

Introduction à la Programmation Orientée Objet

2 - Premiers pas avec le langage Java

Valentin Honoré

valentin.honore@ensiie.fr

FISA 1A

1 Programmation **impérative**

- le plus répandu
- **Principe** : opérations en séquence qui modifient l'état du programme
- **Programmation procédurale** (ex : C) : concevoir en terme de structures de données et de traitements
- **Programmation objet** : conception centrée sur des objets qui interagissent entre eux
 - On ne s'intéresse plus à la mise en œuvre d'un objet, mais aux fonctionnalités qu'il fournit
 - Objet = entité du programme fournissant des fonctionnalités
 - Encapsule une structure de données et des méthodes qui manipulent cette structure de données
 - Expose des fonctionnalités

2 Programmation déclarative

- **fonctionnelle** : applications comme un ensemble de fonctions mathématiques (ex : Lisp)
- **logique** : les composants d'une application sont des relations logiques (ex : Prolog)

3 Programmation concurrente, synchrone etc

- 1 Premiers pas avec le langage JAVA
- 2 Exécution d'un programme Java
- 3 Types de données et variables en Java
- 4 Les opérateurs du langage Java
- 5 Les conversions de type
- 6 Structures conditionnelles et blocs

- ▶ Java est un langage de programmation
 - Première version sortie en 1995 (J.Gosling et P. Naughton, SUN)
 - Actuellement en version 19 (Septembre 2022)
 - L'un des langages les plus utilisés depuis 30 ans

- ▶ Java permet de programmer n'importe type d'application
 - Des jeux, des serveurs, des applications mobiles etc

- ▶ Java possède un ensemble de bibliothèques très complet
 - <https://docs.oracle.com/javase/7/docs/api/overview-summary.html>
 - ~ 2.000.000 de lignes de code
 - Pour tous les domaines applicatifs

- ▶ Syntaxe très proche du C

Dans un programme Java, on trouve :

- ▶ Des mots clés : des mots qui ont un sens réservé dans le langage :
`class`, `if`, `while`, `||`, `&&`, `do`...
- ▶ Des symboles : des noms servant à identifier des éléments
 - Constitué de caractères, chiffres (sauf au début), `_` ou `$`
 - La casse est significative (majuscule vs minuscule)
- ▶ Des types : spécifie la nature de certains symboles
Entier (`int`), chaîne de caractères (`String`), flottant (`float`)...
- ▶ Des valeurs littérales : des valeurs ayant un type donné
`42` (entier), `"Bonjour, monde!"` (chaîne de caractère), `3.14` (flottant)

En bleu : les mots clés du langage

En noir : les symboles (les noms)

En rouge : les types

```
class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello, world!!!");  
    }  
}
```

En vert : des littéraux (ici, une chaîne de caractères)

- ▶ Dans un premier temps, le mot clé `class` sert à définir le nom du programme
 - Ici, le nom du programme est HelloWorld
 - Le code du programme se trouve entre les accolades qui suivent

```
class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello, world!!!");  
    }  
}
```

- ▶ Dans un premier temps, le mot clé `class` sert à définir le nom du programme
 - ❑ Ici, le nom du programme est HelloWorld
 - ❑ Le code du programme se trouve entre les accolades qui suivent

ATTENTION!!!

Le concept de `class` est beaucoup plus complexe que cela.

On le verra dans la suite du cours !

Pour l'instant cette définition suffit (une classe = un programme)



- ▶ La ligne contenant `main` sera expliquée plus tard
 - Cette ligne indique le début du programme
 - Qui se trouve entre les accolades qui suivent

```
class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello, world!!!");  
    }  
}
```

- ▶ Le code du programme est constitué de déclarations
 - System.out.println affiche son paramètre sur le terminal avec retour à la ligne
 - Le "Hello, World!!!" est le paramètre
 - Le point virgule (;) de fin de ligne indique la fin de la déclaration

```
class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello, world!!!");  
    }  
}
```

Mon premier programme JAVA (2/2)

```
/*  
 * Compilation: javac HelloWorld.java  
 * Execution: java HelloWorld  
 *  
 * Prints "Hello, World".  
 *  
 *  
 *****/
```

```
public class HelloWorld {  
  
    public static void main(String[] args) {  
  
        // Prints "Hello, World" in the terminal window.  
        System.out.println("Hello, World");  
    }  
  
}
```

ATTENTION A LA NOTATION DE VOS FICHIERS/CLASSES

- ▶ En JAVA, le nom du fichier et le symbole qui suit `class` doivent coïncider!!!
- ▶ Convention de nommage :
 - Majuscule pour le premier symbole
 - Minuscule pour tous les autres symboles

```
$ emacs HelloWorld.java
```

```
class HelloWorld{  
    ...  
}
```

- 1 Premiers pas avec le langage JAVA
- 2 Exécution d'un programme Java
- 3 Types de données et variables en Java
- 4 Les opérateurs du langage Java
- 5 Les conversions de type
- 6 Structures conditionnelles et blocs

- ▶ **Modulaire**
- ▶ **Portable (JDK)**
- ▶ **Prédictible**
- ▶ Performances - - (virtualisation)
- ▶ Installation :

```
sudo apt install default-jdk
```

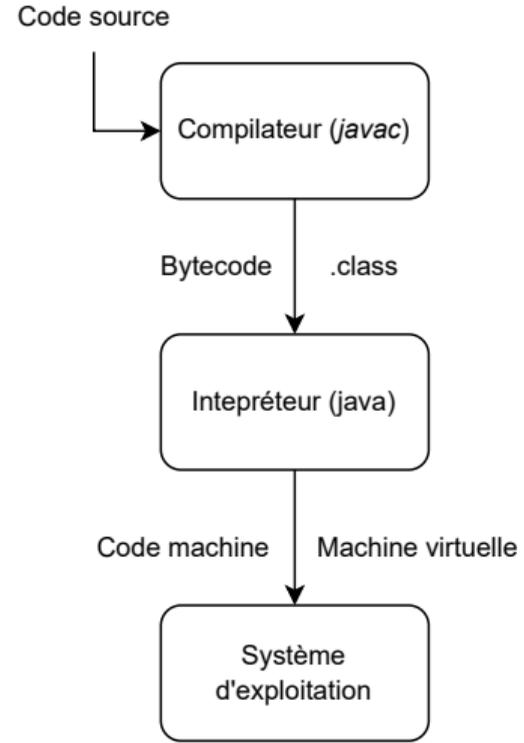


Figure – Processus d'interprétation du langage

▶ Compilation :

```
$ javac HelloWorld.java
```

▶ Virtualisation :

```
$ java HelloWorld
```

▶ Résultat :

```
$ Hello, World
```

▶ le mot clé `-classpath`

- code compilé référence des classes d'autres répertoires
- liste des répertoires, séparés par ' :' (';' sous Windows)
- Exemple :

```
javac -classpath /code/exemple:/code/demo HelloWorld.java
```

▶ Création de "HelloWorld.class" dans un répertoire de compilation

- option `-d` pour destination
- Exemple :

```
javac -d code/bytecode -classpath /code/exemple:/code/demo HelloWorld.java
```

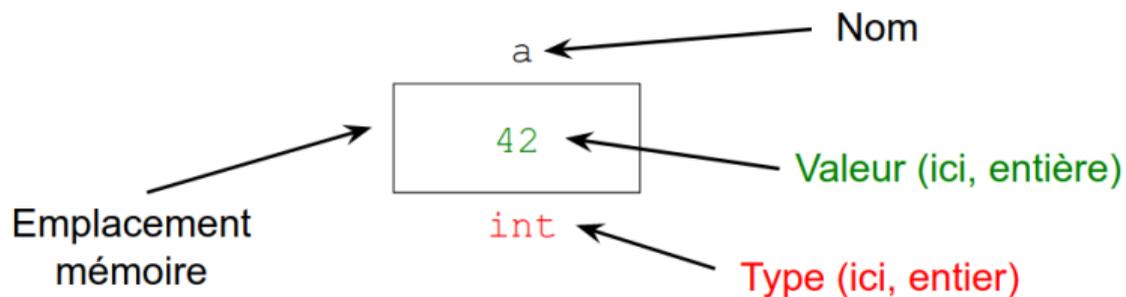
- ▶ le mot clé `-classpath` doit aussi être utilisé
 - même syntaxe que pour la compilation
 - Exemple :

```
java -classpath /code/exemple:/code/demo HelloWorld
```

La suite : syntaxe Java, avec types, opérateurs, conversion de type etc

- 1 Premiers pas avec le langage JAVA
- 2 Exécution d'un programme Java
- 3 Types de données et variables en Java**
- 4 Les opérateurs du langage Java
- 5 Les conversions de type
- 6 Structures conditionnelles et blocs

- ▶ Une variable est un emplacement mémoire
 - Qui possède un nom (le nom de la variable)
 - Un type (la nature de ce que contient la variable)
 - Et une valeur (le contenu de la variable)



En Java, les types de base sont les :

- ▶ Booléens : `boolean` (`true` ou `false`)
- ▶ Entiers (signés) : `byte` (1 octet), `short` (2 octets), `int` (4 octets), `long` (8 octets)
- ▶ Réels : `float` (4 octets), `double` (8 octets)
- ▶ Caractères : `char` (un caractère unicode)
- ▶ Chaînes de caractères : `String` (plusieurs caractères)

Exemple de déclaration de variables

Définition avec :

- ▶ `type` symbole; /* ? valeur indéfinie ? */
- ▶ ou `type` symbole = `literal`;

```
class HelloWorld {  
    public static void main(String[] args) {  
        String msg = "Coucou"; /* Coucou */  
        char c = '!'; /* ! */  
        int x = 2; /* 2 */  
        int y = x + 5; /* 7 (addition entiere) */  
        float z; /* ? valeur indefinie ? */  
        boolean b = true; /* true */  
    }  
}
```

- ▶ Un entier : une valeur entière sans symbole particulier
 - Si suivi de `l`, valeur de type `long` (`2l`)
 - Sinon de type `int` (`2`)

- ▶ Un réel : une valeur avec un point sans symbole particulier (OU AVEC EXPOSANT $3.14e7 == 3.14 \times 10^7$)
 - Si suivi de `f`, valeur de type `float` (`3.14f`)
 - Sinon de type `double` (`3.14`)

- ▶ Un caractère : un caractère entouré d'apostrophes (`'a'`)

- ▶ Une chaîne de caractères : une suite de caractères entourée de guillemets (`"Ceci est une chaîne"`)

Type	Classe éq.	Valeurs	Portée	Défaut
boolean	Boolean	true ou false	N/A	false
byte	Byte	entier signé	$\{-128 \dots 128\}$	0
char	Character	caractère	$\{\backslash u0000 \dots \backslash uFFFF\}$	$\backslash u0000$
short	Short	entier signé	$\{-32768 \dots 32767\}$	0
int	Integer	entier signé	$\{-2147483648 \dots 2147483647\}$	0
long	Long	entier signé	$\{-2^{31} \dots 2^{31} - 1\}$	0
float	Float	réel signé	$\{-3.4028234^{38} \dots 3.4028234^{38}\}$	0.0
double	Double	réel signé	$\{-1.797693134^{308} \dots 1.797693134^{308}\}$	0.0

Table – Types primitifs de données en Java

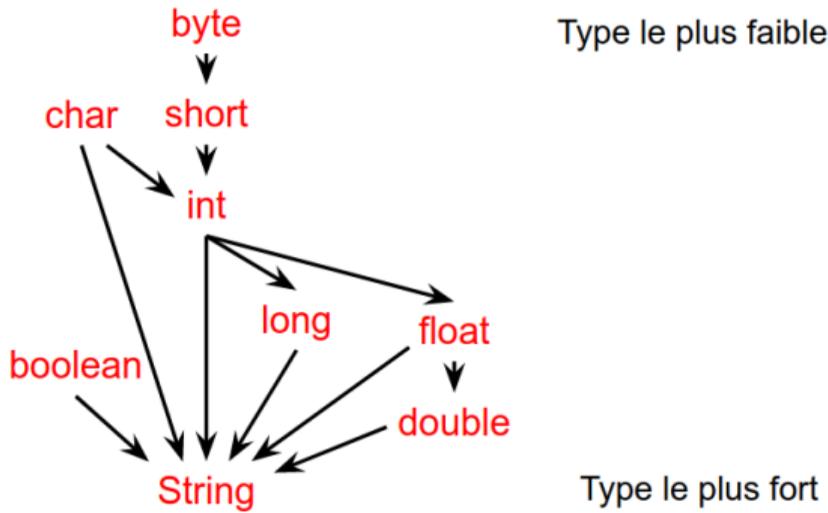
- 1 Premiers pas avec le langage JAVA
- 2 Exécution d'un programme Java
- 3 Types de données et variables en Java
- 4 Les opérateurs du langage Java**
- 5 Les conversions de type
- 6 Structures conditionnelles et blocs

- ▶ Opérations arithmétiques sur les nombres (entiers ou flottants) `+`, `-`, `*`, `/`, `%` (modulo), `++` (incrémente), `--` (décrémente)
- ▶ Opérations bit à bit (sur les entiers) `&` (et), `|` (ou), `^` (xor), `~` (complément), `<<` (décalage à gauche), `>>` (décalage à droite)
- ▶ Opérations sur les booléens `&&` (et), `||` (ou), `!` (non)
- ▶ Opérations sur les chaînes de caractères `+` (concaténation)

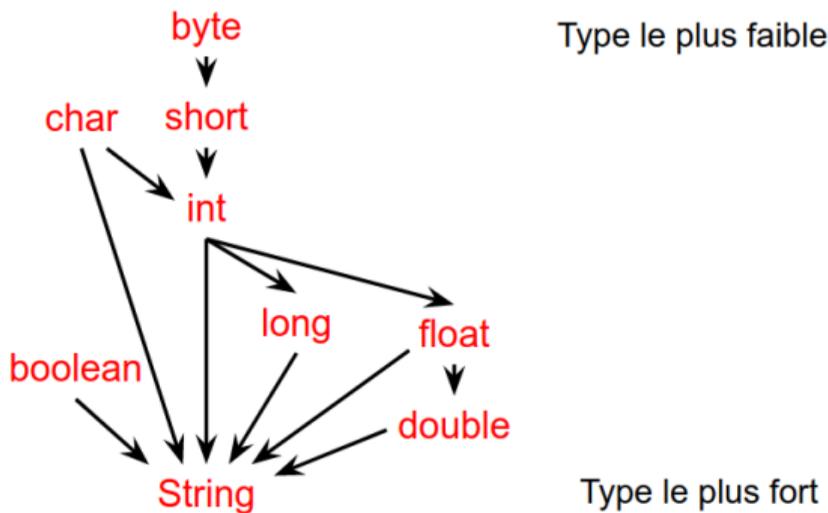
- ▶ Opérations de comparaison `==`, `<=`, `<`, `>=`, `>`, `!=` (différent)
- ▶ Opération d'affectation (tous types) `=`
- ▶ Combinaison d'affectation avec d'autres opérations possible `+=`, `/=`, `»=` etc.
(Exemple : `a += 42`)

- 1 Premiers pas avec le langage JAVA
- 2 Exécution d'un programme Java
- 3 Types de données et variables en Java
- 4 Les opérateurs du langage Java
- 5 Les conversions de type**
- 6 Structures conditionnelles et blocs

- ▶ Avant d'effectuer une opération, un type peut être converti vers un type plus fort



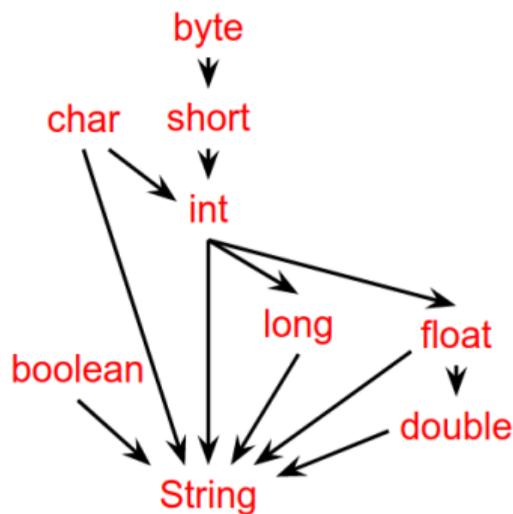
- ▶ Avant d'effectuer une opération, un type peut être converti vers un type plus fort



Remarque

Lors de la conversion d'un char vers un int, c'est le numéro de caractère qui est renvoyé (par exemple 'a' a la valeur 97)

- ▶ Avant d'effectuer une opération, un type peut être converti vers un type plus fort



Type le plus faible

```
int x = 3;
Double y = x + 2.2; // 5.2
String s1 = "Coucou" + x;
/* Coucou3 */
char c = 'a';
String s2 = "Coucou" + c;
/* Coucoua */
int d = c + 1;
/* 98 car 'a' = 97 */
```

Type le plus fort

Quand il n'existe pas de conversion implicite, on utilise la conversion explicite avec (**type**)

- ▶ Dans ce cas, Java tronque le nombre

```
double x = 97.7;  
int y = (int)x; /* 97 */  
char c = (char)y; /* 'a' car c vaut 97 */  
byte b = (byte)(y * 3); /* 291 modulo 256 = 35 */
```

- 1 Premiers pas avec le langage JAVA
- 2 Exécution d'un programme Java
- 3 Types de données et variables en Java
- 4 Les opérateurs du langage Java
- 5 Les conversions de type
- 6 Structures conditionnelles et blocs**

La structure d'un programme Java est donnée par des blocs

- ▶ Commence par un { et termine par un }
- ▶ Regroupe un ensemble de déclarations

```
class Test {  
    public static void main(String[] args) {  
        if(0 == 1) {  
            System.out.println("Cette machine est bizarre");  
            System.out.println("Elle pense que 0 == 1 !");  
        }  
    }  
}
```

- ▶ Accolades optionnelles si seulement une déclaration dans le bloc

Structures conditionnelles (1/3)

Schéma conditionnel simple

- ▶ si alors ... sinon (si alors ... sinon ...)
- ▶ Parties `else if` et `else` optionnelles

```
class Exemple {
    static void main(String[] args) {
        int a = 21 * 2;
        if(a == 42) {
            System.out.println("Super !");
            System.out.println("Java, c'est facile !");
        }
        else if (a<42){
            System.out.println("Gros soucis!!!");
        }
        else
            System.out.println("Perturbation dans la Force");
    }
}
```

Schéma conditionnel plus complexe

- ▶ Si val vaut v1, exécute body1
- ▶ Sinon, si val vaut v2, exécute body2
- ▶ Sinon, si ...
- ▶ Sinon, exécute bodyn

```
class Exemple {
    static void main(String[] args) {
        int c = a;
        switch(c) {
            case 'a': System.out.println("c est un a"); break;
            case 'b': System.out.println("Bizarre..."); break;
            default: System.out.println("Echec?"); break;
        }
    }
}
```

Attention avec le `switch`

- ▶ Si pas de `break`, continue l'exécution
- ▶ Attention au code "spaghetti" !

```
class Exemple {
    static void main(String[] args) {
        char c = a;
        switch(c) {
            case 'a': System.out.println("uniquement a");
            case 'b': System.out.println("a ou b"); break;
            case 'c': System.out.println("uniquement c");
            default: System.out.println("ni a, ni b");
        }
    }
}
```

- ▶ Boucle `while`
Tant que `cond` faire `body`
- ▶ Boucle `do ... while`
Faire `body` tant que `cond`
- ▶ Boucle `for`
Exécute `init`
Tant que `cond` faire
 `body` puis `iter`

```
while(cond) {  
    body  
}
```

```
do {  
    body  
} while(cond)
```

```
for(init; cond; iter) {  
    body  
}
```

Quelques exemples (rappels du C)

```
int x = 0;
while(x < 10) {
    System.out.println(x);
    x++;
}
```

```
for(int x=0; x<10; x++) {
    System.out.println(x);
}
```

```
int x=10;
do {
    x--;
} while(x >= 3);
```

- ▶ En bash :
 - Toutes les variables sont de type chaîne de caractères (PATH=/usr/bin)
 - Les blocs d'instructions sont délimités par des mots clés (do ... done, if ...; then ... fi etc...)

- ▶ En Java :
 - Le type d'une variable est explicite (int x=100 → entier, String name="Arya" → chaîne de caractères)
 - Les blocs sont explicitement délimités par des accolades { ... }

- ▶ Avant d'effectuer une opération arithmétique, Java convertit un **byte**, un **short** ou un **char** en **int**

```
byte x = 1;  
byte y = 2;  
byte z = x + y;
```

- ▶ Interdit car :

Java convertit x et y en **int** avant d'effectuer l'opération

→ le résultat est un **int**

et il est impossible d'affecter un **int** dans un **byte**

- ▶ Si les deux membres de l'opération sont des littéraux entiers (`byte`, `short`, `int`) ou caractères (`char`)
 - alors Java effectue normalement le calcul en `int`
- ▶ Mais après le calcul, s'il faut affecter le résultat à un des types entiers ou au type `char`, Java convertit le résultat si le nombre n'est pas trop grand

Encore plus de bizarreries !

```
char c = 'a';  
char d = c + 1;
```

Interdit car c n'est pas un littéral
(il est impossible de convertir de `int` vers `char`
lors de l'affectation)

```
char c = 'a' + 1;
```

Autorisé car 'a' est un littéral
('a' devient le nombre 97, la somme vaut 98,
qui représente le caractère 'b')