

Tutorial Java Par Rouiha Ayoub et Samir Zahidi

|  |  |
| --- | --- |
|  |  |
| Primitive Valeur par défaut |  |
| Boolean false |  |
| Primitive ‘\u0000’ (null) |  |
| Char (short)0 |  |
| Byte (byte)0 |  |
| Short (Short)0 |  |
| Int 0 |  |
| Long 0L |  |
| Float 0.0f |  |
| Double 0.0d |  |
| Rimitive | syntaxe |
| char | ‘x’ |
| int | 5 (décimal), 05 (octal), 0x5 (hexadécimal) |
| long | 5L, 05L, 0x5L |
| float | 5.5f ou 5f ou 5.5E5f |
| double | 5.5 ou 5.5d ou 5.5E5d ou 5.5E5 |
| boolean | false ou true |

**Chaînes littérales**

Les chaînes de caractères existent aussi sous forme littérale. Il suffit de placer la chaîne entre guillemets comme dans l’exemple suivant :

 "Bonjour samir !"

Les chaînes littérales peuvent contenir des caractères spéciaux issues du type *char* :

|  |  |
| --- | --- |
| **caractères spéciaux** |  |
| \b | backspace |
| \f | saut de page |
| \n | saut de ligne |
| \r | retour chariot |
| \t | tabulation horizontale |
| \\ | \ |
| \’ | ‘ |
| \" | " |
| \012 | caractère en code octal |
| \uxxxx | caractère en code héxadécimal (unicode) |

Description des principaux opérateurs

•    **Opérateur d’affectation**

|  |  |  |  |
| --- | --- | --- | --- |
| **symbole** | **description** | **arité** | **exemple** |
|  |  |  |  |
| opérateur d’affectation | *=* | *affectation* | *2* | *x = 2* |
|  | *–=* | *soustraction et affectation* | *2* | *x–= 2* |
| *+=* | *addition et affectation* | *2* | *x += 2* |

On dispose du raccourci : *x = y = z = 2*.

•    **Opérateurs arithmétiques à deux operands**

|  |  |  |  |
| --- | --- | --- | --- |
| **symbole** | **description** | **arité** | **exemple** |
| opérateurs arithmétiques à deux opérandes | *–* | *soustraction* | *2* | *y – x* |
| *\** | *multiplication* | *2* | *3 \* x* |
| */* | *division* | *2* | *4 / 2* |
| *%* | *modulo (reste de la division)* | *2* | *5 % 2* |

•    **Opérateurs à un opérande**

|  |  |  |  |
| --- | --- | --- | --- |
| **symbole** | **description** | **arité** | **exemple** |
| **opérateurs à un opérande** | *–* | *opposé* | *1* | *–x* |
| *++* | *pré-incrémentation* | *1* | *++x* |
| *++* | *post-incrémentation* | *1* | *x++* |
| *--* | *pré-décrémentation* | *1* | *--x* |
| *--* | *post-décrémentation* | *1* | *x--* |

•    **Opérateurs relationnels**

|  |  |  |  |
| --- | --- | --- | --- |
| **symbole** | **description** | **arité** | **exemple** |
| **opérateurs relationnels** | *==* | *équivalent* | *2* | *x == 0* |
| *<* | *plus petit que* | *2* | *x < 2* |
| *>* | *plus grand que* | *2* | *x > 2* |
| *<=* | *plus petit ou égal* | *2* | *x <= 3* |
| *>=* | *plus grand ou égal* | *2* | *x >= 3* |
| *!=* | *non équivalent* | *2* | *a != b* |

**Opérateurs logiques**

|  |  |  |  |
| --- | --- | --- | --- |
| **symbole** | **description** | **arité** | **exemple** |
| **opérateurs logiques** | *&&* | *et* | *2* | *a && b* |
| *||* | *ou* | *2* | *a || b* |
| *!* | *non* | *1* | *!a* |

**Opérateurs d’arithmétique binaire**

Les opérateurs d’arithmétiques binaires agissent au niveau des bits de données, sans tenir comte de ce qu’ils représentent.

|  |  |  |  |
| --- | --- | --- | --- |
| **symbole** | **description** | **arité** | **exemple** |
| **opérateurs d’arithmétique binaire** | *&* | *et* | *2* | *a & b* |
| *|* | *ou* | *2* | *a | b* |
| *^* | *ou exclusif* | *2* | *a ^  b* |
| *~* | *non* | *1* | *~a* |
| *<<* | *décalage à gauche* | *2* | *a << 2* |
| *>>* | *décalage à droite* | *2* | *b >> 2* |
| *>>>* | *décalage à droite sans extension du signe* | *2* | *b >>> 2* |

**L’opérateur à trois operands**

|  |  |  |  |
| --- | --- | --- | --- |
| **symbole** | **description** | **arité** | **exemple** |
| **opérateur à trois opérandes** | *? :* | *condition ? vrai : faux* | *3* | *y < 5 ? 4 \* y : 2 \* y* |

Si *condition* est vrai, alors on retourne l’évaluation de l’expression *vrai*, sinon on retourne celle de *faux*.

### L’opérateur + pour *String*

Les chaînes de caractères dispose de l'opérateur + qui permet de concaténer deux chaînes ; += est aussi valide pour les chaînes de caractères.

String x = "samir" ;

.println("bonjour" + x) ;

**Les structures de contrôle**

Les structures de contrôle sont presque les mêmes que celles utilisées en C ou en C++. On les rappelle ci-dessous en précisant les spécificités *Java*.

|  |  |  |
| --- | --- | --- |
| **structure de contrôle** | **syntaxe** | **commentaires** |
| *retour d’une méthode* | **return***;* | *cas d’une méthode void* |
|  | **return***(type de retour) ;* |  |
| *instruction conditionnelle****if*** | **if***(expression) instruction ;* | *les parenthèses sont obligatoire en Java autour de l’expression booléenne. De même le point-virgule est obligatoire en fin de bloc.* |
| *i****f****(expression) {bloc d’instructions }* |
| *instruction conditionnelle****else*** | **if***(expression) instruction ;****else****instruction ;* |  |
| **if***(expression) {bloc d’instructions}****else****{bloc d’instructions}* |  |
| **if***(expression) ;****else****{bloc d’instructions}* | *On ne peut pas donner une instruction vide  après la condition if.* |
| *instruction conditionnelle****elseif*** | **if***(expression)  {bloc d’instructions}****elseif****{bloc d’instructions}…*  **else*if****{bloc d’instructions}****else****{bloc d’instructions}* | *Il s’agit en fait d’une structure de contrôle à part entière écrite en deux mots !* |
| *la boucle****for*** | **for***(initialisation ; test ; incrémentation)*  *{bloc d’instructions}* | *la partie initialisation se compose d’une ou plusieurs initialisations (séparées par des virgules. La partie test est une expression booléenne. La partie incrémentation peut contenir plusieurs instruction séparées par des virgules.* |
| *utilisation de****break****et****continue****dans les boucles* | **break***;* | *Interruption de l’itération en cours et passage à l’instruction suivant la boucle.* |
| **continue***;* | *Interruption de l’itération en cours et retour au début de la boucle avec exécution de la partie incrémentation.* |
| *l’instruction****while*** | **while***(expression booléenne)*  *{bloc d’instructions}* |  |
| *l’instruction****switch*** | **switch***(variable)****case****valeur1 : instructions1 ;****case****valeur2 : instructions2 ;*  *…*  **default***: instructions ;* | *Les bloc sont délimités par deux instructions****case****! Lorsqu’une égalité est trouvée, le bloc d’instruction correspondant est exécuté, ainsi que tous les blocs suivant ! Pour qu’un seul bloc ne soit exécuté, il faut utiliser explicitement l’instruction****break****.* |