Informatique - Java

Nicolas Rousset

Java

Java - basique

Variables & type

Une variable est objet auquel on associe une valeur, et qui, comme son nom l'indique, peut varier.
Exemple de variable :

```
int a = 5; // On crée la variable
while a < 10: // on utilise sa valeur courante
    a = a + 1 // on change sa valeur
```

Lorsque l'on fait appel à la variable ensuite, elle est remplacée par sa valeur courante, qui peut avoir changer depuis sa déclaration.

Type de base d'une variable

Les 4 types fondamentaux que l'on manipule sont :

int : entier, 0, 1, 2, 3
double : nombre à virgule flottante, 3.14, 15,5, 1.0
boolean : true ou false
String : chaîne de caractère, représente du texte

La plupart des types que l'on définit soit-même sont des compositions de ces types + des containers.

Type d'une variable (2)

Le type d'une variable est sa nature, elle détermine la façon dont elle est stockée dans la mémoire de l'ordinateur et les opérations possibles avec.
Par exemple on ne peut pas multiplier deux chaines de caractères entre elles (même si elles représentent des entiers) mais on peut multiplier des entiers entre eux.
De même les méthodes attendent en entrée des arguments d'un type particulier, et renvoit des valeurs d'un type fixé.

Par exemple les trois opérations suivantes n'aboutissent pas au même résultat :

```
int a = 2147483647;
a = a + 100;
System.out.println( a ); // affiche -2147483549
  // a est dévenu négatif !
float b = 2147483647;
b = b + 100;
System.out.println( b ); // affiche 2.14748365E9
  // la valeur de b n'a pas changé !
String c = "2147483647";
c = c + 100;
System.out.println( c ); // affiche 2147483647100
```

On observe ici deux opérations proches qui aboutissent toutes les deux à des résultats erronés, et une troisième qui effectue quelque chose de totalement différent.
Les raisons de ces résultats erronés sont dues aux limitations des représentations de ces deux types numériques, que nous verrons plus tard. Ce qu'il faut retenir ici c'est qu'une même opération ne donne pas forcément le même résultat selon le type de la variable.

Sauf quelques cas particuliers (notamment pour la lisibilité du code) une variable dont la valeur ne change jamais est inutile et peut être remplacer par sa valeur.

Type d'une variable (3)

Ce sont bien les types et pas les valeurs qui déterminent les opérations

```
int[] a = new int[]{1,5,9};

int nombre = a[0]; // => on demande la première valeur
int nombre_2 = a[0.0]; // Erreur de compilation
```

Typage statique

Java est un langage dit à typage statique, cela signifie que le type de chaque variable est déterminée à la compilation (avant l'exécution du programme) et ne peut plus changer.
Il se rapproche pour cela des langages de la famille du C (C / C++ / C# / objective-C) et se distingue des langages de script.
Le type d'une variable est déclaré en même temps que la variable elle-même, par exemple :

```
int    a = 0;   // entier
long   b = 0;   // entier sur 64 bits
double d = 0.0; // nombre à virgule flottante
  // sur 64 bits -> plus précis
char   e = 'a'; // caractère
String f = "Bonjour"; // chaîne de caractère
```

Ce sont la plupart des types de base (ou primitifs). On peut ensuite retrouver une quasi-infinité de type différent en java, sachant que l'on peut définir soi-même ces types (on appelle cela des classes dans ce cas).

Opérateurs

Les opérateurs désignent les symboles arithmétiques :

symbole de l'opérateur	type	signification
+	opérateur arithmétique	addition
-	opérateur arithmétique	soustraction
*	opérateur arithmétique	multiplication
/	opérateur arithmétique	division
>=	opérateur de comparaison	supérieur ou égal
==	opérateur de comparaison	égalité / identité
!=	opérateur de comparaison	différence
!	opérateur logique	négation
&&	opérateur logique	et
||	opérateur logique	ou

[1] => aussi appelé reste de la division entière

Limite des types numériques

Les principales limites à connaître sur les types sont :

int => peut aller de -2147483648 à 2147483647 (environ - 2 milliards à 2 milliards)
long => peut aller de environ - 8 milliards à 8 milliards
double => précision relative maximum de l'ordre de 10^-15 (ie 1 + 10^-16 = 1)

int et Integer, long et Long

Ces types sont largement interchangeable :

```
int     entier1  = 1;
Integer integer1 = 1;
int     entier2  = integer1;
Integer integer2 = entier1;
```

Les conversions sont implicites. Nous verrons les différences plus tards. Retenez pour l'instant :

1/ Les types avec majuscule ont beaucoup plus de méthodes associées (parseLong / parseInt)
2/ Les listes et autres structures ne peuvent pas contenir de type primitif
3/ Il y a des différences avec le symbôle "=="

Conversion

String => int

```
String s = "1000";
int i = Integer.parseInt(s);
```

String => long

```
String s = "1000";
Long i = Long.parseLong(s);
```

int => String

```
int i = 1000;
String s = "" + i;
```

Tous les objets sont convertibles

Problème tordus ...

On peut avoir des soucis avec les surcharges de méthode de type

```
public class Main
{
    public static Long multiplier(Long l1, Long l2)
    {
        return l1*l2;
    }

    public static Long multiplier(long l1, long l2)
    {
        return l1*l2;
    }
}
```

Les classes de base

On peut définir des classes pour associer des attributs et des méthodes de manipulation de ces attributs :

```
public class Personne
{
    private String prenom;
    private String nom;
    private int age;
}
```

Il s'agit d'une classe possédant 3 attributs, un prénom sous forme de champ texte, un nom sous forme de champ texte, un age.

Les classes de base (2)

private => par défaut on n'expose pas les attributs (pas d'accès ou de modification directe)

Du coup on définit des méthodes associées pour accéder et modifier les valeurs. Un attribut peut n'avoir ni l'un ou l'autre.

```
public class Personne
{
    // Convention de nommage, les noms commencent par une minuscule
    private String prenom;
    private String nom;
    private int age;

    // convention de nommage, get + le nom de l'attribut avec une majuscule au début
    public String getNom()
    {
        return nom;
    }


    public void setNom(String nom)
    {
        this.nom = nom;
    }
}
```