Limitations des types entiers et réels

V. Lepetit, J. Sam, et J.-C. Chappelier

1 Limitations des types entiers

Le type `int` n'est pas le seul type qu'on peut utiliser pour déclarer des variables contenant des valeurs positives. Il existe aussi les types : `long` et `short`. Par exemple, on peut écrire :

```
int m;
long n;
short p;
```

Cependant, tous ces types ne peuvent représenter que des valeurs comprises dans un certain intervalle. Pour donner une idée de l'ordre de grandeur, voici les intervalles utilisés par Java :

type	valeur minimale	valeur maximale
`short`	-32'678	+32'767
`int`	-2'147'483'648	+2'147'483'647
`long`	environ -10^{18}	$+10^{18}$

Plus l'intervalle est grand, plus une variable du type correspondant occupera de place en mémoire.

Pour voir l'impact que peuvent avoir ces limitations en pratique, vous pouvez exécuter le programme suivant [1] :

```
public class Depassement {

    public static void main(String[] args) {

      int n = 10;

      System.out.println( n );
      n = n * n;
      System.out.println( n );
```

1. Ce programme pourra se réécrire de façon plus compacte dès que vous aurez vu les boucles.

```
        n = n * n;
        System.out.println( n );
        n = n * n;
        System.out.println( n );
        n = n * n;
        System.out.println( n );
        n = n * n;
        System.out.println( n );

    }

}
```

Ce programme initialise la variable `n` à 10, et l'élève au carré plusieurs fois. Les valeurs affichées devraient donc être toutes des puissances de 10, mais en exécutant le programme, vous verrez que ce n'est pas le cas pour les dernières valeurs, qui sont trop grandes pour être représentées correctement par le type `int`. Vous pouvez également changer le type de `n` de `int` à `long` et `short` pour voir l'impact sur les valeurs calculées.

Soyez donc vigilants quand votre programme doit travailler avec de grandes valeurs !

2 Limitations des types réels

De la même façon, le type `double` ne peut pas représenter n'importe quel nombre réel, puisqu'il faudrait pour cela une précision infinie. Il existe également le type `float` qui est plus limité que le type `double`, mais prend moins de place en mémoire. Voici les intervalles de valeurs pour ces types :

type	valeur minimale	valeur maximale
`float`	$-3.4\ 10^{38}$	$+3.4\ 10^{38}$
`double`	$-1.8\ 10^{308}$	$+1.8\ 10^{308}$

Mais en pratique, la limitation de ces types affecte surtout la *précision* des valeurs représentées : les valeurs réelles, y compris celles entre les intervalles donnés ci-dessus, ne peuvent pas toutes être représentées. Considérons le programme suivant :

```
public class Imprecision {

    public static void main(String[] args) {

        double a = 37.0;
        double racine = Math.sqrt(a);

        System.out.println( a - racine * racine );
```

```
    }
}
```

Si le type `double` pouvait représenter parfaitement les valeurs réelles, ce programme afficherait 0. Or, sur mon ordinateur, j'obtiens environ la valeur $7\ 10^{-15}$, qui est une valeur très petite mais pas nulle. C'est parce que la variable `racine` ne peut stocker exactement la racine de `a`, et donc l'expression `racine * racine` ne vaut pas exactement `a`.

C'est pour ça que les tests d'égalité ou d'inégalité entre `double` (ou `float`) NE DEVRAIENT PAS ÊTRE utilisés [2]. Par exemple, le code suivant :

```
double a = 37.0;
double racine = Math.sqrt(a);

if (a == racine * racine) {
  System.out.println( "ok" );
}
```

n'affiche rien, contrairement à ce qu'on pourrait s'attendre. Si vous devez absolument comparer des valeurs de type `double` vous pouvez utiliser un test tel que celui-ci :

```
double a = 37.0;
double racine = Math.sqrt(a);

if (abs(a - racine * racine) < epsilon) {
  System.out.println( "ok" );
}
```

où `epsilon` est une très petite valeur et `abs` calcule la valeur absolue. Cette valeur devrait être choisie selon la précision du type utilisé, mais comment déterminer cette valeur idéalement sort largement du cadre de ce cours.

2. Nous utilisons parfois de tels tests dans notre cours, mais uniquement par souci de simplicité.