

# Programmer en Java

## Programmation Orientée Objet

Thomas BOCQUELET

6 mai 2018

Résumé

Document réalisé à partir du cours de M<sup>r</sup> Grégory BOURGIN :  
bourguin@lisic.univ-littoral.fr

## 1 Présentation

- Qu'est-ce que le Java ?
- Les outils
- Références

## 2 Les éléments du langage

- Types primitifs
- Variables
- Expressions
- Méthodes
- Structures de contrôle

## 3 Suppléments L<sup>A</sup>T<sub>E</sub>X

## 1 Présentation

- Qu'est-ce que le Java ?
- Les outils
- Références

## 2 Les éléments du langage

## 3 Suppléments $\text{\LaTeX}$

## 1 Présentation

- Qu'est-ce que le Java ?
- Les outils
- Références

## 2 Les éléments du langage

- Types primitifs
- Variables
- Expressions
- Méthodes
- Structures de contrôle

## 3 Suppléments $\text{\LaTeX}$

# Présentation

Qu'est-ce que le Java ?

## Histoire

- Alan KEY conçoit le langage « SmallTalk » qui est encore *la* référence dans les langages orientés objets.

# Présentation

Qu'est-ce que le Java ?

## Histoire

- Alan KEY conçoit le langage « SmallTalk » qui est encore *la* référence dans les langages orientés objets.
- Création de nombreuses extensions objet pour le C, C++, Object Pascal, etc.

# Présentation

Qu'est-ce que le Java ?

## Histoire

- Alan KEY conçoit le langage « SmallTalk » qui est encore *la* référence dans les langages orientés objets.
- Création de nombreuses extensions objet pour le C, C++, Object Pascal, etc.
- Au milieu des années 90, Sun publie Java.

# Présentation

Qu'est-ce que le Java ?

Le Java est :

- un langage orienté objet (POO<sup>1</sup>)

- 
1. Programmation Orientée Objet
  2. Java Development Kit
  3. Java Virtual Machine



# Présentation

Qu'est-ce que le Java ?

Le Java est :

- un langage orienté objet (POO<sup>1</sup>)
- une architecture *Virtual Machine*

- 
1. Programmation Orientée Objet
  2. Java Development Kit
  3. Java Virtual Machine

# Présentation

Qu'est-ce que le Java ?

Le Java est :

- un langage orienté objet (POO<sup>1</sup>)
- une architecture *Virtual Machine*
- un ensemble d'*API* variées

- 
1. Programmation Orientée Objet
  2. Java Development Kit
  3. Java Virtual Machine

# Présentation

Qu'est-ce que le Java ?

Le Java est :

- un langage orienté objet (POO<sup>1</sup>)
- une architecture *Virtual Machine*
- un ensemble d'*API* variées
- un ensemble d'outils : le *JDK*<sup>2</sup>

- 
1. Programmation Orientée Objet
  2. Java Development Kit
  3. Java Virtual Machine

# Présentation

Qu'est-ce que le Java ?

Le Java est :

- un langage orienté objet (POO<sup>1</sup>)
- une architecture *Virtual Machine*
- un ensemble d'*API* variées
- un ensemble d'outils : le *JDK*<sup>2</sup>
- portable

- 
1. Programmation Orientée Objet
  2. Java Development Kit
  3. Java Virtual Machine

# Présentation

## Qu'est-ce que le Java ?

Le Java est :

- un langage orienté objet (POO<sup>1</sup>)
- une architecture *Virtual Machine*
- un ensemble d'*API* variées
- un ensemble d'outils : le *JDK*<sup>2</sup>
- portable
  - *JVM*<sup>3</sup> présente sur systèmes Windows, Mac et Unix

- 
1. Programmation Orientée Objet
  2. Java Development Kit
  3. Java Virtual Machine

# Présentation

## Qu'est-ce que le Java ?

Le Java est :

- un langage orienté objet (POO<sup>1</sup>)
- une architecture *Virtual Machine*
- un ensemble d'*API* variées
- un ensemble d'outils : le *JDK*<sup>2</sup>
- portable
  - *JVM*<sup>3</sup> présente sur systèmes Windows, Mac et Unix
  - accompagné d'une librairie standard

- 
1. Programmation Orientée Objet
  2. Java Development Kit
  3. Java Virtual Machine

# Présentation

## Qu'est-ce que le Java ?

Le Java est :

- un langage orienté objet (POO<sup>1</sup>)
- une architecture *Virtual Machine*
- un ensemble d'*API* variées
- un ensemble d'outils : le *JDK*<sup>2</sup>
- portable
  - *JVM*<sup>3</sup> présente sur systèmes Windows, Mac et Unix
  - accompagné d'une librairie standard
- robuste

- 
1. Programmation Orientée Objet
  2. Java Development Kit
  3. Java Virtual Machine

# Présentation

## Qu'est-ce que le Java ?

Le Java est :

- un langage orienté objet (POO<sup>1</sup>)
- une architecture *Virtual Machine*
- un ensemble d'*API* variées
- un ensemble d'outils : le *JDK*<sup>2</sup>
- portable
  - *JVM*<sup>3</sup> présente sur systèmes Windows, Mac et Unix
  - accompagné d'une librairie standard
- robuste
  - mécanisme d'*exceptions*

- 
1. Programmation Orientée Objet
  2. Java Development Kit
  3. Java Virtual Machine



# Présentation

Qu'est-ce que le Java ?

## Attention !

- Le Java n'est pas du JavaScript. Le Java est un langage « généraliste », contrairement au JavaScript qui est orienté sur la programmation Web.

# Présentation

Qu'est-ce que le Java ?

## Attention !

- Le Java n'est pas du JavaScript. Le Java est un langage « généraliste », contrairement au JavaScript qui est orienté sur la programmation Web.
- Le Java n'est pas du C++. Java est un langage *purement* objet et de plus haut niveau.

## 1 Présentation

- Qu'est-ce que le Java ?
- Les outils
- Références

## 2 Les éléments du langage

- Types primitifs
- Variables
- Expressions
- Méthodes
- Structures de contrôle

## 3 Suppléments $\text{\LaTeX}$

Environnements de développement :

- SunJDK
- Eclipse
- IntelliJ (version « community » gratuite, commerciale payante)
- NetBeans

Environnements de développement :

- SunJDK
- Eclipse
- IntelliJ (version « community » gratuite, commerciale payante)
- NetBeans

### Remarque

Dans les TPs, nous utiliserons les environnements de développement IntelliJ et Eclipse.

Liste des outils de Java :

`javac` : compilateur de sources Java

`java` : interpréteur de byte code

`appletviewer` : interpréteur d'applet

`javadoc` : générateur de documentation (HTML, MIF)

`javah` : générateur de header pour l'appel de méthodes natives

`javap` : désassembleur de byte code

`jdb` : debugger

`javakey` : générateur de clés pour la signature de code

Liste des *API* standards :

`java.lang` : types de bases, etc.

`java.util` : HashTable, Vector, Stack, Date...

`java.io` : accès aux entrées/sorties par flux

`java.net` : socket, URL...

`java.sql` : accès homogène aux bases de données

`java.security` : signatures, cryptographie, authentification...

## 1 Présentation

- Qu'est-ce que le Java ?
- Les outils
- Références

## 2 Les éléments du langage

- Types primitifs
- Variables
- Expressions
- Méthodes
- Structures de contrôle

## 3 Suppléments L<sup>A</sup>T<sub>E</sub>X



Java dispose d'un grand nombre de ressources sur internet.  
La version actuelle de Java est la n° 8.

### Documentation officielle

<https://docs.oracle.com/javase/8/>

## 1 Présentation

## 2 Les éléments du langage

- Types primitifs
- Variables
- Expressions
- Méthodes
- Structures de contrôle

## 3 Suppléments $\text{\LaTeX}$

## 1 Présentation

- Qu'est-ce que le Java ?
- Les outils
- Références

## 2 Les éléments du langage

- Types primitifs
- Variables
- Expressions
- Méthodes
- Structures de contrôle

## 3 Suppléments L<sup>A</sup>T<sub>E</sub>X

# Les éléments du langage

## Types primitifs

### Tableau des types primitifs :

Type	Taille	Valeur minimale	Valeur maximale	Exemple
<code>byte</code>	8 bit	-128	127	<code>byte b = 64;</code>
<code>char</code>	16 bit	0	$2^{16} - 1$	<code>char c = 'A'; char d = 64;</code>
<code>short</code>	16 bit	$-2^{15}$	$2^{15} - 1$	<code>short s = 65;</code>
<code>int</code>	32 bit	$-2^{31}$	$2^{31} - 1$	<code>int i = 1;</code>
<code>long</code>	64 bit	$-2^{63}$	$2^{63} - 1$	<code>long l = 65L;</code>
<code>float</code>	32 bit	$-2^{-149}$	$2 - 2^{-23} \times 2^{127}$	<code>float f = 65f;</code>
<code>double</code>	64 bit	$-2^{-1074}$	$2 - 2^{-52} \times 2^{1023}$	<code>double d = 65.55;</code>
<code>boolean</code>	1 bit			<code>boolean b = true; boolean c = false;</code>
<code>void</code>				

## 1 Présentation

- Qu'est-ce que le Java ?
- Les outils
- Références

## 2 Les éléments du langage

- Types primitifs
- Variables
- Expressions
- Méthodes
- Structures de contrôle

## 3 Suppléments L<sup>A</sup>T<sub>E</sub>X

# Les éléments du langage

## Variables

### Fact

*En Java, les variables sont typées, et peuvent être déclarées dans n'importe quel bloc du code.*

# Les éléments du langage

## Variables

### Fact

*En Java, les variables sont typées, et peuvent être déclarées dans n'importe quel bloc du code.*

### Exemple

```
if (...) { //BLOC 1
    int x;
    if (...) { //
        BLOC 2
        int y;
    }
}
```

# Les éléments du langage

## Variables

### Fact

*En Java, les variables sont typées, et peuvent être déclarées dans n'importe quel bloc du code.*

### Exemple

```
if (...) { //BLOC 1
    int x;
    if (...) { //
        BLOC 2
        int y;
    }
}
```

### Résultat

La variable x sera utilisable dans

La variable y sera utilisable



# Les éléments du langage

## Variables

### Fact

*En Java, les variables sont typées, et peuvent être déclarées dans n'importe quel bloc du code.*

### Exemple

```
if (...) { //BLOC 1
    int x;
    if (...) { //
        BLOC 2
        int y;
    }
}
```

### Résultat

La variable x sera utilisable dans  
les blocs 1 et 2.

La variable y sera utilisable

# Les éléments du langage

## Variables

### Fact

*En Java, les variables sont typées, et peuvent être déclarées dans n'importe quel bloc du code.*

### Exemple

```
if (...) { //BLOC 1
    int x;
    if (...) { //
        BLOC 2
        int y;
    }
}
```

### Résultat

La variable x sera utilisable dans les blocs 1 et 2.

La variable y sera utilisable uniquement dans le bloc 2.

### Opérateurs d'affectation

- =
- +=
- -=
- \*=
- /=
- %=

## 1 Présentation

- Qu'est-ce que le Java ?
- Les outils
- Références

## 2 Les éléments du langage

- Types primitifs
- Variables
- Expressions
- Méthodes
- Structures de contrôle

## 3 Suppléments $\text{\LaTeX}$

### Definition

Une *expression ternaire* est une notation « simplifiée » d'un *test logique*.

### Definition

Une *expression ternaire* est une notation « simplifiée » d'un *test logique*.

### Exemple (Test classique)

```
int i=100;
int y=20;
int maximum;
if(x > y) {
    maximum = x;
}
else {
    maximum = y;
}
```

# Les éléments du langage

## Expressions

### Definition

Une *expression ternaire* est une notation « simplifiée » d'un *test logique*.

#### Example (Test classique)

```
int i=100;
int y=20;
int maximum;
if(x > y) {
    maximum = x;
}
else {
    maximum = y;
}
```

#### Example (Test ternaire)

```
int i=100;
int y=20;
int maximum = (x > y)
               ? x : y;
```

### Attention !

Il est nécessaire de *caster* des affectations lorsque celles-ci ne sont pas implicites, sinon des erreurs de compilation sont détectées.



# Les éléments du langage

## Expressions

### Attention !

Il est nécessaire de *caster* des affectations lorsque celles-ci ne sont pas implicites, sinon des erreurs de compilation sont détectées.

### Exemple

```
int i=258;
long l=i; //OK
byte b=i; //ERROR: Explicit cast needed to
           convert int to byte
byte b=258; //ERROR: Explicit cast needed to
            convert int to byte
byte b=(byte)i; //OK mais b=2
```

### Remarque

Levé d'ambiguïté entre `float` et `double` :

```
float f=2564.5; //ERREUR de compilation
```

```
float f=2564.5f; //OK
```

## 1 Présentation

- Qu'est-ce que le Java ?
- Les outils
- Références

## 2 Les éléments du langage

- Types primitifs
- Variables
- Expressions
- **Méthodes**
- Structures de contrôle

## 3 Suppléments L<sup>A</sup>T<sub>E</sub>X

### Definition

Une *méthode* est une fonction appartenant à une classe.

### Definition

Une *méthode* est une fonction appartenant à une classe.

### Syntaxe

```
TypeRetour nomMethode(parametre1, parametre2...)  
{  
    ... corps ...  
}
```

### Remarque

### Remarque

- 1 Le type de retour est un type primitif, une classe ou `void`.

### Remarque

- ② La liste des paramètres peut être vide.



### Remarque

- ③ Si le type de retour n'est pas un `void`, la fonction doit se terminer par un `return`.

### Remarque

- ❶ Le type de retour est un type primitif, une classe ou `void`.
- ❷ La liste des paramètres peut être vide.
- ❸ Si le type de retour n'est pas un `void`, la fonction doit se terminer par un `return`.

Passage de paramètres :

Type simple : passés *par valeur* **uniquement**

Type objet ou tableaux : passés *par référence*

## 1 Présentation

- Qu'est-ce que le Java ?
- Les outils
- Références

## 2 Les éléments du langage

- Types primitifs
- Variables
- Expressions
- Méthodes
- Structures de contrôle

## 3 Suppléments L<sup>A</sup>T<sub>E</sub>X

# Les éléments du langage

## Structures de contrôle (IF)

### Fact

*Le code à l'intérieur d'un `if` s'exécute uniquement si la condition est vraie.*

# Les éléments du langage

## Structures de contrôle (IF)

### Fact

*Le code à l'intérieur d'un `if` s'exécute uniquement si la condition est vraie.*

### Attention !

Plusieurs notations sont possibles ! Soyez vigilants au nombre de lignes de code dans votre bloc d'instructions (si votre condition est vraie) pour bien choisir la notation.

# Les éléments du langage

## Structures de contrôle (IF)

### Fact

*Le code à l'intérieur d'un `if` s'exécute uniquement si la condition est vraie.*

### Attention !

Plusieurs notations sont possibles ! Soyez vigilants au nombre de lignes de code dans votre bloc d'instructions (si votre condition est vraie) pour bien choisir la notation.

### Syntaxe

- `if(condition){...} else {...}`

# Les éléments du langage

## Structures de contrôle (IF)

### Fact

*Le code à l'intérieur d'un `if` s'exécute uniquement si la condition est vraie.*

### Attention !

Plusieurs notations sont possibles ! Soyez vigilants au nombre de lignes de code dans votre bloc d'instructions (si votre condition est vraie) pour bien choisir la notation.

### Syntaxe

- `if(condition){...} else {...}`
- `if(condition)instruction;`

# Les éléments du langage

## Structures de contrôle (IF)

### Fact

*Le code à l'intérieur d'un `if` s'exécute uniquement si la condition est vraie.*

### Attention !

Plusieurs notations sont possibles ! Soyez vigilants au nombre de lignes de code dans votre bloc d'instructions (si votre condition est vraie) pour bien choisir la notation.

### Syntaxe

- `if(condition){...} else {...}`
- `if(condition)instruction;`
- `if(condition)instruction; else instruction;`



# Les éléments du langage

## Structures de contrôle (IF)

### Fact

*Le code à l'intérieur d'un `if` s'exécute uniquement si la condition est vraie.*

### Attention !

Plusieurs notations sont possibles ! Soyez vigilants au nombre de lignes de code dans votre bloc d'instructions (si votre condition est vraie) pour bien choisir la notation.

### Syntaxe

- `if(condition){...} else {...}`
- `if(condition)instruction;`
- `if(condition)instruction; else instruction;`
- `if(condition)instruction; else {...}`

# Les éléments du langage

## Structures de contrôle (IF)

### Fact

*Le code à l'intérieur d'un `if` s'exécute uniquement si la condition est vraie.*

### Attention !

Plusieurs notations sont possibles ! Soyez vigilants au nombre de lignes de code dans votre bloc d'instructions (si votre condition est vraie) pour bien choisir la notation.

### Syntaxe

- `if(condition){...} else {...}`
- `if(condition)instruction;`
- `if(condition)instruction; else instruction;`
- `if(condition)instruction; else {...}`
- `if(condition){...} else instruction;`

# Les éléments du langage

## Structures de contrôle (IF)

### Example

```
int i=0;
//NOTATION 1
if(i == 0) {
    i++; //i=i+1;
}
else {
    i+=2; //i=i+2;
}

//NOTATION 2
if(i==0) i++; else i+=2;
```

# Les éléments du langage

## Structures de contrôle (WHILE)

### Fact

*Le code à l'intérieur d'un `while` s'exécute tant que la condition est vraie.*

# Les éléments du langage

## Structures de contrôle (WHILE)

### Fact

*Le code à l'intérieur d'un `while` s'exécute tant que la condition est vraie.*

### Syntaxe

- `while(condition){...}`
- `while(condition)instruction;`

# Les éléments du langage

## Structures de contrôle (WHILE)

### Fact

*Le code à l'intérieur d'un `while` s'exécute tant que la condition est vraie.*

### Syntaxe

- `while(condition){...}`
- `while(condition)instruction;`

### Exemple

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

# Les éléments du langage

## Structures de contrôle (WHILE)

### Remarque

Le `while`, tel que définit jusqu'à présent, vérifie la condition avant d'exécuter (au moins une fois) les instructions.

Dans certains cas, il peut être utilisé d'exécuter les instructions au moins une fois, avant de vérifier si il faut les répéter.

# Les éléments du langage

## Structures de contrôle (WHILE)

### Remarque

Le `while`, tel que définit jusqu'à présent, vérifie la condition avant d'exécuter (au moins une fois) les instructions.

Dans certains cas, il peut être utilisé d'exécuter les instructions au moins une fois, avant de vérifier si il faut les répéter.

On utilisera l'instruction : `do`.



# Les éléments du langage

## Structures de contrôle (WHILE)

### Remarque

Le `while`, tel que définit jusqu'à présent, vérifie la condition avant d'exécuter (au moins une fois) les instructions.

Dans certains cas, il peut être utilisé d'exécuter les instructions au moins une fois, avant de vérifier si il faut les répéter.

On utilisera l'instruction : `do`.

### Syntaxe

```
do{ ... } while(condition)
```

# Les éléments du langage

## Structures de contrôle (DO ... WHILE)

### Example

```
int i=0;  
do{  
    System.out.println(i);  
    i++;  
}while(i >= 1 && i < 10)
```

# Les éléments du langage

## Structures de contrôle (DO ... WHILE)

### Exemple

```
int i=0;
do{
    System.out.println(i);
    i++;
}while(i >= 1 && i < 10)
```

### Remarque

Sans l'instruction `do`, aucune instruction ne se serait exécutée.

# Les éléments du langage

## Structures de contrôle (FOR)

### Fact

*Le code à l'intérieur d'une boucle `for` s'exécute un nombre défini de fois.*

# Les éléments du langage

## Structures de contrôle (FOR)

### Fact

*Le code à l'intérieur d'une boucle `for` s'exécute un nombre défini de fois.*

### Syntaxe

- `for(initialisation, condition, incrementation){ ... }`
- `for(initialisation, condition, incrementation)`  
    `instruction;`

Avec :

# Les éléments du langage

## Structures de contrôle (FOR)

### Fact

*Le code à l'intérieur d'une boucle `for` s'exécute un nombre défini de fois.*

### Syntaxe

- `for`(initialisation, condition, incrementation){ ... }
- `for`(initialisation, condition, incrementation)  
instruction;

Avec :

`initialisation` : initialisation de la / les variable(s) de boucle

# Les éléments du langage

## Structures de contrôle (FOR)

### Fact

*Le code à l'intérieur d'une boucle **for** s'exécute un nombre défini de fois.*

### Syntaxe

- `for(initialisation, condition, incrementation){ ... }`
- `for(initialisation, condition, incrementation)`  
    `instruction;`

Avec :

**initialisation** : initialisation de la / les variable(s) de boucle

**condition** : la boucle sera répétée tant que la condition sera vraie

# Les éléments du langage

## Structures de contrôle (FOR)

### Fact

*Le code à l'intérieur d'une boucle **for** s'exécute un nombre défini de fois.*

### Syntaxe

- **for**(initialisation, condition, incrementation){ ... }
- **for**(initialisation, condition, incrementation)  
instruction;

Avec :

**initialisation** : initialisation de la / les variable(s) de boucle

**condition** : la boucle sera répétée tant que la condition sera vraie

**incrementation** : incrémente la variable de boucle



# Les éléments du langage

## Structures de contrôle (FOR)

### Example

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

# Les éléments du langage

## Structures de contrôle (FOR)

### Exemple

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

### Attention !

Ce programme affiche les nombres de 0 à 9, et non jusqu'à 10 !

# Les éléments du langage

## Structures de contrôle (SWITCH)

### Definition

Un `switch` est un bloc contenant une succession de tests. On peut le comparer à une succession de `if`.

### Definition

Un `switch` est un bloc contenant une succession de tests. On peut le comparer à une succession de `if`.

- Structure très utile pour effectuer beaucoup de tests avec la même variable.
- Elle évite d'écrire un programme avec une succession de tests en `if` qui utilisent la même variable d'entrée.

# Les éléments du langage

## Structures de contrôle (SWITCH)

### Syntaxe

```
switch(variable) {  
    case valeur:  
        instructions;  
    case valeur:  
        instructions;  
    default: //si tout est faux avant  
        instructions;  
}
```

# Les éléments du langage

## Structures de contrôle (BREAK et CONTINUE)

### Remarque

Il peut être utile :

# Les éléments du langage

## Structures de contrôle (BREAK et CONTINUE)

### Remarque

Il peut être utile :

- 1 d'arrêter une boucle avant sa fin

# Les éléments du langage

## Structures de contrôle (BREAK et CONTINUE)

### Remarque

Il peut être utile :

- ① d'arrêter une boucle avant sa fin
- ② de passer automatiquement à l'itération suivante sans exécuter les instructions qui suivent dans le bloc



# Les éléments du langage

## Structures de contrôle (BREAK et CONTINUE)

### Remarque

Il peut être utile :

- ① d'arrêter une boucle avant sa fin
- ② de passer automatiquement à l'itération suivante sans exécuter les instructions qui suivent dans le bloc

Pour cela, on utilisera respectivement les instructions suivantes :

# Les éléments du langage

## Structures de contrôle (BREAK et CONTINUE)

### Remarque

Il peut être utile :

- ① d'arrêter une boucle avant sa fin
- ② de passer automatiquement à l'itération suivante sans exécuter les instructions qui suivent dans le bloc

Pour cela, on utilisera respectivement les instructions suivantes :

- ① `break;`

# Les éléments du langage

## Structures de contrôle (BREAK et CONTINUE)

### Remarque

Il peut être utile :

- ① d'arrêter une boucle avant sa fin
- ② de passer automatiquement à l'itération suivante sans exécuter les instructions qui suivent dans le bloc

Pour cela, on utilisera respectivement les instructions suivantes :

- ① `break;`
- ② `continue;`

# Plan du cours

- 1 Présentation
- 2 Les éléments du langage
- 3 Suppléments  $\text{\LaTeX}$

