Examen en "Réutilisation et Composants" - FMIN304 M2 Informatique (Master IFPRU).

Christophe Dony, Chouki Tibermacine

Janvier 2010

Durée : 2h. Tous documents autorisés.

La précision et la concision des réponses sont notées, ainsi que la lisibilité. Dans tous les cas il est important de présenter du code extensible et réutilisable utilisant les techniques connues pour cela. Il est important de lire les sections 1 et 2 dans l'ordre mais les sections sont indépendantes du point de vue des questions posées.

1 DP et Réutilisation (environ 4 points)

Considérons la hiérarchie des éléments de stockage (Directory, File, Link), respectant le schéma *Composite*, et dotée d'un mécanisme de visite, selon le schéma *Visiteur*, étudiée dans le TD/TP no 1. Vous avez réalisé en TP plusieurs visiteurs dont `RazVisitor` et `CountVisitor`.

Questions

1. Soit le code suivant de la classe `Visitor`

```
public abstract class Visitor {
    public void visitFile(File f){}
    public void visitDirectory(Directory f){}
    public void visitLink(Link f){}
}
```

et soit le code (incomplet) suivant de la classe `Directory` dans lequel l'attribut `elements` sert à stocker les éléments contenus dans un directory (répertoire) :

```
public class Directory extends ElementStockage{
    protected Collection<ElementStockage> elements ;
    ...
}
```

Donnez dans ce contexte le code Java, le plus court possible, d'une classe `CountVisitor` qui permet de compter le nombre total (récursivement) de sous-répertoires et de fichiers (uniquement, les liens ne doivent pas être comptés) contenus dans un répertoire donné, comme dans l'exemple ci-dessous.

```
Directory d = new Directory("UnProgramme") ;
Directory d2 = new Directory("src");
Directory d3 = new Directory("bin");
d.add(d2);
d.add(d3);
d2.add(new File("F1.java","...")) ;
d2.add(new File("F2.java","...")) ;
d3.add(new File("F1.class","...")) ;
d3.add(new File("F2.class","...")) ;
d3.add(new Link("sources", d2));
CountVisitor c = new CountVisitor();
d.accept(c);
System.out.println(c.getCount()); --> imprime 7
```

2. Quel intérêt y a-t-il, dans la classe `Visitor`, à avoir défini les méthodes avec un corps vide plutôt que de les avoir déclarées abstraites.
3. Dans *wikipedia*, il est écrit à propos du schéma "visiteur" :

> "Aside from potentially improving separation of concerns, the visitor pattern has an additional advantage over simply calling a polymorphic method : a visitor object can have state. This is extremely useful in many cases where the action performed on the object depends on previous such actions."

Utilisez cet exemple de `CountVisitor` pour illustrer en trois phrases courtes les trois termes suivants : *separation of concerns*, *calling a polymorphic method*, *visitor oject can have state*.

2 DP et Réutilisation - suite - Environ 5 points

Soit à réaliser un visiteur (`PrettyPrintVisitor` capable d'afficher au terminal le contenu d'un répertoire **avec indentation**. Par exemple, pour les objets donnés en section 1, le code "`d.accept(new PrettyPrintVisitor());`" imprime ce qui suit :

```
Directory UnProgramme
    Directory src
        File F1.java
        File F2.java
    Directory bin
        File F1.class
        File F2.class
        Link sources
```

Questions

1. Le schéma visiteur dans sa version standard ne permet pas de réaliser cette fonctionnalité. Pour remédier à ce problème, on propose ci-dessous la modification suivante de la classe `Visitor`. Expliquez quel était le problème et pourquoi cette nouvelle version permet de le résoudre.

   ```
   public abstract class Visitor {
       public void visitFile(File f){}
       public void visitBeforeDirectory(Directory f){}
       public void visitAfterDirectory(Directory f) {}
       public void visitLink(Link f) {};
   ```

2. Donnez en Java la nouvelle version de la méthode `accept(Visitor v)` de la classe `Directory` compatible avec cette nouvelle version de la classe `Visitor`.
3. Soit un attribut `int indent` destiné à stocker le niveau d'indentation courant et soit la méthode ci-dessous, tous deux définis sur la classe `PrettyPrintVisitor`.

   ```
   public void printIndent(){
       for (int i = 0; i < indent; i = i+1)
           System.out.print(" ");
   }
   ```

   Donnez le code Java de `PrettyPrintVisitor`.

3 Composants JavaBeans et Aspects - Environ 6 points

1. Donnez en Java le code d'une classe `TrackablePoint` définissant des points classiques mais en plus compatibles avec l'environnement *JavaBeans*, qui pourront être disponible sur la palette, comme dans le TP NetBeans . Les `TrackablePoint` seront des *beans* de type "écouté", les propriétés liées seront les propriétés `X` et `Y` représentant l'abcisse et l'ordonnée du point. Ne définissez que ce qui est important du point de vue de la question, ne perdez pas de temps avec les détails (constructeurs par exemple).
2. Définissez en **AspectJ** un aspect `PointTrace` tel que le message `''un point va être modifié''` soit affiché au terminal à chaque fois que l'abcisse d'un `TrackablePoint` va être modifiée.
3. Seconde version avec paramètres : définissez un aspect `PointTrace`, avec un *advice* de type *around*, tel que le code suivant :

   ```
   public static void main (String[] args){
       TrackablePoint p1 = new TrackablePoint();
       System.out.println("Begin ---");
       p1.setX(33);
       p1.setY(44);
   }
   ```

   produise l'affichage ci-desous. On suppose que la méthode `toString()` existe sur `TrackablePoint`, inutile de l'écrire.

   ```
   Begin ---
   (0, 0) va être modifié --- 33
   (33, 0) a été modifié ---
   (33, 0) va être modifié --- 44
   (33, 44) a été modifié ---
   ```

4 Composants EJB - Environ 5 points

Questions

1. Donnez en Java le code définissant un EJB `DoFAct`, capable de calculer la factorielle d'un entier n, et utilisable en environnement JEE. Vous indiquerez quel type de composant vous choisissez.
2. Donnez le code d'un client, soit web (html+jsp) soit lourd (application java) utilisant l'EJB précédent.