TELECOM SudParis

IP PARIS

Premiers pas avec Java

CSC 3101

Algorithmique et langage de programmation

Gaël Thomas

Plan du cours

1. Mon premier programme Java
2. Exécution d'un programme Java
3. Variables et types de données
4. Les opérateurs du langage Java
5. Les conversions de type
6. Structures algorithmiques

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 caractère (String), flottant (float)…
- Des valeurs littérales : des valeurs ayant un type donné

  42 (entier), "Bonjour, monde!" (chaîne de caractère), 3.14 (flottant)…

Dans un programme Java, on trouve

- Et des commentaires
  - Utilisés pour expliquer ce que fait le programme
  - Non exécutés, uniquement pour documenter le code

```
/*
 * ceci est un commentaire multi-lignes
 */

/* ceci est un commentaire multi-lignes sur une ligne */

// ceci est un commentaire mono-ligne
```

Mon premier programme Java

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)

Mon premier programme Java

- 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!!!");
  }
}
```

Mon premier programme Java

Interlude :

L'équipe pédagogique tient à s'excuser pour un petit mensonge : le mot clé `class` est infiniment plus complexe que ce qui est présenté ici

Vous comprendrez le rôle exact du mot clé `class` dans les CI3 et CI4

Pour le moment, imaginer que `class` donne le nom du programme n'est pas totalement faux

Mon premier programme Java

- La ligne contenant `main` sera expliquée dans les CI3 à CI5
- Pour le moment
  - 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!!!");
  }
}
```

Mon premier programme Java

- Le code du programme est constitué de déclarations
  - `System.out.println` affiche son paramètre sur le terminal
  - 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!!!");
  }
}
```

Plan du cours

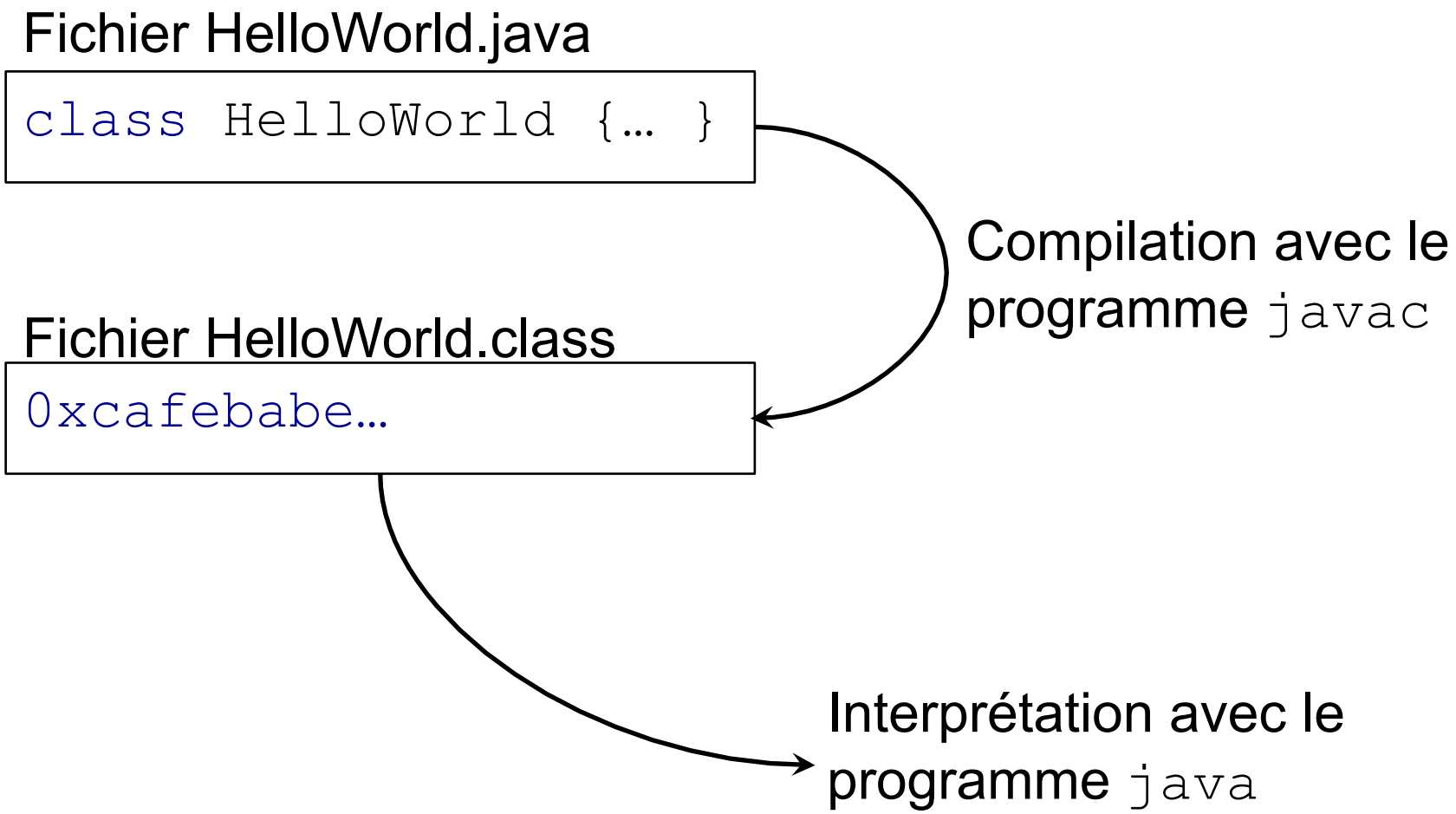
1. Mon premier programme Java
2. Exécution d'un programme Java
3. Variables et types de données
4. Les opérateurs du langage Java
5. Les conversions de type
6. Structures algorithmiques

Exécution d'un programme Java

- Étape 1 : traduction vers un fichier appelé bytecode
  - Le fichier source (contenant le programme Java) n'est pas exécutable en l'état, il doit être traduit
  - Le fichier bytecode contient une forme plus rapide à exécuter que le fichier source (validation syntaxique et sémantique effectuée)
  - La transformation source vers bytecode s'appelle la compilation

- Étape 2 : exécution dans une machine virtuelle Java
  - Un fichier de bytecode n'est pas directement exécutable par un processeur
  - Le programme `java` est capable d'interpréter un fichier bytecode

Exécution d'un programme Java



Mise en perspective

- En `bash` (resp. `python`)
  - Un programme source est un fichier `.sh` (resp. `.py`)
    (contient des déclarations `bash`, resp `python`)
  - Directement exécutable par l'interpréteur `bash` (resp. `python`)

- En `Java`
  - Un programme source est un fichier `.java`
    (contient des déclarations `Java`)
  - Doit être compilé en bytecode (fichier `.class`) avec `javac`
  - Le bytecode est exécutable par l'interpréteur `java`

Conception, compilation, exécution

```
$ emacs HelloWorld.java
```

```
class HelloWorld {
   …
}
```

Éditeur emacs
(fichier HelloWorld.java)

Première étape :
la conception

Conception, compilation, exécution

```
$ emacs HelloWorld.java
```

```
class HelloWorld {
   …
}
```

Attention : le nom du fichier
et le symbole qui suit class
doivent coïncider

Convention majuscule en début
de symbole dans ce cas,
minuscule pour tous les autres
symboles

Conception, compilation, exécution

```
$ emacs HelloWorld.java
$
```

Conception, compilation, exécution

```
$ emacs HelloWorld.java
$ ls
HelloWorld.java
$
```

Conception, compilation, exécution

```
$ emacs HelloWorld.java
$ ls
HelloWorld.java
$ javac HelloWorld.java
$
```

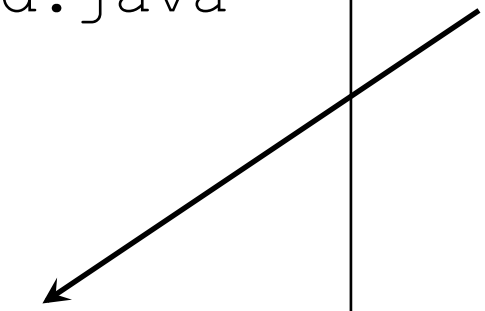
Deuxième étape :
la compilation

Conception, compilation, exécution

```
$ emacs HelloWorld.java
$ ls
HelloWorld.java
$ javac HelloWorld.java
$ ls
HelloWorld.java
HelloWorld.class
$
```

Conception, compilation, exécution

```
$ emacs HelloWorld.java
$ ls
HelloWorld.java
$ javac HelloWorld.java
$ ls
HelloWorld.java
HelloWorld.class
$ java HelloWorld
Hello, world!!!
$
```



Lance l'exécution du programme `HelloWorld.class`

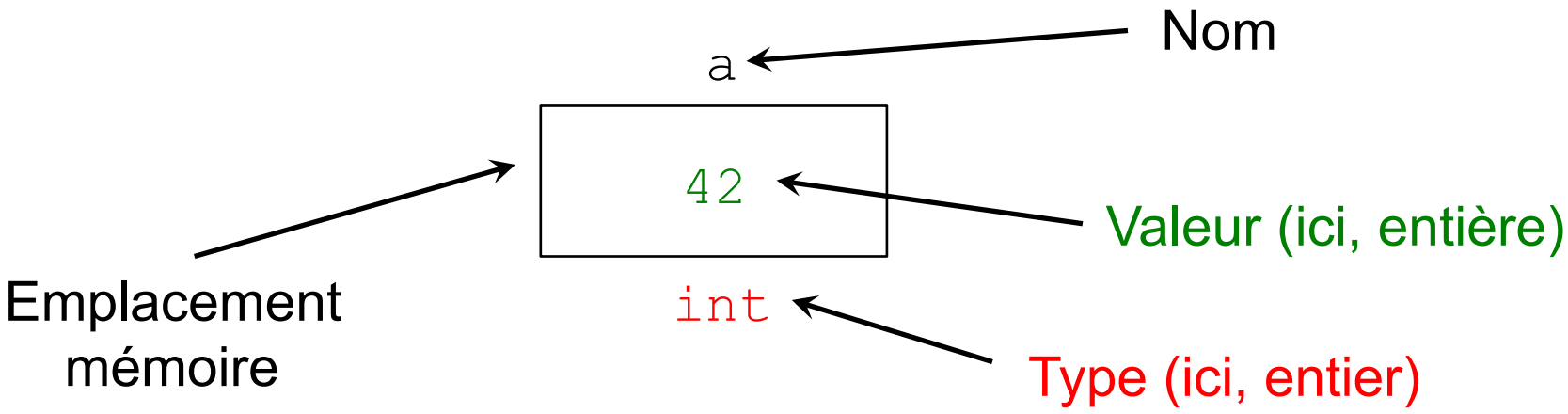
Troisième étape : L'exécution

Plan du cours

1. Mon premier programme Java
2. Exécution d'un programme Java
3. Variables et types de données
4. Les opérateurs du langage Java
5. Les conversions de type
6. Structures algorithmiques

Les variables en Java (1/2)

- 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)



Les variables en Java (2/2)

- 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)

Les variables en Java

- 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 entière) */
    float z;              /* ? valeur indéfinie ? */
    boolean b = true;     /* true */
  }
}
```

Les littéraux en Java

- 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 == $3.14 * 10^7$)
  - 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")

Plan du cours

1. Mon premier programme Java
2. Exécution d'un programme Java
3. Variables et types de données
4. Les opérateurs du langage Java
5. Les conversions de type
6. Structures algorithmiques

Les opérateurs du langage Java (1/2)

- 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)

Les opérateurs du langage Java (2/2)

- Opérations de comparaison
  ==, <=, <, >=, >, != (différent)

- Opération d'affectation (tous types)
  =

- Combinaison d'affectation avec d'autres opérations possible
  +=, /=, >>= etc.

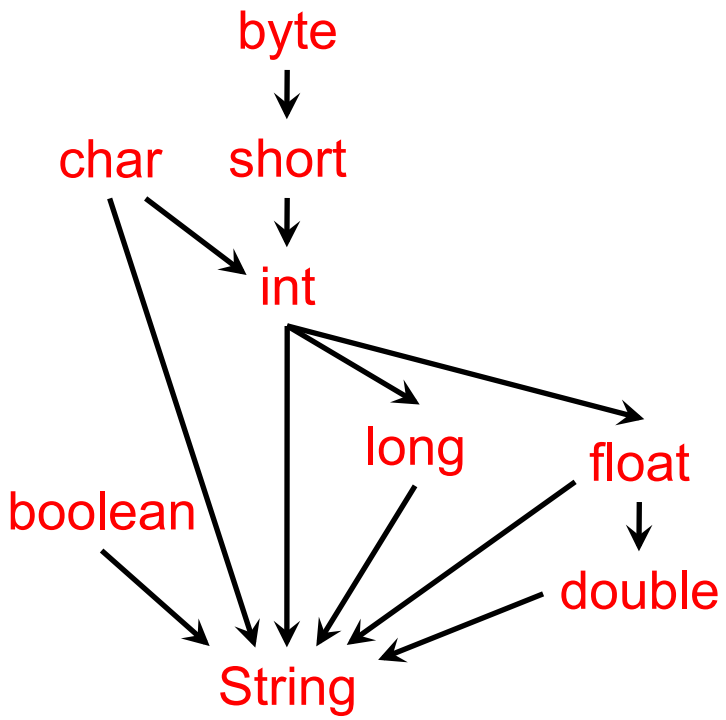
  (Exemple : `a += 42`)

Plan du cours

1. Mon premier programme Java
2. Exécution d'un programme Java
3. Variables et types de données
4. Les opérateurs du langage Java
5. Les conversions de type
6. Structures algorithmiques

Conversion automatique de type

- Avant d'effectuer une opération, les valeurs peuvent être converties vers un type plus fort

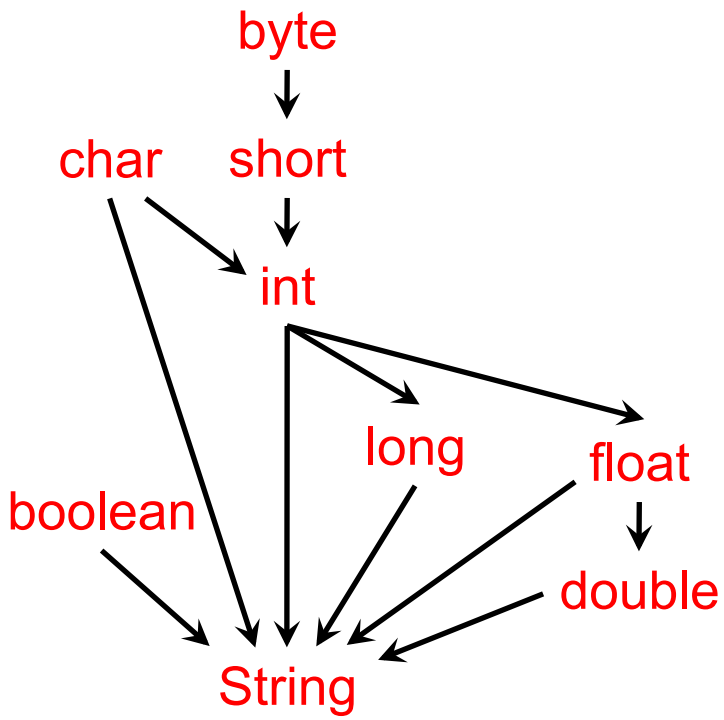


Type le plus faible

Type le plus fort

Conversion automatique de type

- Avant d'effectuer une opération, les valeurs peuvent être converties vers un type plus fort



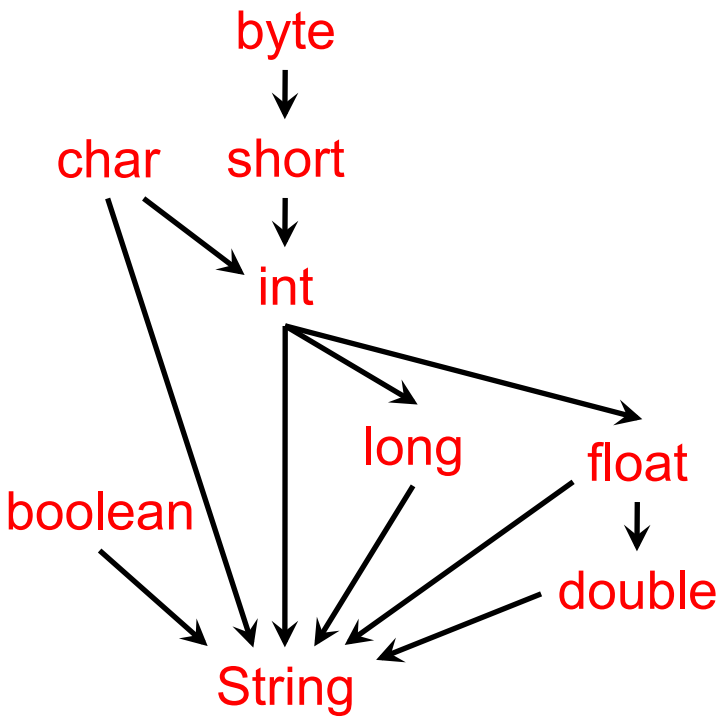
Type le plus faible

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)

Type le plus fort

Conversion automatique de type

- Avant d'effectuer une opération, les valeurs peuvent être converties vers un type plus fort




```
int x = 3;
float 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 */
```

Conversion explicite de type

- 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 */
```

Plan du cours

1. Mon premier programme Java
2. Exécution d'un programme Java
3. Variables et types de données
4. Les opérateurs du langage Java
5. Les conversions de type
6. Structures algorithmiques

Structures algorithmiques 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 !");
    }
  }
}
```

Bloc exécuté si 0 == 1

  - Accolades optionnelles si une unique déclaration dans un bloc
  - Pour être plus précis, un bloc est simplement une déclaration…

Schéma alternatif (1/2)

- Schéma alternatif simple
  - si alors … sinon (si alors … sinon …)
  - Parties `else if` et `else` optionnelles

```
if(cond1) {
  body1
} else if(cond2) {
  body2
} else {
  body3
}
```

Exemple.java

```
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 !");
    }
  }
}
```

Schéma alternatif (2/2)

- Schéma alternatif complexe
  - Si val vaut v1, exécute body1
  - Sinon, si val vaut v2, exécute body2
  - Sinon, si …
  - Sinon, exécute bodyn

```
switch(val) {
  case v1:
    body1;
    break;
  case v2:
    body2;
    break;
  …
  default:
    bodyd;
    break;
}
```

```
switch(c) {
  case 'a': System.out.println("Ceci est un a"); break;
  case 'b': System.out.println("Ceci est un b"); break;
  …
  default: System.out.println("Bizarre"); break;
}
```

Schéma alternatif (2/2)

- Schéma alternatif complexe
  - Si pas de `break`, continue l'exécution avec le `body` suivant
  - Attention au code spaghetti !

```
switch(val) {
  case v1:
    body1;
    break;
  case v2:
    body2;
    break;
  …
  default:
    bodyd;
    break;
}
```

```
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 ☺");
}
```

Schémas itératifs

- Boucle `while`

  Tant que `cond` faire `body`

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

- Boucle `do … while`

  Faire `body` tant que `cond`

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

- Boucle `for`

  Exécute `init`

  Tant que `cond` faire

  `body` puis `iter`

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

Schémas itératifs – exemples

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

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

```
int x;

do {
   x = Math.random() % 10;
} while(x == 3);
```

Java versus python versus bash

- 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 `{ … }`
- En `python`
  - Le type d'une variable est implicite
    (`x=100` ⇒ entier, `name="Arya"` ⇒ chaîne de caractères)
  - Les blocs sont implicitement donnés par l'indentation
- 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…)

Notions clés

- Conception, compilation et exécution d'un programme Java
- Déclaration et typage des variables : `type var;`

  `boolean`, `byte`, `short`, `int`, `long`, `float`, `double`, `char`, `String`
- Opérateurs de base
- Structures algorithmiques
  - Schéma itératif (`if/else`)
  - Schéma itératif (`while`, `for`, `do … while`)

If you want to know more

A propos des conversions de type en Java

Étrangeté

- 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`

Encore plus d'étrangeté (pour votre culture)

- 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**, si 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 d'étrangeté (pour votre culture)

- Si les deux membres de l'opération sont des littéraux entiers

```
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')