# Informatique 1
# L1 Portail IE
# L1 Portail MA

*Responsables:*
*olivier.dameron@univ-rennes1.fr*
*patrick.derbez@univ-rennes1.fr*

# Module INF1

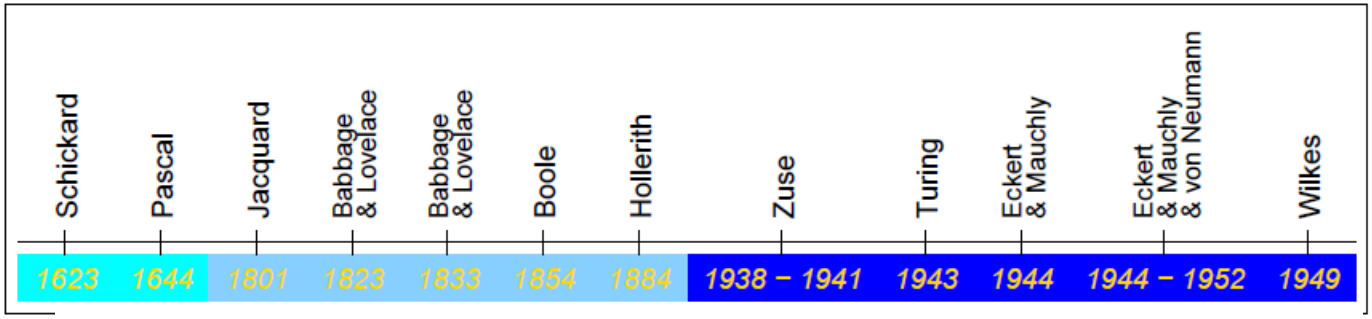
- Objectifs pédagogiques:
  - Maîtriser les fondements de la programmation impérative
  - Savoir concevoir un algorithme à partir d'une spécification
  - Savoir traduire un algorithme dans un langage informatique (Java)
- Déroulement:
  - 1 séance de cours + capsules vidéos (~1h par semaine)
  - 10 séances de TD (20h)
  - 10 séances de TP (20h)
- Modalités de Contrôle
  - Un projet en binôme
  - Un examen à mi-semestre (S43)
  - Un examen final

# Module INF1

## L'informatique est la « science du traitement de l'information »

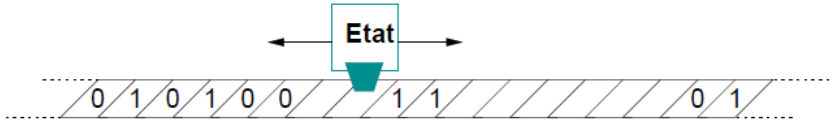
- Savoir modéliser l'information
  - Comment représenter numériquement notre environnement?
- Savoir traiter l'information
  - Comment transformer, résumer, fusionner et interagir avec ces représentations?

# Un peu d'histoire…



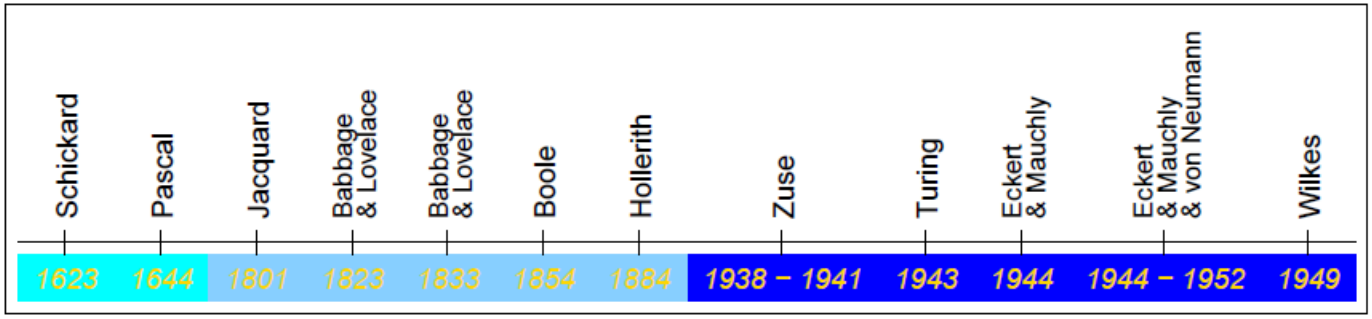

## Alan Turing, 1912–1954

- 1936 : Définition de la *machine universelle de Turing*
  - Lecture
  - Écriture
  - Aller à gauche/droite d'un carré
  - Changer d'état
  - S'arrêter



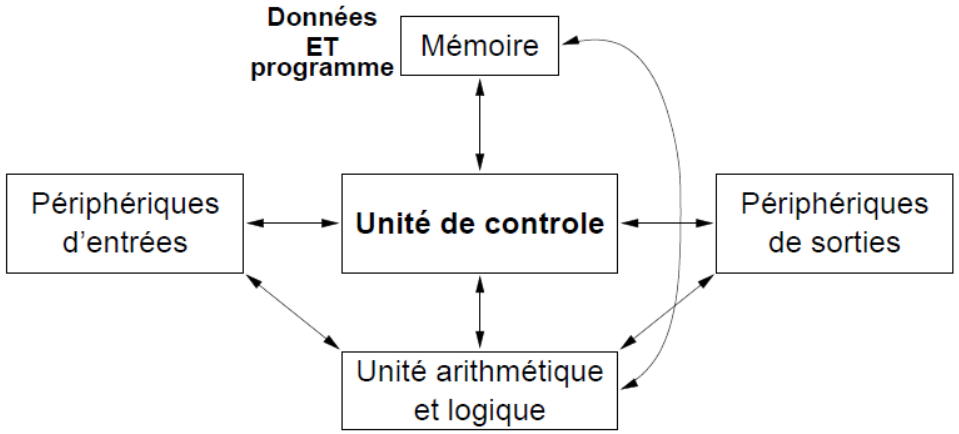

⇒ code *et* données en mémoire

# Un peu d'histoire…




John von Neumann (1903–1957)

**Architecture de von Neumann**



# Comment parle-t-on à un ordinateur ?

| | |
|---|---|
| **Java** | **a+b** |
| **Assembleur** | **add a, b** |
| **Machine** | **1000110010100000** |

# Qu'est-ce qu'un calcul ?

- Modèle de base: séquentiel -- les instructions sont exécutées l'une après l'autre
- Algorithmes: déterministes ou non (probabilistes/randomisés)
- Autres modèles: vectoriels, parallèles (SIMD), distribués.
- Modèles du futur: quantique, biologique, etc.

# Qu’est-ce qu’un programme ?

- C'est la traduction dans un langage compréhensible par la machine d'un ensemble d'algorithmes permettant de résoudre un problème

- Un ordinateur effectue les calculs beaucoup plus efficacement que les humains

# Pourquoi Java ?

- Langage haut niveau
- Relativement simple
- Très répandu
- Gestion de la mémoire automatique

- Dès que l'on connaît un langage, il est facile d'en apprendre d'autres

# À quoi ressemble un programme Java ?

```
public class Wagon{
  final static int WMAX = 100; // constante
  static int n; // variable de classe
  String nom;   // champ d'un objet
  // une fonction (méthode de classe)
  public static void print(Wagon w){
    System.out.println(w.nom);
  }
  // fonction principale
  public static void main(String[] args){
    Wagon w = new Wagon();
    w.nom = "Thalis";
    print(w);
  }
}
```

# Constituants de base du langage

- Identificateurs : séquences de lettres et de chiffres commençant par une lettre, séparées par des espaces, caractères de tabulation, retours à la ligne ou des caractères spéciaux (`+`, `-`, `*`, etc.).

- mots-clefs : **`class`**, **`public`**, **`static`**, etc.

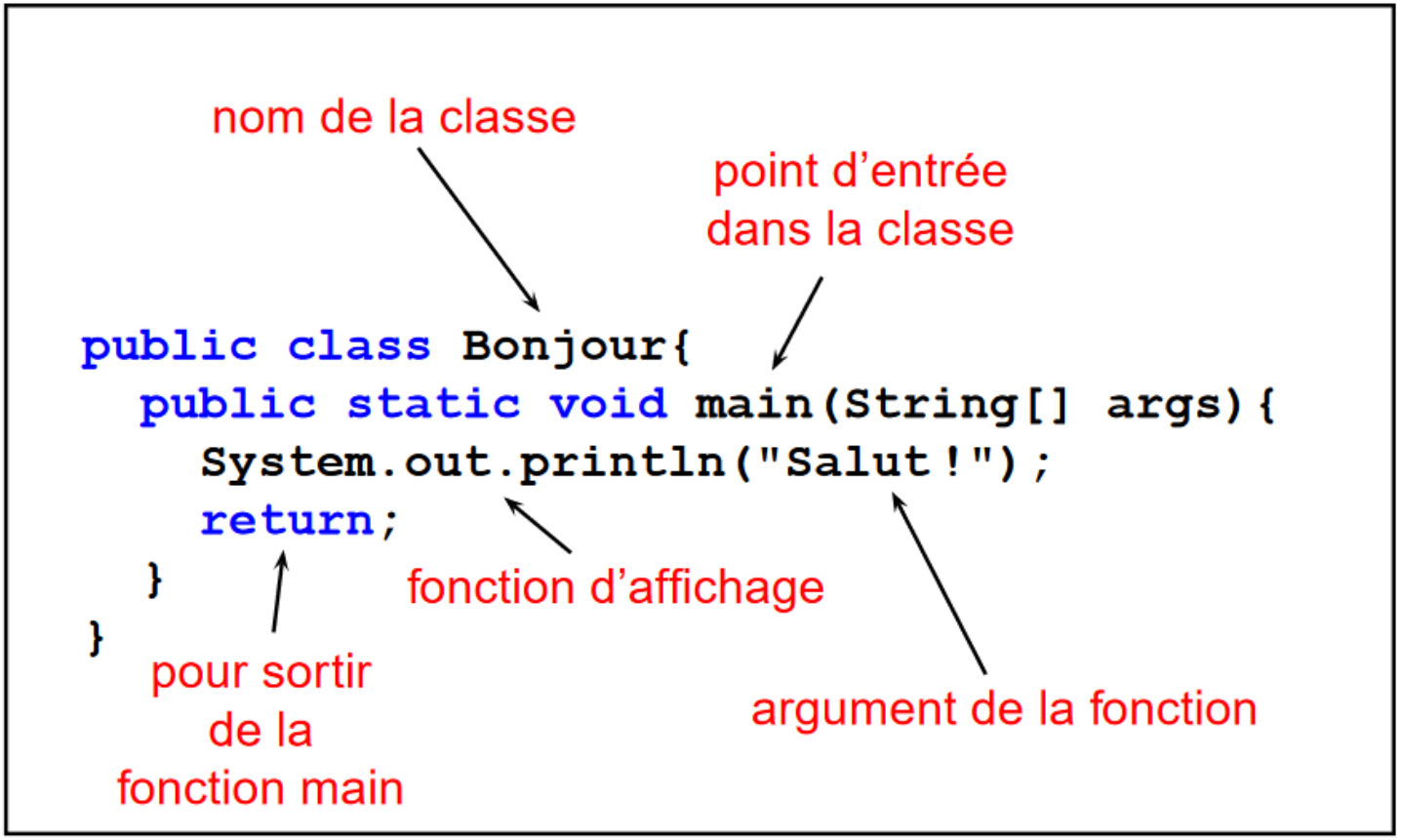
- types primitifs: entiers (**`int`**), réels (**`double`**), caractères (**`char`**).
- Opérations arithmétiques: `+`, `-`, `*`, `/`, `%` (modulo).
- Bibliothèque mathématique : **`Math.sqrt()`**, **`Math.PI`**, etc.

# Premier programme Java

```
public class Bonjour{
   public static void main(String[] args){
      System.out.println("Salut !");
      return;
   }
}
```

Mots en bleu: mots-clefs de Java

# Premier programme Java



# Compilation, Interprétation

- On écrit le programme dans un fichier `Bonjour.java`, puis on le compile avec la commande:

```
javac Bonjour.java
```

- Puis on l'exécute avec la commande:

```
java Bonjour
```

# Introduction aux variables

- On étend notre machine avec de la « **mémoire** » de façon à pouvoir stocker des valeurs (on « représente l'information »)
- La mémoire fait partie de l'environnement de la machine
- La mémoire est accessible au travers de la notion de « **variables** » (une variable permet de stocker une unité d'information)
- On peut accéder et modifier les variables (on « traite l'information »)

# Variables

Une variable est représentée par un identificateur, elle a un type (c'est-à-dire qu'on doit savoir dans quel ensemble elle prend ses valeurs).

On peut utiliser sa valeur, lui en affecter une (nouvelle).

Une variable doit toujours être déclarée :

```
int x;
double u;
```

# Types primitifs: Entiers

| Type | Taille (en octets) | Valeur minimal | Valeur maximale |
|---|---|---|---|
| byte | 1 | -128 | 127 |
| short | 2 | -32 768 | 32 767 |
| int | 4 | -2 147 483 648 | 2 147 483 647 |
| long | 8 | -9 223 372 036 854 775 808 | 9 223 372 036 854 775 807 |

- Attention à l'utilisation du bon type en fonction de vos besoins

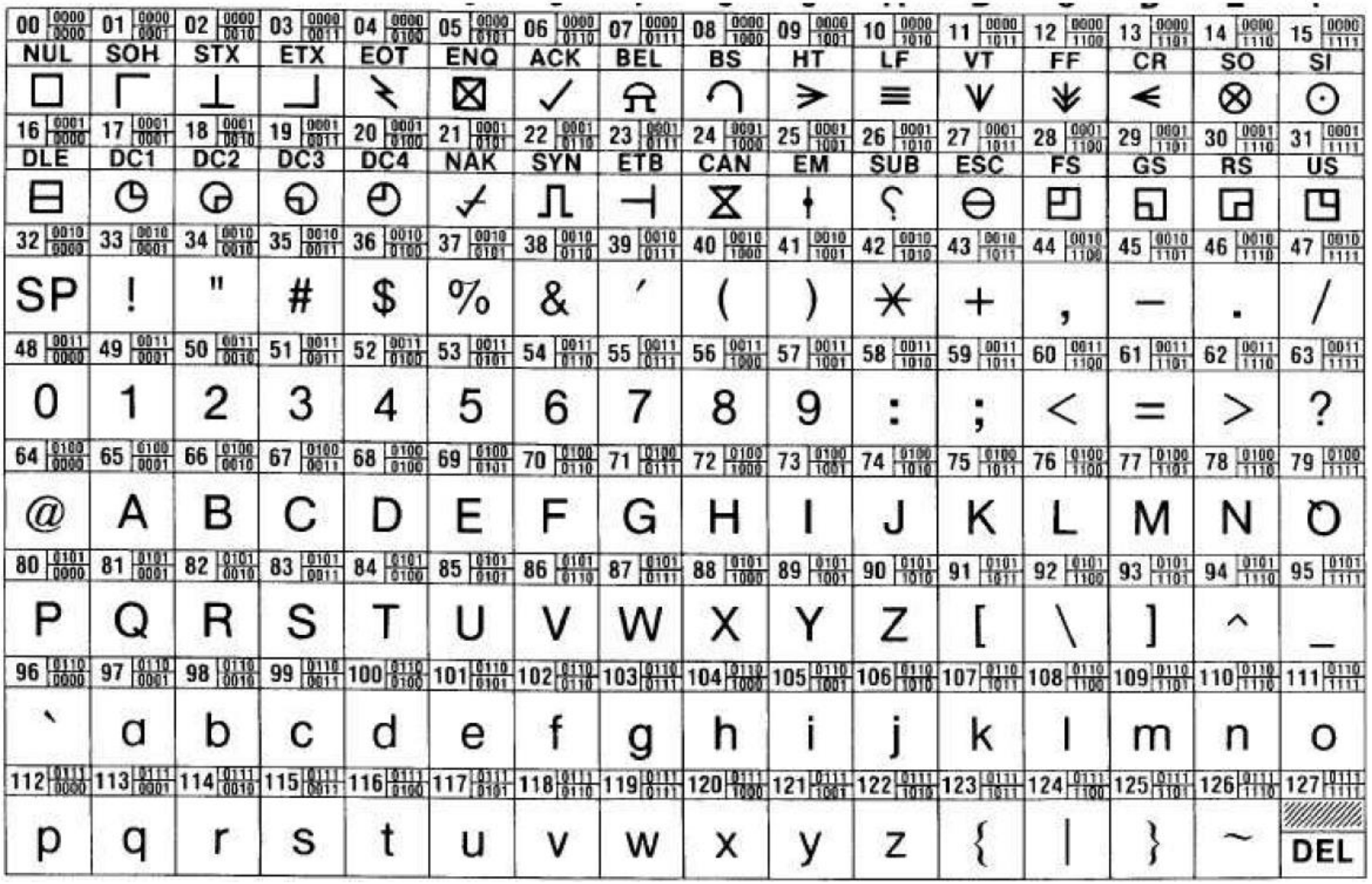
```
short x = 32767 + 1; // valeur de x? -32768
```

  - Dans ce module on utilisera seulement le type int

# Type primitif: Caractères

- Type Java: **char**
- Codage des caractères alphanumériques : réalisé par une association arbitraire et conventionnelle entre un caractère et une suite de bits.
- | Code ASCII | Caractère |
  |---|---|
  | 00100011 | # |
  | 00100100 | $ |
  | 00100101 | % |
  | 01111010 | z |
  | 01111011 | { |
  | 00100001 | ! |
  | 00110001 | 1 |

# Table ASCII des caractères



| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NUL | SOH | STX | ETX | EOT | ENQ | ACK | BEL | BS | HT | LF | VT | FF | CR | SO | SI |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB | CAN | EM | SUB | ESC | FS | GS | RS | US |
| 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| SP | ! | " | # | $ | % | & | ' | ( | ) | * | + | , | − | . | / |
| 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 |
| @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 |
| P | Q | R | S | T | U | V | W | X | Y | Z | [ | \ | ] | ^ | _ |
| 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
| ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 |
| p | q | r | s | t | u | v | w | x | y | z | { | \| | } | ~ | DEL |

# Types primitifs: Réels

- Types Java: **float** et **double**
- Impossibilité de représenter tous les réels (donc seul un sous-ensemble est encodé)
- Un arrondi est nécessairement opéré lors des conversions et des calculs (ce qui donne lieu à beaucoup d'erreurs informatiques!)

# Types primitifs: Réels

- Norme IEEE 754:
  - on stocke le signe, l'exposant et la partie fractionnaire

| s | e | f |
|---|---|---|
| 1 | 10101010101 | 101010101010...10100101 |
| *Signe* | *Exposant biaisé* | *Partie fractionnaire* |

# Précision

- Le premier bit de la mantisse d'un nombre *normalisé* étant toujours 1, il n'est représenté dans aucun de ces deux formats : on parle de bit implicite. Pour ces deux formats, les précisions sont donc respectivement de 24 et de 53 bits.

| | encodage | Signe | Expos-ant E | Man-Tisse M | Valeur D'un nombre | Chiffres significatifs |
|---|---|---|---|---|---|---|
| Simple précision | 32 bits | 1 bit | 8 bits | 23 bits | $(-1)^S \times M \times 2^{(E-127)}$ | 7 |
| Double précision | 64 bits | 1 bit | 11 bits | 52 bits | $(-1)^S \times M \times 2^{(E-1023)}$ | 16 |

# Type primitif: Booléens

- Type Java: **boolean**
- Seulement 2 valeurs possibles: **true** ou **false**

**Expressions logiques:** évaluées à **`true`** ou **`false`**, combinaisons d'opérateurs de comparaison arithmétiques:
`<`, `<=`, `>`, `>=`, `==`, `!=`
et d'opérateurs logiques: `!` (négation), `&&` (et), `||` (ou).

```
x > 8;
x == 7;
y != ((x == 1) && (z > 9));
```

# Affectation

```
u = e;
```

- `u` est une variable et `e` une expression ;
- les deux ont même type ;
- l'expression `e` est évaluée, puis la variable `u` prend pour valeur le résultat de cette évaluation : on a affecté la valeur de `e` à `u`.

```
x = -155;
z = x + 3;
```

Une variable doit toujours être initialisée.

# Affectation

On peut condenser déclaration et initialisation :

```
int x = -155;
```

Une instruction idiomatique : avec le mécanisme d'affection, les instructions :

```
int i = 3;
i = i + 1;
```

sont valides, car on calcule d'abord **`i+1`** et le résultat 4 est mis dans **`i`** à la place de 3.

# Instructions compactes

`i = i+1;` ou `i += 1;` (idem pour `-`, `*`, `/`).
`i++;` ou `++i;` (idem avec `-`).

Post-incrémentation

| | |
|---|---|
| `n = 5;` | |
| `m = n++;` | `m = n;` |
| | `n += 1;` |

Pré-incrémentation

| | |
|---|---|
| `n = 5;` | |
| `m = ++n;` | `n += 1;` |
| | `m = n;` |

# Expressions logiques et arithmétiques

- Priorité des opérateurs

| Niveau | Opérateur | Description | Exemple |
|---|---|---|---|
| 1 | ++ | incrémentation postfixe | x++ |
| | -- | décrémentation postfixe | x-- |
| 2 | ++ | incrémentation préfixe | ++x |
| | -- | décrémentation préfixe | --x |
| | - | changement de signe | -x |
| | + | signe + | +x |
| | ! | NON logique | ! x |
| 4 | * | multiplication | x * y |
| | / | division | x / y |
| | % | modulo | x % y |
| 5 | + | addition | x + y |
| | - | soustraction | x - y |
| 7 | < | plus petit que | x < y |
| | <= | plus petit ou égal à | x <= y |
| | > | plus grand que | x > y |
| | >= | plus grand ou égal à | x >= y |
| 8 | == | égal | x == y |
| | != | différent | x != y |
| 10 | ^ | OU logique exclusif | x ^ y |
| 12 | && | ET logique | x && y |
| 13 | \|\| | OU logique inclusif | x \|\| y |
| 14 | ? : | opérateur conditionnel | x ? y : z |
| 15 | = | affectation | x = y |
| | +=, -=, *=, /=, %= | affectations composées | x += y |

**Figure 15.** Priorité de quelques opérateurs Java.

Evaluation d'expressions de gauche à droite:

Int x = 1 - 2 - 3; // donc x= (1-2)-3

(sauf pour les opérateurs 2,3, 14 et 15)

Int x = y = 2 + 3; // donc x= (y = (2+3))

Exemples d'expressions

- Int x = 1 + 2 * 3 / 6; // x=?
- Int x = 3 * 5 - 2 * 1 - 12 / 2 * 3; // x=?
- b=-9 >= 6 && true || 7*3 < 2; // b=?

- b=2 * 7 + 12 >= 17 - 1 / 2.0 || !true; // b=?

(parenthésez les expressions pour vous aider)

# & | &&?

```
boolean a, b;

Operation     Meaning                          Note
---------     -------                          ----
   a && b     logical AND                       short-circuiting
   a || b     logical OR                        short-circuiting
   a &  b     boolean logical AND               not short-circuiting
   a |  b     boolean logical OR                not short-circuiting
   a ^  b     boolean logical exclusive OR
  !a          logical NOT

short-circuiting        (x != 0) && (1/x > 1)   SAFE
not short-circuiting    (x != 0) &  (1/x > 1)   NOT SAFE
```