UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Licence INFO
Info0201

Travaux dirigés n° 3

Méthodes d'instances / Méthodes de classes

Exercice 1 (Synthèse technique)

Nous considérons les deux classes suivantes :

```
class O1 {
    public  static final int a = 12;
    private static int b = 1;

    public  int c;
    public  int d = 3;
    private int e;

    public O1() { c=1; d=2; e=3; }
    public O1(int c, int d, int e) {
         this.c=c; this.d=d; this.e=e;
    }

    public  static void pa(int a) { O1.b += a; }
    private static int  fb() { return a * b; }

    public  void pc(int a) {
        System.out.println( a + b + fb() );
    }
    public  int  fd(int a) {
        return a + b + c + d + e;
    }
} // fin de la classe O1
```

```
class TestO1 {
    static final int t = 5;

    public static void main(String[] args) {
        int a = 2;                          // 1
        int c = O1.a + a + 5;               // 2

        O1 ref  = new O1();                 // 3
        O1 ref2 = new O1(a, O1.b, c);       // 4

        t++;                                // 5
        c = a + O1.a;                       // 6
        O1.a += 5;                          // 7
        O1.d += c;                          // 8
        a += O1.fb(c);                      // 9
        ref.d = ref.fd(t);                  // 10
        ref.pa(7);                          // 11
    }
}
```

1°) Pour chacune des expressions suivantes précisez (en justifiant) si elle est valide, si elle est visible depuis le `main` de la classe de test (et donnez sa valeur) et si elle est modifiable.

```
O1.a        O1.b        O1.c        O1.d        O1.e
ref.a       ref.b       ref.c       ref.d       ref.e
```

2°) Pour chacune des expressions suivantes précisez (en justifiant) si elle est valide et si elle est visible depuis le `main` de la classe de test (et donnez sa valeur).

```
O1.fb()     O1.fd(5)     ref.fb()     ref.fd(5)
```

3°) Pour chacune des instructions suivantes précisez (en justifiant) si elle est valide et si elle est visible depuis le `main` de la classe de test (et donnez son effet).

```
O1.pa(1);     O1.pc(2);     ref.pa(3);     ref.pc(4);
```

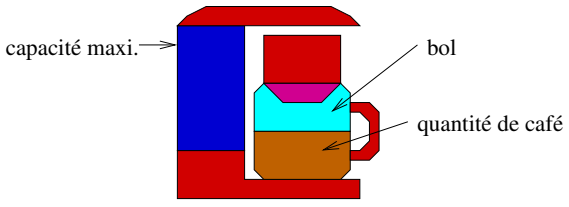
4°) Commentez la méthode suivante, ajoutée à la classe **O1** :

```
public static void pe(int b) {
    c = a + b;
    this.b = a;
}
```

5°) Pour chacune des instructions de la méthode principale de la classe de test, précisez ses effets si elle est valide (si elle ne l'est pas, justifiez pourquoi).

Exercice 2 (Cafetière)

Nous souhaitons modéliser une cafetière caractérisée par la capacité de son bol (en ml) et par la quantité de café qu'elle contient (en ml). Lorsqu'une cafetière est construite (*i.e.* instanciée), sa capacité est fixée (soit par défaut à 200ml, soit spécifiée en paramètre). Par contre, elle ne possède pas de café dans son bol lors de sa création. La quantité de café ne doit jamais dépasser la capacité [maximale].



1°) Écrivez la classe `Cafetiere` :

- Écrivez la déclaration des attributs, les constructeurs par défaut et par initialisation, le(s) *getter(s)*.
- Écrivez une méthode `faireCouler` qui prend en paramètre la quantité de café à ajouter à la quantité déjà présente dans le bol. Il n'est pas possible de faire couler plus de café que la cafetière courante n'est en mesure de contenir.
- Écrivez les méthodes `afficher` et `toString`. Une cafetière doit s'afficher de la manière suivante :

```
Cafetiere : 10ml de cafe sur 200ml max.
```

2°) Écrivez une classe `TestCafetiere` permettant de tester les méthodes de la classe `Cafetiere`.

3°) Nous souhaitons créer une nouvelle instance de `Cafetiere` qui est une copie d'une des instances créées précédemment. Ecrivez un *constructeur par copie* dans la classe `Cafetiere`.

4°) Autres méthodes :

- Ajoutez la méthode `egalA` permettant de vérifier si la cafetière passée en paramètre est identique à la cafetière courante.
- Nous souhaitons écrire une méthode `transferer` qui prend en paramètre une cafetière. Le café de la cafetière passée en paramètre est transféré dans la cafetière courante. Il n'est pas possible de transférer plus de café que la cafetière courante n'est en mesure de contenir.

5°) Nous souhaitons calculer le nombre d'instances de la classe `Cafetiere` créées lors de l'exécution d'un `main`. Pour ce faire, nous ajoutons un attribut à la classe `Cafetiere` : un compteur global.

- Donnez la déclaration du compteur. Que doit-on modifier dans la classe `Cafetiere` pour le mettre à jour ?
- Écrivez la méthode `getCompteur` permettant de récupérer la valeur du compteur.

6°) Complétez votre classe de Test et donnez la représentation mémoire correspondante.

Exercice 3 (Un autre exemple)
Nous souhaitons écrire une classe `Memoire` représentant des cases mémoire, chaque case pouvant contenir un entier. Lorsqu'une `Memoire` est créée, il faut indiquer le nombre maximal de cases qu'il est possible de sauvegarder, ce nombre dépend de l'évolution des générations de la mémoire. Chacune de ses cases contient la valeur 0 à la création de la mémoire. En effet, la technologie avançant très vite, il est possible à chaque nouvelle génération de sauvegarder plus de cases. Par défaut, la première génération est fixé à 8 cases. Chaque nouvelle génération permet d'augmenter d'un certain nombre de cases supplémentaires la capacité des mémoires créées.

1°) Imaginez la représentation mémoire qu'on devrait obtenir après avoir créé une mémoire de 6 cases (ce qui est possible à la génération initiale) et y avoir sauvegardé la valeur -3 dans la case numéro 2.

2°) Écrivez la déclaration des attributs et le constructeur par défaut et par initialisation.

3°) Écrivez la méthode `getCase(int n)` retournant l'entier situé dans la case mémoire dont l'indice est `n`. Si l'indice est incorrect, la méthode retourne 0.

4°) Écrivez la méthode `boolean setEntier(int n, int v)` permettant de placer la valeur `v` dans la case mémoire dont l'indice est `n`. Cette méthode renvoie un booléen pour permettre d'indiquer si la sauvegarde a pu avoir lieu.

5°) Écrivez la méthode `nouvelleGeneration(int nb)` qui augmente de `nb` le nombre de cases maximum pour les futures instances de `Memoire`.

6°) Écrivez la méthode `toString`. La description d'une `Memoire` est du style : `[-8, 0, 0, 1, 2, 5]`

7°) Écrivez la méthode `egalA` permettant de tester si la mémoire passée en paramètre est identique à la mémoire courante.

8°) Écrivez la méthode `ajouter` permettant d'ajouter une valeur (passée en paramètre) à toutes les cases mémoire.

9°) Écrivez la méthode `ajouter` permettant d'ajouter aux cases mémoire, les valeurs situées dans la `Memoire` passée en paramètre. Proposez une solution pour gérer le cas où les nombres de cases mémoire sont différents.