

Programmation Orienté Objet JAVA

Cheikh MBENGUE

Email: cheikh23.mbengue@ucad.edu.sn

Site: <https://daaratech.sn>

Licence 1 TDSI

UCAD/FST/TDSI

Année académique 2023-2024

Introduction

- Plan
 - Introduction
 - Caractéristiques du langage Java
 - Concepts de base
 - Structure d'un programme Java

Introduction générale

- La **Programmation Orientée Objet (POO)** consiste à modéliser informatiquement un ensemble d'éléments d'une partie du monde réel (que l'on appelle domaine) en un ensemble d'entités informatiques. Ces entités informatiques sont appelées objets. Il s'agit de données informatiques regroupant les principales caractéristiques des éléments du monde réel (taille, couleur, ...).

Introduction générale

- Java a été développé à partir de décembre 1990 par une équipe de Sun Microsystems dirigée par James Gosling.
- Les fondateurs de Java ont réalisé un langage indépendant de toute architecture de telle sorte que Java devienne idéal pour programmer pour des réseaux hétérogènes, notamment Internet.

Caractéristiques du langage Java

- **Portabilité ;**
 - Indépendance par rapport aux plateformes(os+ architectures).
- **Sécurité et robustesse;**
 - Le compilateur interdit toute manipulation en mémoire
 - En interdisant les manipulations directes et dangereuses de la mémoire, Java offre une couche de sécurité importante. Cela permet de réduire les erreurs liées à la mémoire et de rendre les applications Java plus robustes et sécurisées.
- **Gratuité;**
 - Les outils de développement Java sont fournis gratuitement.
- **Richesse;**
 - Disponibilité d'une vaste collection de bibliothèques de classes.

Compilation et exécution d'un programme en JAVA

- Exemple

- `javac MaClasse.java`: **Compilation du fichier source.**

Cette commande va générer un fichier **MaClasse.class** dans le même répertoire que le fichier source **MaClasse.java**.

- `java MaClasse` **Exécution du programme MaClasse.**

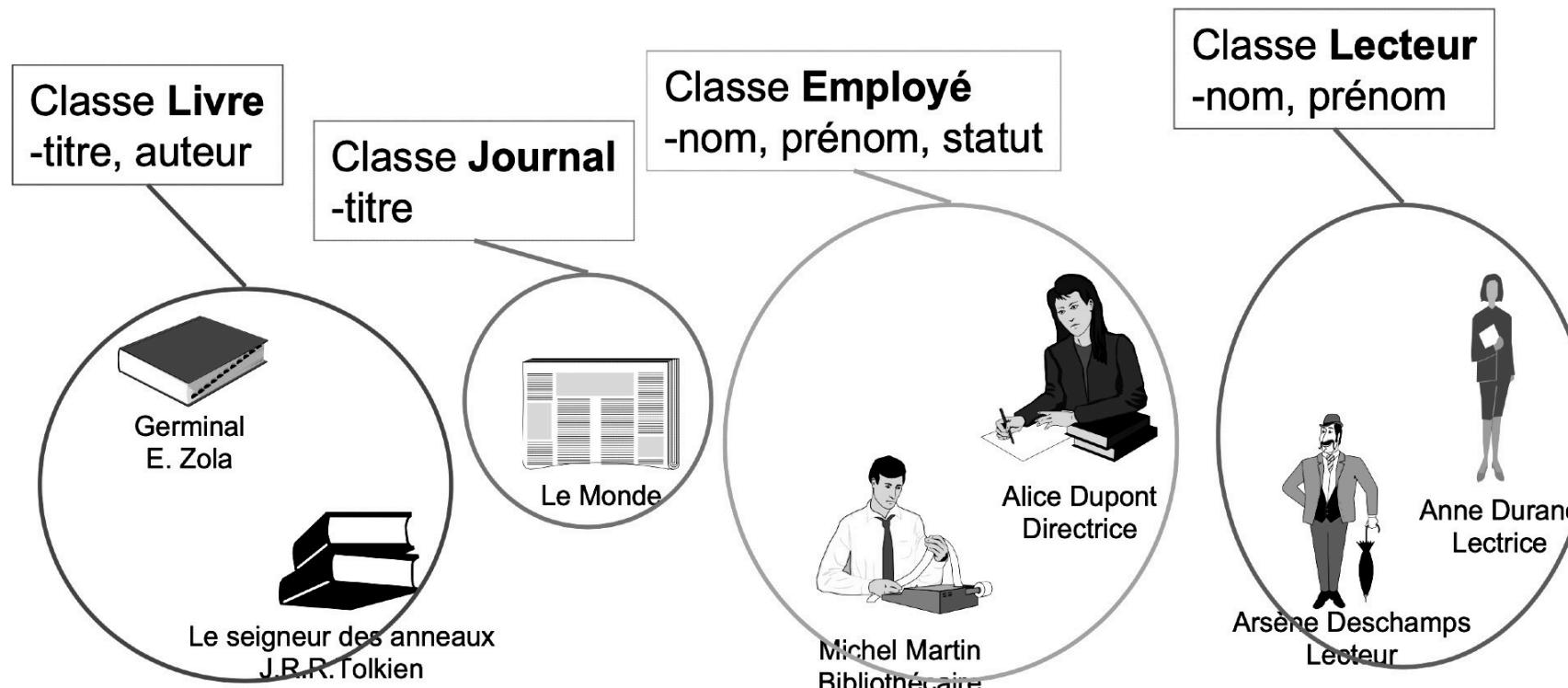
Cette commande est censé exécuter les instruction du programme de la classe **MaClasse** s'ils sont bien sûr exécutable sinon il génère des erreurs d'exécution.

Concepts de base

- Une **classe** est la structure d'un objet, c'est-à-dire la déclaration de l'ensemble des entités qui composeront un objet.
- Un **objet** est donc "issu" d'une classe, c'est le produit qui sort d'un moule. En réalité on dit qu'un objet est une instantiation d'une classe, c'est la raison pour laquelle on pourra parler indifféremment d'objet ou d'instance.
- Une classe est composée de deux parties:
 - Les **attributs** ou champs (parfois appelés données membres): il s'agit des données représentant l'état de l'objet.
 - Les **méthodes** (parfois appelées fonctions membres): il s'agit des opérations applicables aux objets.

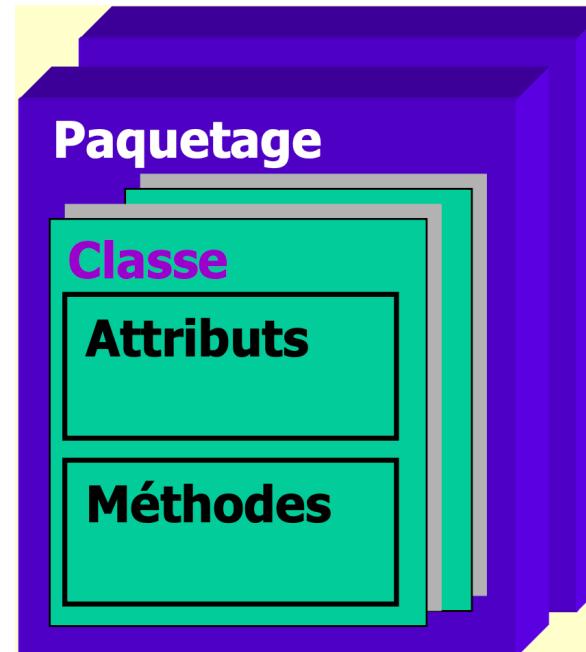
Concepts de base

- Illustration



Structure d'un programme Java

- Un programme Java utilise un ensemble de **classes**.
- Les classes sont regroupées par **paquetage** (en anglais, **package**).
- Une classe regroupe un ensemble **d'attributs** et de **méthodes**.



Syntaxe du langage Java

- Plan
 - Déclaration d'une classe
 - Définition d'une méthode
 - Les commentaires
 - Instructions, blocs et blancs
 - Déclaration d'une variable
 - Portée d'une variable
 - Les opérateurs arithmétiques élémentaires
 - Les opérateurs de comparaison
 - Les opérateurs logiques
 - Les opérateurs d'affectation
 - Point d'entrée d'un programme Java
 - Exemple de programme
 - Compilation et exécution
 - Les identificateurs
 - Les mots réservés de Java

Déclaration d'une classe

- Le nom de la classe est spécifié derrière le mot clé « **class** ».
- Le corps de la classe est délimité par des accolades.
- On définit dans le corps les attributs et les méthodes qui constituent la classe.

```
class Test {
```

```
    < corps de la classe >
```

```
}
```

Définition d'une méthode

- Une méthode est constituée:
 - D'un nom.
 - D'un type de retour.
 - De paramètres ou arguments (éventuellement aucun).
 - D'un bloc d'instructions.
- Un paramètre est constitué:
 - D'un type
 - D'un nom
- « **void** » est le mot-clé signifiant que la méthode ne renvoie pas de valeur.

Définition d'une méthode

- Exemple:

```
class Test {  
    int calculer (int taux, float delta) {  
        < corps de la méthode >  
    }  
}
```

Les commentaires

- `/*` Commentaires sur une ou plusieurs lignes `*/`
 - `//` Commentaires sur une ligne.
 - `/**` Commentaires d'explication `*/`
- Les commentaires d'explication se placent généralement juste avant une déclaration (d'attribut ou de méthode).

Instructions, blocs et blancs

- Les instructions Java se terminent par un « ; ».
- Les blocs sont délimités par deux accolades:
 - { pour le début de bloc
 - } pour la fin du bloc
 - Un bloc permet de définir un regroupement d'instructions. La définition d'une classe ou d'une méthode se fait dans un bloc.
- Les espaces, tabulations, sauts de ligne qui forment **l'indentation** sont autorisés. Cela permet de présenter un code plus lisible.
- Instructions possibles:
 - Déclaration d'une variable
 - Appel de méthode.
 - Affectation.
 - Instruction de boucle (while, for...).
 - Instruction de test (if, switch).

Déclaration d'une variable

- Une **variable** possède un type et un nom.
- Le type peut être un type de **base**(primitif) ou une **classe**(composé).
- L'initialisation d'une variable peut se faire au moment de la déclaration.

```
{  
    int compteur; // Déclaration  
    int indice = 0; // Déclaration + Initialisation  
  
    Voiture golf;  
    Voiture twingo = new Voiture();  
}
```

Portée d'une variable

- La portée d'une variable s'étend jusqu'à la fin du bloc dans lequel elle est définie.

```
{  
{  
    int compteur;  
    ...  
    // compteur est accessible  
}  
  
// compteur n'est plus accessible  
}
```

Les opérateurs arithmétiques élémentaires

- Règles de priorité sur les opérateurs arithmétiques:

Niveau	Symbol	Signification
1	()	Parenthèse
2	*	Produit
	/	Division
	%	Modulo
3	+	Addition ou concaténation
	-	Soustraction

Les opérateurs de comparaison

Opérateur	Exemple	Renvoie TRUE si
>	v1 > v2	v1 plus grand que v2
\geq	v1 \geq v2	Plus grand ou égal
<	v1 < v2	Plus petit que
\leq	v1 \leq v2	Plus petit ou égal à
\equiv	v1 \equiv v2	égal
\neq	v1 \neq v2	différent

Les opérateurs logiques

Opérateur	Usage	Renvoie TRUE si
&& &	expr1 && expr2 expr1 & expr2	expr1 et expr2 sont vraies Idem mais évalue toujours les 2 expressions
 	expr1 expr2 expr1 expr2	Expr1 ou expr2, ou les deux sont vraies idem mais évalue toujours les 2 expressions
!	! expr1	expr1 est fausse
!=	expr1 != expr2	si expr1 est différent de expr2

Les opérateurs d'affectation

- L'opérateur de base est « `=` ».
- Il existe des opérateurs d'affectation qui réalisent à la fois une opération arithmétique et l'affectation proprement dite:

Opérateur	Exemple	Équivalent à
<code>+=</code>	<code>expr1 += expr2</code>	<code>expr1 = expr1 + expr2</code>
<code>-=</code>	<code>expr1 -= expr2</code>	<code>expr1 = expr1 - expr2</code>
<code>*=</code>	<code>expr1 *= expr2</code>	<code>expr1 = expr1 * expr2</code>
<code>/=</code>	<code>expr1 /= expr2</code>	<code>expr1 = expr1 / expr2</code>
<code>%=</code>	<code>expr1 %= expr2</code>	<code>expr1 = expr1 % expr2</code>

Point d'entrée d'un programme Java

- Pour pouvoir faire un programme exécutable il faut toujours une classe qui contienne une méthode particulière: la méthode « **main** ».
 - C'est le point d'entrée dans le programme: le microprocesseur sait qu'il va commencer à exécuter les instructions à partir de cet endroit.

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

Exemple de programme

Fichier Bonjour.java

```
public class Bonjour
{
    //Accolade débutant la classe Bonjour

    public static void main(String args[])
    {
        //Accolade débutant la méthode main

        /* Pour l'instant juste une instruction */

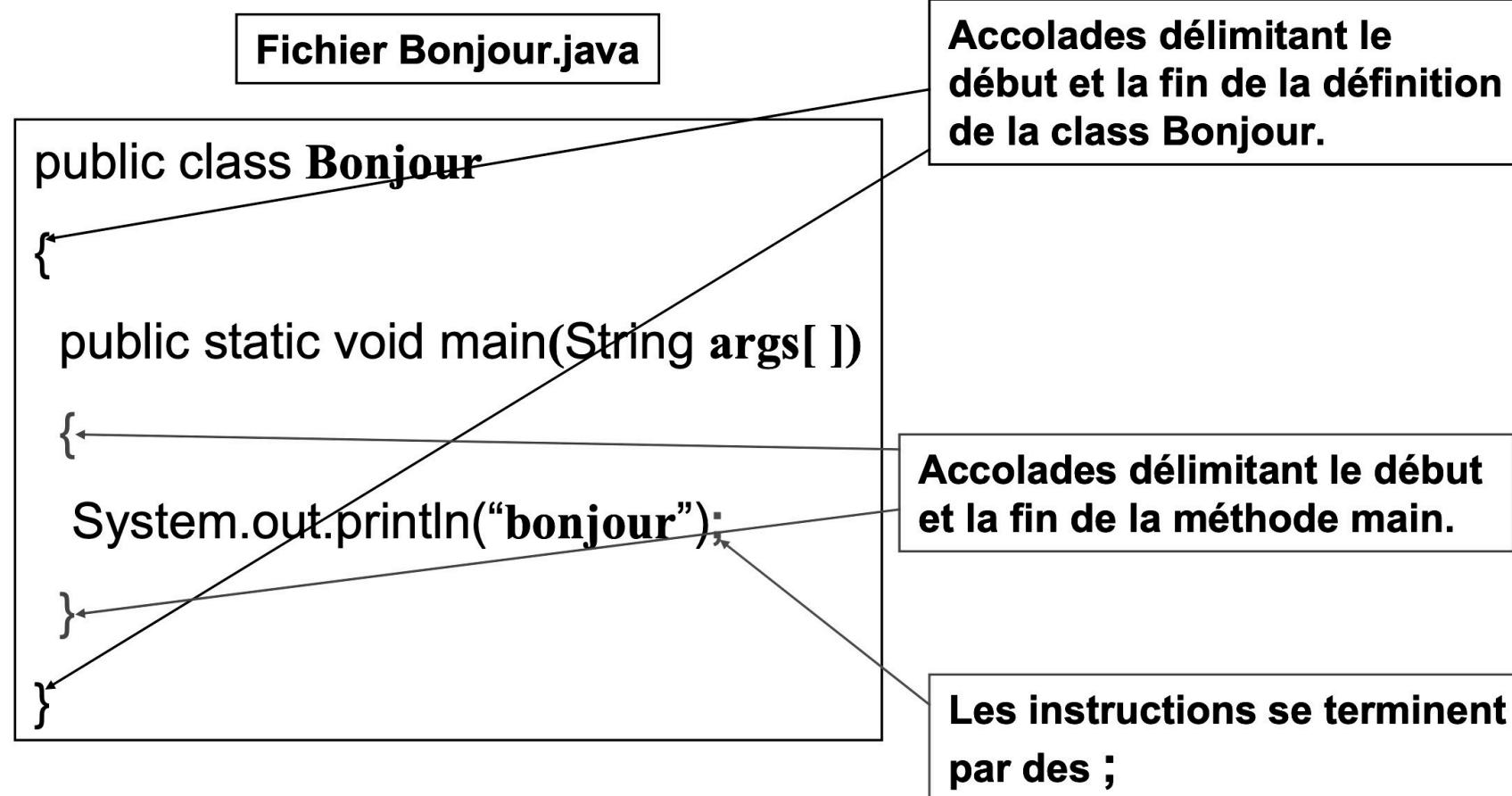
        System.out.println("bonjour");

    } //Accolade fermant la méthode main

} //Accolade fermant la classe Bonjour
```

La classe est l'unité de base de nos programmes.
Le mot clé en Java pour définir une classe est class.

Exemple de programme



Exemple de programme

Fichier Bonjour.java

```
public class Bonjour
{
    public static void main(String args[ ])
    {
        System.out.println("bonjour");
    }
}
```

Une méthode peut recevoir des paramètres. Ici la méthode main reçoit le paramètre args qui est un tableau de chaîne de caractères.

Compilation et exécution

Fichier Bonjour.java

Compilation d'un programme java dans une console DOS:
javac Bonjour.java
Génère un fichier Bonjour.class
Exécution du programme (toujours depuis la console DOS) sur la JVM :
java Bonjour
Affichage de « bonjour » dans la console.

Le nom du fichier est nécessairement celui de la classe suivi de l'extension « .java ». Java est sensible à la casse des lettres (exemple: Bonjour ≠ bonjour)

```
public class Bonjour
{
    public static void main(String[ ] args)
    {
        System.out.println("bonjour");
    }
}
```

Compilation et exécution

- Pour résumer, dans une console DOS ou un terminal, si j'ai un fichier Bonjour.java pour la classe Bonjour:
 - **javac** Bonjour.java
 - **javac** est la commande qui lance le compilateur Java.
 - Compilation en bytecode java.
 - Indication des erreurs (éventuelles) de syntaxe.
 - Génération d'un fichier Bonjour.class s'il n'y a pas d'erreurs.
 - **java** Bonjour
 - **java** est la commande qui lance la machine virtuelle (JVM).
 - Exécution du bytecode.
 - Remarque:
 - Nécessité de la méthode main, qui est le point d'entrée dans le programme.

Compilation et exécution

- On peut utiliser ce qu'on appelle un environnement de développement intégré (**IDE**) pour compiler et exécuter des applications Java.
- Un **IDE** Java est un éditeur spécifique du code Java ayant une interface intuitive qui permet de faciliter l'édition, la compilation, la correction d'erreurs et l'exécution des applications Java.
- Exemples d'IDE Java:
 - NetBeans,
 - Eclipse,
 - VsCode,
 - IntelliJ,
 -

Les identificateurs

- Un **identificateur**(ou nom) permet de désigner une classe, une méthode, une variable...
- Règles à respecter pour les identificateurs:
 - Interdiction d'utiliser les mots-clés (mots réservés de Java).
 - Les identificateurs peuvent commencer par:
 - Une lettre.
 - Un « **\$** ».
 - Un « **_** » (underscore).
 - Les identificateurs ne peuvent pas commencer par:
 - Un chiffre.
 - Un signe autre que « **\$** » ou « **_** ».

Les mots réservés de Java

abstract	default	goto	null	synchronized
boolean	do	if	package	this
break	double	implements	private	throw
byte	else	import	protected	throws
case	extends	instanceof	public	transient
catch	false	int	return	true
char	final	interface	short	try
class	finally	long	static	void
continue	float	native	super	volatile
const	for	new	switch	while

Les types primitifs

- Plan
 - Introduction
 - Les entiers
 - Les réels
 - Les booléens
 - Les caractères
 - Lecture des entrées en JAVA

Introduction

- En Java, tout est objet sauf les types primitifs ou types de base.
- Il y a huit types de base:
 - Un type booléen pour représenter les variables ne pouvant prendre que 2 valeurs (vrai et faux, 0 ou 1, etc.):
 - Notation: **boolean** avec les valeurs associées **true** et **false**.
 - Un type pour représenter les caractères:
 - Notation: **char**.
 - Quatre types pour représenter les entiers de divers taille:
 - Notation: **byte**, **short**, **int** et **long**.
 - Deux types pour représenter les réelles:
 - Notation: **float** et **double**.

Les entiers

- Les entiers (avec signe):
 - **byte**: codé sur 8 bits, peuvent représenter des entiers allant de -2^7 à $2^7 - 1$ (-128 à +127).
 - **short**: codé sur 16 bits, peuvent représenter des entiers allant de -2^{15} à $2^{15} - 1$.
 - **int**: codé sur 32 bits, peuvent représenter des entiers allant de -2^{31} à $2^{31} - 1$.
 - **long**: codé sur 64 bits, peuvent représenter des entiers allant de -2^{63} à $2^{63} - 1$.
- Opérations sur les entiers:
 - + : addition.
 - - : soustraction.
 - * : multiplication.
 - / : division entière.
 - % : reste de la division entière.
 - Exemples:
 - $15/4$ donne 3
 - $17\%3$ donne 2

Les entiers

- Opérations sur les entiers:
 - Les opérateurs d'incrémentation et de décrémentation **++** et **--** :
 - **++** : ajoute 1 à la valeur d'une variable entière.
 - **--** : retranche 1 de la valeur d'une variable entière.
 - **n++;** « équivalent à » $n = n+1$;
 - **n--;** « équivalent à » $n = n-1$;
 - Exemple:
 - `int n=12;`
 - `n ++; // Maintenant n vaut 13.`

Les entiers

- Opérations sur les entiers:
 - `8++`; est une instruction illégale.
 - L'incrémentation (resp. la décrémentation) peut être utilisée de manière préfixée: `++n`.
 - La différence avec la version suffixée se voit quand on les utilisent dans les expressions.
 - En version préfixée l'incrémentation (resp. la décrémentation) s'effectue en premier, alors qu'elle s'effectue en dernier en version suffixée.
 - Exemple illustratif:
 - `int m=7; int n=7;`
 - `int a=2 * ++m; // a vaut 16, m vaut 8`
 - `int b=2 * n++; // b vaut 14, n vaut 8`

Les réels

- Les réels:
 - **float**: codé sur 32 bits, peuvent représenter des nombres allant de -10^{35} à $+10^{35}$.
 - **double**: codé sur 64 bits, peuvent représenter des nombres allant de -10^{400} à $+10^{400}$.
- Notation:
 - 4.55 ou 4.55**D**: réel en double précision.
 - 4.55**f**: réel en simple précision.

Les réels

- Opération sur les réels:
 - Opérateurs classiques: +, -, *, /
 - Attention pour la division:
 - $15/4$ donne 3.
 - $15.0 / 4$ donne 3.75 (si l'un des termes de la division est un réel, la division retournera un réel).

Les booléens

- Les booléens:
 - `boolean`: contient soit vrai (`true`) soit faux (`false`).
- Les opérateurs logiques de comparaison:
 - Egalité: opérateur `==`
 - Différence: opérateur `!=`
 - Supérieur et inférieur strictement à: opérateurs `>` et `<`
 - Supérieur et inférieur ou égal: opérateurs `>=` et `<=`
- Exemple:
 - `boolean x;`
 - `x= true;`
 - `x= false;`
 - `x= (5==5);` // l'expression `(5==5)` est évaluée et la valeur est affectée à `x` qui vaut alors vrai
 - `x= (5!=4);` // `x` vaut vrai, ici on obtient vrai car 5 est différent de 4 `x= (5>5);` // `x` vaut faux, 5 n'est pas supérieur strictement à 5 `x= (5<=5);` // `x` vaut vrai, 5 est bien inférieur ou égal à 5

Les booléens

- Autres opérateurs logiques:
 - Et logique: `&&`
 - Ou logique: `||`
 - Non logique: `!`
- Exemples:
 - `boolean a,b, c;`
 - `a= true;`
 - `b= false;`
 - `c= (a && b); // c vaut false`
 - `c= (a || b); // c vaut true`
 - `c= !(a && b); // c vaut true`
 - `c=!a; // c vaut false`

Les caractères

- Les caractères:
 - « char » contient un seul caractère (lettre, symbole, ponctuation...).
 - Le type char désigne des caractères en représentation Unicode.
 - Codage sur 2 octets contrairement à ASCII/ANSI codé sur 1 octet.
 - Notation hexadécimale des caractères Unicode de ‘ \u0000 ’ à ‘ \uFFFF ’.
- Remarque:
 - Le codage ASCII/ANSI est un sous-ensemble de la représentation Unicode.
 - Pour plus d'information sur Unicode: www.unicode.org
- Exemple:
 - `char a,b,c; // a,b et c sont des variables de type char`
 - `a='a'; // a contient la lettre « a »`
 - `b= '\u0022' // b contient le caractère guillemet " c=97; // c contient le caractère de rang 97: 'a'`

Lecture des entrées en JAVA

- Pour lire les entrées de l'utilisateur à partir de différentes sources comme le clavier, on peut utiliser la classe **Scanner** et ses différentes méthodes de récupération des saisies des utilisateurs.
- Importation de la classe **Scanner**:
 - `import java.util.Scanner;`
- Création d'un objet **Scanner**:
 - Pour lire l'entrée de l'utilisateur à partir du clavier, on crée un objet Scanner en passant **System.in** en paramètre :
 - Exemple: `Scanner sc = new Scanner(System.in);`

Lecture des entrées en JAVA

- Méthodes de la classe Scanner:

```
//Création de l'objet Scanner
Scanner sc = new Scanner(System.in);

//Lire une ligne de texte (String)
String line = sc.nextLine();
System.out.println("Vous avez saisi : " + line);
//Lire un mot (String)
String word = sc.next();
System.out.println("Vous avez saisi : " + word);
//Lire un entier (int)
int number = sc.nextInt();
System.out.println("Vous avez saisi : " + number);
//Lire un nombre à virgule flottante (double or float)
double decimal = sc.nextDouble();
float fl1 = sc.nextFloat();
System.out.println("Vous avez saisi : " + decimal);
//Lire un booléen (boolean)
boolean bool = sc.nextBoolean();
System.out.println("Vous avez saisi : " + bool);
```

Les structures de contrôle

- Plan
 - If else
 - while
 - do-while
 - for
 - break-continue
 - switch

If else

- Comme la plupart des langages impératifs, Java propose un ensemble de structures de contrôle.

- L'expression **if** permet d'exécuter un bloc d'instructions uniquement si l'expression booléenne est évaluée à vrai :

- **Exemple:** `if (i % 2 == 0) {`

// instructions à exécuter si i est pair
}

- L'expression **if** peut être optionnellement suivie d'une expression **else** pour les cas où l'expression est évaluée à faux :

- **Exemple:** `if (i % 2 == 0) {`

// instructions à exécuter si i est pair
} **else** {
// instructions à exécuter si i est impair
}

If else

- L'expression **else** peut être suivie d'une nouvelle instruction **if** afin de réaliser des choix multiples :
- Exemple : **if (i % 2 == 0) {**

```
// instructions à exécuter si i pair
} else if (i > 10) {
// instructions à exécuter si i est impair et supérieur à 10
} else {
// instructions à exécuter dans tous les autres cas
}
```

- Remarque: Si le bloc d'instruction d'un **if** ne comporte qu'une seule instruction, alors les accolades peuvent être omises :
 - Exemple: **if (i % 2 == 0)**

i++; //Cependant, beaucoup de développeurs Java préfèrent utiliser systématiquement les accolades.

while

- L'expression **while** permet de définir un bloc d'instructions à répéter tant que l'expression booléenne est évaluée à vrai.
- **Exemple:** `while (i % 2 == 0) {
 // instructions à exécuter tant que i est pair
}`
- L'expression booléenne est évaluée au départ et après chaque exécution du bloc d'instructions.
- **Remarque:** Si le bloc d'instruction d'un **while** ne comporte qu'une seule instruction, alors les accolades peuvent être omises :

do-while

- Il existe une variante de la structure précédente, nommée **do-while** :

- Exemple: do {

// instructions à exécuter

```
} while (i % 2 == 0);
```

- Exemple: while (i % 2 == 0) {

// instructions à exécuter tant que i est pair

```
}
```

- L'expression booléenne est évaluée au départ et après chaque exécution du bloc d'instructions.

- Remarque: Si le bloc d'instruction d'un **while** ne comporte qu'une seule instruction, alors les accolades peuvent être omises :

for

- Une expression `for` permet de réaliser une itération. Elle commence par réaliser une initialisation puis évalue une expression booléenne. Tant que cette expression booléenne est évaluée à vrai, le bloc d'instructions est exécuté et un incrément est appelé.

- **Syntaxe:** `for (initialisation; expression booléenne; incrément)`

```
{ bloc d'instructions }
```

- **Exemple:** `for (int i = 0; i < 10; ++i)`

```
{ // instructions }
```

- Il existe une forme améliorée de l'expression `for` (souvent appelée *for-each*) qui permet d'exprimer plus succinctement un parcours d'une collection d'éléments.

- Pour que cette expression compile, il faut que la variable désignant la collection à droite de `:` implémente le type `Iterable` ou qu'il s'agisse d'un tableau. Il faut également que la variable à gauche de `:` soit compatible pour l'assignation d'un élément de la collection.

- **Exemple:**
`short arrayOfShort[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
for (int k : arrayOfShort) {
 System.out.println(k);
}`

break-continue

- Pour les expressions **while**, **do-while**, **for** permettant de réaliser des itérations, il est possible de contrôler le comportement à l'intérieur de la boucle grâce aux mots-clés **break** et **continue**.
- **break** quitte la boucle sans exécuter le reste des instructions.

- **Exemple:**

```
int k = 10;
for (int i = 1 ; i < 10; ++i){
    k *= i
    if (k > 200) {
        break;
    }
}
```

- **continue** arrête l'exécution de l'itération actuelle et commence l'exécution de l'itération suivante.

- **Exemple:**

```
for (int i = 1 ; i < 10; ++i) {
    if (i % 2 == 0) {
        continue;
    }
    System.out.println(i);
}
```

switch

- Un expression **switch** permet d'effectuer une sélection parmi plusieurs valeurs.

- **Syntaxe:** `switch (s) {`

```
    case "valeur 1":  
        // instructions  
        break;  
    case "valeur 2":  
        // instructions  
        break;  
    case "valeur 3":  
        // instructions  
        break;  
    default:  
        // instructions  
}
```

- **switch** évalue l'expression entre parenthèses et la compare dans l'ordre avec les valeurs des lignes `case`. Si une est identique alors il commence à exécuter la ligne d'instruction qui suit. Attention, un `case` représente un point à partir duquel l'exécution du code commencera. Si on veut isoler chaque cas, il faut utiliser une instruction `break`. Au contraire, l'omission de l'instruction `break` peut être pratique si on veut effectuer le même traitement pour un ensemble de cas :

Les tableaux

- Plan
 - Définition
 - Déclaration
 - Création
 - Taille et indices
 - Initialisation, création et initialisation simultanée
 - Valeur par défaut des éléments
 - Tableaux bidimensionnels

Définition, Déclaration et Création

- Les tableaux permettent de stocker plusieurs valeurs de même type dans une variable.
 - Les valeurs contenues dans la variable sont repérées par un indice.
 - En langage java, les tableaux sont des objets.
- Déclaration:
 - **Type Nom_Tableau[]; ou Type[] Nom_Tableau;**
 - **Exemple:**
 - `int tab[]; // Ou int[] tab;`
 - `String chaines[]; // Ou String[] chaines;`
- Création d'un tableau:
 - **Nom_Tableau = new Type[Taille_Tableau];**
 - **Exemple:**
 - `tab = new int [20]; // tableau de 20 entiers`
 - `chaines = new String [100]; // tableau de 100 chaînes`

La taille et les indices

- Le nombre d'éléments du tableau est mémorisé. Java peut ainsi détecter, lors de l'exécution, le dépassement d'indice et générer une exception.
- Mot clé pour récupérer la taille d'un tableau: **length** .
 - **Syntaxe:**
 - **Nom_Tableau.length;**
 - **Exemple:**
 - `int taille = tab.length; // taille vaut 20`
- Comme en C, les **indices** d'un tableau commencent à ' 0 '. Donc un tableau de taille 10 aura ses indices qui iront de 0 à 9.

Initialisation, Création et initialisation simultanée

- Initialisation:
 - `tab[0] = 5;`
 - `tab[1] = 3; // etc.`
 - `chaines[0] = new String("Pierre");`
 - `chaines[1] = new String("Paul"); // etc.`
- Crédit et initialisation simultanées:
 - `int tab [] = {5, 3};`
 - `String chaines [] = {"Pierre", "Paul"};`

Valeurs par défaut des éléments dans un tableau

- Voici un tableau qui résume les valeurs par défaut des éléments dans un tableau en Java pour chaque type primitif :

Type primitif	Valeur par défaut dans un tableau
int	0
double	0.0
float	0.0f
long	0L
short	0
byte	0
char	'\u0000' (caractère null)
boolean	false

Tableaux bidimensionnels

- Syntaxe de déclaration et de création:

- Type[][] Nom_Tableau = new Type[Nombre_Lignes][Nombre_Colones];
- Exemple:

```
public class UneMatrice {  
    Run | Debug  
    public static void main(String[ ] args) {  
        int[ ][ ] M = new int[3][5];  
        for (int i=0; i<3; i++){  
            for (int j=0; j<5; j++){  
                M[i][j] = 10*(i+1) + j;  
            }  
        }  
        for (int i=0; i<3; i++){  
            print(M[i]);  
        }  
        System.out.println("M[2][4] = " + M[2][4]);  
    }  
    public static void print(int[ ] a) {  
        for (int i=0; i<a.length; i++){  
            System.out.print(a[i] + " ");  
        }  
        System.out.println();  
    }  
}
```

```
10 11 12 13 14  
20 21 22 23 24  
30 31 32 33 34  
M[2][4] = 34
```

```
○ (base) user@Cheikh-MBENGUE-DISI ~ %
```