Types et instructions de base Java

Philippe Genoud
Philippe.Genoud@imag.fr

dernière modification : 13/12/2023 01:31

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Identificateurs

- Noms choisis par le programmeur pour désigner les classes, les variables, les méthodes (fonctions), ...
- Un identificateur Java
  - est de longueur quelconque
  - commence soit par
    - une lettre (caractères unicodes acceptés),
    - _ caractère souligné (*underscore*)
    - $
  - peut ensuite contenir des lettres, des chiffres, _ ou $
  - ne doit pas être un mot réservé du langage (noms prédéfinis)
    - `abstract, assert, boolean, break, byte, case, catch, char, class, continue, default, do, double, else, extends, final, finally, float, for, foreach, if, implements, import, instanceof, int, interface, long, native, new, null, package, private, protected, public, return, short, static, super, switch, synchronized, this, throw, throws, transient, try, var, void, volatile, while`

```
maVariable1
ma_variable1
_maVariable1
$maVariable1
maVariable1$
_uneVariableCachée
```

possibilité d'utiliser des caractères accentués mais pas recommandé

```
1Variable
ma Variable1
maVariable1.
```

Conventions pour les identificateurs

- Les noms de classes commencent par une majuscule (ce sont les seuls avec les constantes)
  - exemples: `Visage` `String`
- Les noms de variables et méthodes commence par une minuscule
  - exemples : `age`, `nom`, `prenom`, `avancer(…)` …
- Les mots contenus dans un identificateur commencent par une majuscule (camel Case)
  - exemples : `UneClasse`, `uneMethode`, `uneVariable`
  - On préférera `ageDuCapitaine` à `ageducapitaine` ou `age_du_capitaine`
- Les constantes sont en majuscules et les mots sont séparés par le caractère souligné « _ » :
  - `UNE_CONSTANTE`

Commentaires

- Sur une ligne

```
// Comme en “C++”, après un slash-slash
int i; // commentaire jusqu’à la fin de la ligne
```

- Sur plusieurs lignes

```
/* Comme en C, entre un slash-étoile et
   un étoile-slash,
   sur plusieurs lignes */
```

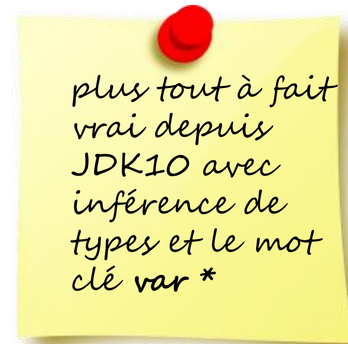
- Commentaires documentant pour l’outil javadoc

```
/**
 * pour l’utilisation de Javadoc
 * à réserver pour la documentation automatique avec javadoc
 */
```

précédent les déclarations de classe et de membres (variables, ou méthodes) d'une classe

Variables et Types de données en Java

- Java langage **typé statiquement**
  - le type de donnée est associé au nom de la variable, plutôt qu'à sa valeur.
  - avant de pouvoir être utilisée une variable doit être déclarée en associant un type à son identificateur.
  - la compilation ou l'exécution peuvent détecter des erreurs de typage



* Bruce Eckel : on simplifying local variable type inference in Java (4 min lecture) https://blogs.oracle.com/javamagazine/post/java-local-reference-type-inference

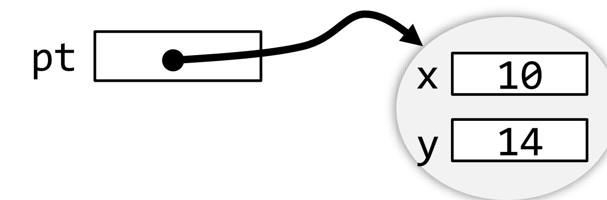
- instructions de déclaration
  - `nomDeType identificateur ;`
  - `nomDeType identificateur = expression d'initialisation ;`

- Le type peut être
  - soit un type primitif (ou type simple)
    - la variable contient alors une **valeur** de ce type
  - soit un nom de classe
    - la variable contient alors une **référence** vers un objet instance de cette classe

```
double prix = 99.99;
```

prix | 99.99 |

```
Point pt = new Point(10,14);
```



Types primitifs

- Valeur logique
  - `boolean` (true/false)
- Nombres entiers
  - `byte` (1 octet), `short` (2octets), `int` (4 octets), `long` (8 octets)
- Nombres non entiers (à virgule flottante)
  - `float` (4 octets), `double` (8 octets).
- Caractère (un seul), utilisation du standard Unicode
  - `char` (codage UTF-16 , 2 octets, sauf pour les caractères étendus)

- types indépendants de l'architecture
  - En C/C++, représentation dépendante de l'architecture (compilateur, système d'exploitation, processeur)
    ex: int = 32 bits sous x86, mais 64 bits sous x64
    Portage difficile, types numériques signés/non signés

© Philippe GENOUD – Université Grenoble Alpes UGA

Types pour les entiers

Type	Taille	Valeurs possibles
`byte`	8	-128 à 127 (-2^7 à $+ 2^7-1$)
`short`	16	-32 768 à 32 767 (-2^{15} à $+ 2^{15}-1$)
`int`	32	-2 147 483 648 à 2 147 483 647 (-2^{31} à $+ 2^{31}-1$)
`long`	64	-9 223 372 036 854 775 808 à 9 223 372 036 854 775 808 (-2^{63} à $+ 2^{63}-1$)

- il n'existe pas en Java de type entier non signé (`unsigned` en C)
- le compilateur vérifie que les valeurs affectées sont compatibles avec le type de la variable

```
byte b1 = 125;
short s1 = -32000;
```

```
byte b2 = -320;
short s2 = 45678;
```

```
Error:
incompatible types: possible lossy conversion from int to byte
```

- depuis Java 7 possibilité d'utiliser le groupage de chiffres pour rendre les constantes numériques plus lisibles

```
int i1 = 1147483647;
```

```
int i1 = 1_147_483_647;
```

utilisation du caractère _ pour définir des blocs de taille quelconque

- expressions littérales pour les longs

```
long l1 = 3_147_483_647;
```

```
Error:
integer number too large
```

```
long l1 = 3_147_483_647L;
```

on rajoute le caractère `L` (ou `l`) à la fin de la valeur

Types pour les entiers

```
int i1 = 010;
System.out.println(i1);
```

```
---> 8
```

? la constante `010` définit une valeur exprimée en base 8 (octal) car débute par `0` (même règle qu'en langage C)

- Possibilité d'utiliser différentes bases numériques pour exprimer des valeurs entières
  - binaire : valeur préfixée par `0b`, seuls les deux chiffres `0` et `1` sont autorisés.

```
int i1 = 0b10;     // == 2
```

n'existe pas en C

  - octale : valeur préfixée par `0` , seuls les huit chiffres `0` à `7` sont autorisés.

```
int i1 = 010;     // == 8
```

identique au C

  - décimale : par défaut, les dix chiffres `0` à `9` sont autorisés.

```
int i1 = 10;
```

  - hexadécimale : valeur préfixée par `x` , 16 chiffres de `0` à `9` puis de `A` à `F`.

```
int i1 = x10;     // == 16
```

identique au C

- possibilité d'utiliser le groupage de chiffres quelle que soit la base numérique utilisée

```
int color = xff_00_ff; // == 16 711 935  couleur fuchsia en codage RGB
```

Passage d'un type entier à un autre

- transfert de la valeur d'une variable entière vers une autre variable entière dépend du type des variables
  - 1er cas : la taille de la variable affectée est supérieure ou égale à la taille de la variable transférée
    - conversion de type implicite autorisée (il n'y a pas de perte de valeur)

```
byte c = 127;
short s = c;
int i = s;
long l = i;
```

toutes les variables ont la valeur 127

- 2ème cas : la taille de la variable affectée est inférieure à la taille de la variable transférée

```
long l = 127;
int i = l;
short s = i;
byte b = s;
```

```
Error: incompatible types: possible lossy conversion from long to int
Error: incompatible types: possible lossy conversion from int to short
Error: incompatible types: possible lossy conversion from short to byte
```

- le changement de type doit être explicité à l'aide d'un opérateur de transtypage (cast)

```
long l = 127;
int i = (int) l;
short s = (short) i;
byte b = (byte) s;
```

le transtypage peut occasionner une perte de données

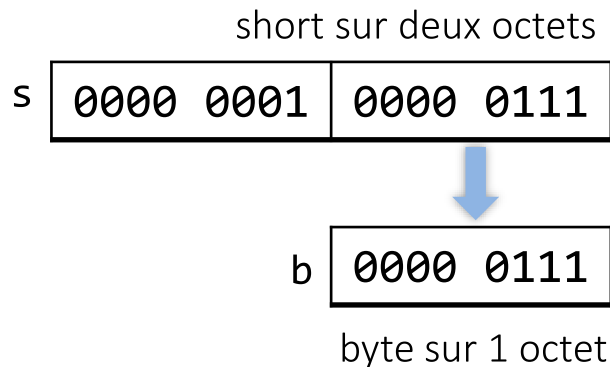
```
short s = 263;
byte b = (byte) s;
System.out.println(b);  ---> 7
```

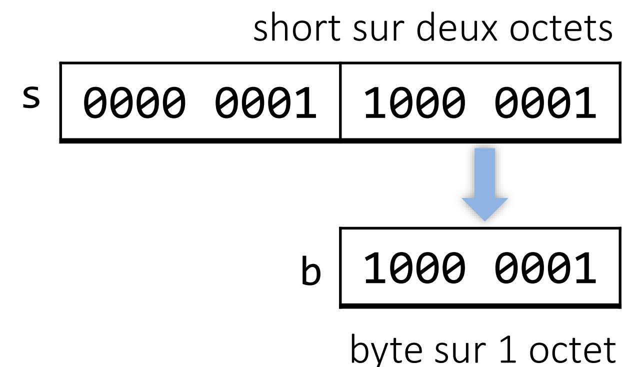
Passage d'un type entier à un autre

- transtypage avec perte de données

```
short s = 263;
byte b = (byte) s;
System.out.println(b);   ---> 7
```

```
short s = 385;
byte b = (byte) s;
System.out.println(b);   ---> -127
```





bit de signe									
0	1	1	1	1	1	1	1	=	127
0				…				=	…
0	0	0	0	0	0	1	0	=	2
0	0	0	0	0	0	0	1	=	1
0	0	0	0	0	0	0	0	=	0
1	1	1	1	1	1	1	1	=	−1
1	1	1	1	1	1	1	0	=	−2
1				…				=	…
1	0	0	0	0	0	0	1	=	−127
1	0	0	0	0	0	0	0	=	−128

Représentation en complément à deux sur 8 bits.

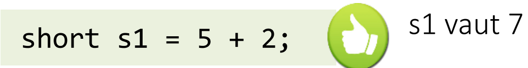
https://fr.wikipedia.org/wiki/Compl%C3%A9ment_%C3%A0_deux

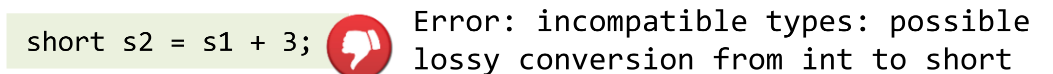
Entiers : Opérateurs arithmétiques

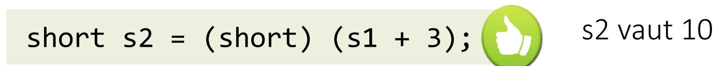
- `+ - * /` addition, soustraction, multiplication et division (division entière)
- `%` reste de la division entière (modulo)
- `++ --` pré ou post incrémentation et décrémentation
- règles de priorités
  - la multiplication (ou la division) est prioritaire par rapport à l'addition (ou la soustraction)
  - en cas de priorités identiques, les opérateurs sont évalués de la gauche vers la droite
  - utilisation des parenthèses pour forcer les priorités (à utiliser en cas de doute)

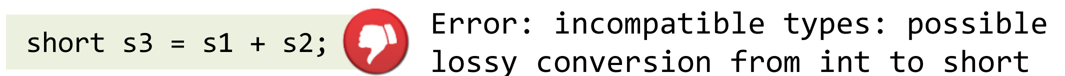
```
public class DemoOpArithmétiques {

    public static void main( String [] args ) {
        int nb1 = 5
        int div = nb1 / 2;
        int reste = nb1 % 2;

        int nb2 = div * 2 + rest; // ⇔ nb2 = (div * 2) + reste;

        System.out.println( "5 / 2 => " + div );    // 2
        System.out.println( "5 % 2 => " + reste );  // 1
        System.out.println( "nb2 => " + nb2 );      // 5

    }
}
```

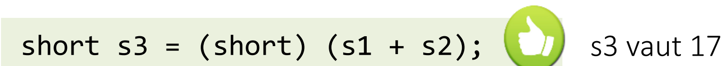
dans un expression arithmétique les variables de type `byte` ou `short` sont converties vers le type `int`











Entiers : Opérateurs de comparaison

- opérateurs de comparaison
  - opérateurs d'égalité
    - **==** l'égalité , **!=** la différence
  - opérateur relationnels
    - **<=** l'infériorité , **<** l'infériorité stricte
    - **>=** la supériorité , **>** la supériorité stricte

```java
public class DemoOpComparaison {

    public static void main( String [] args ) {
        int nb1 = 5
        int nb2 = 6
        System.out.println("nb1 égal nb2 : " + (nb1 == nb2d));          // false
        System.out.println("nb1 plus grand que nb2 : " + (nb1 > nb2)); // false
        System.out.println("nb1 plus petit que nb2" + (nb1 < nb2));    // true
    }
}
```

- opérateurs arithmétique binaire
  - décalage
    - **>>** décalage à droite, **<<** décalage à gauche
  - opérateurs bits à bits
    - **&** et binaire, **|** ou binaire, **^** exclusif

```java
public class DemoOpBinaire {

    public static void main( String [] args ) {
        byte nb1 = 5;
        byte nb2 = 6;
        System.out.println("nb1 & nb2 : " + (nb1 & nb2d));   // 4
        System.out.println("nb1 | nb2 : " + (nb1 | nb2));    // 7
        System.out.println("nb1 << 2" + (nb1 << 2));         // 20
     }
}
```

nb1	0000 0101	nb1 & nb2	0000 0100	nb1 << 2	0001 0100
nb2	0000 0110	nb1 | nb2	0000 0111		

© Philippe GENOUD – Université Grenoble Alpes UGA

Types pour les nombres flottants

Type	Taille	Valeurs possibles
`float`	32	1.40239846e-45 à 3.40282347e38
`double`	64	4.94065645841246544e-324 à 1.79769313486231570e308

- expressions littérales pour les flottants

```
float f1 = 0.55;
```

```
Error: incompatible types: possible
lossy conversion from double to float
```

```
float f1 = 0.55F;
float f2 = 4.567e2f;
float f3 = 1274;
```

conversion implicite de `int` vers `float`

```
double d1 = 0.55;
double d2 = 4.567e-2;
double d3 = 1274;
```

conversion implicite de `int` vers `double`

- Un transtypage vers un flottant peut provoquer une perte de précision

```
long l1 = 928999999L;
float f = (float) l1;
System.out.println(f);    ---> 9.29E8
long l2 = (long) f;
System.out.println(l2);   ---> 929000000
```

Par exemple, la conversion d'un `long` vers un `float` peut faire perdre des chiffres significatifs mais pas l'ordre de grandeur

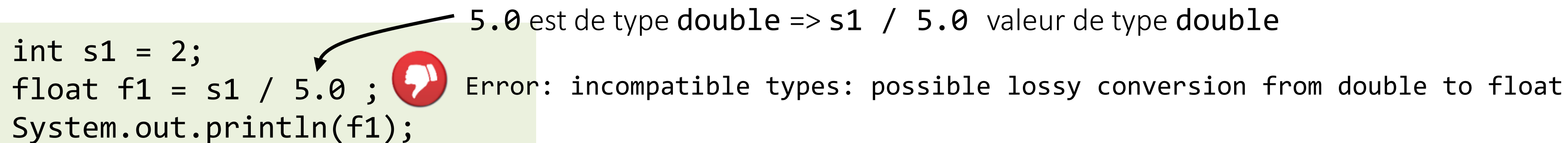
Flottants : Opérateurs

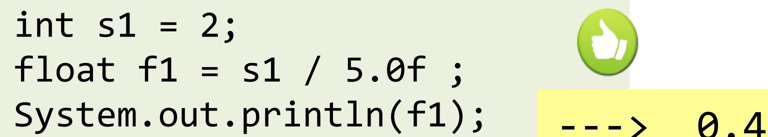
- Opérateurs arithmétiques
  - `+ - * /` addition, soustraction, multiplication et division
- Opérateurs de comparaison
  - `==` l'égalité , `!=` la différence
  - `<=` l'infériorité , `<` l'infériorité stricte, `>=` la supériorité , `>` la supériorité stricte

```
int s1 = 2;
float f1 = s1 / 5;
System.out.println(f1);
```

```
---> 0
```

```
int s1 = 2;
float f1 = s1 / 5.0 ;
System.out.println(f1);
```





```
int s1 = 2;
float f1 = s1 / 5.0f ;
System.out.println(f1);
```

```
---> 0.4
```

```
int s1 = 2;
float f1 = (float) s1 / 5 ;
System.out.println(f1);
```

```
---> 0.4
```

Type caractère

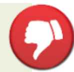
Type	Taille	Valeurs possibles
char	16	0 à 65535 (pour les points de codes supérieurs, 32 bits (4 octets) sont utilisés

- caractères unicodes codés sur 2 (ou 4) octets (UTF-16)
- un variable de type char code **un seul** caractère

```
char c1 = 'a';
```

' (simple quote) pour délimiter littéraux de type char

```
char c1 = "a";
```



```
Error:
incompatible types: java.lang.String cannot be converted to char
```

" (double quote) pour délimiter littéraux de type chaîne de caractères qui ne sont pas des types primitifs mais des objet de type `String`

- utilisation du code numérique du caractère

```
char c2 = 65 ;
System.out.println(c2);   ---> A
```

- utilisation de la notation unicode

```
char c3 = '\u03a9' ;
System.out.println(c3);   ---> α
```

pour que le caractère s'affiche correctement, il faut que le système d'exploitation ait une police de caractères connaissant les glyphes de caractères utilisés (ici des caractères Grecs). Par exemple sous Windows on obtient `---> ?`

Type caractère

Type	Taille	Valeurs possibles
char	16	0 à 65535 (pour les points de codes supérieurs, 32 bits (4 octets) sont utilisés

- caractères unicodes codés sur 2 (ou 4) octets (UTF-16)
- utilisation de caractère spéciaux

```
char c1 = '\n';
```
retour à la ligne

```
char c2 = '\t';
```
tabulation

```
char c1 = '\\';
```
caractère \

```
char c2 = '\'';
```
caractère '

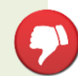
- caractères et conversions de types
  - La correspondance **char** → **int**, **long** s'obtient par cast implicite

```
char c = '@';
int i = c;
System.out.println(i);   --->  64
```
⇔ `int i = (int) c;`

point de code du caractère @

  - Les correspondances **char** → **short**, **byte** , et **long**, **int**, **short** ou **byte** → char nécessitent un cast explicite (entiers sont signés et pas les char)

```
byte b = 64;
char c = b
System.out.println(i);
```



```
Error: incompatible types: possible lossy
conversion from byte to char
```

```
byte b = 64;
char c = (char) b;
System.out.println(i);   --->  @
```

Type booléen

Type	Taille	Valeurs possibles
boolean	1	true false

- seules valeurs possibles true ou false
- pas de conversion vers les autres types

```
boolean bool = true ;
int i = (int) bool;
```

```
Error:incompatible types: boolean cannot be converted to int
```

- pas de conversion depuis les autres types

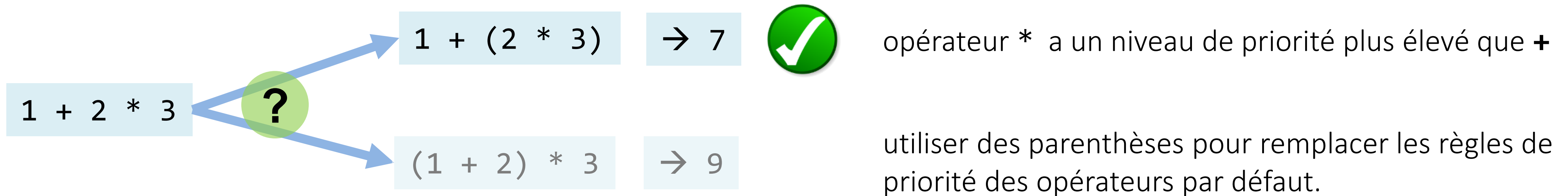
```
int i = 1;
boolean bool = (boolean) i;
```

```
Error:incompatible types: int cannot be converted to boolean
```

- opérateurs booléens
  - && et logique
  - || ou logique
  - ! négation

Précédence des opérateurs

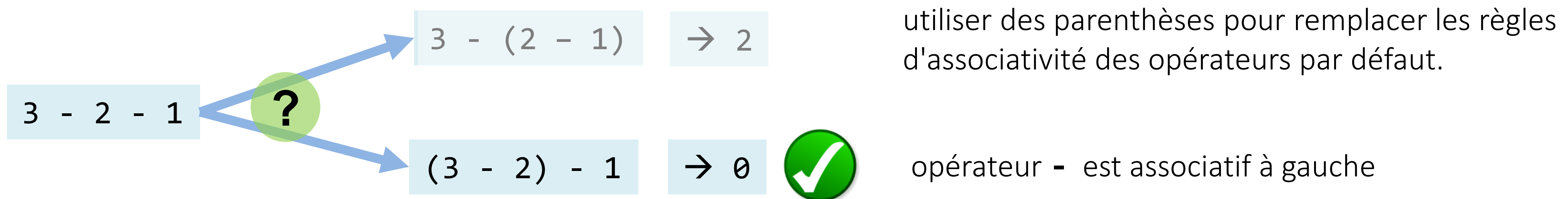
- Java (comme tout langage de programmation) propose des règles bien définies pour l'évaluation des expressions
- **priorité (*précédence*) des opérateurs :** spécifie la manière dont les opérandes sont regroupés avec les opérateurs



opérateur * a un niveau de priorité plus élevé que **+**

utiliser des parenthèses pour remplacer les règles de priorité des opérateurs par défaut.

- **associativité des opérateurs** : spécifie la manière dont opérateurs et opérandes sont regroupés (associés) lorsqu'une expression possède deux opérateurs de même priorité



utiliser des parenthèses pour remplacer les règles d'associativité des opérateurs par défaut.

opérateur **-** est associatif à gauche

Précédence des opérateurs

- Tableau complet des précédences des opérateurs en Java

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

source : https://introcs.cs.princeton.edu/java/11precedence/

Pas besoin de mémoriser toutes les règles de priorités, utilisez les parenthèses pour privilégier la clarté !

à

```
year % 4 == 0 && year % 100 != 0 || year % 400 == 0
```

on préfèrera

```
((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0)
```

Déclarations

- Avant toute utilisation dans un programme une variable doit être déclarée

- syntaxe: ***type identificateur***
  - type : un type primitif ou un nom de classe

- Exemples

```
byte age;
boolean jeune;
float poids;
double x, y ,z;
```

- Une variable est accessible (**visible**) depuis l'endroit où elle est déclarée jusqu'à la fin du bloc où sa déclaration a été effectuée (similaire à `let` en JavaScript)

Affectation

- Syntaxe : *`lvalue = expression`*

  `lvalue` est une expression qui doit délivrer une variable (par exemple un identificateur de variable, élément de tableau...., mais pas une constante)

- Exemples

```
int age;
age = 10;
boolean jeune = true;
float poids = 71.5f;
float taille = 1.75f;
float poidsTaile = poids / taille;
```

- Attention en JAVA comme en C, l'affectation est un opérateur. L'affectation peut donc être utilisée comme une expression dont la valeur est la valeur affectée à la variable

```
i = j = 10; // à i est affectée la valeur de l'expression j = 10, c'est-à-dire 10
```

© Philippe GENOUD – Université Grenoble Alpes UGA

Flot de contrôle

bloc d'instructions - instruction composée

- permet de grouper un ensemble d'instructions en lui donnant la forme syntaxique d'une seule instruction

- syntaxe:

```
{
   séquence d'énoncés (statements)
}
```

- exemple

```
int k;
{
      int i = 1;
      int j = 12;
      j = i+1;
      k = 2 * j - i;
}
```

© Philippe GENOUD – Université Grenoble Alpes UGA

Flot de contrôle

Instruction conditionnelle - instruction if

- Syntaxe **if** (*expression booléenne*) *instruction1*

  ou bien

```
if ( expression booléenne )
   instruction1
else
   instruction2
```

- exemple

```
if (i==j){
   j = j -1;
   i = 2 * j;
}
else
   i = 1;
```

Un bloc car *instruction1* est composée de deux instructions

Flot de contrôle

boucle tantque ... faire - instruction while ()

- Syntaxe

```
while ( expression booléenne )
        instruction
```

- Exemple

```
int i = 0;
int somme = 0;
while (i <= 10) {
   somme += i;
   i++;
}
System.out.println("Somme des 10 premiers entiers" + somme);
```

© Philippe GENOUD – Université Grenoble Alpes UGA

Flot de contrôle

boucle répeter … jusqu'à – instruction do while ()

- Syntaxe

```
do
        instruction
while ( expression booléenne ) ;
```

- Exemple

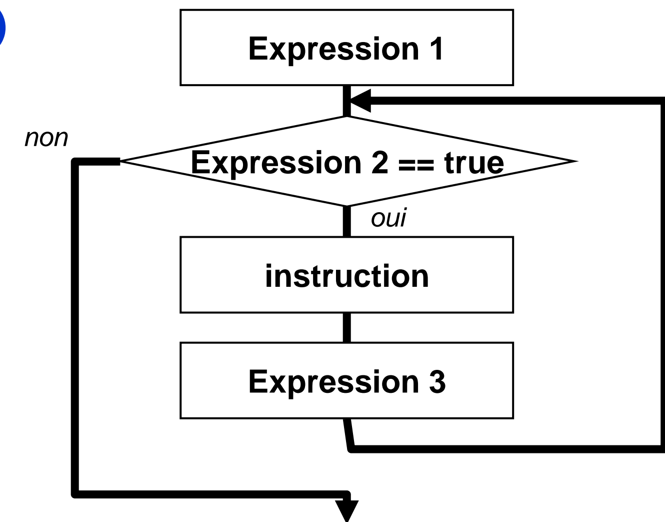
```
int i = 1 n;
int somme = 0;
do {
   somme += i;
   i++;
}  while (i <= 10);
System.out.println("Somme des 10 premiers entiers" + somme);
```

Flot de contrôle

boucle pour – instruction for

- Syntaxe

```
for (expression1 ; expression2; expression3)
   instruction
```



- Exemple

```
int  i;
int somme = 0;
for (i = 0; i <= 10; i++) {
   somme += i;
}
System.out.println("Somme des 10 premiers entiers " + somme);
```

Entrées/sorties sur la console

- Affichage sur la console
  - `System.out.println(`*chaîne de caractères à afficher*`)` avec retour à ligne
  - `System.out.print(`*chaîne de caractères à afficher*`)` sans retour à la ligne

  - chaîne de caractères peut être :
    - une constante chaîne de caractères (`String`)

```
System.out.println("coucou");
```

    - une expression de type `String`

```
int age = 30;
System.out.println(age);
```

Ici **age** est une variable de type **`int`**
Elle est **automatiquement** convertie en **`String`**

    - une combinaison (concaténation) de constantes et d'expressions de type String. La concaténation est exprimée à l'aide de l'opérateur **+**

```
double poids = 64.5;
System.out.println("L'age de la personne est " + age + " son poids " + poids);
```

age (int) et **poids** (double) sont **automatiquement** converties en **`String`**

© Philippe GENOUD – Université Grenoble Alpes UGA

Entrées/sorties sur la console

- Lecture de valeurs au clavier

  - utiliser la classe `Scanner` du package standard `java.util` (version JDK 1.5 et supérieur).

  - Dans le fichier `ScannerDemo.java`

    vous trouverez un exemple d'utilisation de cette classe pour lire des données au clavier.

Mon premier programme Java

Le code de la classe doit être enregistré dans un fichier de même nom (casse comprise) que la classe ③

`HelloWorld.java`

① Tout code java doit être défini à l'intérieur d'une classe

```
public class HelloWorld {

  public static void main(String[] args) {

    int i = 0;

    while (i < 5) {
       System.out.println("Hello World !");
       i++;
    }

  }
}
```

② La description de la classe est effectuée à l'intérieur d'un bloc `{ }`

④ Le point d'entrée pour l'exécution est la méthode `main()`

Compilation :
`javac HelloWorld.java`

Exécution :
`java HelloWorld`

`HelloWorld.java`

`HelloWorld.class`

```
Hello World !
Hello world !
Hello World !
Hello World !
Hello World
```