TD n° 2
Licence Informatique (L2)
« Programmation objet avancée »

F. BERTRAND

Année universitaire 2020–2021

Éléments de correction

Concepts abordés :
- Polymorphisme et liaison dynamique
- Transtypage dans une hiérarchie de classes
- Construction d'un objet complexe

1 Polymorphisme élémentaire

Soit la hiérarchie de classes présentée sur la Figure 1 :

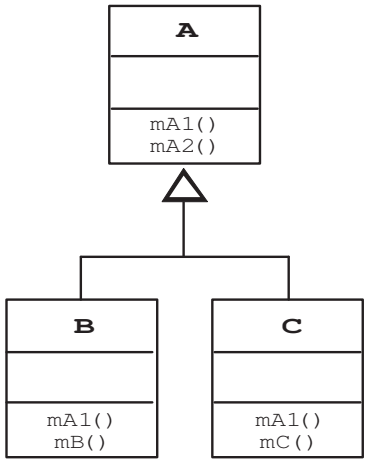

FIGURE 1 – Relations d'héritage entre les classes `A`, `B` ou `C`

Répondre aux questions suivantes :

1. Indiquez la méthode appelée à l'exécution par l'instruction « *x*`.mA1()` » dans le cas où *x* référence soit une instance de `A`, soit une instance de `B`, soit une de `C` ?
2. Indiquez la méthode appelée à l'exécution par l'instruction « *x*`.mA2()` » dans le cas où *x* référence soit une instance de `A`, soit une instance de `B`, soit une `C` ?
3. Dans l'instruction « *x*`.mB()` » que se passerait-il à l'exécution si *x* référençait une instance de `A` ou de `C` ?

Notes

Rappel dans le cas d'un appel `x.m()` : à l'exécution la machine virtuelle détermine le type dynamique de `x` c'est-à-dire la classe de l'objet référencée par `x`. Si cette classe définit localement une méthode `m` alors celle-ci est appelée, sinon la recherche se poursuit dans les super-classes.

1. *Réponse :*
   - *Si* x *référence une instance de `A` alors ce sera la méthode `mA1` de `A` qui sera appelée*
   - *Si* x *référence une instance de `B` alors ce sera la méthode `mA1` de `B` qui sera appelée*
   - *Si* x *référence une instance de `C` alors ce sera la méthode `mA1` de `C` qui sera appelée*
2. *Réponse :*
   - *Si* x *référence une instance de `A` alors ce sera la méthode `mA2` de `A` qui sera appelée*
   - *Si* x *référence une instance de `B` alors ce sera la méthode `mA2` de `A` qui sera appelée*
   - *Si* x *référence une instance de `C` alors ce sera la méthode `mA2` de `A` qui sera appelée*
3. *Il n'y aurait rien qui se passerait à l'exécution car il y aurait une erreur à la compilation. En effet, si le type statique de `x` est `A` ou `C`, la méthode `mB` n'existe dans aucune de ces 2 classes, donc la recherche de `mB()` (à la compilation) échoue.*

2 Polymorphisme avec les méthodes d'instance et de classe

Le but de cet exercice est de montrer la différence de comportement entre les méthodes d'instance et les méthodes de classe vis à vis de l'héritage.

1. Donner la trace d'exécution du code suivant :

```
1  class A {
2    void m1() {
3      System.out.println("m1 de A");
4    }
5    static void m2() {
6      System.out.println("m2 de A");
7    }
8  }
9
10 class B extends A {
11   void m1() {
12     System.out.println("m1 de B");
13   }
14   static void m2() {
15     System.out.println("m2 de B");
16   }
17 }
18
19 public class TestFinal {
20   public static void main(String[] args) {
21     A[] t = {new A(), new B()};
22     for(A elt : t) {
23        elt.m1();
24        elt.m2();
25     }
26   }
27 }
```

2. Que se passerait-il si on modifiait le type du tableau en `B[]` ?
3. Que se passerait-il si on retirait la méthode `m1` de la classe `A` ?
4. Que se passerait-il si on retirait la méthode `m1` de la classe `B` ?

Notes

1. *Trace d'exécution :*

```
m1 de A
m2 de A
m1 de B
m2 de A
```

   Pour déterminer la méthode à appeler le compilateur va :
   - *dans le cas d'une méthode **d'instance**, générer du code pour déterminer **à l'exécution** le type dynamique de l'objet sur lequel la méthode est appelée et, en fonction, du résultat de ce test, ce code appellera la méthode dans la classe correspondant à ce type ;*
   - *dans le cas d'une méthode **de classe**, générer directement le code de l'appel de la méthode correspondant au type statique (déclaré) de l'objet (ici le type du tableau).*

2. *Si le type du tableau était modifié, cela créerait une erreur à la compilation sur le « `new A()` » du tableau car un super-type (`A` ici) ne peut être substitué à un sous-type...*

   ```
   Found 1 semantic error compiling "TestFinal.java":

       21.     B[] t = {new A(), new B()};
                               <----->
   *** Error: The type of the left-hand side (or array type) in this
   initialization (or array creation expression), "B", is not compatible
   with the type of the right-hand side expression, "A".
   ```

3. *Si on retirait la méthode `m1` de la classe `A` il y aurait une erreur à la compilation car le compilateur ne trouverait pas cette méthode qu'il recherche dans la classe `A` (type du tableau `t`)..*
4. *Si on retirait la méthode `m1` de la classe `B` alors ce serait la méthode `m1` de `A` qui serait appelée.*

   Se rappeler de ce principe en Java, si le code se compile c'est qu'il existe quelque part une version de la méthode appelée. Si cette méthode a été redéfinie dans des sous-classes alors la détermination de la version à appeler se fera à l'exécution.

Maintenant on ajoute, à la fin de la méthode `main`, les lignes suivantes :

```
B b1 = (B) t[1];
b1.m1();
b1.m2();
```

Questions :

1. Est-ce que la première ligne se compile correctement ?
2. Si oui, quel sera le résultat de l'exécution des deux lignes suivantes ?...

Notes

1. *Oui car le transtypage, nécessaire pour la compilation, est vérifié à l'exécution et il est valide : le type dynamique de `b[1]` est `B` (`b[1]` stocke une référence à une instance de `B`) et il est logiquement possible de stocker une référence à un objet `B` dans une variable de type `B` (`b1` en l'occurrence).*
2. *À l'exécution :*
   — *à la ligne 2, le choix d'une méthode d'instance s'effectuant en fonction du type dynamique (ici `B`), le message affiché sera « `m1 de B` ».*
   — *à la ligne 3, le choix d'une méthode de classe s'effectuant en fonction du type statique (ici `B` pour la variable `b1`), le message affiché sera « `m2 de B` ».*

Pour terminer on ajoute ces deux dernières lignes :

```
1 B b2 = (B) t[0];
2 b2.m1();
```

Répondre aux mêmes questions posées au point précédent.

Notes

1. *Pour la ligne 1, oui car le transtypage indique au compilateur que `b[0]` stocke une référence à une instance de `B`. Mais ce n'est pas le cas. Cependant le compilateur ne peut pas le détecter lors de la phase de compilation, il se fonde uniquement sur l'indication de transtypage.*
2. *Concernant la ligne 2, elle ne sera jamais exécutée. En effet, à l'exécution, le programme va se terminer immédiatement sur la première ligne (l'affectation), une exception `ClassCastException` va être lancée indiquant que la machine virtuelle a détecté que `t[0]` ne référence pas une instance de `B`, mais une instance de `A`, et donc que l'affectation est impossible.*

3 Construction d'objets...

La construction d'une instance avec Java s'effectue en plusieurs étapes qu'il est intéressant de connaître pour interpréter certaines erreurs.

Soit le code suivant :

```
1  class Str {
2     public Str(char x) { System.out.print(x); }
3  }
4
5  class Base {
6     private static Str s1=new Str('H');
7     public Base() {
8        System.out.print ('I');
9     }
10    private static Str s2=new Str('E');
11    public int methode1(char c) {
12       System.out.print(c);
13       return 1;
14    }
15 }
16
17 class Derive extends Base {
18    private static Str t=new Str('R');
19    private int c = super.methode1('T');
20    public Derive() {
21       System.out.print ('A');
22    }
23    public int methode1(char c) {
24       System.out.print(c);
25       return 1;
26    }
27 }
28
29 public class TestConstruction {
30    private Str b = new Str('S');
31    public static void main(String[] argv) {
32       Base b = new Derive();
33       b.methode1('G');
34       System.out.print('E');
35       //TestConstruction tc = new TestConstruction();
36    }
37 }
```

La première question est : quel est le mot affiché par ce programme ?...

Si maitenant on décommente la ligne 35 :

```
1  TestConstruction tc = new TestConstruction();
```

Que se passe-t-il ?...

À noter qu'il est possible de visualiser les actions effectuées par la machine virtuelle et notamment le chargement des classes en indiquant l'option « `-verbose` » lors de son lancement. Ainsi le code précédent peut être compilé puis exécuté avec les commandes suivantes :

```
1  javac TestConstruction.java        // compilation
2  java -verbose TestConstruction     // exécution par la machine virtuelle
```

Notes

Le mot affiché est « HERITAGE ».

Si on ajoute le code indiqué à la ligne 35 on obtient « HERITAGES » (« S » en plus)

Ci-dessous la fin de la trace d'exécution de ce programme :

```
1  [Opened file:/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/jre/lib/rt.jar]
2  ...
3  [Loaded TestConstruction from file:/tmp/TD/src/]
```

```
4 [Loaded Base from file:/tmp/TD/src/]
5 [Loaded Derive from file:/tmp/TD/src/]
6 [Loaded Str from file:/tmp/TD/src/]
7 HERITAGE
8 ...
```

Java est un langage dynamique dans le sens où il n'y a pas création d'un code binaire exécutable à la compilation (comme en C) mais création, pour chaque classe, d'un code (en anglais bytecode*) destiné à être interprété par la machine virtuelle (sorte de processeur « logiciel » et non matériel).*

La machine virtuelle charge d'abord automatiquement un ensemble de classes nécessaires à tout programme Java (présentes sous forme compressées dans `rt.jar`[1] *(rt =* runtime*) puis charge les classes présentes dans le fichier Java compilé. Pour rappel, plusieurs classes Java peuvent être déclarées dans le même fichier (même si ce n'est pas une bonne pratique) mais une seule peut être déclarée* `public` *et dans ce cas, le nom du fichier doit correspondre au nom de la classe qualifiée* `public`*.*

Le chargement de ces classes se fait selon leur dépendance (quelle classe a besoin de quelles(s) autre(s) classe(s)). Ainsi la première classe chargée est celle qui contient la méthode `main` *: la classe* `TestConstruction` *(ligne 3 de la trace d'exécution).*

L'exécution des instructions du `main` *peut se décomposer en plusieurs étapes :*

1. *Chargement de la classe contenant la méthode* `main`*. Dans le code de cette classe, le compilateur a indiqué les classes nécessaires à son exécution.*
2. *Puis ces classes sont donc chargées par ordre de dépendance. Dans notre cas, les classes sont chargées dans l'ordre suivant (cf. trace d'exécution donnée ci-dessus) :*
   (a) `Base`
   (b) `Derive`
   (c) `Str`

   Lors du chargement de chaque classe, il y a initialisation de ses variables de classes (`static`*) ce qui donne les affichages suivants :*
   (a) *'H' pour l'initialisation* `s1` *dans* `Base`*;*
   (b) *'E' pour l'initialisation de* `s2` *dans* `Base`*;*
   (c) *'R' pour l'initialisation de* `t` *dans* `Derive`*;*
3. *Exécution de la ligne 32 avec allocation de la mémoire nécessaire à l'instance de* `Derive` *qui est créée (ligne 32) mais aucun affichage ne traduit ce fait;*

   Puis pour chaque classe impliquée dans la création de l'instance :
   — *Pour* `Base` *:*
      (a) *Initialisation de toutes les variables d'instance avec leurs valeurs par défaut : pour* `Base` *il n'y en a pas;*
      (b) *puis exécution du constructeur appelé : affichage de 'I' pour le constructeur de* `Base`*;*
   — *Pour* `Derive` *:*
      (a) *Initialisation de toutes les variables d'instance avec leurs valeurs par défaut : pour* `Derive` *il y a l'attribut* `c` *ce qui provoque l'affichage de 'T';*
      (b) *puis exécution du constructeur appelé : affichage de 'A' pour le constructeur de* `Base`*;*
4. *Enfin exécution des autres instructions du* `main` *(lignes 33 et 34) qui provoquent l'affichage de 'G' puis 'E'*

1. Un fichier jar (Java ARchive) est un regroupement en un seul fichier de plusieurs (pour `rt.jar`, de milliers) classes Java compilées et compressées afin de réduire la taille du fichier.

Donc pour résumer :

1. *L'initialisation des variables d'instance (attributs) lors de leur définition s'effectue avant les initialisations contenues dans le constructeur.*
2. *Ces initialisations peuvent contenir une exécution de code, pas simplement l'affectation d'une valeur, comme à la ligne 19 où il y a appel d'une méthode*

Donc pour être sûr de l'ordre des initialisations, il vaut mieux initialiser l'ensemble des attributs dans le constructeur plutôt que de mixer les différents types d'initialisations...

Dernier point, la variable d'instance `b` *(ligne 30) dans la classe* `TestConstruction` *ne provoquera aucun affichage car aucune instance de cette classe n'a été sera créée.*

Ce sera fait lorsque la ligne 35 sera décommentée et exécutée.