



Le langage Java

Littéraux & Opérateurs

Programme détaillé ou sommaire

Les littéraux

Les types primitifs

Nombres

Caractère

Chaine de caractères

Conversion implicite

Règles de nommage

Opérateurs arithmétiques

Opérateurs relationnels

Opérateurs logiques

Opérateurs conditionnels

Chaine de caractères

Opérateur d'affectation

Caractère _

Littéraux

Un littéral représente l'information qu'on peut stocker dans des variables.

Exemples :

`int a = 5;` (le littéral est une valeur située à droite du égal, la variable à gauche)

`double maValeur = 4.28;`

`String monAdresse = "1, place du Marché, 34000 MONTPELLIER";`

`boolean test = true;`

Types de littéraux

Nombres:

Entiers : -1, -2, -3, 0, 1, 2, 3, ...

Réels: -0.784, 1.205, 0.0015, ...

Caractères

'e', 'i', 'A', ...

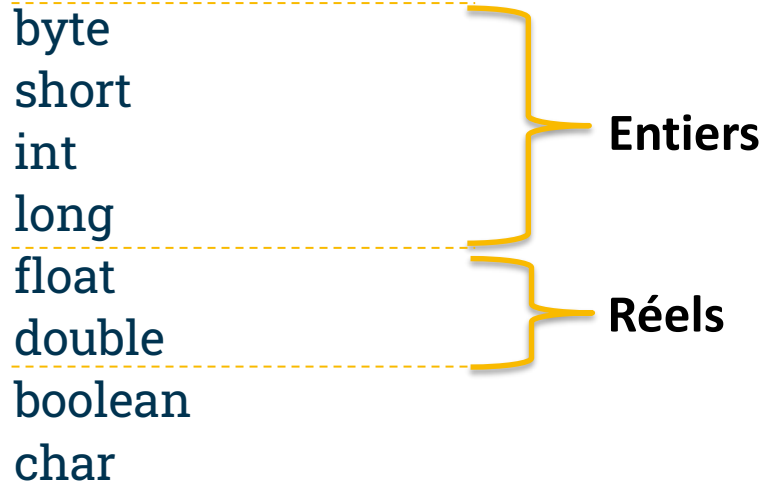
Chaînes de caractères

"Bonjour"

Booléens

true ou false

Types de variables



String

(n'est pas un type primitif officiel)

Déclaration de variable

Type nom = littéral;

Le caractère **=** sert de caractère d'affectation de valeur à une variable.

Le caractère **;** est obligatoire

Il sert à délimiter les différentes instructions.

En général une instruction par ligne.

Déclaration de variable - exemples

Exemples qui compilent :

```
int var1 = 2;
```

```
String chaine = "Coucou";
```

Exemples qui ne compilent pas :

```
int var1 = "Coucou";
```

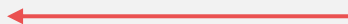
```
String chaine = 2;
```

Pourquoi cela ne compile pas ?

Déclaration de variable – sens de lecture

Sens de lecture, de droite à gauche

```
int var1 = 2;
```



J'affecte la valeur littérale 2 à var1, variable de type int.

Déclaration de variable - byte

Les variables de type **byte** stockent de petits nombres entiers entre -128 et +127.

```
byte a = 12;
```

Occupe 1 octet en mémoire.

Exemples:

- peut convenir pour un numéro de département
- ne convient pas pour un numéro de rue

Déclaration de variable - short

Les variables de type **short** stockent des nombres entiers entre -32 768 et +32 767.

```
short a = -15 325;
```

Occupe 2 octets en mémoire.

Exemples:

- peut convenir pour un numéro de rue
- ne convient pas pour un code postal

Déclaration de variable - int

Les variables de type **int** stockent des nombres entiers entre -2,1 milliards et +2,1 milliards.

```
int a = 1250326;
```

Occupe 4 octets en mémoire.

Exemples:

- peut convenir pour code postal
- ne convient pas pour le nombre d'habitants sur Terre.

Déclaration de variable - long

Les variables de type **long** stockent des nombres entiers entre $-9,22 \cdot 10^{18}$ et $9,22 \cdot 10^{18}$.

long a = 10000L;

Occupe 8 octets en mémoire.

Exemples:

- convient pour le nombre d'habitants sur Terre
- ne convient pas pour le nombre de particules dans l'univers.

Déclaration de variable - float

Les variables de type **float** stockent des nombres réels entre $-3,4 \cdot 10^{38}$ et $3,4 \cdot 10^{38}$.

float a = 1.8F;

A noter que le caractère F (ou f) est obligatoire pour le littéral de type float.

Le caractère décimal est le point (.)

Occupe 4 octets en mémoire

Exemples:

- convient pour stocker un prix
- convient pour la plupart des applications financières et scientifiques.
- ne convient pas pour le nombre de particules dans l'univers.

Déclaration de variable - double

Les variables de type **double** stockent des nombres réels entre $-1,79 \cdot 10^{308}$ et $1,79 \cdot 10^{308}$.

double a = 1.8;

ou

double a = 1.8D;

A noter que le caractère D (ou d) est facultatif pour le littéral de type double.

Occupe 8 octets en mémoire

Exemples:

- convient pour tout.

Récapitulatif

	Type	Déclaration
Entiers naturels	byte	<code>byte a = 1;</code>
	short	<code>short a = 1;</code>
	int	<code>int a = 1;</code>
	long	<code>long a = 1L;</code>
Réels	float	<code>float a = 1.0f;</code>
	double	<code>double a = 1.0;</code> <code>double a = 1.0d;</code>

Déclaration de variables - char

```
char cc = 'a';
```

Littéral entre apostrophes

```
'A'
```

Avec une barre inverse si nécessaire

```
'\'' (apostrophe)  '\\\' (barre inverse)  '\t' (tabulation 9)  
'\n' (ligne 10)
```

Avec un code Unicode en hexadécimal 4 chiffres

```
'\u00a9' (©)      '\u0153' (œ)      '\u20ac' (€)
```


Chaînes de caractères - String

```
String chaine = "Bonjour";
```

Littéral entre guillemets
"Bonjour"

Utiliser la barre inverse si nécessaire

"Une chaîne avec \" un guillemet"

"Une chaîne avec \\ une barre inverse"

"29,99 \u20ac"

(29,99 €)

"Un n\u00e9ud"

(un nœud)

...

Déclaration de variable - boolean

Sert à stocker une information qui est soit vraie soit fausse.

```
boolean a = true;
```

ou

```
boolean a = false;
```

ATTENTION : pas de majuscule

```
boolean a = False; ➔ ne compile pas
```

Utilisation des variables

Une fois qu'une variable est déclarée, il est inutile de la redéclarer à chaque utilisation:

```
int i = 1;
```

```
i = 2;
```

```
i = i + 1;
```

Utilisation des variables

Dans les initialisations de variables

```
int i = 1;  
char a = 'P';  
boolean trouve = false;
```

Dans les expressions (voir plus loin les opérateurs)

```
k = 10 * i + 1;  
a = code + 'A';  
i = i + 1;
```

Dans les appels

```
System.out.println(trouve);  
traitement(a, 100);
```

Sens de lecture

De droite à gauche:

```
int i = 0;
```

```
i = i + 1;
```



Étape 1: j'affecte la valeur 0 à la variable i de type int.

Étape 2: j'augmente la valeur i de 1 et je stocke le résultat à nouveau dans la variable i.

Noms de variables

Pour les noms de variables sont autorisés les caractères suivants :

- *Toutes les lettres de a à z, en minuscules ou majuscules*
- *Les chiffres de 0 à 9*
- *Les caractères \$ et _*
- *Les caractères unicode*

Règle

Les chiffres ne sont pas autorisés comme premier caractère

Atelier (TP)

OBJECTIFS :

Déclarer des variables.

DESCRIPTION : Dans ce TP vous allez déclarer une variable de chaque type.

Quizz

Quelles lignes ci-dessous sont correctes ?

```
int a = +2;  
byte c = 128;  
long d1 = 10;  
long d2 = 10L;  
long d3 = 101;  
long d4 = 100_000_000;  
float e = 3,14;  
double f = 3.14;  
float g = 2.1E-3F;  
double j = 127f;  
char k = 'A';  
char k = '\u0043';
```


Quizz

Quelles sont les lignes qui compilent ?

- `int $ = 0;`
- `float _0 = 1f;`
- `double $_$ = 2d;`
- `int v0 = 125;`
- `int 0b = 2;`

Noms de classes et packages

Même règles que pour les variables à l'exception des caractères unicode non autorisés dans les noms de classes et packages.

Les exemples suivants compilent:

```
package __1;  
class $_ { ... }
```

Dans les faits, l'utilisation du caractère \$ est **proscrite**.
Le caractère _ est autorisé pour nommer les constantes.

Opérateurs

Proviennent du langage C

Pour la compatibilité

Pour la souplesse et les performances

Assument des conversions implicites

Conversions implicites

Depuis La version 1.5 de Java il existe un mécanisme de conversion dit « implicite » basé sur cette hiérarchie:

double > float > long > int > short > byte

Les conversions possibles

double > float > long > int > short > byte

	Type	Déclaration	Conversion implicite
Entiers naturels	byte	byte a = 1;	
	short	short a = 1;	
	int	int a = 1;	
	long	long a = 1L; long a = 1;	De int vers long
Réels	float	float a = 1.0f; float a = 1L; float a = 1;	De long vers float De int vers float
	double	double a = 1.0; double a = 1.0d; double a = 1.0f; double a = 1L; double a = 1;	De float vers double De long vers double De int vers double

Opérateurs arithmétiques

Les opérations sur les nombres

Valables sur les char (assimilés à leur code)

Attention aux débordements (pas d'erreur dans ce cas)

+ Addition :	7+8	résultat: 15
- Soustraction :	7-8	résultat: -1
* Multiplication :	7*8	résultat: 56
/ Division :	7/8	résultat: 0
	7.0/8.0	résultat: 0.875
% Modulo (reste)	7%8	résultat: 7
	16%7	résultat: 2
++ Incrémentation (*) :	i++	
-- Décrémentation (**) :	k--	

Opérateur de concaténation

Concaténation

Les String se concatènent avec l'**opérateur +**

Conversions implicites vers String lorsqu'on ajoute des nombres à des chaînes.

```
String result = "indice = " + i;
```

Il est possible de réaliser une **opération** dans une concaténation

```
String result = "indice = " + i * 2;
```

Attention dans le cas de l'**addition** les parenthèses sont conseillées

```
int i = 1;
```

```
String result = "indice = " + i + 2;
```

indice = 12



```
String result = "indice = " + (i + 2);
```

indice = 2



Opérateurs relationnels

Opérations de comparaison

Valable pour tous types

== Égal(*)

!= Différent (non égal)

Opérations de relation d'ordre

Seulement pour les nombres

< Plus petit

<= Plus petit ou égal

> Plus grand

>= Plus grand ou égal

Opérateurs logiques

AND, OR, NON

&&	Conjonction (AND) :	$(x < 0) \ \&\& \ (a > -1.0)$
 	Disjonction (OR) :	$(x == 0) \ \ (a > 1)$
!	Négation (NOT) :	$!(x == 0)$ équivalent de $x != 0$

Opérateur un peu plus exotique

^	Disjonction exclusive (XOR) :	$(x == 0) \ ^ \ (a > 1)$
----------	-------------------------------	--------------------------

Opérateurs logiques

&&	true	false
true	true	false
false	false	false

Retourne **vrai** seulement si les 2 membres de l'expression sont **vrais**.

Exemple:

```
int a = 0;
int b = 1;
a++;

if (a==1 && b==1){
    System.out.println("Vrai");
}
else {
    System.out.println("Faux");
}
```

Opérateurs logiques

	true	false
true	true	true
false	true	false

Retourne **vrai** seulement si 1 membre de l'expression est **vrai** ou si les 2 sont **vrais**.

Exemple:

```
int a = 0;
int b = 1;

if (a==1 || b==1){
    System.out.println("Vrai");
}
else {
    System.out.println("Faux");
}
```

Opérateur d'affectation

Permet de positionner la valeur d'une variable

Opérateur =

Evalue l'expression à droite puis l'affecte à la variable de gauche

Affectation simple :

←
`int a=10;`

`int b=(a+1)*3;`

`int c = additionner(a, b); // Le résultat de la méthode
additionner est stocké dans la variable c`

Opérateur d'affectation combiné

Affectation combinée avec un opérateur

Les plus courants:

`a += 10` (raccourci pour `a = a + 10`)

`a -= 10` (raccourci pour `a = a - 10`)

Opérateur conditionnel

Opérateur ternaire permettant des choix

(condition) ? valeur pour true : valeur pour false

Simplifie certaines expressions (ne pas en abuser)

Exemples:

```
double x = -2.9;
```

```
char sign = (x < 0) ? '-' : '+';
```

*Valeur que prend **sign** si la condition est vraie*

*Valeur que prend **sign** si la condition est **fausse***

Que vaut sign ?

Caractère _

Underscore est un caractère autorisé afin d'améliorer la lisibilité des longues valeurs

```
int a = 100_000_000;
```

Underscore ne peut être utilisé qu'entre 2 chiffres

Ne compile pas:

```
int a = 100_;
```

```
int a = _100;
```

```
float x = 100.0_f;
```

Atelier (TP)

OBJECTIFS :

Utiliser des opérateurs.

DESCRIPTION : Dans ce TP vous allez tester les différents opérateurs vus en cours.