TD de Compléments en Programmation Orientée Objet n° 2 : Objets, classes, encapsulation
(Correction)

I) Questions de cours

À faire sur le sujet.

À propos des QCM : Dans chaque case, on peut inscrire vrai ou faux ou laisser blanc pour ne pas répondre. En contrôle évalué, de score obtenu est le solde du nombre de bonnes réponses moins le nombre de mauvaises réponses, ramené à zéro si négatif.

Exercice 1 : Généralités

1. [V] Un même code-octet JVM est exécutable sur plusieurs plateformes physiques (x86, PPC, ARM, ...)
2. Les entités suivantes peuvent être membres d'une classe :
   [V] attribut [V] méthode [V] classe [F] constructeur

   Correction : Ok, c'était un bête piège : par convention, la JLS considère que les constructeurs ne sont pas des membres. Cela dit, leurs déclarations sont aussi dans des classes.

3. [F] La JVM interprète du code source Java.

   Correction : La JVM interprète seulement (ou bien compile à la volée, cf. JIT) du code-octet, mais en aucun cas du code source. C'est le rôle du compilateur (javac) de comprendre le code source.

4. [V] Le mot-clé `this` est une expression qui s'évalue comme l'objet sur lequel la méthode courante a été appelée.
5. [F] Toute classe dispose d'un constructeur par défaut, sans paramètre.

   Correction : Toute classe non munie d'un constructeur écrit par le programmeur se verra ajouter automatiquement le constructeur par défaut par le compilateur.
   Les autres classes n'ont pas ce constructeur par défaut, mais seulement ceux que le programmeur aura écrits.

6. [F] On peut écrire `public` devant la déclaration d'une variable locale pour qu'elle soit visible depuis les autres classes.

   Correction : Les variables locales sont... locales ! Ceci signifie qu'elles n'ont du sens que dans le bloc où elles sont déclarées. Les modificateurs de visibilité ne s'y appliquent donc pas.

7. [F] Dès lors qu'un objet n'est plus utilisé, il faut penser à demander à Java de libérer la mémoire qu'il occupe.

Correction : Non, le ramasse-miettes détermine automatiquement quels objets ne sont plus référencés et peuvent donc être libérés (ce qu'il va donc faire périodiquement sans qu'on ait à le demander).

8. Examinez le programme suivant et évaluez les propositions.

```
1  class Truc {
2      static int v1 = 0;
3      int v2 = 0;
4      public int getV1() { return v1; }
5      public int getV2() { return v2; }
6      public Truc() {
7          v1++; v2++;
8      }
9  }
10
```

```
11 public class Main {
12     public static void main(String args[]) {
13         System.out.println(new
               Truc().getV1());
14         System.out.println(new
               Truc().getV2());
15         System.out.println(new
               Truc().getV1());
16     }
17 }
```

F La ligne 14 affichera "2". F La ligne 15 affichera "1".

Correction : `v1` et `v2` n'ont pas le même statut. `v1` est statique et donc n'existe qu'en un seul exemplaire, incrémenté à chaque instanciation de `Truc` (donc 3 fois avant la ligne 15) . Donc La ligne 15 affiche "3".
`v2`, elle, est un attribut d'instance, donc un nouvel exemplaire existe pour chaque nouvelle instance de `Truc`, dont la valeur vaut 1 à la sortie du constructeur. Donc la ligne 14 affiche "1'".

9. F La durée de vie d'un attribut statique est liée à celle d'une instance donnée de la classe où il est défini.

Correction : Non, la durée de vie d'un attribut statique est liée à la durée d'existence de la classe dans la mémoire de la JVM (typiquement : toute la durée de l'exécution du programme).

Exercice 2 : Constructeurs des classes imbriquées

Soient les classes :

```
1  public class A{
2      private int a;
3
4      public class AA{
5          private int a;
6          public AA(int y){ this.a = y;}
7      }
8
9      public static class AB{
10         private int b;
11         public AB(int x){ this.b = x;}
12     }
13
```

```
14     public A(int a){
15     this.a = a;
16     }
17 }
18 public class Main{
19     public static void main(String[] args){
20         A unA = new A(2);
21         // A.AA unAA = new A.AA(3);
22         // A.AB unAB = new A.AB(4);
23         // A.AA autreAA = unA.new AA(3);
24         // A.AB autreAB = unA.new AB(4);
25     }
26 }
```

— Parmi les lignes 21 à 24, quelles sont celles pour qui le programme compile encore quand on les « dé-commente » ?

Correction : Les lignes 22 et 23 compilent.

— Dans la classe `AA`, je peux accéder à la valeur `a` de `A`, en faisant
[F] `A.a` [V] `A.this.a` [F] `this.A.a`
— même question dans la classe `AB`

Correction : la question ne se pose pas : une instance de `AB` n'a pas de lien vers une instance de `A`. Donc faux partout.

Exercice 3 : Classes imbriquées et visibilité

Soit le programme suivant :

```
1  public class A{
2      private int x;
3      private AA aa;
4      public AA aa2;
5      private AB ab;
6
7      private class AA{
8          private int y;
9          public AA(int y){ this.y = y;}
10     }
11
12     public static class AB{
13         private int z;
14         private AB(int z){ this.z = z;}
15         public static int getYDUnAA(A a){
16             return a.aa.y;
17         }
18     }
19
20     public A(int x, int n, int z){
21         this.x = x;
22         this.aa = new AA(n);
23         this.aa2 = new AA(n*2);
24         this.ab =new AB(z);
25     }
26 }
```

— Dites s'il y des erreurs de visibilté.

Correction : Il n'y en a aucune. Dans une classe, toutes les classes imbriquées voient tout et la classe englobante aussi.

— Soit la classe `Main` suivante :

```
1  public class Main{
2      public static void main(String[] args){
3          A unA = new A(2, 3, 4);
4          System.out.println(A.AB.getYDUnAA(unA));
5      }
6  }
```

— Dessinez l'objet `unA` et son contenu. Qu'est-ce qui est affiché ?

Correction : C'est mieux avec un dessin, mais `unA` contient un pointeur vers sa classe `A`, un entier `x` de valeur 2, deux pointeurs vers des objets de type `A.AA` (`aa` et `aa2`) et un pointeur vers un objet de type `A.AB` (`ab`). Chacun des objets de type `A.AA`, contient un entier `y` et un pointeur vers `A.this`, plus le pointeur vers la classe `A.AA` Pour l'objet de type `AB`, juste un entier et le pointeur vers la classe `A.AB`
Affichage : 3

Exercice 4 : Encapsulation

Examinez le programme suivant et dites ce qui se passe ? Est-ce que le comportement est conforme aux commentaires du `main` ? Que faut-il corriger et comment ?

```java
public class Personne{
    private String nom;
    private final int numSecu;

    public Personne(String nom, int numSecu){
        this.nom = nom;
        this.numSecu = numSecu;
    }

    public void changeNom(String nom){
        this.nom = nom;
    }

    @Override
    public String toString(){
        return nom + " "+ numSecu;
    }
}

public class Cours{
    private String nom;
    private HashSet<Personne> inscrits;

    public Cours(String nom, HashSet<Personne>
         inscrits){
        this.nom = nom;
        this.inscrits = inscrits;
    }
    //return false si pas dans la liste
    //  des inscrits au départ
    public boolean exclut(Personne p){
        return inscrits.remove(p);
    }
    //return false si déjà  dans liste
    //  des inscrits au départ
    public boolean inscrit(Personne p){
        return inscrits.add(p);
    }

    @Override
    public String toString(){
        String s = nom;
        for(Personne p : inscrits){
            s += "\n" + p;
        }
        return s;
    }
}

public class Main{
    public static void main(String[] args){
        //on crée un certain nombre de personnes:
        Personne p1 = new Personne ("Adèle",
             1254);
        Personne p2 = new Personne ("Brian",
             1287);
        Personne p3 = new Personne ("Coralie",
             2546);
        Personne p4 = new Personne ("Désiré",
             2546);
        //on crée un groupe de copains:
        HashSet<Personne> copains = new
             HashSet<Personne>();
        copains.add(p1);
        copains.add(p2);
        copains.add(p3);
        //pour ce groupe, on crée 2 cours:
        Cours c1 = new Cours("Couture", copains);
        Cours c2 = new Cours("Karate", copains);
        //Coralie est exclue du cours de Couture
        c1.exclut(p3);
        //Désiré s'inscrit à celui de Karaté
        c2.inscrit(p4);
        //Adèle change de prénom, parce que
        // l'ancien ne lui plaisait pas
        p1.changeNom("Adeline");
        System.out.println(c1);
        System.out.println();
        System.out.println(c2);
    }
}
```

Remarque : En général, pour des raisons de persistence des données, ce genre de données a vocation à être géré par une base de données en interfaçage éventuel avec un langage de programmation.

Correction : Comportement :

```
Couture
Adeline 1254
Brian 1287
Désiré 2546

Karate
Adeline 1254
Brian 1287
Désiré 2546
```

Il faut faire une copie du `HashSet` à la création de cours, par contre le changement de nom ne pose pas de problème.

II) Programmer

Finir le TD 1 !