c2 references c1. For example, in sheet F described above cell (2,1) is a subject of cell (2,2), because the latter's expression references cell (2,1). As a result of this pattern, any change to (2,1) will cause its own recalculation, followed by a call of its update() method, which in turn will provoke a recalculation of (2,2). In this manner recalculations are kept to a minimum.

Question 5. Provide an extension to your class diagram to insert the *Observer* design pattern in it. To save time you can draw on your existing diagram using another ink color (but not red).

Question 6. Using the F sheet defined above provide a sequence diagram that explains how the recomputation is done using *Observer*.

Question 7. Provide a Java implementation of the *Observer* pattern applied to the recompute() implementation.

Applying Visitor

There are many processing possibilities on a spreadsheet, for instance print it in a file, check for circular dependencies, etc.

Question 8. Explain why the *Visitor* design pattern is useful here.

Question 9. Define an extension of your architecture that relies on the instanciation of the *Visitor* design pattern.

Question 10. Provide a Java implementation of this *Visitor* instantation in the case of sheet recomputation.