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.

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 primitifs

byte
short
int
long

float
double

boolean
char

Entiers

Réels

String (n’est pas un type primitif officiel)

Déclaration de variables

```
type nom = littéral;
```

Le caractère **;** sert à délimiter les différentes instructions. En général une instruction par ligne.

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

Exemple:

```
int var1 = 2;
String chaine = "Coucou";
```

Sens de lecture, de droite à gauche

```
int var1 = 2;
```


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

Exemple qui ne compile pas:

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

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 $-9,22.10^{18}$ et $9,22.10^{18}$.

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 $-3,4.10^{38}$ et $3,4.10^{38}$.

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 $-1,79.10^{308}$ et $1,79.10^{308}$.

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.

Déclaration de variable – notations spécifiques

```
double reel1 = 2.1e3;          // Notation scientifique : 2.1x10^3
double reel2 = 2.1e-4;         // Notation scientifique : 2.1x10^-4

float reel1 = 2.1e3F;          // Notation scientifique : 2.1x10^3
float reel2 = 2.1e-4F;         // Notation scientifique : 2.1x10^-4

double reel3 = 100_000_000;    // Notation financière avec le signe _
                               // Attention signe autorisée uniquement entre 2 nombres
```

Comment afficher la valeur d'une variable ?

```
double a = 1.8;
System.out.println(a);
```

Récapitulatif

	Type	Déclaration
Entiers naturels	byte	`byte a = 1;`
	short	`short a = 1;`
	int	`int a = 1;`
	long	`long a = 1L;`
Réels	float	`float a = 1.0f;`
	double	`double a = 1.0;` `double a = 1.0d;`

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) `'\"'` (guillemet)

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\u0153ud"          (un nœud)
...
```

Stockés sous la forme d'objet de type `String`

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 n°2 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 = 10l;
long d4 = 100_000_000;
float e = 3,14;
double f = 3.14;
float g = 2.1E-3F;
double j = 127;
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;`

Question subsidiaire

Que produit la ligne suivante ?

```
int a = 053;
System.out.println(a);
```

Pourquoi ?

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
  - byte < short < int < long < float < double
  - char<->int
  - Permettent d'éviter toute perte d’information

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 de String en utilisant les **parenthèses**

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

Opérateurs relationnels

Opérations de comparaison

Valable pour tous types **sauf les String**

==	Égal
!=	Différent de (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

ET, OU, NON

&&	ET (AND) :	(x < 0) && (a > -1.0)
||	OU (OR) :	(x == 0) || (a > 1)
!	Négation (NOT) :	!(x == 0)

Opérateur un peu plus exotique

^	Ou exclusif(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 **sont 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");
}
```

Opérateurs logiques

^	true	false
true	false	true
false	true	false

Retourne vrai seulement si 1 membre de l'expression est **vrai et l'autre est faux.**

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);
```

Opérateur d'affectation combiné

Affectation combinée avec un opérateur

Forme « var **[op]**= expression »

Raccourci pour « var = var **[op]** expression »

Valable pour tous opérateurs

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) ? '-' : '+';
String prefix = (x <= 9) ? "0" : "";
double valeur = (x < 0) ? –x : x;

System.out.println(sign + prefix + valeur);
```

Précédence des opérateurs

Operator	Description	Level	Associativity
[ ] . () ++ --	access array element access object member invoke a method post-increment post-decrement	1	left to right
++ -- + - ! ~	pre-increment pre-decrement unary plus unary minus logical NOT bitwise NOT	2	right to left
() new	cast object creation	3	right to left
* / %	multiplicative	4	left to right
+ - +	additive string concatenation	5	left to right
<< >> >>>	shift	6	left to right
< <= > >= instanceof	relational type comparison	7	left to right
== !=	equality	8	left to right
&	bitwise AND	9	left to right
^	bitwise XOR	10	left to right
|	bitwise OR	11	left to right
&&	conditional AND	12	left to right
||	conditional OR	13	left to right
?:	conditional	14	right to left
= += -= *= /= %= &= ^= |= <<= >>= >>>=	assignment	15	right to left

Recommandation : simplifier l'écriture avec des parenthèses

Caractère _

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

```
int a = 100_000_000;
int a = 0b10000000_10011000;
```

Underscore ne peut être utilisé qu'entre 2 chiffres

Underscore est interdit en début et en fin de valeurs

```
int a = 100_;
int a = _100;
```

Underscore est interdit avant et après le marqueur décimal .

Underscore est interdit avant et après les marqueurs b, x, f, d et l.