Programmation orientée objet : Java

LA SYNTAXE JAVA

LES VARIABLES

LES VARIABLES

- Déclaration d'une variable :

```
Type Nom;
```

- Initialisation d'une variable :

```
Type Nom=Valeur;
```

Nom--->Identificateur

IDENTIFICATEURS

- Un identificateur est un nom symbolique qui fait référence à un élément d'un programme Java. Les noms de classes, de méthodes, de paramètres et de variables sont tous des identificateurs

- Un identificateur doit débuter par une lettre, un blanc souligné (_) ou un symbole monétaire Unicode (Par exemple ¥).

- Apres sa première lettre un identificateur peut être suivi par un nombre quelconque de lettres, de chiffres, de blancs soulignés ou de symboles monétaires.

IDENTIFICATEURS ET MOTS RÉSERVÉS

- Les identificateurs peuvent inclure des nombres mais ne peuvent pas commencer par un nombre, de plus ils ne peuvent pas contenir des caractères de ponctuation autres que les blancs soulignés et les caractères monétaires;
- Les identificateurs Java sont sensible a la casse;
  - foo et FOO sont différents.
- Un identificateur doit être différent des mots clés et les littéraux qui font partie du langage Java lui-même.
  - Ces mots réservés sont énumérés dans le tableau suivant:

• MOTS RÉSERVÉS EN JAVA

abstract	**const**	**final**	**int**	**public**	**throw**
assert	continue	finally	interface	return	throws
boolean	default	float	long	short	transient
break	do	for	native	static	true
byte	double	goto	new	strictfp	try
case	else	if	null	super	void
catch	enum	implements	package	switch	volatile
char	extends	import	private	synchronized	while
class	false	instanceof	protected	this	

LITTÉRAUX

- Un identificateur ne doit pas contenir :
- Les séparateurs :
  - ( ) { } [ ] < > : ; , . @
- Les opérateurs :
  - + - * / %
  - & | ^ << >> >>>
  - += -= *= /= %= &=
  - |= ^= <<= >>= >>>==
  - = = != < <= > >=! ~ && || ++ -- ? :

Identificateurs: exemples

- int _a;
  - // OK
- int .f;
  - //KO
- int $c;
  - // OK
- int this_is_a_very_detailed_name_for_an_identifier;
  - // OK
- int :b;
  - //KO

Identificateurs: exemples

- int –d;
  - //KO
- int _____2_w;
  - // OK
- int e#;
  - //KO
- int _$;
  - // OK
- int 7g;
  - //KO

LES VARIABLES

- En java, il y a deux types de variables :
  - des variables de type simple ou « primitif » ;
  - des variables de type complexe ou des « objets ».

1. VARIABLE DE TYPES PRIMITIFS

- Java prend en charge huit types de base connus sous le nom de types primitifs:
  - Booléen;
  - Caractère;
  - Quatre types entiers;
  - Deux types à virgule flottante.

TYPES DE DONNÉES PRIMITIFS EN JAVA

Type	Contient	Par défaut	taille	gamme
boolean	true (vrai) or false(faux)	false	1 bit	NA
char	Caractère Unicode	\uoooo	16 bits	\uoooo to \uFFFF
byte	Entier signé	0	8 bits	-128 to 127
short	Entier signé	0	16 bits	-32768 to 32767
int	Entier signé	0	32 bits	-2147483648 to 2147483647
long	Entier signé	0	64 bits	-9223372036854775808 to 9223372036854775807
float	Virgule flottante	0.0	32 bits	1.4E-45 to 3.4028235E+38
double	Virgule flottante	0.0	64 bits	4.9E-324 to 1.79769313486231570E+308

LES TYPES ENTIERS

- Les types entiers de Java sont:
  - byte,
  - short,
  - int,
  - long.
- Tous les types entiers représentent des nombres signés.
- Il n'existe pas le mot clé unsigned (non signé) comme c'est le cas avec C et C++.

LES TYPES ENTIERS :EXEMPLES

```
byte b;
b = 4;
```

```
short  s;
s=1234;
```

```
long lg;
lg=12345678L ;
```

```
int  i;
i=1234567;
```

LES TYPES ENTIERS

- Un littéral commençant par ox ou oX (Zéro X) est considéré comme un nombre hexadécimal .
- Les littéraux entiers commençant par un o (Zéro) sont considérés comme des nombres octaux (base 8) .
- Les littéraux hexadécimaux et octaux valables comprennent:
  - oxff // 255, exprimé en notation hexadécimale
  - 0377 // 255, exprimé en notation octale.

LES TYPES ENTIERS

- L’arithmétique entière de Java est modulaire, ce qui signifie qu'elle ne produit jamais de dépassement de capacité supérieure ou inférieure lorsque vous dépassez la portée d'un type entier donné. En lieu et place, les nombres s'adaptent. Par exemple:

  - byte b1 = 127, b2 = 1; // la plus grande valeur du type byte est 127
  - byte sum = (byte)(b1 + b2); // la somme s’adapte et produit la valeur-128,
    // qui représente la plus petite valeur du type byte

- **Ni le compilateur Java, ni l'interpréteur Java ne vous avertissent lorsque cette situation se présente.**

LES TYPES ENTIERS

- La division entière par zéro et le modulo par zéro sont interdits et soulèvent une exception **ArithmeticException**.
- Chaque type entier possède une classe correspondante : Byte, Short, Integer, et Long.
- Chacune de ces classes définit des constantes:
  - MIN_VALUE
  - MAX_VALUE
- Ces constantes décrivant la portée du type en question.

LES TYPES A VIRGULE FLOTTANTE

- float correspond à une valeur à virgule flottante en simple précision sur 32 bits.
- double correspond à une valeur à virgule flottante en double précision sur 64bits.

LES TYPES A VIRGULE FLOTTANTE : EXEMPLES

```
float pi;
pi = 3.14159f;

float  ft;
ft=4.0f;

double  d;
d=0.123456789d;
```

LES TYPES A VIRGULE FLOTTANTE

- Les littéraux à virgule flottante peuvent également s'exprimer en notation exponentielle, ou scientifique:
  - 1.2345E02 // 1.2345×10^2

LES TYPES A VIRGULE FLOTTANTE

- Les littéraux à virgule flottante ne peuvent pas être exprimés en notation octale ou hexadécimale.

- Les types float et double peuvent représenter quatre valeurs particulières:
  - l'infini positif, l'infini négatif, zéro et NaN.

LES TYPES A VIRGULE FLOTTANTE

- Les valeurs infinies apparaissent lorsqu'un calcul à virgule flottante produit une valeur qui dépasse la gamme représentable d'un float ou d'un double.
- Lorsqu'un un calcul à virgule flottante subit un dépassement de capacité inférieure sur une valeur float ou double, la valeur zéro est retournée.
- Les types à virgule flottante de java font une distinction entre le zéro positif et le zéro négatif, en fonction de la direction des dépassements de capacité inférieure.

LES TYPES A VIRGULE FLOTTANTE

- La valeur NaN "not-a-number" apparait lors d'une opération à virgule flottante, non autorisée telle que 0.0/0.0.
- Quelques exemples d'instructions produisant des valeurs particulières:
  - double inf = 1.0/0.0; // infini
  - double neginf = -1.0/0.0; // -infini
  - double negzero = -1.0/inf; // zéro négatif
  - double NaN = 0.0/0.0; // NaN

LES TYPES A VIRGULE FLOTTANTE

- L'arithmétique décimale ne soulève jamais d'exception, même en cas d'opérations non autorisées, telles qu'une division zéro par zéro ou l'extraction de la racine carrée d'un nombre négatif.

LES TYPES A VIRGULE FLOTTANTE

- Les types primitifs float et double possèdent des classes correspondantes, nommées Float et Double. Chacune de ces classes définit les constantes utiles suivantes :
  - MIN_VALUE ,
  - MAX_VALUE,
  - NEGATIVE_INFINITY,
  - POSITIVE_INFINITY,
  - NaN.

LES TYPES A VIRGULE FLOTTANTE

- Une addition ou une soustraction impliquant l'infini, par exemple, produit l'infini.
- Un zéro négatif se comporte de manière quasi-identique à un zéro positif.
- L'opérateur d'égalité = = considère que le zéro négatifs est équivalent au zéro positif.

LES TYPES A VIRGULE FLOTTANTE

- La seule manière de distinguer un zéro négatif d'un zéro positif, ou standard, consiste à effectuer une division par lui-même. 1.0/0.0 produit l'infini positif mais 1.0 divisé par un zéro négatif produit l'infini négatif.

- Enfin, étant donné que NaN n'est pas à un nombre, l'opérateur = = affirme qu'il n'est égal à aucun autre nombre, y compris lui-même!

LE TYPE CHAR

- La différence de byte, short, int, et long, char est un type non signé.

- Les caractères Java, sont placés entre apostrophe :

```
char c = 'A';
```

- Vous pouvez utiliser n'importe quel caractère Unicode sous la forme d'un caractère littéral:
  - char tab = '\t', apostrophe = '\'', nul = '\000', aleph='\u05Do';

CARACTÈRES D'ÉCHAPPEMENT EN JAVA

Séquence d'échappement	signification
\b	Retour arrière
\t	Tabulateur horizontal
\n	Saut de ligne
\f	Saut de page
\r	Retour chariot
\"	Apostrophe double
\'	Apostrophe
\\	Barre oblique arrière
\xxx	Caractère latin-1 avec encodage xxx, où xxx représente un nombre octale (base 8) entre ooo et 377.
\uxxxx	Caractère Unicode xxxx représentent quatre chiffres hexadécimaux.

EXERCICE

- Ecrire un programme qui affiche la plus grande Valeur int et float?

SOLUTION

- public class MaxAndMinValues{
- public static void main(String [] args)
- {
- System.out.println("La plus grande valeur int est:"+Integer.MAX_VALUE+ " la plus grande valeur float est : "+ Float.MAX_VALUE);
- }
- }
- **Plus grand integer : 2 147 483 647**
- **Plus grand float   : 3.40282e+38**

2.VARIABLE DE TYPE OBJET : STRING

```
//Première méthode de déclaration
String str;
str =  "Salam";

//Deuxième méthode de déclaration
String str = new String();
str = "Salaaaaam";

//Troisième méthode de déclaration
String str =  "Salaaaaam";

//Quatrième méthode de déclaration
String str = new String("Salaaaaam");
```

OPERATEURS

OPÉRATEURS ARITHMÉTIQUES : L'OPÉRATEUR +

- L'opérateur + additionne deux nombres.
- L'opérateur + peut également être utilisé pour concaténer des chaînes de caractères.
- Si l'un des deux opérandes de + est une chaîne caractères, l'autre est alors converti en chaîne de caractères.
- Il faut utiliser des parenthèses lorsqu'on combine l'addition la concaténation:
  - System.out.println("Total: " + 3 + 4); // Affiche "Total: 34", et non pas 7!

OPÉRATEURS ARITHMÉTIQUES : L'OPÉRATEUR -

- Lorsque le - est utilisé comme opérateur binaire, il soustrait le second opérande du premier.

  Par exemple : 7-3 retourne 4.

- Le moins monadique (-):

Lorsque le - est utilisé comme opérateur monadique devant un seul opérande, il convertit une valeur positive en une valeur négative correspondante et vice versa.

OPÉRATEURS ARITHMÉTIQUES : L'OPÉRATEUR *

- L'opérateur * multiplie ses deux opérandes.

Exemple: 7 * 3 → 21.

OPÉRATEURS ARITHMÉTIQUES : L'OPÉRATEUR /

- L'opérateur / divise son premier opérande par le second. Si les deux opérandes sont des entiers, le résultat est un nombre entier et le reste est perdu.
- Si l'un des deux opérandes est une valeur à virgule flottante, le résultat est une valeur à virgule flottante.
- Lors de la division de deux entiers, une division par zéro lève une exception ArithmeticException.
- Avec les calculs à virgule flottante, la division par zéro provoque un résultat infini ou NaN:
  - 7/3 // retourne 2
  - 7/3.0f // retourne 2.333333f
  - 7/0 // lève une exception ArithmeticException
  - 7/0.0 // retourne l'infini positif
  - 0.0/0.0 // retourne NaN

OPÉRATEURS ARITHMÉTIQUES : L'OPÉRATEUR %

- L'opérateur % calcule le premier opérande modulo le second opérande
  - Par exemple, 7%3 retourne 1.
- Le signe du résultat est le même que le signe du premier opérande.
- L'opérateur modulo est généralement utilisé avec des opérandes entiers
- Mais il fonctionne également avec des valeurs à virgule flottante:
  - Par exemple 4.3% 2.1 retourne 0.1.
- En travaillant sur des valeurs entières, le calcule d'une valeur modulo zéro provoque une exception ArithmeticException.
- Lorsque vous travaillez avec des valeurs à virgule flottante, une valeur quelconque modulo 0.0 retourne NaN, de même que l'infini modulo une valeur quelconque.

OPÉRATEURS RELATIONNELS

Opérateur	Exemple	Renvoie TRUE si
> >=	v1 > v2 v1 >= v2	v1 plus grand que v2 Plus grand ou égal
< <=	v1 < v2 v1 <= v2	Plus petit que Plus petit ou égal à
== !=	v1 == v2 v1 != v2	égal différent

OPÉRATEURS LOGIQUES

Opérateur	Usage	Renvoie TRUE si
&& &	expr1 && expr2 expr1 & expr2	expr1 et expr2 sont vraies Idem mais évalue toujours les 2 expressions
|| |	expr1 || expr2 expr1 | expr2	Expr1 ou expr2, ou les deux sont vraies idem mais évalue toujours les 2 expressions
!	! expr1	expr1 est fausse
!=	expr1 != expr2	si expr1 est différent de expr2

OPÉRATEURS D'AFFECTATION

- La forme générale de ces opérateurs de combinaison d'affectation est:
  - var op= value
- Ce qui est équivalent (à moins qu'il n'y ait des effets de bord dans var) à:
  - var = var op value
- Les opérateurs disponibles sont:
  - += -= *= /= %= // Opérateurs arithmétiques avec affectation
  - &= |= ^= // Opérateurs orientés bit avec affectation

OPÉRATEURS D'INCRÉMENTATION ET DE DÉCRÉMENTATION

- ++ i : expression qui incrémente de 1 la valeur de i, et sa valeur est celle de i après incrémentation

- si la valeur de i est 5, l'expression : n = ++i - 5 affectera à i la valeur 6 et à n la valeur 1.
- n = i++ - 5 (n==0 , i++ vaut 5, i vaut 6)

- On dit que ++ est:

  -un opérateur de **pré incrémentation** lorsqu'il est placé à gauche

  -un opérateur de **post incrémentation** lorsqu'il est placé à droite