Java - Les types de données

Septembre 2015

Les types primitifs

Java est un langage orienté objet, c'est-à-dire qu'il manipule des classes, ou plus exactement des objets, qui sont des instances de ces classes. Les données manipulées avec Java, et que l'on utilise au travers de variables, sont donc typées, le type d'un objet correspond à la classe qu'il instancie.

Toutefois il existe quelques types primitifs, permettant de manipuler directement les données les plus courantes. Ces données sont notamment spécifiées par une représentation en mémoire, et donc à un nombre d'octets prédéfinis.

Voici un tableau répertoriant les huit types primitifs du langage Java :

Type	Signification	Taille (en octets)	Plage de valeurs acceptées
char	Caractère Unicode	2	'\u0000' ? '\uffff' (0 à 65535)
byte	Entier très court	1	-128 ? +127
short	Entier court	2	-32 768 ? +32 767
int	Entier	4	-2^{31}?$-2{,}147\times10^{9}$? $+2^{31}-1$?$2{,}147\times10^{9}$
long	Entier long	8	-2^{63}?$-9{,}223\times10^{18}$? $+2^{63}-1$?$9{,}223\times10^{18}$
float	Nombre réel simple	4	$\pm2^{-149}$?1.4×10^{-45} ? $\pm2^{128}-2^{104}$?3.4×10^{38}
double	Nombre réel double	8	$\pm2^{-1074}$?$4{,}9\times10^{-324}$? $\pm2^{1024}-2^{971}$?$1{,}8\times10^{308}$
boolean	Valeur logique (booléen)	1	true (vrai), ou false (faux)

Les enveloppeurs (Wrappers)

Chacun des types primitifs peut être "enveloppé" dans un objet provenant d'une classe prévue à cet effet et appelée *Wrapper* (mot anglais signifiant *enveloppeur*). Les enveloppeurs sont donc des objets représentant un type primitif.
Avantages :

- Les Wrapper peuvent être utilisés comme n'importe quel objet, ils ont donc leurs propres méthodes.

Inconvénients :

- L'objet enveloppant utilise plus d'espace mémoire que le type primitif. Par exemple, un int prends 4 octets en mémoire mais un Integer utilisera 32 octets sur une machine virtuelle en 64 bits (20 octets en 32 bits).
- L'objet enveloppant est immuable, c'est à dire qu'il ne peut pas être modifié, toute modification de sa valeur nécessite de créer un nouvel objet et de détruire l'ancien, ce qui augmente le temps de calcul.

Voici la liste des enveloppeurs de chaque type primitif Java :

Enveloppeur	**Type primitif**
Character	char
Byte	byte
Short	short
Integer	int
Long	long
Float	float
Double	double
Boolean	boolean

Nombre entier (*byte, short, int, long*)

Un nombre entier est un nombre sans virgule qui peut être exprimé dans différentes bases :

- Base décimale: L'entier est représenté par une suite de chiffre unitaires (de 0 à 9) ne devant pas commencer par le chiffre 0
- Base hexadécimale: L'entier est représenté par une suite d'unités (de 0 à 9 ou de A à F (ou a à f)) devant commencer par *0x* ou *0X*
- Base octale: L'entier est représenté par une suite d'unités (incluant uniquement des chiffres de 0 à 7) devant commencer par *0*

Lorsqu'un nombre est trop grand pour être représenté par un int, il faut explicitement le déclarer comme étant un long en lui rajoutant un L :

```
long n = 9876543210L;
```

Les entiers sont signés par défaut, cela signifie qu'ils comportent un signe. Pour stocker l'information concernant le signe (en binaire), les ordinateurs utilisent le complément à deux

Nombre à virgule (*float, double*)

Un nombre à virgule flottante est un nombre à virgule, il peut toutefois être représenté comme :

- un entier décimal : 895
- un nombre à virgule (en utilisant la notation américaine avec un point) : 845.32
- un nombre exponentiel, c'est-à-dire un nombre (éventuellement à virgule) suivi de la lettre *e* (ou *E*), puis d'un entier correspondant à la puissance de 10 (signé ou non, c'est-à-dire précédé d'un + ou d'un -)

2.75e-2 35.8E+10 .25e-2

En réalité, les nombres réels sont des nombres à virgule flottante, c'est-à-dire un nombre dans lequel la position de la virgule n'est pas fixe, et est repérée par une partie de ses bits (appelée l'exposant), le reste des bits permettent de coder le nombre sans virgule (la mantisse).

Les nombres de type **float** sont codés sur 32 bits dont :

- 23 bits pour la mantisse
- 8 bits pour l'exposant
- 1 bit pour le signe

Les nombres de type **double** sont codés sur 64 bits dont :

- 52 bits pour la mantisse
- 11 bits pour l'exposant
- 1 bit pour le signe

La précision des nombres réels est approchée. Elle dépend du nombre de décimales, elle sera au moins :

- de 6 chiffres significatifs pour le type **float**
- de 15 chiffres significatifs pour le type **double**

Lorsque l'on écrit directement une valeur flottante dans le code, elle est considérée par défaut comme un double. Si on veut réduire sa précision pour qu'elle représente un float, il faut explicitement lui rajouter un F :

```
float x = 2.F;
```

Caractère (*char*)

Le type **char** (provenant de l'anglais *character*) permet de stocker la valeur Unicode, codée sur 16 bits, d'un caractère, c'est-à-dire un nombre entier codé sur 16 bits, soit 65535 caractères !
Par conséquent il est possible de stocker un caractère accentué dans une variable de type *char*.

Si jamais on désire par exemple stocker la lettre B, on pourra définir cette donnée soit par son code Unicode '\u0066', soit en plaçant directement le caractères entre apostrophes 'B', on peut ainsi copier-coller un caractère Unicode et le mettre directement dans son code sans avoir à connaître sa valeur Unicode.

Chaînes de caractères (*String*)

Les chaînes de caractères ne correspondent pas à un type de données mais à une classe, ce qui signifie qu'une chaîne de caractère est un objet possédant des attributs et des méthodes. Une chaîne peut donc être déclarée de la façon suivante :

```
String s = "Chaine de caractères";
```

Conversion de type de données (casting)

On appelle *conversion de type de données*, parfois *transtypage* (traduction de l'anglais *casting*), le fait de modifier le type d'une donnée en une autre.

- **conversion implicite**: une conversion implicite consiste en une modification du type de donnée effectuée automatiquement par le compilateur. Cela signifie que lorsque l'on va stocker un type de donnée dans une variable déclarée avec un autre type, le compilateur ne retournera pas d'erreur mais effectuera une conversion *implicite* de la donnée avant de l'affecter à la variable. Par exemple la conversion d'un type primitif vers son Wrapper est implicite.

```
int n = 8;
Integer m = n;
```

- **conversion explicite**: une conversion explicite (appelée aussi *opération de cast*) consiste en une modification du type de donnée forcée. Cela signifie que l'on utilise un opérateur dit *de cast* pour spécifier la conversion. L'opérateur de cast est tout simplement le type de donnée, dans lequel on désire convertir une variable, entre des parenthèses précédant la variable.

```
double x = 8.324;
int n = (int) x;
```

x contiendra après affectation la valeur 8.

‹ Précédent

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

Suivant ›

Réalisé sous la direction de Jean-François PILLOU, fondateur de CommentCaMarche.net.