Exercices d'introduction orientée objet en Java : personnes, enseignants et étudiants

Université Paris Sud

Préambule

Cette suite d'exercices a pour but de passer en revue les notions orientées objet les plus simples du langage. La rédaction utilise volontairement le vocabulaire dédié. Il est plus que conseillé de vous reporter au notes de cours qui vous ont été distrribuées ainsi qu'à l'aide sur les API Java.

Exercice 1 Classe Personne (*constructeurs*)

Nous allons créer deux classes, l'une représentant une personne, nommée `Personne` et l'autre pour contenir le main, nommée `PersonneMain`.

1. *Constructeur exhaustif.*
   a) Créer la classe nommée `PersonneMain`, *publique* (avec le mot clé `public` devant `class`), ne comportant pas de champ et comportant une unique méthode `main()` de signature
   ```
   public static void main(String args[])
   ```
   Au sein de cette méthode, créer, à l'aide de l'opérateur `new` une instance d'objet de type `Personne` (voir la description de la classe `Personne` ci-dessous) et appeler sa méthode `afficher()`.
   On rappelle qu'un fichier source java peut contenir au plus une classe publique et que, si c'est le cas, le nom du fichier doit correspondre (y compris les majuscules/minuscules) au nom de la classe publique.
   b) Créer une classe, nommée `Personne`, *non publique* (sans le mot clé `public` devant `class`), contenant 2 champs :
   – un champ `nom` de type `String`, et
   – un champ `age` de type primitif `int`.
   Créer un constructeur exhaustif (c.à.d. initialisant tous les champs), de signature
   ```
   Personne(String leNom, int lAge)
   ```
   initialisant les deux champs de la classe à `leNom` et `lAge`.

c) Créer la méthode `afficher()` de signature
```
void afficher()
```
qui affiche le `nom` et l'`age` au format décrit ci-après. Pour un objet de type `Personne` dont la variable d'instance `nom` vaut `"Coupdetrix"` et la variable d'instance `age` vaut `25`, la méthode `afficher()` affichera à l'écran :
```
Nom : Coupdetrix
Age : 25 ans
```

d) Tester ces classes en exécutant `PersonneMain`.

2. *Champ de type tableau ; test de conformité d'un constructeur exhaustif.*

a) Ajouter un champ à la classe `Personne`, de type tableau de réels
```
double[] comptes;
```
susceptibles de représenter les divers soldes des comptes en banque de la personne. Adapter le constructeur exhaustif ainsi que la méthode `afficher()` (qui doit afficher, outre le nom et l'age de la personne les différents éléments du tableau `comptes`. Créer, au sein de la classe `Personne`, la méthode `void diviserParDeux()`, qui modifie le tableau `comptes` en divisant chaque élément par deux.

b) Afin de tester que le constructeur exhaustif précédent est bien conforme à ce qu'il doit être, dans la classe `PersonneMain`, créer deux objets de type `Personne`, de même `nom`, même `age` et à partir du *même objet* `nombres` comportant deux éléments. Appeler `diviserParDeux()` sur le premier des objets de type `Personne`. Appeler la méthode `afficher()` de ces deux objets. Le code utilisé sera par exemple le suivant :
```
int[] mesNombres = new int[2];
mesNombres[0] = 100;
mesNombres[1] = 92;
Personne sage = new Personne("Agecanonix", 80,
                                    mesNombres);
Personne sageBis = new Personne("Agecanonix", 80,
                                    mesNombres);
sage.diviserParDeux();
sage.afficher();
sageBis.afficher();
```
Si l'affichage est identique pour les deux objets, le présent constructeur exhaustif, ainsi que celui que vous avez réalisé à la question précédente ne sont pas corrects. Les initialisations des champs `nom` et `nombres` doivent alors avoir été faites avec ce que l'on nomme de la copie superficielle (copie des références uniquement), alors qu'elles auraient dues être faites avec de la copie profonde (copie du contenu des objets). Corriger le constructeur exhaustif si nécessaire.

3. *Constructeurs par défaut* et *de recopie.*

Par souci de simplicité, le champ `comptes` précédent ne sera pas conservé. Son seul but était de faire prendre conscience de la nécessité des copies profondes au sein des constructeurs. Nous reprendrons également le constructeur exhaustif et la méthode `afficher()` de la première question.

a) Au sein de la classe `Personne`, créer un constructeur par défaut, de signature
```
Personne()
```
appelant le constructeur exhaustif de la question précédente au moyen du mot clé `this` (voir l'utilisation de `this(...)` comme appel de constructeur) et

initialisant respectivement les champs `nom` et `age` à `"Asterix"` et `30`. Si l'évocation du mot clé `this` comme appel de constructeur ne vous semble pas parfaitement claire, reportez vous au polycopié, au chapitre des bases orientée objet de java, section "Classes, instance d'objet, héritage", sous section "constructeurs".

b) Créer un constructeur de recopie, de signature
```
Personne(Personne p)
```
qui initialise les champs de la classe à ceux de `p`.

c) Modifier la méthode `main()` de `PersonneMain` en créeant 3 personnes, l'une à l'aide du constructeur exhaustif, une autre avec le constructeur par défaut et la dernière avec le contructeur de recopie. Les afficher toutes les trois en appelant la méthode `afficher()`.

Exercice 2 Classes Enseignants et étudiants (*Héritage*)

1. Classes `Enseignant` et `Etudiant`.

   a) Créer une classe `Enseignant` héritant de `Personne` et ayant deux champs :
   – Un champ `nbHeuresCours`, de type primitif `int`,
   – Un champ `grincheux`, de type primitif `boolean`.
   Créer un constructeur exhaustif appelant le constructeur de la classe mère (celui de `Personne`) par `super` (cet appel doit être la première ligne du constructeur d'`Enseignant`), puis initialisant ses champs propres (`nbHeuresCours` et `grincheux`).

   b) Créer de même une classe `Etudiant` héritant de `Personne` et ayant deux champs :
   – Un champ `noteMoyenne`, de type primitif `int`,
   – Un champ `faineant`, de type primitif `boolean`.
   Créer un constructeur exhaustif appelant le constructeur de la classe mère (celui de `Personne`) par `super` (cet appel doit être la première ligne du constructeur d'`Etudiant`), puis initialisant ses champs propres (`noteMoyenne` et `faineant`).

   c) Créer une classe `ProfEleveMain` ne comportant pas de champ et comportant une unique méthode `main()`. Au sein de cette méthode, créer une instance d'objet de type `Personne`, de `nom` égal à `"Nimbus"` et d'`age` égal à `45`.
   Créer un `Enseignant`, de même `nom` et `age`, ayant `nbHeuresCours` égal à `50` et `grincheux` mis à `true`.
   Créer un `Etudiant` de nom `"Soupaloigonycrouton"`, d'`age` égal à `20`, ayant `2` comme `noteMoyenne` et `faineant` mis à `true`.
   Appeler la méthode `afficher()` (qui est héritée de `Personne`) de ces trois instances. On constate que les deux premières instances ont des affichages indistinguables.

2. Différentiation par le type.

   a) Créer un tableau de 5 éléments de type `Personne` dans la méthode `main()` de la classe `ProfEleveMain` ; initialiser ce tableau avec 2 instances d'objet de type `Enseignant` et 3 instances de type `Etudiant`. Effectuer une boucle `for` sur les éléments du tableau ; dans le corps de la boucle, effectuer un appel

à la méthode `afficher()`, puis afficher s'il s'agit d'un `Etudiant` ou d'un `Enseignant` en se servant de l'opérateur `instanceof`.

Exercice 3 Différents affichages (*Surcharge et redéfinition*)

1. *Surcharge de* la *méthode* `afficher()`.
   Dans la classe `Personne`, créer une méthode `afficher()`, de signature
   ```
   void afficher(boolean compact)
   ```
   qui affiche selon l'un des formats suivants. Si `compact` est égal à `false`, l'affichage est le même que celui de signature `void afficher()` (la méthode de même nom décrite en question 1c de l'Exercice 1). Si `compact` est égal à `true`, l'affichage d'un objet de `nom` égal à `Coupdetrix` et d'`age` égal à `25` sera de la forme `[Coupdetrix, 25]`
2. *Redéfinition de* la *méthode* `afficher()`.
   Créer, au sein de la classe `Enseignant` (resp. au sein de la classe `Etudiant`) une méthode de signature `void afficher()` qui appelle la méthode `afficher()` de `Personne` (au moyen de `super`) puis qui affiche la chaîne `"Enseignant"` (resp. `"Etudiant"`).
3. Modifier la méthode `main()` de `ProfEleveMain` élaborée dans la question précédente en ne conservant, dans le corps de la boucle `for`, qu'un appel à la méthode `afficher()` sur les cinq éléments du tableau.

Exercice 4 Délégation d'affichage (*Classes abstraites*)

Supposons que la classe `Personne` fasse partie d'une bibliothèque que vous distribuez et que vous vouliez obliger les programmeurs qui conçoivent des classes héritant de `Personne` (telles `Enseignant` et `Etudiant`) à munir ces dernières de certaines méthodes.

1. Rendre la classe `Personne` abstraite (par le mot clé `abstract` devant `class`).
2. Déclarer une méthode abstraite, ne retournant rien (c'est-à-dire `void`), sans arguments, nommée `afficherType()`.
3. La méthode `afficher()` de `Personne` va affectuer l'affichage décrit en question 1c de l'Exercice 1, puis va faire appel à `afficherType()`.
4. Les classes `Enseignant` et `Etudiant` héritent de la classe abstraite `Personne` et ne sont pas abstraites. Leur constructeur doit donc faire appel à la méthode `createPersonne()` (puisqu'il n'y a plus de constructeur de `Personne`).
   La méthode `afficherType()` dans la classe `Enseignant` (resp. dans la classe `Etudiant`) n'effectue qu'une chose : afficher la chaîne `Enseignant` (resp. la chaîne `Etudiant`).
5. Adapter les classes `Enseignant` et `Etudiant`.

De cette manière, toute classe héritant de `Personne` doit nécessairement implanter la méthode `afficherType()`. L'affichage est *délégué* aux sous types concrets de `Personne`.