TP de CPOO n° 1 : Révisions

Exercice 1 :

Les affirmations suivantes sont elles **rigoureusement exactes** ?

1. Un même code-octet JVM est exécutable sur plusieurs plateformes physiques (x86, PPC, ARM, ...)
2. La JVM interprète du code source Java
3. Le mot-clé this est une expression qui s'évalue comme l'objet sur lequel la méthode courante a été appelée.
4. Toute classe dispose d'un constructeur par défaut, sans paramètre.
5. On peut écrire public devant la déclaration d'une variable locale pour qu'elle soit visible depuis les autres classes.
6. Dès lors qu'un objet n'est plus utilisé, il faut penser à demander à Java de libérer la mémoire qu'il occupe.
7. La durée de vie d'un attribut statique est liée à celle d'une instance donnée de la classe où il est défini.

Exercice 2 : statique et non-statique

Etant donné le code suivant, ces affirmations sont-elles vrai ou fausse :
La ligne 14 affichera "2" ?
La ligne 15 affichera "1" ?

```
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 }
```

Exercice 3 : 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 que le programme compile encore quand on les dé-commente ?

Exercice 4 : Encapsulation et sûreté

Voici deux classes avec leur spécification. Pour chaque cas :
— soit la spécification est satisfaite par la classe, dans ce cas, justifiez-le ;
— soit la spécification n'est pas satisfaite, dans ce cas écrivez un programme qui la met en défaut (sans modifier la classe fournie), puis proposez une rectification de la classe.

1.
```
1  public class EvenNumbersGenerator {
2      static int MAX = 42;
3      public int previous = 0;
4      public int next() {
5          previous += 2; previous %= MAX;
6          return previous;
7      }
8  }
```

Spécification : la méthode `next` ne retourne que des entiers pairs.

2.
```
1  public class VectAdditioner {
2      private Point sum = new Point();
3
4      public void add(Point p) {
5          sum.x += p.x; sum.y += p.y;
6      }
7
8      public Point getSum() {
9          return sum;
10     }
11 }
```

Spécification : la méthode `getSum` retourne la somme de tous les vecteurs qui ont été passés en paramètre par la méthode `add` depuis l'instanciation.

Exercice 5 : Nombres complexes

Pour le vocabulaire, référez vous à `https://fr.wikipedia.org/wiki/Nombre_complexe`.

1. Écrivez une classe `Complexe`, avec :
   — les attributs (`double`) : parties réelle et imaginaire du nombre (`static` ou pas ?) ;
   — le constructeur, prenant comme paramètres les parties réelle et imaginaire du nombre ;
   — la méthode `public String toString()`, permettant de convertir un complexe en chaîne de caractères lisible par l'humain ;
   — les opérations arithmétiques usuelles (somme, soustraction, multiplication, division) ;
   — le test d'égalité (méthode `public boolean equals(Object other)`) ;
   — les fonctions et accesseurs spécifiques aux complexes : partie réelle, partie imaginaire, conjugaison, module, argument...

   Vous pouvez vous aider des fonctionnalités de génération de code de votre IDE.

   Attention, style demandé : attributs non modifiables (si vous savez le faire, faites une vraie classe immuable), les opérations retournent de nouveaux objets.
2. Ajoutez à votre classe :
   — des attributs (constants : vous pouvez ajouter `final`) pour les valeurs les plus courantes du type `Complexe` : à savoir 0, 1 et i (le nombre i tel que $i^2 = -1$).
   Ces attributs doivent-ils être `static` ou non ?
   — une méthode (fabrique statique)
   ```
   1  public static Complexe fromPolarCoordinates(double rho, double theta)
   ```
   qui construit un complexe depuis son module ρ et son argument θ (on rappelle que la partie réelle vaut alors $\rho \cos\theta$ et la partie imaginaire $\rho \sin\theta$).

Remarquez que cette méthode joue le rôle d'un constructeur. Pourquoi ne pas avoir fait plutôt un autre constructeur alors ? (essayez de compiler avec 2 constructeurs puis expliquez pourquoi ça ne marche pas)

3. Améliorez l'encapsulation de votre classe, afin de permettre des évolutions ultérieures sans casser les clients/dépendents de celle-ci : en l'occurrence, les attributs doivent être privés et des accesseurs publics [1] doivent être ajoutés pour que la classe reste utilisable.
4. Testez en écrivant un programme (méthode `main()`, dans une autre classe), qui fait entrer à l'utilisateur une séquence de nombre complexes et calcule leur somme et leur produit. Améliorez le programme pour permettre la saisie des nombres au choix, via leurs parties réelles et imaginaires ou via leurs coordonnées polaires.
5. Écrivez une version mutable de cette classe (il faut donc des méthodes `set` pour chacune des propriétés).

   Changez la signature [2] et le comportement des méthodes des opérations arithmétiques afin que le résultat soit enregistré dans l'objet courant (`this`), plutôt que retourné.

1. Remarquez qu'il n'y a pas de raison de favoriser le couple parties réelle/imaginaire par rapport au couple module/argument (les deux définissent de façon unique un nombre complexe) ; et qu'il faut donc considérer ce dernier couple comme un couple de propriétés, pour lequel il faudrait utiliser aussi la notation `get`. Ainsi, cette classe aurait 4 propriétés (peu importe si elles sont redondantes : les attributs ne le sont pas ; cela limite les risques d'incohérences).
2. Si la méthode déjà programmée est `static` rendre non statique et enlever un paramètre ; dans tous les cas retourner `void`.