Université Paris Diderot **Programmation Orientée Objet**
L2 **Année 2020-2021**

TD – Séance n°7 - Correction

Classes Internes

Exercice 1 *Compréhension détaillée du cours*

1. Trouvez toutes les erreurs de compilation du programme suivant.
2. Après avoir *enlevé* toutes les lignes erronées, dites quel sera l'affichage produit par la méthode `main` de la classe `D`.

A.java

```
1  public class A {
2    private static int h = 5;
3    private B x = new B();
4    private int i = x.b;
5    // classe interne statique
6    public static class StaticA {
7      public int nstat = h;
8      public static int astat = h;
9      public void increase() {
10       h++;
11       i++;
12     }
13   }
14   // classe interne membre
15   private class B {
16     private int b = 2;
17     private static int bstat = 3;
18   }
19   // classe interne membre
20   public class InstanceA {
21     private int i = A.StaticA.astat;
22     private int j = A.StaticA.nstat;
23     public int k = A.x.b;
24     public static void staticMethod() {}
25     public void method(int i) {
26       int i1 = this.i;
27       int i2 = A.this.i;
28       System.out.println(i + " " + i1 + " " + i2);
29     }
30   }
31 }
```

D.java

```
32 public class D {
33   public void localMethod(int i) {
34     i++;
35     int j = i;
36     // declaration de classe dans une methode
37     class Locale {
38       int k = i;
39       public void increase1() {
40         return ++i;
41       }
42       public int increase2() {
43         return ++j;
44       }
45       public int increase3() {
46         return ++k;
47       }
48     }
49     Locale l = new Locale();
50     System.out.println(l.increase3());
51   }

53   public static void main(String[] args) {
54     A a = new A();
55     A.StaticA.astat = 4;
56     A.StaticA.nstat = 3;
57     A.StaticA sa1 = new A.StaticA();
58     A.StaticA sa2 = A.new StaticA();
59     sa1.nstat = 3;
60     A.InstanceA nsa1 = new A.InstanceA();
61     A.InstanceA nsa2 = a.new InstanceA();
62     A.InstanceA nsa3 = new a.InstanceA();
63     nsa2.i = 3;
64     nsa2.method(6);
65     A.B b = a.new B(5);
66     D d = new D();
67     int i = 2;
68     i++;
69     d.localMethod(i);
70   }
71 }
```

Correction :

1. Ligne 11 :
   `non-static variable i cannot be referenced from a static context`
   Cette méthode d'objet de `A.StaticA` fait référence à une variable `i` qui est inconnue, parce qu'une classe intèrne statique n'a pas d'accès à des membres non-statiques de sa classe englobante. Un accès à `nstat` serait possible.

2. Ligne 17 :
   `Illegal static declaration in inner class A.B`
   `modifier 'static' is only allowed in constant variable declarations`
   `private final static int bstat = 3;` fonctionnerait.
   Explication : à la compilation le type commun pour ces objets est `A.B`, même si plus précisément `B` est une classe membre. Ce choix de type fixé, faire référence à une variable statique ne peut donc plus dépendre de l'instance de A considérée.
3. Ligne 22 :
   `non-static variable nstat cannot be referenced from a static context`
   Il faut un objet de la classe `StaticA` pour faire reference à `nstat` car c'est un champ d'instance. Remarquer également qu'à la ligne 21 on peut aussi bien écrire `StaticA.astat`.
4. Ligne 23 :
   `non-static variable x cannot be referenced from a static context`
   Ce qui est erroné est `A.x` : `x` n'est pas une variable statique de la classe `A`.
   Par contre écrire `k = x.b` ou `k = A.this.x.b` ne poserait pas de problème (même si tout est privé).
5. Ligne 24 :
   `Illegal static declaration in inner class A.InstanceA`
   `modifier 'static' is only allowed in constant variable declarations`
   Des méthodes statiques peuvent seulement être déclaré dans une classe membre statique ou dans une classe au plus haut niveau.
   Une classe membre non-statique ne peut pas avoir de méthodes statiques (ni de champs statics sauf si final).
   C'est la même explication que pour l'erreur 2 : la frontière entre le type commun utilisé à la compilation est distincte du type réel pour la classe membre. C'est considéré comme une imprécision du programmeur qui est trop proche d'une erreur. C'est donc interdit. Remarquer que dans le cas de l'erreur 2 il était accepté de définir la variable `static final`. Dans le cas d'une méthode non, on préfère imposer que la méthode figure dans le type supérieur.
6. Ligne 38 :
   `local variables referenced from an inner class must be final or effectively final`
   Cette restriction est liée à des problèmes techniques. Ici elle se contourne facilement. Cette initialisation de `k` est un raccourci pour éviter d'écrire un constructeur. Il faut imaginer qu'à la création d'un objet `Locale`, `i` n'étant pas constant, Java a un travail à faire pour le retrouver. Or le compilateur voudrait remplacer tout de suite `i` par une valeur définitive, c'est pourquoi il voudrait qu'elle soit "constante".
   Pour faire les choses plus clairement, on peut écrire un constructeur :

```
1  class Locale {
2    int k;
3    Locale(int x) { k = x; }
4  }
```

   et il n'y a pas de problème à faire `new Locale(i);` ensuite.
7. Ligne 40 :
   `incompatible types: unexpected return value`

Déjà une méthode `void` ne doit pas retourner un `int`, donc on change en `public int increase1()`.
`local variables referenced from an inner class must be final or effectively final`
Ensuite, supposons que le type retour soit `int`, on tombe sur la restriction de `i` non effectivement finale. Et on peut comprendre un peu mieux cette restriction : Imaginons que ce soit autorisé, comme l'objet `Locale` pourrait très bien être utilisé ailleurs que dans ce bloc (parce qu'il est passé en argument à une methode), il y aurait donc un moyen de modifier le contenu de `i`. On aurait alors un peu la même chose qu'en C ou quelque part on aurait accès à `i` par un pointeur, et ce n'est pas le choix fait en Java.

8. Ligne 43 :
   `local variables referenced from an inner class must be final or effectively final`
   `j` doit être finale
9. Ligne 49 :
   on avait dit qu'on supprimait les lignes qui posaient problèmes, en particulier la ligne 38, du coup `k` est à présent inconnu. Mais si on remplace ligne 38 par `int k;` alors c'est bon.
10. Ligne 56 :
    `non-static variable nstat cannot be referenced from a static context`
    Accès à `nstat` de qui ? `nstat` est un champ d'instance, on a donc besoin d'un objet de la classe `A.staticA` pour y accéder.
11. Ligne 58 :
    `cannot find symbol A`
    La bonne syntaxe est celle dans la ligne précédente.
12. Ligne 60 et 62 :
    60 : `an enclosing instance that contains A.InstanceA is required`
    62 : `package a does not exist`
    La bonne syntaxe est celle dans la ligne entre les deux.
13. Ligne 63 :
    `i has private access in A.InstanceA`
    `i` est privé
14. Ligne 64 : affiche `6 4 2`
15. Ligne 65 :
    `A.B has private access in A`
    `B` est privée dans `A`, invisible donc.
16. Ligne 69 : affiche `1` (à condition d'avoir corrigé les problèmes avec la classe `Locale` c'est-à-dire éliminé `increase1` et `increase2` et remplacé `int k = i` par `int k;`, mais pas la suggestion du constructeur).

Exercice 2 *Questions diverses*

1. Est-ce qu'une classe de niveau supérieur est statique ou non-statique par défaut ? Et une classe membre ?
   Correction : Une classe de niveau supérieur est statique par défaut. C'est parce qu'elle n'a pas de classe englobante, donc il ne fait pas de sens d'avoir une référence implicite à un objet d'une classe englobante.
   Une classe membre est non-statique par défaut, et donc a une référence implicite à son objet de la classe englobante.

Exercice 3 *Itérateurs* – Illustration des classes internes

Le but de cet exercice est de construire plusieurs itérateurs qui parcourent une structure donnée à un rythme différent. Leurs implémentations se feront de manières différentes pour illustrer différentes techniques possibles.

On rappelle qu'un objet qui implémente l'interface `Iterator<Integer>` doit disposer des méthodes :

```
1 boolean hasNext() // returns true if the iteration has more elements
2 Integer next()    // returns the next element in the iteration
```

On travaille sur une classe `Datas` qui encapsule un tableau d'entiers :

```
1  public class Datas {
2    private final static int SIZE = 16;
3    private Integer[] monTableau = new Integer[SIZE];
4    public Datas() {
5      for (int i = 0; i < SIZE; i++) {
6        monTableau[i] = Integer.valueOf(i);
7      }
8    }

10   // on definit une interface interne
11   private interface DatasIterator
12           extends java.util.Iterator<Integer> {}

14   // la methode de base pour afficher en suivant un iterateur
15   private void print(DatasIterator i) {
16     while (i.hasNext()) {
17       System.out.print(i.next() + " ");
18     }
19     System.out.println();
20   }

22   public void printEven() {...}
23   public void printOdd() {...}
24   public void printBackwards() {...}

26   public static void main(String s[]) {
27     Datas d = new Datas();
28     d.printEven();
29     d.printOdd();
30     d.printBackwards();
31   }
32 }
```

1. Définir une classe privée membre interne `EvenIterator` qui implémente l'interface `DatasIterator`. Elle doit permettre de parcourir tous les éléments situé en positions paires de `monTableau`. Puis écrivez la méthode `printEven` qui en construit une instance et fait appel à `print`.

Correction :

```
// dans la classe Datas
private class EvenIterator implements DatasIterator {
  int i = 0;
  public boolean hasNext() { return (i < SIZE); }
  public Integer next() {
    Integer retValue = monTableau[i];
    i += 2;
    return retValue;
  }
}

public void printEven() {
  print(new EvenIterator());
  // equivalent : print(this.new EvenIterator());
}
```

2. Définissez une classe interne locale `OddIterator` directement dans le bloc de la méthode `printOdd()`, puis l'utilisez pour parcourir et afficher les positions impaires du tableau.

Correction :

```
// dans la classe Datas
public void printOdd() {
  class OddIterator implements DatasIterator {
    int i = 1;
    public boolean hasNext() { return (i < SIZE); }
    public Integer next() {
      Integer retValue = monTableau[i];
      i += 2;
      return retValue;
    }
  }
  OddIterator i = new OddIterator();
  print(i);
}
```

3. Pour `printBackward` on souhaite utiliser une classe anonyme qui implémente l'interface `DataIterator`, et qui sera transmise en paramètre à `print`. Le résultat attendu est un affichage dans le sens inverse. Ecrivez cette méthode.

Correction :

```
// dans la classe Datas
public void printBackwards() {
  DatasIterator it = new DatasIterator() {
    int i = SIZE - 1;
    public boolean hasNext() { return (i >= 0); }
```

```
6       public Integer next() {
7         Integer retValue = monTableau[i];
8         i--;
9         return retValue;
10      }
11    };
12    print(it);
13  }
```

Exercice 4 *Comparable* – Illustration des classes internes anonymes On suppose l'interface suivant est déclaré :

```
1  interface Comparateur {
2      boolean plusGrand(int x, int y);
3  }
```

Et nous reprenons le tri à bulles du TP6 dans un style différent. On commence avec le modèle suivant :

```
1  public class Comparer {

3    public static void main(String[] args) {
4      int[] a = { 10, 30, 5, 0, -2, 100, -9 };
5      afficher(a);

7      trier(a, ... );
8      afficher(a);

10     trier(a, ... );
11     afficher(a);

13     trier(a, ... );
14     afficher(a);
15   }

17   static void afficher(int[] x) {
18     for (int i = 0; i < x.length; i++) {
19       System.out.print(x[i] + " ");
20     }
21     System.out.println();
22   }

24   static void trier(int[] x, ... ) {
25     boolean change = false;
26     do {
27       change = false;
28       for (int i = 0; i < ... - 1; i++) {
29         if ( ... ) {
30           ...
```

```
31              change = true;
32            }
33          }
34        } while (change);
35    }
36  }
```

Dans la classe `Comparer` on définit la méthode statique `trier`, qui prend en paramètre un tableau `x` d'entiers à trier, et en second paramètre la méthode à utiliser pour comparer les éléments. Une méthode qu'on a déjà vu en cours pour implémenter cela sont des classes anonymes.

1. Complétez la méthode `trier`. Elle prend un objet d'une classe anonyme de type `Comparateur` en second paramètre.
2. Dans la méthode `main`, complétez les trois appels à `trier`, pour qu'elle trie les éléments
   (a) en ordre croissant,
   (b) en ordre decroissant,
   (c) en ordre croissant de la valeur absolue.

Correction :

```
1  public class Comparer {
2
3    public static void main(String[] args) {
4      int[] a = { -6, 1, -6, 8, 10, 4, 0, -3, 7 };
5      afficher(a);
6
7      trier(a, new Comparateur() {
8        public boolean plusGrand(int x, int y) {
9          return x > y;
10       }
11     });
12     afficher(a);
13
14     trier(a, new Comparateur() {
15       public boolean plusGrand(int x, int y) {
16         return y > x;
17       }
18     });
19     afficher(a);
20
21     trier(a, new Comparateur() {
22       public boolean plusGrand(int x, int y) {
23         return Math.abs(x) > Math.abs(y);
24       }
25     });
26     afficher(a);
27   }
```

```
29    static void afficher(int[] x) {
30      for (int i = 0; i < x.length; i++) {
31        System.out.print(x[i] + " ");
32      }
33      System.out.println();
34    }

36    static void trier(int[] x, Comparateur c) {
37      boolean change = false;
38      do {
39        change = false;
40        for (int i = 0; i < x.length - 1; i++) {
41          if (c.plusGrand(x[i], x[i+1])) {
42            int tmp = x[i];
43            x[i] = x[i+1];
44            x[i+1] = tmp;
45            change = true;
46          }
47        }
48      } while (change);
49    }
50  }
```