

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 A {
2     public A f(A z, B t){
3         B unB;
4         if (!this.estUneSorteDe(z))
5             return z;
6         if (t.estUnB())
7             return t;
8
9         unB = new B();
10        return unB;
11    }
12
13    public boolean estUneSorteDe(A x){
14        return !(x.estUnB());
15    }
16
17    public boolean estUnB(){
18        return false;
19    }
20 }
21
22 class B extends A {
23     public boolean estUneSorteDe(A x){
24         return true;
25     }
26
27     public boolean estUnB(){
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 trans-typage forcé (*cast*) et regarder si une exception `ClassCastException` est levée.

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

```

1 class A {}
2
3 class B extends A {}
4
5 class TestInstanceOf {
6
7     public static void main(String args[]) {
8         Object b = new B();
9         A a = new A();
10        System.out.println("b est une instance de B : " + (b instanceof B));
11        System.out.println("b est une instance de A : " + (b instanceof A));
12        System.out.println("b est une instance de Object : " + (b instanceof Object));
13        System.out.println("b est une instance de Integer : " + (b instanceof Integer));
14        System.out.println("a est une instance de B : " + (a instanceof B));
15        System.out.println("a est une instance de A : " + (a instanceof A));
16        System.out.println("a est une instance de Object : " + (a instanceof Object));
17        System.out.println("a est une instance de Integer : " + (a instanceof Integer));
18    }
19 }

```

▷ **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 :

```

1 class A {
2     A() {
3         System.out.println("constructeur de A");
4     }
5 }
6
7 class B extends A {
8     B() {
9         System.out.println("constructeur de B");
10    }
11
12    B(int x) {
13        this();
14        System.out.println("autre constructeur de B");
15    }
16 }
17
18 class C extends B {
19     C() {
20         super(3);
21         System.out.println("constructeur de C");
22    }
23
24    public static void main(String [] arg) {
25        new C();
26    }
27 }

```

▷ **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 :

```

1 class A {
2     void m(A a) {
3         System.out.println("m de A");
4     }
5     void n(A a){
6         System.out.println("n de A");
7     }
8 }
9
10 class B extends A {
11     void m(A a){
12         System.out.println("m de B");
13     }
14     void n(B b){
15         System.out.println("n de B");
16     }
17 }
18
19 class Test {
20     public static void main(String[] argv){
21         A a = new B();
22         B b = new B();
23         A a1 = new A();
24         a.m(b);
25         a.n(b);
26         b.m(b);
27         b.n(b);
28         a.m(a1);
29         a.n(a1);
30         a1.m(b);
31         a1.n(b);
32     }
33 }
```

▷ **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 A une méthode `m(B b)` dont la définition est la suivante :

```

void m(B b) {
    System.out.println("m de A version 2");
}
```