Université Paris Diderot **Programmation Orientée Objet**
L2-L3 **Année 2014-2015**

TD n° 6

Classes internes

Exercice 1 :

Les programmes suivants sont-ils corrects ? Justifiez.

1.

```
public class A {                                                    1
  class B { public int x; }                                         2
  public static void main(String args []) { B b = new B();}         3
}                                                                   4
```

2.

```
public class A {                                                    1
  class B { public int x; }                                         2
  public static void main(String args []) { B b = (new A()).new B();} 3
}                                                                   4
```

3.

```
public class A {                                                    1
  static class B { public int x; }                                  2
  public static void main(String args []) { B b = new B();}         3
}                                                                   4
```

4.

```
public class A {                                                    1
  int x = 42;                                                       2
  static class B { public int y = x; }                              3
}                                                                   4
```

5.

```
class A {                                                           1
  int x = 42;                                                       2
  static class B {                                                  3
    void f() { x++; }                                               4
  }                                                                 5
}                                                                   6
```

6.

```
class A {                                                           1
  int x = 42;                                                       2
  class B {                                                         3
    static void f() { x++; }                                        4
  }                                                                 5
}                                                                   6
```

7.

```
interface Vect {
    void scale(double lambda);
}

class Test {
  static void main(String args[]) { Vect v = new Vect(); }
}
```

8.

```
interface Vect { void scale(double lambda); }

class TestVect {
  static void main(String args[]) {
    Vect v = new Vect() {
      double x=15; double y=16;
      public void scale(double lambda) { x *= lambda; y *= lambda; }
    };
  }
}
```

9.

```
interface Vect { void scale(double lambda); }

class TestVect {
  static void main(String args[]) {
    Vect v = new Vect() {
      double x=15; double y=16; static int dim;
      public void scale(double lambda) { x *= lambda; y *= lambda; }
    };
  }
}
```

10.

```
interface Vect { void scale(double lambda); }

class TestVect {
  static void main(String args[]) {
    Vect v = new Vect() {
      double x=15; double y=16; static final int dim=2;
      public void scale(double lambda) { x *= lambda; y *= lambda; }
    };
  }
}
```

Exercice 2 : Suites de nombres entiers

Nous définissons les interfaces suivantes :

```
interface ItereSuiteEntiere { long courant(); long suivant(); }
interface SuiteEntiere { ItereSuiteEntiere nouvelIterateur(); }
```

que l'on pourrait utiliser de la façon suivante (étant donnée une suite entière `s`, la méthode `affiche` et un entier `n` affiche les `n` premiers termes de `s`) :

```
public class Test {
  static void affiche( SuiteEntiere s, int n)
  {
     ItereSuiteEntiere it = s.nouvelIterateur();
    for (int i=0; i < n; i++)
```

```
        System.out. println (s. suivant ());
    }
}
```

1. Définissez les classes
   — `class SuiteArithm implements SuiteEntiere` définissant des suites arithmétiques (rappel : la suite arithmétique de terme initial u_0 et de raison r est la suite de valeurs suivantes : u_0, $u_0 + r$, $u_0 + 2r$, $u_0 + 3r$, ...).
   — `class SuiteGeom implements SuiteEntiere` définissant des suites géométriques (rappel : la suite géométrique de terme initial u_0 et de raison r est la suite de valeurs suivantes : u_0, $u_0.r$, $u_0.r^2$, $u_0.r^3$, ...).

   Dans les deux cas, l'objet retourné par `nouvelIterateur()` appartiendra à une classe
   — soit interne à `SuiteArithm` et puis `SuiteGeom` (peut-elle être statique ?),
   — soit locale à la méthode `nouvelIterateur()` (et nommée),
   — soit locale et anonyme.

   Essayez les trois possibilités à chaque fois.
2. On veut maintenant créer des sous-suites, c'est-à-dire, étant donnée une suite (u_n) et une suite croissante (v_n), modéliser en Java la suite (u_{v_n}). Dans ce but, on ajoute à l'interface `SuiteEntiere` la méthode `ItereSuiteEntiere iterateurSousSuite(SuiteEntiere s)`.

   Écrivez cette méthode dans `SuiteArithm` et dans `SuiteGeom`. Argumentez sur l'usage de clases internes/locales/anonymes.

Exercice 3 : Base de données

Dans cet exercice on programme un début de système de gestion de bases de données. L'objet de base est une table (implémentant `TableBD`), c'est à dire une liste d' enregistrements (de lignes implémentant `Enregistrement`). Plus précisément, on utilise les interfaces ci-dessous :

```
interface Enregistrement { Object valeur(String champ); }
interface TableBD { Iterateur itereTable (); }
interface Iterateur { boolean aSuivant (); Enregistrement suivant (); }
```

La méthode `valeur()` doit servir à récupérer le contenu du champ dont le nom est passé en paramètre (le contenu de la colonne ayant cet intitulé).

Par exemple, si on prend pour `t` la table suivante :

```
Article           | Rayon
------------------|---------------
cahier            | papèterie
crayon            | papèterie
clous             | quincaillerie
flan              | pâtisserie
```

et si on définit `Enregistrement e = t.itereTable().suivant();`, alors, l'enregistrement `e`, désignera la première ligne de la table (c'est le premier appel à `suivant()`) et l'instruction `System.out.println(e.valeur("Article"));` affichera "`cahier`".

Dans les questions suivantes, on utilisera au maximum les classes internes et locales afin d'éviter de dupliquer les données en mémoire.

1. Écrivez `class TableBDImpl`, une implémentation possible de `TableBD`. Vous pourrez par exemple prendre un tableau unidimensionnel de `String` pour les intitulés des champs (titres des colonnes), et un tableau bi-dimensionnel d'`Object` pour le contenu de la table en lui-même (une ligne par enregistrement, une colonne par champ).

   Pour éviter la duplication de données en mémoire, les classes implémentant `Enregistrement` et `Iterateur` seront des classes internes ou locales de `TableBDImpl` avec pour attribut principal un numéro de ligne.

2. Écrivez la méthode `TableBD selection(String champ, Object valeur)` qui retourne la "même" table que `this`, à ceci près que l'on ne "garde" que les enregistrements pour lesquels le champ `champ` a pour valeur `valeur`.

   Par exemple l'objet t2 obtenu par `TableBD t2 = t.selection("Rayon", "papèterie");` devra représenter la table suivante :

   ```
   Article           | Rayon
   ------------------|---------------
   cahier            | papèterie
   crayon            | papèterie
   ```

   Là aussi `selection()` doit en fait retourner un objet d'une classe locale qui implémente `TableBD` et qui ne contienne pas une copie (même partielle) du tableau bi-dimensionnel.

3. On veut maintenant ajouter `TableBD selection(String champ, Object valeur)` à l'interface `TableBD`. Quel en serait l'intérêt ? Que faudrait-il ajouter/modifier dans votre programme actuel ?