M1 Informatique - HAI818I - "Modularité et Réutilisation"

Année 2021-2022. Semestre pair. Session 1, mai 2022.

Christophe Dony, Chouki Tibermacine

Durée : 2h00. Les documents ne sont pas autorisés ; quand du code est demandé, la syntaxe exacte du langage utilisé n'est pas exigée mais la logique de l'écriture de code doit être présente dans les réponses.

La précision et la concision des réponses ainsi que la lisibilité des codes sont notées. Rédiger les réponses aux parties A et B de façon séparée (feuilles différentes).

PARTIE A (Environ 10 points - constituée de 3 sous-sections)

1 Limites de l'approche objet : couplage

Les langages à objets sont sous-jacents de la plupart des framework actuels pour le développement logiciel. La construction des architectures extensibles des framework est rendu possible par l'affectation polymorphique combinée à l'envoi de message (ou appel de méthode) avec liaison dynamique. Les langages à objet historiques, issus de la fin du 20ième siécle ont quelques limites, que les systèmes actuels visent à combler par divers moyens.

Considérons le listing 1.

```
2  interface IB {public mb();}

4  class A{
5        IB b;
6        public void setB(IB b) {this.b = b;}
7        public int ma() {return (1 + b.mb());}

9  class B1 implements IB {
10       public int mb() {return(2); }}

12 class B2 implements IB {
13       public int mb() {return(3); }}

15 class Application{
16       public static void main (String[] args) {
17         a.setB(new B2());
18         a.ma();} }
```

Listing 1 – programmation et couplage

Questions

1. dans le `main` que rend l'évaluation de l'expression a.ma()
2. Expliquez pourquoi le couplage fort est un problème
3. Expliquez pourquoi les classes A et (B1 ou B2) sont faiblement couplées.
4. Indiquez quels sont les techniques qui ont été utilisées ici pour réaliser le couplage faible, et celles qui auraient pu l'être mais qui ne l'ont pas été.
5. Discutez l'affirmation suivante : "Dans l'exemple, dans la méthode `setB` de `A`, quel que soit l'argument b, l'instruction this.b = b ne peut pas être une affectation polymorphique car le type statique de this.b est le même que le type statique de b."

2 Limites de l'approche objet : extensibilité

Le découpage modulaire en classes promu par les langages à objet rend aisé l'ajoût d'une nouvelles classes à une application existante mais ne facilite pas l'ajout d'une nouvelle fonctionnalité globale.

On a vu à ce propos dans le TD-TP 1 le schéma *Visiteur* appliqué à des classes représentant les éléments de stockage. On s'intéresse pour les énoncés de cette question uniquement aux dossiers et aux fichiers (on ignore les liens) ; les éléments de stockage y sont équipés d'un mécanisme de visite selon ce schéma *Visiteur*. On considère le code ci-dessous définissant dans ce contexte la classe `Directory` réprésentant les dossiers.

```
2  public class Directory extends ElementStockage {
3    protected Collection<ElementStockage> elements;
4    public Directory(String nom) {
5      super(nom,4) ;
6      elements = new ArrayList<ElementStockage>() ; }
7    public void accept(Visitor v) {
8      v.visitDirectory(this);
9      for (ElementStockage s : elements) { s.accept(v); }
10 }

12 public class Visitor {
13   public void visitFile(File f){}
14   public void visitDirectory(Directory f) {}
15 }

17 public class CountVisitor extends Visitor {
18   protected int count;
19   public CountVisitor() { count = 0;}
20   public int getCount() { return count; }
21   public void visitDirectory(Directory d) { count = count + 1; } }

23 public class TestCountVisitor{
24   public static void main(String[] args) {
25     Directory d = new Directory("UnProgramme") ;
26     Directory d2 = new Directory("src1");
27     Directory d3 = new Directory("bin");
28     d.add(d2);
29     d.add(d3);
30     d2.add(new File("F1.java","....")) ;
31     d2.add(new File("F2.java","....")) ;
32     d3.add(new File("F1.class","....")) ;
33     Visitor c2 = new CountVisitor();
34     d.accept(c2);
35     System.out.println("nb2 = " + ((CountVisitor)c2).getCount()); }
36 }
```

Listing 2 – découpage modulaire et ajout de fonctionnalités

1. Que fait le visiteur c2, défini par la classe `CountVisitor` du listing 2.
   L'exécution du main de TestCountVisitor affiche "nb2 = x" ; que vaut x et pourquoi ?
2. Commentez, d'un point de vue algorithmique, le parcours d'arborescence réalisé par le visiteur c2.
3. A quoi sert le schéma visiteur ?
4. Commentez, pour un programmeur non expérimenté, le lien conceptuel entre entre les schémas *Composite* et *Visiteur*.
5. Si on remplaçait dans le code du listing 2 les noms de méthodes "visitFile" et "visitDirectory" par "visit", le programme compilerait-il sans erreur ? si oui son exécution produirait-elle le même résultat ? Expliquez.

3 Limites de l'approche objet : séparation des préoccupations

Les aspects, ou leur version réduite les annotations, traitent certaines limites de l'approche objet en permettant de séparer les codes réalisant différents requis fonctionnels ou non fonctionnels d'une entité logicielle. On réalise ici en utilisant le langage *AspectJ* un aspect (tel que vus dans le TD-TP-3) nommé `ESAspect` (voir listing 3, relatif aux éléments de stockages (tels que vus dans les TD-TPs 1). Note : on considérera le code des éléments de stockage de base donc sans le mécanisme de visite du listing 2 On ne s'intéresse dans cette question qu'aux dossiers et aux fichiers (pas aux liens).

```
1  public aspect ESAspect {
2    protected static int c = 0;
3    static int getCount() {int aux = c; c = 0; return aux;}

5    pointcut detectCreation (String d) :
6      args(d) && execution (Directory.new(String));

8    after (String s) : detectCreation (s){
9      System.out.println("le ... " + s + " a été créé"); }

11   pointcut count(Directory rec, ElementStockage arg) :
12     target(rec) && args(arg) && call (boolean Directory.add(..));

14   after(Directory rec, ElementStockage arg) : count(rec,arg) { c = c + 1; }
15   } //end ESAspect

17 public class TestESAspect {
18   public static void main(String[] args) {
19     Directory d = new Directory("UnProgramme") ;
20     Directory d2 = new Directory("src1");
21     Directory d3 = new Directory("bin");
22     d.add(d2);
23     d.add(d3);
24     File f2 = new File("F2.java",".....");
25     d2.add(new File("F1.java",".....")) ;
26     d2.add(f2) ;
27     d2.remove(f2);
28     d3.add(new File("F1.class",".....")) ;
29     System.out.println("comptage : " + ESAspect.getCount());
30   } }
```

Listing 3 – séparation des préoccupations

Questions

1. dans le listing 3, dans l'advice associé au point de coupe `detectCreation`, remplacer les "..." par le bon terme.
2. Au vu du code des *advices*, explicitez précisément ce que comptabilise l'attribut statique `c` de l'aspect `ESAspect` ;
3. A la fin du main du listing 3, quelle est la valeur de l'expression `ESAspect.getCount()`. Expliquez.
4. Considérez à nouveau vos classes d' `éléments de stockage` ; décrivez (texte et éventuellement code) une autre solution, différente de celle utilisée dans le listing 2, pour les doter d'un mécanisme de visite sans modifier leur code.

Voir page suivante pour partie B.

PARTIE B (Environ 10 points - 1 seule section)

Les documents ne sont pas autorisés ; quand du code est demandé, la syntaxe exacte du langage utilisé n'est pas exigée mais la logique de l'écriture de code doit être présente dans les réponses.

Soit une classe PasswordManager, qui implémente une interface déclarant deux méthodes. La première retourne à chaque invocation un mot de passe aléatoire différent (de ceux retournés précédemment). Le mot de passe doit être composé de 12 caractères minimum (lettres, chiffres et symboles spéciaux). La deuxième méthode réinitialise (vide) la liste de mots de passe déjà retournés. En Java, pour obtenir un caractère de façon aléatoire, on peut écrire :

```
1 Random rnd = new Random(); // java.util.Random
2 String chars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!&?%$£";
3 char c = chars.charAt(rnd.nextInt(chars.length()));
```

Listing 4 –

Questions :

1. Si l'on définit cette classe dans une application Spring MVC, quelle annotation doit-on donner à la classe : Controller, RestController, Entity, Service ou Repository ? En d'autres termes, à quelle couche les objets de cette classe doivent être attribués. Justifier votre choix.
2. Définir cette classe avec ses deux méthodes, en stockant les mots de passe dans une structure de données (une collection Java) appropriée.
3. On souhaite maintenant associer à ce code une base de données SQL pour stocker les mots de passe générées. L'objectif est de rendre l'application plus fiable et supporter son redémarrage, qui peut être plus ou moins fréquent si l'application est déployée dans un container sur le Cloud. En effet, sans base de données, le redémarrage de l'application est synonyme de perte des mots de passe générés (stockés en mémoire centrale). Définir le code Spring MVC nécessaire pour couvrir les différentes couches de l'application, y compris les contrôleurs Rest (ce qui nous permet d'obtenir une application avec une API Rest). Ne pas écrire la totalité du code. Indiquer toutefois les parties du code jugées les plus importantes. (La même règle s'applique aux questions suivantes.)
4. Définir une configuration Spring Boot dans un fichier application.yml pour déployer l'application dans un environnement de développement local. Justifier le choix de vos paramètres de configuration.
5. Ajouter maintenant une terminaison d'API (on dira une *(endpoint)*) permettant d'obtenir tous les mots de passe *générés*.
6. Mettre en place la configuration nécessaire pour sécuriser cette " API *endpoint*" et celle utilisée pour vider la liste de mot de passe générés, et laisser la première *endpoint* (pour générer un mot de passe aléatoire) accessible à tout utilisateur. Utiliser une authentification en mémoire (*in-memory*) pour sécuriser les *endpoints*.
7. En supposant que la sécurité est désactivée dans l'application, définir une classe de tests unitaires permettant de vérifier le comportement des 3 *endpoints*.
   Donner la structure de votre projet, en termes de dossiers et fichiers.
8. Nous souhaitons réutiliser cette application comme module Java 9+. Que faudra-t-il faire dans le code de cette application (et autour de celui-ci) pour la transformer en module Java ? Faire en sorte que ce composant publie un service dans une JVM (non-web). Indiquer le nom de la classe d'implémentation du service.
   Définir un deuxième module dans lequel on mettra le code du client du service (un client non-web). Appeler les opérations du service dans ce client.
   Où doit on mettre l'interface du service ? (Dans le module qui implémente le service, dans le module client ou dans un module dédié ?) Motiver votre choix.