TD6 : Héritage et concordance des types

POO : Programmation Orientée Objet
Première année

1 Concordance des types : type statique et type dynamique

Concordance de types :
Un type `B` concorde avec un type `A` si l'une des conditions suivantes est vérifiée :
- `A` et `B` sont le même type (*réflexivité de la relation de concordance*).
- `B` hérite de `A`
- il existe un type `T`, tel que `B` concorde avec `T` et `T` concorde avec `A` (*transitivité de la relation de concordance*).

Soient deux variables `a` et `b` de types respectifs `A` et `B` tels que `B` concorde avec `A` mais `A` ne concorde pas avec `B`,
- l'affectation `a=b` est admise par le compilateur,
- l'affectation `b=a` est refusée.

Concernant le passage de paramètres,
- si il existe une méthode `p(A a)`, l'appel `p(b)` est accepté par le compilateur,
- alors pour une méthode `q(B b)` l'appel `q(a)` sera refusé par le compilateur.

On considère la définition des classes A et B suivantes :

```
1  class Animal {
2    public Animal f(Animal z, Canard t){
3      Canard unCanard;
4      if (!this.estUneSorteDe(z))
5        return z;
6      if (t.estUnCanard())
7        return t;
8
9      unCanard = new Canard();
10     return unCanard;
11   }
12
13   public boolean estUneSorteDe(Animal x){
14     return !(x.estUnCanard());
15   }
16
17   public boolean estUnCanard(){
18     return false;
19   }
20 }
21
22 class Canard extends Animal {
23   public boolean estUneSorteDe(Animal x){
24     return true;
25   }
26
27   public boolean estUnCanard(){
28     return true;
29   }
30 }
```

▷ **Question 1.** En considérant les 16 combinaisons possibles de types des variables `x`, `y`, `z` et `t`, déterminer les erreurs de compilation de l'instruction `x = y.f(z,t)`.

▷ **Question 2.** Étudier le type dynamique du résultat de la méthode `f()`, en déduire une post-condition.

2 Utilisation de `instanceof`

Rappel : `instanceof` est un opérateur qui prend à sa gauche un objet et à sa droite un nom de classe ou d'interface. Il retourne `true` si l'objet passé en première opérande est du type donné en seconde opérande. En interne le calcul est effectué est identique à tenter un transtypage forcé (*cast*) et regarder si une exception `ClassCastException` est levée.

▷ **Question 3.** Déterminer l'affichage du programme suivant :

```
class Animal {}

class Canard extends Animal {}

class TestInstanceOf {

  public static void main(String args[]) {
    Object c = new Canard();
    Animal a = new Animal();
    System.out.println("c est une instance de Canard : " + (c instanceof Canard));
    System.out.println("c est une instance de Animal : " + (c instanceof Animal));
    System.out.println("c est une instance de Object : " + (c instanceof Object));
    System.out.println("c est une instance de Integer : " + (c instanceof Integer));
    System.out.println("a est une instance de Canard : " + (a instanceof Canard));
    System.out.println("a est une instance de Animal : " + (a instanceof Animal));
    System.out.println("a est une instance de Object : " + (a instanceof Object));
    System.out.println("a est une instance de Integer : " + (a instanceof Integer));
  }
}
```

▷ **Question 4.** En utilisant l'opérateur de transtypage (*cast*) écrivez une suite d'instructions permettant de simuler le comportement de l'instruction `instanceof`.

3 Constructeurs et héritage

Rappel : Le constructeur d'une sous-classe doit faire appel au constructeur de la super-classe. Dans le cas, où le constructeur ne fait pas appel explicitement à un constructeur de la super-classe, le compilateur générera de manière implicite un appel au constructeur sans paramètre de la super-classe.

On considère la définition des classes suivantes :

```
class A {
  A() {
    System.out.println("constructeur de A");
  }
}

class B extends A {
  B() {
    System.out.println("constructeur de B");
  }

  B(int x) {
    this();
    System.out.println("autre constructeur de B");
  }
}

class C extends B {
  C() {
    super(3);
    System.out.println("constructeur de C");
  }

  public static void main(String [] arg) {
    new C();
  }
}
```

▷ **Question 5.** Déterminer l'affichage de la méthode principale `void main(String args[])`.

▷ **Question 6.** Même question en supposant que l'on supprime l'instruction `this()` dans la classe `B`.

▷ **Question 7.** Même question si l'on supprime l'instruction `super(3)` dans la classe `C`. (Remarque : on supposera que l'instruction `this()` de la question précédente est supprimée.)

▷ **Question 8.** Même question si l'on supprime complètement la définition du constructeur sans paramètre de la classe `B`. (Remarque : cette fois encore l'instruction `super(3)` n'a pas été ré-introduite.)

4 Typage et héritage

Algorithme simplifié de sélection de la méthode à appeler :
À la compilation :
- Regarder le type statique du receveur de l'appel de méthode (type de la référence à la déclaration).
- Sélectionner tous les profils des méthodes définies dans la classe correspondant à ce type. Cette sélection se base sur le nombre de paramètres et les types statiques de ces paramètres. Le type de retour de la méthode n'a aucune influence.

À l'exécution :
- Regarder le type dynamique du receveur de l'appel de méthode (type de l'objet référencé).
- Sélectionner dans la classe correspondant à ce type, la méthode dont le profil correspond à une des méthodes candidates sélectionnées lors de la phase de compilation. Si une telle méthode n'existe pas remonter dans la super-classe et re-exécuter la sélection.

On considère le programme suivant :

```
class Animal {
  void m(Animal a) {
    System.out.println("m de Animal");
  }
  void n(Animal a){
    System.out.println("n de Animal");
  }
}

class Canard extends Animal {
  void m(Animal a){
    System.out.println("m de Canard");
  }
  void n(Canard c){
    System.out.println("n de Canard");
  }
}

class Test {
  public static void main(String[] argv){
    Animal a = new Canard();
    Canard c = new Canard();
    Animal a1 = new Animal();
    a.m(c);
    a.n(c);
    c.m(c);
    c.n(c);
    a.m(a1);
    a.n(a1);
    a1.m(c);
    a1.n(c);
  }
}
```

▷ **Question 9.** Déterminer l'affichage lors de l'exécution de ce programme.

▷ **Question 10.** Déterminer le nouvel affichage si l'on ajoute dans la classe `Animal` une méthode `m(Canard c)` dont la définition est la suivante :

```
void m(Canard c) {
  System.out.println("m de Animal version 2");
}
```