Université Paris Cité

Compléments en Programmation Orientée Objet
TP n° 3
(Correction)

Indication : avec le cours sous les yeux les trois premiers exercices doivent être fait rapidement.

1 Les bases du polymorphisme

Exercice 1 : Définir le sous-typage entre types objet

On suppose déjà définies :

```
1 class A {}
2 class B {}
3 interface I {}
4 interface J {}
5
```

Voici une liste de déclarations :

```
1 class C extends I {}
2 interface K extends B {}
3 class C implements J {}
4 interface K implements B {}
5 class C extends A implements I {}
6 interface K extends I, J {}
7 class C extends A, B {}
8 class C implements I, J {}
9
```

Lesquelles sont correctes ?

Correction : Sont correctes les lignes 3, 5, 6 et 8.
Ligne 1 : incorrecte car une classe n'étend pas une interface.
Ligne 2 : incorrecte car une interface n'étend pas une classe.
Ligne 4 : incorrecte car une interface n'implémente pas une classe.
Ligne 7 : incorrecte car l'héritage de classes multiples est interdit.

Exercice 2 : Transtypage

```
1
2  class A { }
3
4  class B extends A { }
5
6  class C extends A { }
7
8  public class Tests {
9      public static void main(String[] args) {
10         System.out.println((int)true);
11         System.out.println((int) 'a');
12         System.out.println((byte) 'a');
13         System.out.println((byte) 257);
14         System.out.println((char) 98);
15         System.out.println((double) 98);
16         System.out.println((char) 98.12);
17         System.out.println((long) 98.12);
18         System.out.println((boolean) 98.);
19         System.out.println((B) new A());
20         System.out.println((C) new B());
21         System.out.println((A) new C());
22     }
23 }
24
```

Dans la méthode `main()` ci-dessus,

1. Quelles lignes provoquent une erreur de compilation ?

   Correction : 10 (conversion de booléen), 18 (conversion de booléen), 20 (conversion sans qu'un type soit sous-type ou supertype de l'autre).

Les autres lignes ne posent pas de problème à la compilation.

2. Après avoir supprimé ces dernières, quelles lignes provoquent une exception à l'exécution ?

   Correction : (en gardant la numérotation actuelle)
   Seules les conversions d'objet vers une sous-classe (*downcasting*) sont susceptibles de provoquer une erreur à l'exécution. Donc seule la ligne 19 peut être concernée (on convetit de `A` vers `B`).
   Et effectivement, à la ligne 19, on voit que ça va nécessairement provoquer une erreur car la valeur de `new A()` à l'exécution sera toujours un objet de type `A`, donc non-utilisable en tant qu'objet de type `C` (il pourrait, par exemple, manquer des attributs).

3. Après les avoir enlevées, elles aussi, quels affichages provoquent les lignes restantes ?

   Correction : Exécutez pour voir. Les seules valeurs qu'une conversion de type est susceptible de modifier "physiquement" sont celles des types valeur. En cas de *downcasting*, on va perdre de l'information, observez les conséquences.
   Enfin, que ce soit une conversion vers le haut ou vers le bas, la façon d'afficher peut changer énormément d'un type à l'autre (cas le plus criant : la conversion de/vers un nombre vers/de un `char`).
   Pour les types référence, si l'objet "converti" est le récepteur (à gauche du point) de la méthode appelée (c'est le cas ici : on appelle implicitement `toString()`), à cause de la liaison dynamique, ça ne change absolument pas la méthode qui sera appelée (elle dépend uniquement de l'objet réel, pas du type que le compilateur associe à son expression). Si l'expression objet apparaît en tant qu'argument d'une méthode, en cas de surcharge, le résultat peut changer (la surcharge est résolue en regardant le type statique de l'expression).

Exercice 3 : Surcharge

```
1  class A {};
2  class B extends A {};
3  class C extends B {};
4  class D extends B {};
5
6  public class Dad {
7      public static void f(A a, A aa) { System.out.println("Dad : A : A"); }
8      public static void f(A a, B b) { System.out.println("Dad : A : B"); }
9  }
10 public class Son extends Dad {
11     public static void f(A a, A aa) { System.out.println("Son : A : A"); }
12     public static void f(C c, A a) { System.out.println("Son : C : A"); }
13
14     public static void main(String[] args) {
15         f(new B(), new A());
16         f(new D(), new A());
17         f(new B(), new D());
18         f(new A(), new C());
19     }
20 }
```

Dans la méthode `main()` ci-dessus,

1. Quels affichages provoquent les lignes 15 à 18 ?

   Correction : Les lignes 15 et 16 provoquent l'affichage *Son : A : A* et les autres lignes provoquent l'affichage *Dad : A : B*.

2. Que se passe-t-il si on appelle `f(new C()),new C())` ? `f(new C(),new B())` ?

   Correction : Dans les deux cas, les trois signatures possibles de `f` `((A,A),(A,B),(C,A))` subsument la signature d'appel, mais `(A,B)` ne subsume pas `(C,A)` et inversement, ce qui entraine une erreur : appel ambigu.

3. Dans la classe `Son` comment être sûr d'appeler les méthodes `f` de la classe `Dad` ? Quels types de paramètres permettent d'appeler la fonction `f` avec signature `(A,A)` ?

   Correction : Vu que les fonctions `f` sont statiques il suffit d'écrire `F.f(x,y)`. Pour appeler la fonction avec la signature `(A,A)`, il faut que le type de `x` et `y` soit un sous-type de `A` mais pas de `B`.

2 Les dessous du transtypage

Exercice 4 : Transtypages primitifs

Voici un programme (`TranstypagesPrimitifs.java` sur Moodle) :

```
1  public class TranstypagesPrimitifs {
2      public static void main(String[] args) {
3          int vint = 1234567891;
4          short vshort = 42;
5          float vfloat = 9.2E11f;
6          System.out.println("vint = " + vint +
7              ", vshort = " + vshort +
8              ", vfloat = " + vfloat);
9
10     }
11 }
```

1. Compilez et exécutez ce programme (assurez-vous de comprendre la notation `9.2E11f`).
2. Nous allons regarder superficiellement le code-octet produit : dans un terminal, allez dans le répertoire où se trouve `TranstypagesPrimitifs.class` et tapez la commande "`javap -c -v TranstypagesPrimitifs`". Le code-octet apparaît ainsi sous une forme désassemblée quasi lisible. Nous nous intéresserons en particulier au début de la partie `Code :`, qui correspond à la déclaration et l'initialisation de nos trois variables. On peut repérer l'appel à l'instruction suivante, `println`, par l'instruction **getstatic** dans le code-octet.

   Il n'y a donc que 6 ou 7 lignes à regarder. Constatez que certaines variables sont initialisées par une séquence d'instructions comme : **bipush** `42;`**istore_2**, alors que d'autres ont la séquence **ldc** suivie de **istore** ou **fstore** (le i ou le f désigne clairement un type)
3. Nous allons nous intéresser à la façon dont sont fait les transtypages. Ajoutez une ligne avant l'instruction d'affichage : `vint=vshort;` et interpréter les opérations `load,store` qui apparaissent.

   Avec les 3 variables présentes il y a théoriquement 6 transtypages, certains qu'il faut rendre explicites. Essayez les tous et complétez le tableau ci-dessous avec vos remarques.

   Le tableau :

=	vint	vshort	vfloat
vint	XXX		
vshort		XXX	
vfloat			XXX

   Relevez, notamment, dans chaque cas :
   - si ça compile directement, s'il ajouter un *cast* explicite, etc. ;
   - la nature des instructions ajoutées dans le code-octet (notez que les instructions de la forme `f2i` expriment un changement de type) ;
   - l'affichage produit après conversion.

Exercice 5 : Transtypages d'objets (sur machine)

Même exercice que le précédent mais sur le programme suivant :

```java
public class TranstypagesObjets {
    public static void main(String[] args) {
        Object vObject = new Object();
        Integer vInteger = 42;
        String vString = "coucou";
        System.out.println("vObject = " + vObject + ", vInteger = "
            + vInteger + ", vString = " + vString);
    }
}
```

Différence, vous ne verrez plus l'ajout de l'instruction **u2t** mais parfois celle de **checkcast**. Dans quels cas ?
Dans certains cas vous aurez eu besoin, pour compiler, d'un *cast* explicite. Lesquels ? Est-ce les-mêmes que dans la question précédente ?
Dans certains cas, le programme quittera sur `ClassCastException`, lesquels ?

Correction : Tout est écrit explicitement dans le cours.

Exercice 6 : Transtypages mixtes (sur machine)

Faites le même travail sur le programme suivant. Remarquez les instructions, insérées par le compilateur, qui correspondent au boxing et à la vérification de types.

```java
public class TranstypagesMixtes {
    public static void main(String[] args) {
        Object vObject = Integer.valueOf(9);
        Integer vInteger = 42;
        int vint = 111;
        System.out.println("vObject = " + vObject +
            ", vInteger = " + vInteger +
            ", vint = " + vint);

    }
}
```

Correction : Idem : ceci est couvert par le cours. On peut observer ici les appels de méthode que Java insère pour réaliser le *boxing* et l'*autoboxing*.