

## TD – Séance n°7

### 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++; → marche pas car int i n'est pas static
12         }
13     }
14     // classe interne membre
15     private class B {
16         private int b = 2;
17         private static int bstat = 3; → une classe interne ne peut pas contenir attributs static
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 }
```

1x 11: marche pers car i c la base n'est  
pas static

Une classe statice peut reconnaître



```

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 }

```

## Exercice 2 *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**.
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.
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.

**Exercice 3** *Comparable* – Illustration des classes internes anonymes On suppose que l'interface suivante est déclarée :

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

Et nous reprenons le programme du tri à bulles du TP6 dans un style différent. On rappelle que le tri à bulle consiste à parcourir le tableau autant de fois que nécessaire, tant que lors d'un passage on rencontre au moins une fois 2 voisins qui ne sont pas dans le bon ordre (on les permute).

```
1 public class Comparer {
2
3     public static void main(String[] args) {
4         int[] a = { 10, 30, 5, 0, -2, 100, -9 };
5         afficher(a);
6
7         trier(a, ... );
8         afficher(a);
9
10        trier(a, ... );
11        afficher(a);
12
13        trier(a, ... );
14        afficher(a);
15    }
16
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    }
23
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 un objet qui encapsule la méthode à utiliser pour comparer les éléments.

1. Complétez la méthode `trier`. Elle prend un objet d'une classe anonyme de type `Compareur` 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 décroissant,
  - (c) en ordre croissant de la valeur absolue.