

Langage de programmation

Java

Les fondamentaux

MARECAT Gaëtan

CGI

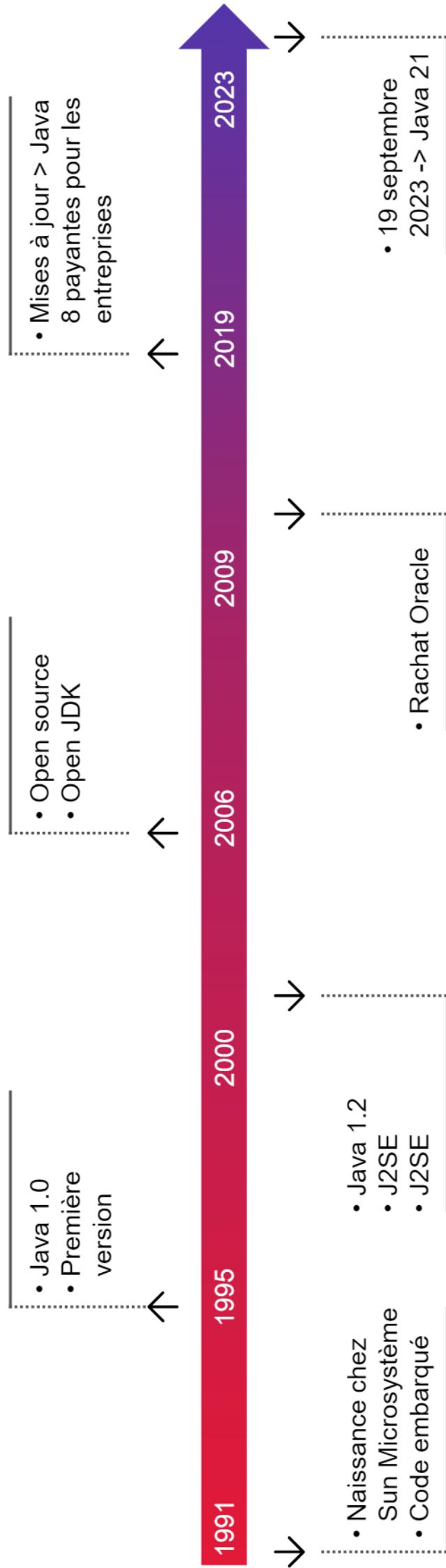
Sommaire



1	Présentation du langage
2	Types
3	Opérateurs
4	Structures
5	Premier programme
6	Let's practice

Présentation du langage

Historique



Cycle de vie

Source : <https://www.oracle.com/fr/java/technologies/java-se-support-roadmap.html>

Oracle Java SE Support Roadmap *†

Release	GA Date	Premier Support Until	Extended Support Until	Sustaining Support
8 (LTS)**	March 2014	March 2022	December 2030****	Indefinite
9 - 10 (non-LTS)	September 2017 - March 2018	March 2018 - September 2018	Not Available	Indefinite
11 (LTS)	September 2018	September 2023	January 2032	Indefinite
12 - 16 (non-LTS)	March 2019 - March 2021	September 2019 - September 2021	Not Available	Indefinite
17 (LTS)	September 2021	September 2026****	September 2029****	Indefinite
18 (non-LTS)	March 2022	September 2022	Not Available	Indefinite
19 (non-LTS)	September 2022	March 2023	Not Available	Indefinite
20 (non-LTS)	March 2023	September 2023	Not Available	Indefinite
21 (LTS)***	September 2023	September 2028	September 2031	Indefinite
22 (non-LTS)***	March 2024	September 2024	Not Available	Indefinite
23 (non-LTS)***	September 2024	March 2025	Not Available	Indefinite
24 (non-LTS)***	March 2025	September 2025	Not Available	Indefinite
25 (LTS)***	September 2025	September 2025****	Not Available	Indefinite

Les bases

Langage Objet

- Object is everywhere
- Types primitifs : bool, char, int, byte, short, long, float, double
- Typage fort

Les opérateurs et structures

- Permet de manipuler des variables
- Opérateurs de calcul, d'assignation, d'incrémantation, de comparaison, logiques
- Structures itératives et conditionnelles

Concepts Objet

- Classes
- Héritage
- Encapsulation

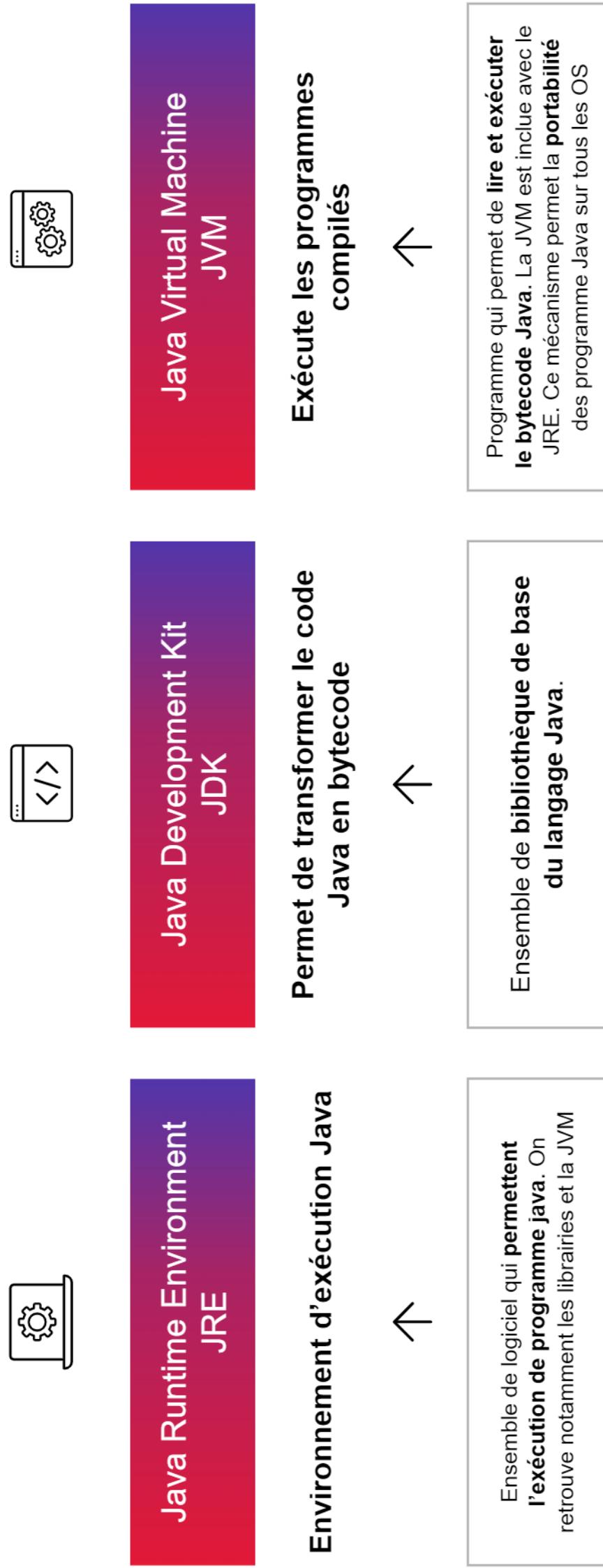
Programmation fonctionnelle

- Stream / Lambda expression
- Clean code
- Modulaire

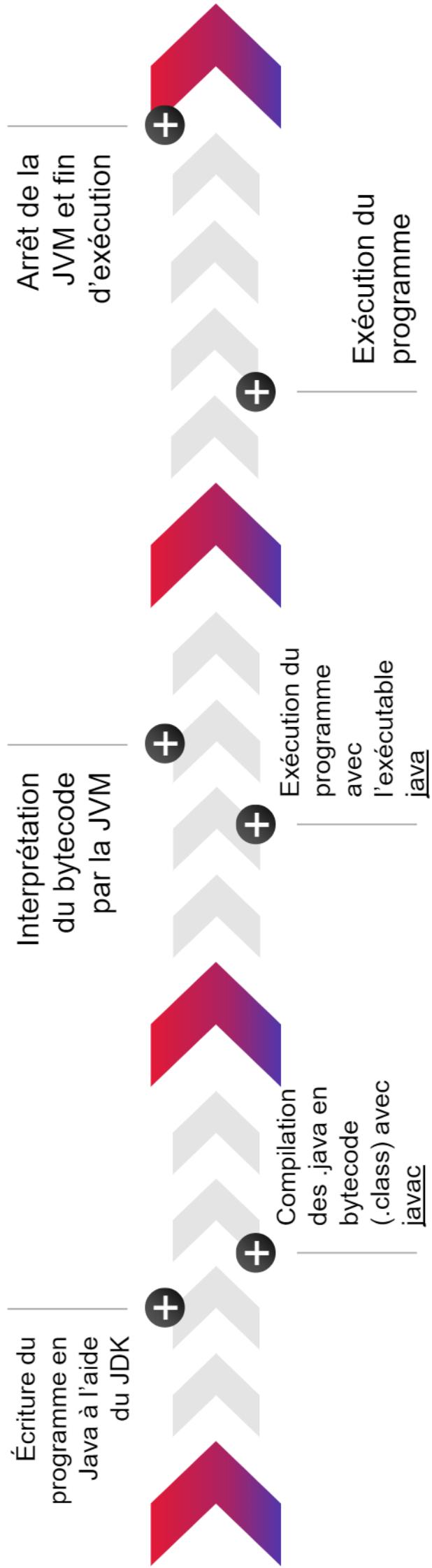
Et bien plus

- Gestion des exceptions
- Java Doc
- Écriture fichier (IO)

Définitions clés



Comment fonctionne un programme Java ?



Types

Type primitif

Notation	Type	Taille	Signé	Intervalle
bool	Booléen	1 bit	non	false ou true
char	Caractère (s'écrit entre quotes simples)	2 octets (16 bits)	non	"\u0000" à "\uFFFF"
int	Entier (integer)	4 octets (32 bits)	oui	-2147483648 à 2147483647
byte	Entier sur 1 octet	1 octet (8 bits)	oui	-128 à 127

Type primitif

Notation	Type	Taille	Signé	Intervale
short	Entier sur 2 octets	2 octets (16 bits)	oui	-32768 à 32767
long	Entier sur 8 octets (<u>noté avec un L à la fin</u>)	8 octets (64 bits)	oui	-9223372036854775808L à 9223372036854775807L
float	Nombre à virgule flottante (<u>noté avec un F à la fin</u>)	4 octets (32 bits)	oui	1,402399846E-45 à 3,40282347E38
double	Nombre à virgule flottante sur 8 octets	8 octets (64 bits)	oui	4,9406564584124654E-324 à 1,797693134862316E308

Déclaration

```
int n; // Déclaration, n n'est pas initialisé  
n = 15; // Initialisation de n à 15  
  
int p = 15; // Il est possible d'initialiser lors de la déclaration
```

Une variable qui n'est pas déclarer ne peut pas être utilisée sinon cela entraîne une erreur de compilation.

Java détecte toutes les situations de variables non initialisées, comme le montre l'exemple suivant :

```
int n;  
if (...) {  
    n = 30;  
}  
  
int p = 2*n; // erreur de compilation : n n'est pas initialisé dans tous les cas
```

```
final int n = 0; // constante  
final int p; // constante  
p = 0; // valeur définitive la constante
```

Les constantes sont déclaré avec le mot clé final.
Elles ne peuvent être initialisées qu'une seule fois.

Tableaux

```
// Déclaration (la dimension ne doit pas être précisée)
int t[];
int [] otherT; // écriture équivalente à la ligne du dessus
int [] t1, t2; // avantage de la notation
int t3[], t4[]; // équivalent à la ligne du dessus
// Il est possible de mélanger tableau et variable de même type
int a, t5[], b;

// Initialisation
t = new int[5]; // créer un tableau vide de dimension 5
int t6[] = {1, 2, 3, 4, 5}; // notation possible uniquement à la déclaration
// la ligne du dessus est équivalente à ces instructions
int t7[] = new int [5];
t7[0] = 1; t7[1] = 2; t7[2] = 3; t7[3] = 4; t7[4] = 5;
```

Strings

Documentation officielle Java :

<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html>

```
// Déclaration d'un string
String s = "Hello world"; // équivalent à : String s = new String("Hello world");

// Concaténation
String s1 = "Hello";
String s2 = "World";
String s3 = s1 + " " + s2;
System.out.println("s3 = " + s3); // s3 = Hello World
```

Opérateurs

Opérateurs de calcul

Opérateur	Dénomination	Effet	Exemple	Résultat (avec x valant 6)
+	addition	Ajoute deux valeurs	$x+3$	9
-	soustraction	Soustrait deux valeurs	$x-3$	6
*	multiplication	Multiplie deux valeurs	$x*3$	18
/	division	Divise deux valeurs	$x/3$	2
=	affectation	Affecte une valeur à une variable	$x=3$	Affecte 3 dans la variable x

Opérateurs d'affectation

Opérateur	Effet	Exemple	Résultat (avec x valant 6)
$\%$	Modulo => Retourne le reste d'une division entière	$x \% 4$	2
$+=$	Additionne la variable à gauche et la valeur à droite puis stock le résultat dans la variable	$x += 3$	Affecte 9 dans la variable x
$-=$	Soustrait la variable à gauche et la valeur à droite puis stock le résultat dans la variable	$x -= 3$	Affecte 3 dans la variable x
$*=$	Multiplie la variable à gauche et la valeur à droite puis stock le résultat dans la variable	$x *= 3$	Affecte 18 dans la variable x
$/=$	Divise la variable à gauche et la valeur à droite puis stock le résultat dans la variable	$x /= 3$	Affecte 2 dans la variable x

Opérateurs d'incrémantation

Opérateur	Dénomination	Effet	Exemple	Résultat (avec $x = 6$)
<code>++</code>	Incrémentation	Augmente d'une unité la variable	<code>X++</code>	7
<code>--</code>	Décrémentation	Diminue d'une unité la variable	<code>X--</code>	5

Attention, dans le cadre de certaines opérations il est possible de constater **des conversions automatiques** de type. Par exemple sur une opérations qui mélange des int et des float, le int sera converti automatiquement en float avant l'opération et le résultat sera de type float. Les conversion ne se font que dans un seul sens :

`int -> long -> float -> double`

Opérateurs de comparaison

Opérateur	Dénomination	Effet	Exemple	Résultat (avec x valant 6)
$==$	Égalité	Compare deux valeur et vérifie leur égalité	$x == 6$	true
$<$	Stricte infériorité	Vérifie si une valeur est strictement inférieur à une autre	$x < 6$	false
\leq	Infériorité	Vérifie si une valeur est inférieur ou égale à une autre	$x \leq 6$	true
$>$	Strict supériorité	Vérifie si une valeur est strictement supérieur à une autre	$x > 6$	false
\geq	Supériorité	Vérifie si une valeur est supérieur ou égale à une autre	$x \geq 6$	true
$!=$	Différence	Vérifie si une valeur est différente d'une autre	$x != 6$	false

Opérateurs d'incrém̄entation

Opérateur	Dénomination	Effet	Exemple	Résultat
<code> </code>	OU logique	Vérifie qu'une des deux conditions est vraie	1. <code>true false</code> 2. <code>true true</code> 3. <code>false false</code>	1. <code>true</code> 2. <code>true</code> 3. <code>false</code>
<code>&&</code>	ET logique	Vérifie que les deux conditions sont vraies	1. <code>true && false</code> 2. <code>true && true</code> 3. <code>false && false</code>	1. <code>false</code> 2. <code>true</code> 3. <code>false</code>
<code>!</code>	NON logique	Inverse l'état d'une variable booléenne	1. <code>!true</code> 2. <code>!false</code>	1. <code>false</code> 2. <code>true</code>

Le NON logique est prioritaire sur le ET logique qui est prioritaire sur le OU logique. Il est possible de contrôler les priorités à l'aide des parenthèses.

Structures

Structures conditionnelles

```
// Condition
int a = 1, b = 2; // ou int a = 1; int b = 2;
if (a < b) {
    // do something
} else if (a == b) {
    // do something else
} else {
    // do something else
}
```

```
if (a < b) System.out.println("a < b");
else if (a == b) System.out.println("a == b");
else System.out.println("Else");
```

Possible si le if ou le else ne comporte qu'une seule instruction.

Switch

```
// Switch
char c;
switch (a) {
    case 1:
        c = 'a';
        break;
    case 2:
        c = 'b';
        break;
    default:
        c = 'd';
        break;
}
```

```
Java >= 13
// ou alors
c = switch (a) {
    case 1:
        yield 'a';
    case 2:
        yield 'b';
    default:
        yield 'd';
}
```

```
Java >= 12
// ou alors
c = switch (a) {
    case 1 -> 'a';
    case 2 -> 'b';
    default -> 'd';
};
```

Structures itératives

```
// While
int t[] = {1, 2, 3, 4, 5};
int i = 0;
int cpt = 0;
while (i < t.length) {
    cpt += t[i];
    i++;
}
System.out.println("cpt = " + cpt); // cpt = 15
```

```
// For
int t[] = {1, 2, 3, 4, 5};
int cpt = 0;
for (int i = 0; i < t.length; i++) {
    cpt += t[i];
}
System.out.println("cpt = " + cpt); // cpt = 15
```

```
// Do... While
int t[] = {1, 2, 3, 4, 5};
int i = 0;
int cpt = 0;
do {
    cpt += t[i];
    i++;
} while (i < t.length);
System.out.println("cpt = " + cpt); // cpt = 15
```

```
// Foreach
int t[] = {1, 2, 3, 4, 5};
int cpt = 0;
for (int value: t) {
    cpt += value;
}
System.out.println("cpt = " + cpt); // cpt = 15
```

Dans ce cas la valeur de *value* n'est pas modifiable.

Structures itératives

Il est possible de sortir prématurément d'une boucle avec le mot clé *break*.

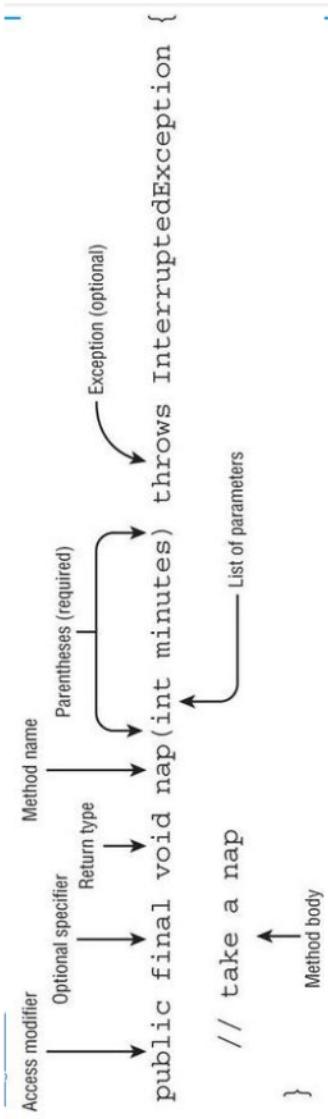
Lorsque le mot-clé *break* est rencontré à l'intérieur d'une boucle, l'exécution de la boucle est immédiatement interrompue, et le contrôle est transféré à l'instruction suivant immédiatement après la boucle.

```
for (int i = 1; i <= 10; i++) {  
    if (i == 5) {  
        break; // Sort de la boucle lorsque i atteint 5  
    }  
    System.out.println(i);  
}
```

L'instruction *continue* est utilisée pour sauter l'itération actuelle d'une boucle et passer à l'itération suivante. Lorsque le mot-clé *continue* est rencontré à l'intérieur d'une boucle, le code restant dans le corps de la boucle pour l'itération actuelle est ignoré, et la boucle passe à l'itération suivante.

```
for (int i = 1; i <= 10; i++) {  
    if (i % 2 == 0) {  
        continue; // Passe à l'itération suivante si i est pair  
    }  
    System.out.println(i);  
}
```

Conception de méthodes



Element	Value in nap() example	Required?
Access modifier	public	No
Optional specifier	final	No
Return type	void	Yes
Method name	nap	Yes
Parameter list	(int minutes)	Yes, but can be empty parentheses
Method signature	nap(int minutes)	Yes
Exception list	throws InterruptedException	No
Method body	{ // take a nap }	Yes, except for abstract methods

Premier programme

Comment écrire un programme ?

Pour rappel, tout est Objet en Java. Il nous faut donc une classe principale dans laquelle mettre le code (le fichier doit porter le même nom que la classe). Pour écrire notre première classe, dans un fichier Application.java :

```
public class Application {  
    ...  
}
```

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

```
public class Application {  
    public static void main(String[] args) {  
        ...  
    }  
}
```

Dans cette classe nous devons avoir une méthode avec une signature particulière pour qu'elle soit détecté par le compilateur et la JVM. Cette méthode va contenir le code principale de l'application :

Comment exécuter un programme ?

Il est nécessaire de compiler la classe avec un programme du JDK : **javac**

Le JDK (contient également le JRE) se télécharge ici : <https://jdk.java.net/21/>.

Ce programme est trop poussé pour en voir tous les aspects, cependant voici comment faire pour générer le bytecode Java avec une compilation sur un simple fichier :

```
/>javac Application.java
```

Cette commande va générer le bytecode dans un fichier Application.class dans le même répertoire.

Pour exécuter le programme, utiliser l'exécutable **java** fourni avec le JRE avec le nom de la classe qui comporte la méthode main.

Il ne faut pas mentionner de .class lors de l'exécution du programme java.

```
/>java Application
```

Outils pratiques

Lecture saisie utilisateur

```
import java.util.Scanner;  
...  
  
Scanner scanner = new Scanner(System.in);  
System.out.print("Entrez une chaîne de caractères : ");  
String chaine = scanner.nextLine();  
System.out.print("Entrez un entier : ");  
int n = scanner.nextInt();
```

Affichage en console

```
System.out.println("phrase avec retour à la ligne");  
System.out.print("phrase sans retour à la fin");  
...
```

Créer une fonction

```
/ public static + type de retour + nom de la méthode + (paramètres)  
public static long malMéthode(int n) {
```

Pour les procédure le type de retour peu être *void*, dans ce cas aucun *return* n'est attendu dans la méthode.

Documentation du JDK :

<https://docs.oracle.com/en/java/javase/21/docs/api/index.html>

Let's practice

