



# PROGRAMMATION JAVA

Jérémy PERROUAULT



# INTRODUCTION

Les bases

# BIT — OCTET

Un humain occidental compte aujourd'hui en base 10

- 0 1 2 3 4 5 6 7 8 9

Une machine compte en base 2 – c'est ce qu'on appelle le « langage binaire »

- 0 1

Avec 2 valeurs (0 ou 1), on ne va pas très loin !

- Introduction de l'octet
  - Suite de 8 bits [0 0 0 0 0 0 0 0]

# BIT — OCTET

On retrouve aussi la base 16, qu'on appelle « hexadécimal »

- 0 1 2 3 4 5 6 7 8 9 A B C D E F

Codé sur un octet

Base 10	Base 2	Base 16
0	0000 0000	0
2	0000 0010	2
15	0000 1111	F
255	1111 1111	FF

# BIT — OCTET

0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	1	1
4	0	0	0	0	0	1	0	0
8	0	0	0	0	1	0	0	0
16	0	0	0	1	0	0	0	0
32	0	0	1	0	0	0	0	0
64	0	1	0	0	0	0	0	0
128	1	0	0	0	0	0	0	0
255	1	1	1	1	1	1	1	1

# BIT — OCTET

Un octet va donc de 0 à 255 (ou  $2^8$ , pour "2 possibilités puissance 8 bits")

- OU de -128 à 127
  - On parle alors de nombres *signed* : le premier bit est utilisé pour préciser le signe (0 = positif, 1 = négatif)
  - 0 0 0 0 0 0 0 1 = 1
  - 1 0 0 0 0 0 0 1 = -1 (au lieu de 129 pour un nombre *unsigned*)

Pour aller jusqu'à 256, l'encodage sera forcément sur 2 octets minimum

- MAIS, 2 octets = 16 bits =  $2^{16}$  possibilités = 65 536 possibilités = 0 à 65 535

En JAVA, tous les types (sauf char) sont *signed*

# BIT — OCTET

Les caractères sont stockés de la même façon ! (*unsigned*)

Une équivalence binaire <-> caractère a été imaginée : jeu de caractères

- Norme ASCII, sur 1 octet
- Norme Unicode, sur 2 octets

A ne pas confondre avec les encodages de caractères

- UTF-8 (pour 8 bits, 1 octet), compatible Unicode ET ASCII
- UTF-16 (pour 16 bits, 2 octets)
- UTF-32 (pour 32 bits, 4 octets)

# BIT — OCTET

Binaire	Numérique	Alphabet
0 1 0 0 0 0 0 1	65	A
0 1 0 0 0 0 1 0	66	B
0 1 1 0 0 0 0 1	97	a
0 1 1 0 0 0 1 0	98	b

Chaque système d'encodage a sa propre équivalence

- Pour les caractères standards, c'est le même code
- Pour les caractères exotiques (accents par exemple), le code peut être différent



# LES VARIABLES

Type algorithmique	Type JAVA	Taille en octets
Entier très Court	byte	1
Entier Court	short	2
Entier	int	4
Entier Long	long	8
Réel	float	4
Réel Long	double	8
Caractère	char	2 (Unicode)
Booléen	boolean	1
Vide	void	



# NOMBRES RÉELS

Les machines ne comptent pas  
comme nous !

# NOMBRE RÉELS

$$0.1 + 0.2 = 0.300000000000000004$$

Base 10	100	10	1	.	1/10	1/100	1/1000		
			0	.	1				
Base 2	4	2	1	.	1/2	1/4	1/8	1/16	1/32
			0	.	0	0	0	1	1..

# NOMBRE RÉELS

$$1/3 = 0.3333333...$$

Base 10	100	10	1	.	1/10	1/100	1/1000	1/10000
			0	.	3	3	3	3...

En tant qu'humain

- $1/3 + 1/3 + 1/3 = 1$
- Parce qu'on est capable de comprendre et de percevoir la récursivité des nombres

Mais imaginons que nous n'en soyons pas capables (comme une machine)

- $1/3 + 1/3 + 1/3$
- $0.3333330 + 0.3333330 + 0.3333330 = \mathbf{0.999999}$
- $0.3333334 + 0.3333334 + 0.3333334 = \mathbf{1.0000002}$

# NOMBRE RÉELS

Donc en base 2

- $0.1 (1/10) = 0.00011001100110011..$
- $0.2 (2/10) = 0.00110011001100110..$
- L'ordinateur stockant un nombre limité des bits, il va couper la récursivité
  - Exemple pour 52 bits (normes IEEE 754 – double)
  - $0.1 = 0.000110011001100110011001100110011001100110011001$
- En décimal, la valeur est de
  - 0.1000000000000000055511151231257827021181583404541015625
- Autrement dit, 0.1 pour un ordinateur est légèrement supérieur à 0.1 ...
- Et donc, pour l'ordinateur :  $0.1 + 0.2 \neq 0.3$

# NOMBRE RÉELS

## Exemples en JAVA

```
System.out.printf("%.15f", (double)0.1);
```

```
0,10000000000000000
```

```
System.out.printf("%.15f", (float)0.1);
```

```
0,100000001490116
```

```
System.out.printf("%.20f", 0.1 + 0.2);
```

```
0,300000000000000004000
```

```
System.out.printf("%.20f", (float)(0.1 + 0.2));
```

```
0,30000001192092896000
```



# APPLICATIONS JAVA

Compilation et exécution

# APPLICATIONS JAVA

## JVM

- Java Virtual-Machine, qui permet d'exécuter du code Java byte code

## JRE

- Java Runtime Environment, nécessaire à l'exécution
- JVM + API Java standards

## JDK

- Java Development Toolkit
- JRE + les outils nécessaires au développement (javac, javadoc, ...)



# APPLICATIONS JAVA

## Java a 2 compilateurs

- javac
  - Compile le code Java (fichiers .java) en byte code (fichiers .class)
  - Avant l'exécution du programme
- JVM (Java Virtual-Machine) JIT (Just-In-Time)
  - Compile le byte code en code natif
  - Après l'exécution du programme
  - Permet de compiler à chaud ce qui est nécessaire au fonctionnement du programme, directement sur l'hôte

# EXERCICE

## Télécharger et installer

- JDK 8
- Eclipse IDE for Java EE Developers (Eclipse JEE), version Oxygen minimum

## Exécuter Eclipse

- Trouver et activer la perspective "Java"



# ECRIRE UN PROGRAMME

En JAVA

# ECRIRE UN PROGRAMME

Dans les langages hors Web (Java, C#, C++, C, ...)

- Une méthode principale
  - Sui s'appelle "main"
  - Qui attend un tableau de chaines de caractères en argument
  - Qui retourne un statut d'exécution (un code erreur, ou un code de programme terminé correctement)

```
int main(char[][] args) {  
}
```

# ECRIRE UN PROGRAMME

Java est un langage orienté Object

- Il existe un Object, **String**, qui encapsule le tableau de caractères pour la chaîne de caractères
- L'application Java est exécutée au sein de la JVM, c'est elle qui gère les codes de retour
  - La méthode "main" n'a pas besoin de le gérer !

```
static void main(String[] args) {  
}
```

- Sans entrer dans les détails techniques
  - On a besoin d'une classe, qu'on peut appeler **Principale** par exemple, ou **ProgrammePrincipal**
  - Il faut qu'elle ait un programme "main", comme décrit ci-dessus (une option dans Eclipse permet de l'auto-générer)
  - Il faut accéder aux programmes sans instancier la classe, il faudra tous les préfixer du mot-clé "static"
  - On va demander à Eclipse d'exécuter la classe **Principale**, la classe qui a la méthode "main"

# ECRIRE UN PROGRAMME

Déclaration d'un programme sans paramètre, qui ne retourne rien

```
static void nomDuProgramme() {  
}
```

Déclaration d'un programme sans paramètre, qui retourne un Entier

```
static int nomDuProgramme() {  
}
```

Déclaration d'un programme avec 1 paramètre Booléen, qui retourne un Entier

```
static int nomDuProgramme(boolean nomParametre) {  
}
```

Déclaration d'un programme avec 2 paramètres Booléen et Entier, qui ne retourne rien

```
static void nomDuProgramme(boolean nomParametre1, int nomParametre2) {  
}
```

# ECRIRE UN PROGRAMME

Condition Si, Sinon

```
if (a == 0) {  
}  
  
else {  
}
```

OU

```
if ((a == 0) || (b == 0)) {  
}
```

ET

```
if ((a >= 0) && (a <= 10)) {  
}
```

Condition Si, Sinon Si, Sinon

```
if (a == 0) {  
}  
  
else if (b == 0) {  
}  
  
else {  
}
```

# ECRIRE UN PROGRAMME

## Boucle TantQue

```
while (a < 10) {  
}
```

## Boucle Faire..TantQue

```
do {  
} while (a < 10);
```

## Boucle Pour

```
for (int i = 0; i < 10; i = i + 1) {  
}  
  
for (int i = 0; i < 10; i++) {  
}
```



# ECRIRE UN PROGRAMME

Déclarer un entier

```
int a;
```

Déclarer et affecter un entier

```
int a = 42;
```

Déclarer un tableau d'entiers

```
int[] monTab = new int[10];
```

Déclarer et affecter un tableau d'entiers

```
int[] monTab = new int[] {  
    1, 2, 3, 4, 5, 6, 7, 8, 9, 10  
};
```

Déclarer un tableau 2-dimensions

```
int[][] monTab = new int[3][10];
```

# ECRIRE UN PROGRAMME

Suffixe long

```
long a = 425L;
```

Suffixe float

```
float a = 42.23f;
```

Suffixe double

```
double a = 42.23d;
```

# ECRIRE UN PROGRAMME

## Paramètres transmis

- int, boolean, char, long, byte, ..      Transmission par valeur
- Tableaux, objets      Transmission par référence

# EXERCICE

Implémenter les algorithmes en JAVA

- Le sous-programme « taille » sera fourni
- Le sous-programme « chaine » sera fourni



# INTERACTIONS

Avec l'utilisateur

# INTERACTIONS

Nouveaux sous-programmes disponibles (qui vous seront fournis)

- lireEntier()                      Retourne l'entier saisi par l'utilisateur
- lireChaine()                    Retourner un tableau de caractères
- ecrire(..)                        Afficher un texte ou une information à l'utilisateur
- nouvelleLigne()                Afficher une nouvelle ligne à l'utilisateur

```
int age;  
  
ecrire("Saisir votre age : ");  
age = lireEntier();  
nouvelleLigne();  
ecrire("Vous avez saisi : ");  
ecrire(age);
```

# EXERCICES

1. Demander 2 nombres à l'utilisateur puis calculer sa puissance ( $n1$  puissance  $n2$ ), afficher le résultat
2. Demander 3 prénoms à l'utilisateur, et lui afficher s'ils ont été donnés dans un ordre croissant ou non
3. Demander un CA à l'utilisateur, et l'informer de la catégorie du client
  1.  $< 0$  Plus client
  2.  $0 - 200$  Petit client
  3.  $201 - 10000$  Client
  4.  $> 10000$  Grand client
4. Demander  $n$  nombres à l'utilisateur, et afficher quel est le plus grand saisi, et à quelle position il se trouve
  1. La demande s'arrête quand l'utilisateur saisi 0
  2. 5, 10, 4, 74, 25, 0 74 est le plus grand, à la position 4
5. Demander une phrase à l'utilisateur, ranger et afficher les caractères par ordre croissant
6. Demander une phrase à l'utilisateur, ranger et afficher les mots par ordre croissant
7. Constituer une liste de mots, demander à l'utilisateur de saisir un mot, afficher la position de ce mot s'il existe dans votre liste