Introduction

Helloworld

Pour démarrer l'apprentissage d'un programme, rien de tel que l'écriture du fameux HelloWorld.

Fichier **Helloworld.java**

```java
public class Helloworld {

    /**
     * point d'entrée des programmes java. Il est nécessa&ire de respecter exactement la signature
     * de cette méthode afin que le programme puisse s'exécuter.
     *
     * @param args tableau d'arguments passés lors du lancement du programme
     */
    public static void main(String[] args) {
        System.out.println("Hello world !");
    }
}
```

Pour compiler le programme, il est nécessaire de se positionner dans le repertoire **src**

```
cd src/main/java
javac Helloworld.java
java Helloworld
```

> En java, toute ligne d'instructions est terminée par un **point virgule**

> Depuis Java 23, il est possible de créer une méthode *main* sans être dans une classe JEP 477

Déclaration d'une variable

Pour utiliser une variable, il est nécéssaire de la déclarer et de lui assigner une valeur. Lors de la déclaration, on definit le type de la variable (primitif, classe, record, enum) suivi du nom de la variable.

Fichier **Variables1.java**

```java
public class Variables1 {

    public static void main(String[] args) {
        /**
         * Declaration  et assignement d'une variable
```

```
         */
        int magicNumber = 62;

        /**
         * Une chaine de caractères peut être concaténée à tout type
d'élements
         */
        System.out.println("Le nombre magique est " + magicNumber);

        /*
         * L'utilisation de la méthode printf permet d'écrire des chaines
de caractères
         * en formattant la chaine de chaine de caratères.
         */
        System.out.printf("Le nombre magique est %d", magicNumber);

        // des variable
        short year = 2024;
        char a = 'a';

        double montant = 3_000;

        long longNumber = 5_678_232;
        long longNumber2 = 5_678_232l;

        LocalDateTime now = LocalDateTime.now();

        boolean yes = true;
    }
}
```

```
mvn --quiet compile exec:java -Dexec.mainClass=Variables1
Le nombre magique est 62
Le nombre magique est 62
```

Il est tout à fait possible de déclarer puis, par la suite, d'assigner une valeur à cette variable.

Fichier **Variables2.java**

```
public class Variables2 {

    public static void main(String[] args) {
        /**
         * Declaration puis assignement
         */
        String name;
        name = "Java";
        System.out.printf("Un langage toujours au top : %s", name);
```

```
    }
}
```

```
mvn --quiet compile exec:java -Dexec.mainClass=Variables2
Un langage toujours au top : Java
```

> tant qu'une variable, déclarée dans le corps d'une méthode, n'est pas assignée à une valeur, elle ne peut pas être utilisée.

Fichier **Variables3.java**

```
public class Variables3 {

    public static void main(String[] args) {
        /**
         * Declaration sans assignement
         */
        int ageDuCapitaine;
        System.out.printf("L'age du capitaine est %d", ageDuCapitaine);
    }
}
```

```
javac Variables3.java
Variables3.java:8: error: variable ageDuCapitaine might not have been
initialized
        System.out.printf("L'age du capitaine est %d", ageDuCapitaine);
```

Les primitives

Java n'est pas un lanqage purement objet. 8 primitives sont mises à disposition au développeur.

Catégorie	Type	Taille	Signé	Min	Max
	byte	2^8	signed	-128	127
	char	2^{16}	unsigned	0	$2^{16}-1$
Entier	short	2^{16}	signed	-2^{15}	$2^{15}-1$
	int	2^{32}	signed	2^2	$2^{31}-1$
	long	2^{64}	signed	-2^{63}	$2^{63}-1$
Decimaux	float	2^{32}	signed	-2^{-149}	$(2-2^{-23}) * 2^{127}$
	double	2^{64}	signed	-2^{-1074}	$(2-2^{-52}) * 2^{21023}$

Catégorie	Type	Taille	Signé	Min	Max
Autre	boolean				

Les opérateurs d'assignement permettent de modifier la valeur d'une primitive.

Operateur	Nom	Exemple
=	Assigne une valeur	int i = 18;
+=	Ajoute la valeur de droite	i = 1;i += 56 // i vaut 57
-=	Soustrait la valeur de droite	i = 10;i -= 2 // i vaut 8
/=	Division par la valeur de droite	i = 10;i /= 2 // i vaut 5
*=	Multiplication par la valeur de droite	i = 10;i *= 2 // i vaut 20
%=	Application du modulo	i = 10;i %= 2 // i vaut 0

Les opérateurs mathématiques sont utilisés sur les primivites pour réaliser des opérations.

Operateur	Nom	Exemple
+	Addition	1 + 5
-	Soustraction	1 – 3
*	Multiplication	45 * 90
/	Division	1 / 10
%	Modulo (Reste de la division Euclidienne)	65 % 3
++	Incrémente de 1	i++
--	Décrémente de 1	i--

Il est possible de convertir un paramétre d'un certain type (de primitif) en un autre.

> Attention, il peut y avoir des pertes de precision lorsqu'on utilise un type dont l'intervalle de definition est plus petit.

```
public class Variable5 {

    public static void main(String[] args) {
        //Explicit
        int value = 56;
        short castValue = (short) value;
        System.out.println(castValue);

        //Implicit
        short valueS = 56;
        int valueI = valueS;
        System.out.println(valueI);
```

```
        //Perte de precision
        int number = (int) 34.78; // numb
        System.out.println(number);
    }
}
```

```
mvn --quiet compile exec:java -Dexec.mainClass=Variable5
56
56
34
```

Combinaison de type

Calcul	Resultat	Régle
2/3	0	(int, int) -> (int)
2/3.0	0,666666666	(int, double) -> (double)
2.0/3.0	0,666666666	(double, double) -> (double)
2.0 + "3.0"	2.03.0	(double, string) -> (string)

Régle :

String > double > int > char > boolean

Et si on parlait de *null*

Une variable peut :

- ne pas être définie.
- être associée à une valeur numerique, booleen, chaine de caractères, enumération, objet.
- être associée à **null**

null représente l'absence de valeur. il n'est ni un objet, ni un type. Il répresente une valeur spéciale. Une variable associée à **null** mal utilisée peut entrainer l'exception **NullPointerException**.

null ne peut pas être assigné à une primitive

Le mot clé *var*

Depuis Java5, il est possible d'éviter de déclarer le type d'une variable grâce au mot clé **var**.

Dans ce cas, il est nécessaire d'assigner une valeur afin que le compilateur infère le type de la variable. Une fois le type inféré, il n'est plus possible d'assigner à la variable à une valeur d'un autre type.

Fichier **Variable4.java**

```java
public class Variables4 {

    public static void main(String[] args) {
        var birthday = 1995;
        var author = "James Gosling";

        System.out.printf("birthday :  %s, author : %s\n",
((Object)birthday).getClass().getName(), author.getClass().getName());

        var message = "Java est apparu en " + birthday + " et un des
auteurs est "+ author;
        System.out.println(message);
    }
}
```

```
mvn --quiet compile exec:java -Dexec.mainClass=Variables4
birthday :  java.lang.Integer, author : java.lang.String
Java est apparu en 1995 et un des auteurs est James Gosling
```

Loop et conditions

Avant : les opérateurs logiques

Operateur	Nom	Exemple
&&	Et	x == y && x < z
||	Ou	x == y || x < z
!	Not	!(x == y && x < z)

if / else if / else

Le mot clé **if** permet d'éxécuter un bloc d'instructions si la condition est vraie.

Pour comparer des primitives, on peut utiliser une des comparateurs suivants :

Comparateur	Description
==	Egalité
!=	Différent
<	Plus petit
<=	Plus petit
>	Plus grand
>=	Plus grand ou égal

> Pour definir un bloc, on utilise les accolades { }

Fichier **Condition1.java**

```
public class Condition1 {

    public static void main(String[] args) {
        int age1 = 25;
        int age2 = 43;

        if(age1 < age2){
            System.out.printf("La variable age1 (%d) est à une valeur
inférieure à la variable age2  (%d) ", age1, age2);
        }
    }
}
```

```
mvn --quiet compile exec:java -Dexec.mainClass=Condition1
La variable age1 (25) est à une valeur inférieure à la variable age2
```

Le mot clé **else** permet d'exécuter le second bloc d'instructions dans le cas où la condition du **if** n'est pas réalisée.

Fichier **Condition2.java**

```
public class Condition2 {

    public static void main(String[] args) {
        int nombre = 19;
        if(nombre % 2 == 0){
            System.out.printf("Le nombre (%d) est un nombre pair ",
nombre);
        }else{
            System.out.printf("Le nombre (%d) est un nombre impair ",
nombre);
        }
    }
}
```

```
mvn --quiet compile exec:java -Dexec.mainClass=Condition2
Le nombre (19) est un nombre impair
```

En utilisant **else if**, il est possible d'enchainer des tests conditionnels et associant un bloc d'instructions à chaque test. Dés qu'un test est vrai, le bloc d'instructions est executé. Les conditions suivantes seront

ignorées.

Fichier **Condition3.java**

```java
public class Condition3 {

    public static void main(String[] args) {
        int angle = 85;
        if(angle == 90) {
            System.out.println("Angle droit");
        }else if(angle == 0) {
            System.out.println("Angle plat");
        }else if(angle > 90) {
            System.out.println("Angle obtus");
        }else {
            System.out.println("Angle aigu");
        }
    }
}
```

```
mvn --quiet compile exec:java -Dexec.mainClass=Condition3
Angle aigu
```

> dans le cas d'une succession de **else if**, le mot clé seul **else** sera toujours à la fin de cette succession.

switch ... case

Dans sa forme de base, le switch case ressemble au **if**. Il permet de tester plusieurs alternatives.

```java
switch(variable) {
    case constante1:
    // bloc d'instructions
    case constante2:
    // bloc d'instructions
    //...
    default:
    // bloc d'instructions
}
```

> Attention : dés qu'un case est vraie, le bloc d'instructions est évalué. Si ce bloc ne se termine paa par le mot clé **break**, les cases situés en dessous seront évalués jusqu'au prochain **break** ou jusqu'à la fin du swicth.

> on ne peut pas avoir 2 cases avec la même constante

Depuis Java 12, le switch évolue afin d'intégrer le pattern matching.

for

Le mot clé **for** permet d'executer plusieurs fois un même bloc d'instructions.

Fichier **Looping1.java**

```
public class Looping1 {

    public static void main(String[] args) {
        for(int index=0; index<5; index++){
            System.out.printf("Hello %d ! \n", index);
        }
    }
}
```

Le bloc de code contenant l'instruction `System.out.printf("Hello %d ! \n", index);` est executé 5 fois (index pouvant varié de 0 à 4)

```
mvn --quiet compile exec:java -Dexec.mainClass=Looping1
Hello 0 !
Hello 1 !
Hello 2 !
Hello 3 !
Hello 4 !
```

Si on decompose les parties de **for**

statement	description
int index=0	bloc de déclaration. La variable **index** est déclarée et sa valeur est initialisée à **0**
index<5	bloc d'évaluation. A chaque itération, la condition est évaluée et tant qu'elle est vraie, le bloc d'instructions sera évalué
index++	bloc d'instruction. A chaque itération, les instructions dans cette partie sont évaluées. C'est ici que l'on trouve le plus souvent une incrementation. Ici, à chaque passage la valeur de la variable **index** s'incrémente de 1

> on utilise la boucle **for** lorsque l'on peut déterminer à l'avance le nombre d'itérations.

L'operateur ternaire ?

Il est possible d'évaluer sur une meme ligne et d'assigner une valeur spécifique en fonction de la réalisation d'une condition. On utilise l'operateur ternaire **?**

Voici un exemple long avec le mot clé **if**

Fichier **Looping2.java**

```
public class Looping2 {

    public static void main(String[] args) {
        for(int nombre=0; nombre<5; nombre++){
            if(nombre % 2 == 0){
                System.out.printf("Le nombre %d est un nombre pair.\n",
nombre);
            }else{
                System.out.printf("Le nombre %d est un nombre impair.\n",
nombre);
            }
        }
    }
}
```

```
mvn --quiet compile exec:java -Dexec.mainClass=Looping2
Le nombre 0 est un nombre pair.
Le nombre 1 est un nombre impair.
Le nombre 2 est un nombre pair.
Le nombre 3 est un nombre impair.
Le nombre 4 est un nombre pair.
```

il est possible d'utiliser l'operateur ternaire à la place du mot clé **if**

Fichier **Looping3.java**

```
public class Looping3 {

    public static void main(String[] args) {
        for(int nombre=0; nombre<5; nombre++){
            var type = nombre % 2 == 0 ? "pair": "impair";
            System.out.printf("Le nombre %d est un nombre %s.\n", nombre,
type);
        }
    }
}
```

```
mvn --quiet compile exec:java -Dexec.mainClass=Looping3
Le nombre 0 est un nombre pair.
Le nombre 1 est un nombre impair.
```

```
Le nombre 2 est un nombre pair.
Le nombre 3 est un nombre impair.
Le nombre 4 est un nombre pair.
```

While

Le mot clé **while** permet d'itérer tant que la condition associée au **while** est vérifiée. while est souvent utilisé lorsque le nombre d'itérations ne peut être déterminé.

Fichier **Looping4.java**

```java
public class Looping4 {

    public static void main(String[] args) {
        var total = 0;
        var inc = new SecureRandom().nextInt(10);
        while(total<20){
            total += inc;
        }
        System.out.printf("Inc : %d, Total : %d", inc, total);
    }
}
```

```
mvn --quiet compile exec:java -Dexec.mainClass=Looping4
Inc : 6, Total : 24
```

> il éxiste également l'instruction **do ...while()**. Le bloc d'instructions sera toujours évalué **au moins une fois**

Les méthodes

Une méthode est une fonction réalisation un traitement spécifique.

Une méthode :

- porte un nom
- prend 0 à n arguments
- peut retourner une valeur.
- posséde des accolades. Les accolades vont introuduire le bloc d'instructions qui sera évalué lors de l'appel de la méthode.
- est déclarée à l'intérieur d'une classe.

Il existe 2 types de méthodes :

- **les méthodes de classe** Ces méthodes, en fonction de leur portée, peuvent être appelé n'importe où dans le code en utilisant directement la classe.

- **les méthodes d'instance**. Ces méthodes manipules les propriétés d'un objet et altére le comportement de ce dernier. Ces méthodes ne peuvent pas être appelée à partir du nom de la classe.

> Depuis **Java 23**, il est possible de créer la méthode **main** en dehors d'une classe. JEP - 477

Squelette d'une méthode :

```
<scope> (static) <return type> methodName(<parameters>…) {
 ....
}
```

Les méthodes de classe

Le mot clé **static** permet de definir une mathode de classe.

Une méthode peut ne rien retourner. Dans ce cas, on utilisera le mot clé **void**

Fichier **Methode1.java**

```java
public class Methode1 {

    public static void displayHello(String name){
        System.out.println("Hello " + name  + " !");;
    }

    public static void main(String[] args) {
        displayHello("Bob");
    }
}
```

```
mvn --quiet compile exec:java -Dexec.mainClass=Methode1
Hello Bob !
```

Elle peut également retourner une donnée. Danc il faut indiquer dans la signature le type (primitif ou classe) que va retourner la méthode

Fichier **Methode2.java**

```java
public class Methode2 {

    public static int sum(int a, int b){
        return a + b;
    }
```

```
    public static void main(String[] args) {
        System.out.printf("Somme 10 + 7 : %d", sum(10, 17));
    }
}
```

```
mvn --quiet compile exec:java -Dexec.mainClass=Methode2
Somme 10 + 7 : 27
```

Depuis Java5, il est possible de déclarer un nombre infini de paramétres d'un même type. On parle alors de **varargs** Un seul vararg est autorisé par méthode et il doit être le dernier paramétre.

Pour introduire un paramétre **vararg**, il suffit d'ajouter **...** apr&s le type du paramatré suivi du nom de celui ci. Le paramétre sera considéré comme un tableau.

Fichier **Methode3.java**

```
public class Methode3 {

    public static int substract(int... numbers){
        int ret = 0;
        for(var number: numbers){
            ret -= number;
        }
        return ret;
    }

    public static void main(String[] args) {
        System.out.printf("Sousoutraction : %d\n", substract(1,3,4,5));
        System.out.printf("Sousoutraction : %d", substract(9, 8));
    }
}
```

```
mvn --quiet compile exec:java -Dexec.mainClass=Methode3
Sousoutraction : -13
Sousoutraction : -17
```

> Les exemples ci-dessus utilisent la notion de vararg. En effet, la méthode **System.out.printf** est uné méthode avec un vararg. Le premier paramétre est la chaine de caractéres, suivi de plusieurs paramétres qui seront utilisés pour remplacer les élements introduits par **%**.

La classe System

La classe *System* fournit des méthodes pour interagier avec les entrées/sorties standards, récuperer les variables/propriétés d'environnements et quelques méthodes utilitaires. La classe ne peut être instanciée.

Sortie console : System.out

La propriété *System.out* permet de réaliser des sorties console. La propriété est de type **PrintStream** possédant des nombreux méthodes pour écrire dans le terminal.

Méthode	Description
println()	réalise une sortie console du paramétre en ajoutant un retour à la ligne
print()	réalise une sortie console du paramétre
printf(str, ...args)	réalise une sortie en console en utlisant une chaine formattée. Les arguments seront intégrés à la chaâine

> la propriété *System.err* ressemble à la propriété *System.out* mais réalise la sortie d'erreurs

Entrée standard : System.in

La propriété *System.in* permet de lire les éléments saisies en console. La propriété est de type **InpuStream**.

La classe **InputStream** founit des méthodes de base pour lire des octets. Pour faciliter la récupération des élements saisis, nous utilisons la classe *java.util.Scanner*

```
import java.util.Scanner;

public class Main1 {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Saisissez un mot : ");
        var mot = scanner.next();
        System.out.println("Résultat : " + mot);

        System.out.print("Saisissez un nombre : ");
        var nb = scanner.nextInt();
        System.out.println("Résultat  : " + nb);

        //pour forcer le scanner à lire jusqu'au retour charriot
        scanner.useDelimiter(System.getProperty("line.separator"));
        System.out.print("Saisissez une phrase : ");
        var phrase = scanner.next();
        System.out.println("Résultat : " + phrase);
    }
}
```

```
mvn --quiet compile exec:java -Dexec.mainClass=Main1
Saisissez un mot : java
Résultat : java
Saisissez un nombre : 8
Résultat  : 8
Saisissez une phrase : Java dans le top 3 des langages !
Résultat : Java dans le top 3 des langages !
```