# UNIPRO - Java - Mr Niang

I. **Module 1:** Concepts de base
   A. Introduction Java
   B. Programme “HelloWorld”
   C. Les Commentaires en Java
   D. Les Variables en Java
   E. Les Opérateurs Primitifs
   F. Incrémentation et Décrémentation
   G. Strings
   H. Récupérer les données des utilisateurs
II. **Module 2:** Conditionnelles et boucles
   A. Déclaration Conditionnelle
   B. L’instruction SI imbriqué
   C. Les instructions SINON SI
   D. Déclaration logique
   E. L’instruction Switch
   F. Boucle While
   G. Boucle For
   H. Boucle Do While
III. **Module 3:** Tableaux
   A. Les tableaux
   B. Additionner des éléments dans des tableaux
   C. Boucle For Améliorer
   D. Tableaux multidimensionnel
IV. **Module 4:** Classes et objets
   A. Programmation Orienté Objet
   B. Méthodes
   C. Type de retour d’une méthode
   D. Création de Classes et d’objets
   E. Les attributs d’une classe
   F. Modificateurs d'accès
   G. Getters et Setters
   H. Constructeurs
   I. Types de valeur et de référence
   J. La classe Math
   K. Static
   L. Final
   M. Packages
V. **Module 5:** Plus sur les Classes
   A. Encapsulation

# I. Module 1: Concepts de base

## A- Introduction Java !

Java est un langage de programmation moderne de haut niveau conçu au début des années 1990 par Sun Microsystems et actuellement détenu par Oracle.

Java est indépendant de la plate-forme, ce qui signifie que vous n'avez besoin d'écrire le programme qu'une seule fois pour pouvoir l'exécuter sur un certain nombre de plates-formes différentes!
Java est portable, robuste et dynamique, avec la capacité de répondre aux besoins de pratiquement n'importe quel type d'application.
Plus de 3 milliards d'appareils exécutent Java.
Java est utilisé pour développer des applications pour le système d'exploitation Android de Google, diverses applications de bureau, telles que des lecteurs multimédias, des programmes antivirus, des applications Web, des applications d'entreprise (c.-à-d. Bancaires), et bien d'autres!

### B - Programme “HelloWorld”

Commençons par créer un programme simple qui imprime «Hello World» à l'écran

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

En Java, chaque ligne de code qui peut réellement s'exécuter doit se trouver dans une classe.
Dans notre exemple, nous avons nommé la classe MyClass.
Vous en apprendrez plus sur les cours dans les prochains

modules.

En Java, chaque application a un point d'entrée, ou un point de départ, qui est une méthode appelée main. Avec main, les mots-clés public et static seront également expliqués plus tard.

## C - Les Commentaires en Java

Le but d'inclure des commentaires dans votre code est d'expliquer ce que fait le code.
Java prend en charge les commentaires sur une ou plusieurs lignes. Tous les caractères qui apparaissent dans un commentaire sont ignorés par le compilateur Java.

- **Commentaire sur une ligne**

Un commentaire sur une seule ligne commence par deux barres obliques et se poursuit jusqu'à la fin de la ligne.
Par exemple:

```
// c’est un commentaire sur une ligne
x = 5; // une ligne de commentaire après le code
```

Ajouter des commentaires lorsque vous écrivez du code est une bonne pratique, car ils apportent des éclaircissements et une compréhension lorsque vous devez vous y référer, ainsi que pour ceux qui pourraient avoir besoin de le lir**e.**

- **Commentaire sur plusieurs ligne**

Java prend également en charge les commentaires qui s'étendent sur plusieurs lignes.
Vous commencez ce type de commentaire par une barre oblique suivie d'un astérisque et vous le terminez par un

astérisque suivi d'une barre oblique.
Par exemple:

```
/*  C'est aussi un
   commentaire couvrant
   plusieurs lignes
*/
```

Notez que Java ne prend pas en charge les commentaires multilignes imbriqués.
Cependant, vous pouvez imbriquer des commentaires sur une seule ligne dans des commentaires sur plusieurs lignes.

## D - Les Variables en Java

Les variables stockent des données pour le traitement.
Une variable reçoit un nom (ou identifiant), tel que la zone, l'âge, la taille, etc. Le nom identifie de façon unique chaque variable, en attribuant une valeur à la variable et en récupérant la valeur stockée.

Les variables ont des **types**. Quelques exemples:
- **int**: pour les entiers (nombres entiers) tels que 123 et -456
- **double**: pour les nombres à virgule flottante ou réels avec des décimales facultatives et des parties fractionnaires dans des notations fixes ou scientifiques, telles que 3.1416, -55.66.
- **Chaîne**: pour les textes tels que "Bonjour" ou "Bonjour!". Les chaînes de texte sont placées entre guillemets doubles.

Vous pouvez déclarer une variable d'un type et lui affecter une valeur. Exemple:

```
String nom = "David";
```

Cela crée une variable appelée **nom** de type **String** et lui affecte la valeur "David".

Il est important de noter qu'une variable est associée à un type et n'est capable que de stocker des valeurs de ce type particulier. Par exemple, une variable int peut stocker des valeurs entières, telles que 123; mais il ne peut pas stocker des nombres réels, tels que 12,34, ou des textes, tels que "Bonjour".

Exemples de déclarations de variables:

```
class MyClass {
 public static void main(String[ ] args) {
   String name = "David";
   int age = 42;
   double score = 15.9;
   char group = 'Z';
 }
}
```

**char** représente le caractère et détient un seul caractère.
Un autre type est le type **Boolean**, qui n'a que deux valeurs possibles: **true** et **false**.
Ce type de données est utilisé pour les indicateurs simples qui suivent les conditions vraies / fausses.
Par exemple:

```
boolean online = true;
```

Vous pouvez utiliser une liste séparée par des virgules pour

déclarer plusieurs variables du type spécifié.
**Exemple:** `int a = 42, b = 11;`

## E - Les Opérateurs Primitifs

- **Les Opérateurs mathématiques**

Java fournit un riche ensemble d'opérateurs à utiliser pour manipuler des variables. Une valeur utilisée de chaque côté d'un opérateur est appelée opérande.
Par exemple, dans l'expression ci-dessous, les nombres 6 et 3 sont des opérandes de l'opérateur plus:

```
int x = 6 + 3;
```

Opérateurs arithmétiques Java:
**+ ajout**
**- soustraction**
*** multiplication**
**/ division**
**% modulo**

Les opérateurs arithmétiques sont utilisés dans les expressions mathématiques de la même manière qu'ils sont utilisés dans les équations algébriques.

**Addition**
L'opérateur **+** additionne deux valeurs, telles que deux constantes, une constante et une variable, ou une variable et une variable. Voici quelques exemples d'ajout:

```
int sum1 = 50 + 10;
int sum2 = sum1 + 66;
int sum3 = sum2 + sum2;
```

**Subtraction**

L'opérateur **-** soustrait une valeur d'une autre.

```
int sum1 = 1000 - 10;
int sum2 = sum1 - 5;
int sum3 = sum1 - sum2;
```

Tout comme en algèbre, vous pouvez utiliser les deux opérations sur une seule ligne.
Par exemple: `int val = 10 + 5 - 2;`

**Multiplication**

L'opérateur ***** multiplie deux valeurs.

```
int sum1 = 1000 * 2;
int sum2 = sum1 * 10;
int sum3 = sum1 * sum2;
```

**Division**

L'opérateur **/** divise une valeur par une autre.

```
int sum1 = 1000 / 5;
int sum2 = sum1 / 2;
int sum3 = sum1 / sum2;
```

Dans l'exemple ci-dessus, le résultat de l'équation de division sera un nombre entier, car **int** est utilisé comme type de données. Vous pouvez utiliser **double** pour récupérer une valeur avec un point décimal.

**Modulo**

L'opération mathématique **modulo** (ou le reste) effectue une division entière d'une valeur par une autre et renvoie le reste de cette division.

L'opérateur de l'opération modulo est le pourcentage (%).

Exemple:

```
int value = 23;
int res = value % 6; // res is 5
```

La division de 23 par 6 renvoie un quotient de 3, avec un reste de 5. Ainsi, la valeur de 5 est affectée à la **variable res.**

## F - Incrémentation et Décrémentation

- **Opérateurs d'incrémentation**

Un opérateur d'**incrémentation** ou de **décrémentation** offre un moyen plus pratique et compact d'augmenter ou de diminuer la valeur d'une variable **d'une unité**.

Par exemple, l'instruction **x = x + 1;** peut être simplifié en **++ x**;

Exemple:

```
int test = 5;
++test;  //test est maintenant 6
```

L'opérateur de **décrémentation** (-) est utilisé pour diminuer la valeur d'une variable d'une unité.

```
int test = 5;
--test;  //test is now 4
```

Utilisez cet opérateur avec prudence pour éviter les erreurs de calcul.

**Prefix & Postfix**

Deux formes, **préfixe** et **postfixe**, peuvent être utilisées avec les opérateurs d'incrémentation et de décrémentation.
Sous forme de préfixe, l'opérateur apparaît avant l'opérande, tandis que sous forme de suffixe, l'opérateur apparaît après l'opérande. Voici une explication du fonctionnement des deux formulaires:
**Préfixe**: incrémente la valeur de la variable et utilise la nouvelle valeur dans l'expression.
Exemple:

```
int x = 34;
int y = ++x; // y is 35
```

La valeur de x est d'abord incrémentée à 35, puis affectée à y, de sorte que les valeurs de x et y sont désormais de 35.

**Postfix:** la valeur de la variable est d'abord utilisée dans l'expression, puis augmentée.
Exemple:

```
int x = 34;
int y = x++; // y is 34
```

x est d'abord affecté à y, puis incrémenté de un. Par conséquent, x devient 35, tandis que y reçoit la valeur 34.
Il en va de même pour l'opérateur de décrémentation.

**Opérateurs d'affectation**

Vous connaissez déjà l'opérateur d'**affectation** (=), qui attribue une valeur à une variable.

```
int value = 5;
```

Cela a affecté la valeur 5 à une variable appelée **valeur** de type **int**.

Java fournit un certain nombre d'opérateurs d'affectation pour faciliter l'écriture de code.
**Addition et affectation** (+ =):

```
int num1 = 4;
int num2 = 8;
num2 += num1; // num2 = num2 + num1;

// num2 is 12 and num1 is 4
```

**Subtraction and assignment (-=):**

```
int num1 = 4;
int num2 = 8;
num2 -= num1; // num2 = num2 - num1;

// num2 is 4 and num1 is 4
```

De même, Java prend en charge
la multiplication et l'affectation (* =),
la division et l'affectation (/ =),
le reste et l'affectation (% =).

**Strings**

Une chaîne est un objet qui représente une séquence de caractères.
Par exemple, "Bonjour" est une chaîne de 5 caractères.

```
String s = "SoloLearn";
```

Vous êtes autorisé à définir une chaîne vide. Par exemple,
String str = " ";

**String Concatenation**
L'opérateur + (plus) entre les chaînes les additionne pour créer une nouvelle chaîne. Ce processus est appelé **concaténation**.
La chaîne résultante est la première chaîne assemblée avec la deuxième chaîne.
Par exemple:

```
String firstName, lastName;
firstName = "David";
lastName = "Williams";

System.out.println("My name is " + firstName + "" + lastName);

// Prints: My name is David Williams
```

Le type de données **char** représente un seul caractère.

## H - Obtention de la saisie utilisateur

Bien que Java propose de nombreuses méthodes différentes pour obtenir des entrées utilisateur, l'objet Scanner est le plus courant et peut-être le plus facile à implémenter. Importez la classe Scanner pour utiliser l'objet Scanner, comme illustré ici:

```
import java.util.Scanner;
```

Pour utiliser la classe Scanner, créez une **instance** de la classe

en utilisant la syntaxe suivante:

```
Scanner myVar = new Scanner(System.in);
```

Vous pouvez désormais lire différents types de données d'entrée saisies par l'utilisateur.
Voici quelques méthodes disponibles via la classe Scanner:
Lire un octet (byte) - nextByte()
Lire un court (short) - nextShort()
Lire un entier (**int**) - nextInt()
Lire un long (long) - nextLong()
Lire un flotteur (**float**) - nextFloat()
Lire un double (double) - nextDouble()
Lire un booléen (**boolean**) - nextBoolean()
Lire une ligne complète - nextLine()
Lire un mot (word) - next()

Exemple de programme utilisé pour obtenir une entrée utilisateur:

```
import java.util.Scanner;

class MyClass {
    public static void main(String[ ] args) {
        Scanner myVar = new Scanner(System.in);
        System.out.println(myVar.nextLine());
    }
}
```

Cela attendra que l'utilisateur entre quelque chose et imprime cette entrée.

# II. Module 2: Conditionnelles et Boucles

## A. Déclarations Conditionnelle

La prise de décision :

**Les instructions conditionnelles** sont utilisées pour effectuer différentes actions en fonction de différentes conditions.

**L'instruction if** est l'une des instructions conditionnelles les plus fréquemment utilisées.

Si l'expression de condition de l'instruction **if** prend la valeur true, le bloc de code à l'intérieur de l'instruction if est exécuté.

Si l'expression s'avère fausse, le premier ensemble de code après la fin de l'instruction if (après l'accolade fermante) est exécuté.

**Syntaxe:**

```
if (condition) {
  //Executes when the condition is true
}
```

L'un des opérateurs de comparaison suivants peut être utilisé pour former la condition:

**<** Inférieur à
**>** supérieur à
**! =** différent de
**==** égal à
**<=** inférieur ou égal à
**>=** supérieur ou égal à

Par exemple:

```
int x = 7;
if(x < 42) {
  System.out.println("Hi");
```

```
}
```

N'oubliez pas que vous devez utiliser deux signes égaux (**==**) pour tester **l'égalité**, puisqu'un seul signe égal est l'opérateur d'affectation.

**L'instruction if ... else**

Une instruction **if** peut être suivie d'une instruction **else** facultative, qui s'exécute lorsque la condition est évaluée à false.

Par exemple:

```
int age = 30;

if (age < 16) {
  System.out.println("Trop jeune");
} else {
  System.out.println("Bienvenue !");
}
//Outputs "Welcome!"
```

L'âge étant égal à 30, la condition de l'instruction if est évaluée à false et l'instruction else est exécutée.

### B - Les instructions SI imbriqué

Vous pouvez utiliser une instruction if-else dans une autre instruction if ou else.

Par exemple:

```
int age = 25;
if(age > 0) {
  if(age > 16) {
```

```
      System.out.println("Welcome!");
   } else {
      System.out.println("Trop Jeune");
   }
} else {
   System.out.println("Error");
}
//Outputs "Welcome!"
```

Vous pouvez imbriquer autant d'instructions if-else que vous le souhaitez

### C- Les instructions SINON SI

Au lieu d'utiliser des instructions if-else imbriquées, vous pouvez utiliser l'instruction else if pour vérifier plusieurs conditions.

Par exemple:

```
int age = 25;

if(age <= 0) {
   System.out.println("Error");
} else if(age <= 16) {
   System.out.println("Too Young");
} else if(age < 100) {
   System.out.println("Welcome!");
} else {
   System.out.println("Really?");
}
//Outputs "Welcome!"
```

Le code vérifiera la condition à évaluer comme vraie et exécutera les instructions à l'intérieur de ce bloc.

### D - Les Opérateurs Logiques

Les opérateurs logiques sont utilisés pour combiner plusieurs conditions.

Supposons que vous vouliez que votre programme affiche "Bienvenue!" uniquement lorsque l'âge variable est supérieur à 18 ans et que l'argent variable est supérieur à 500.
Une façon d'y parvenir consiste à utiliser des instructions if imbriquées:

```
if (age > 18) {
  if (money > 500) {
     System.out.println("Welcome!");
  }
}
```

Cependant, l'utilisation de l'opérateur logique **AND** (**&&**) est une meilleure solution:

```
if (age > 18 && money > 500) {
  System.out.println("Welcome!");
}
```

Si les deux opérandes de l'opérateur AND sont vrais, alors la condition devient vraie

**L'opérateur OR**

L'opérateur OR (||) vérifie si l'une des conditions est vraie.
La condition devient vraie, si l'un des opérandes est évalué

comme vrai.

Par exemple:

```
int age = 25;
int money = 100;

if (age > 18 || money > 500) {
  System.out.println("Welcome!");
}
//Outputs "Welcome!"
```

Le code ci-dessus affichera "Bienvenue!" si l'âge est supérieur à 18 ans ou si l'argent est supérieur à 500.

L'opérateur logique **NOT** (**!**) Est utilisé pour inverser l'état logique de son opérande. Si une condition est vraie, l'opérateur **NOT** logique la rendra fausse.

Exemple:

```
int age = 25;
if(!(age > 18)) {
  System.out.println("Too Young");
} else {
  System.out.println("Welcome");
}
//Outputs "Welcome"
```

!(age > 18) se lit comme "*si l'âge N'EST PAS supérieur à 18*".

### E - L'instruction switch

Une instruction switch teste l'égalité d'une variable par rapport à une liste de valeurs. Chaque valeur est appelée un cas et la variable activée est vérifiée pour chaque cas.
Syntaxe:

```
switch (expression) {
  case value1 :
    //Statements
    break; //optional
  case value2 :
    //Statements
    break; //optional
   //You can have any number of case statements.
   default : //Optional
      //Statements
}
```

- Lorsque la variable activée est égale à un cas, les instructions suivant ce cas s'exécutent jusqu'à ce qu'une instruction break soit atteinte.

- Lorsqu'une instruction break est atteinte, le commutateur se termine et le flux de contrôle passe à la ligne suivante après l'instruction switch.

- Tous les cas n'ont pas besoin de contenir une pause. Si aucune interruption n'apparaît, le flux de contrôle passe aux cas suivants jusqu'à ce qu'une interruption soit atteinte.

L'exemple ci-dessous teste day par rapport à un ensemble de valeurs et imprime un message correspondant.

```
int day = 3;
```

```
switch(day) {
 case 1:
   System.out.println("Monday");
   break;
 case 2:
   System.out.println("Tuesday");
   break;
 case 3:
   System.out.println("Wednesday");
   break;
}
// Outputs "Wednesday"
```

Vous pouvez avoir n'importe quel nombre d'instructions de cas dans un commutateur. Chaque cas est suivi de la valeur de comparaison et de deux points.

**La déclaration par défaut**

Une instruction switch peut avoir un cas par défaut facultatif. Le cas par défaut peut être utilisé pour effectuer une tâche lorsqu'aucun des cas ne correspond.

Par exemple:

```
int day = 3;

switch(day) {
 case 6:
   System.out.println("Saturday");
   break;
 case 7:
   System.out.println("Sunday");
```

```
   break;
default:
   System.out.println("Weekday");
}
// Outputs "Weekday"
```

Aucune pause n'est nécessaire dans le cas par défaut, car il s'agit toujours de la dernière instruction du commutateur.

### F - Boucle While

Une instruction de boucle permet d'exécuter de manière répétée une instruction ou un groupe d'instructions.

Une instruction de boucle while exécute de manière répétée une instruction cible tant qu'une condition donnée est vraie.

Exemple:

```
int x = 3;

while(x > 0) {
  System.out.println(x);
  x--;
}
/*
Outputs
 3
 2
 1
*/
```

Les boucles while vérifient la condition x > 0. Si elle prend la valeur true, elle exécute les instructions dans son corps. Ensuite, il vérifie à nouveau l'instruction et répète.

Remarquez la déclaration **x--**. Cela diminue x à chaque exécution de la boucle et arrête la boucle lorsque x atteint 0. Sans l'instruction, la boucle s'exécutera pour toujours.

**while Loops**

Lorsque l'expression est testée et que le résultat est faux, le corps de la boucle est ignoré et la première instruction après la boucle while est exécutée.

Exemple:

```
int x = 6;

while( x < 10 )
{
 System.out.println(x);
 x++;
}
System.out.println("Loop ended");

/*
6
7
8
9
Loop ended
*/
```

### G - Boucle For

Une autre structure de boucle est la boucle for. Une boucle for vous permet d'écrire efficacement une boucle qui doit s'exécuter un certain nombre de fois.

Syntaxe:

```
for (initialization; condition; increment/decrement)
{
  statement(s)
}
```

Initialisation: l'expression ne s'exécute qu'une seule fois au début de la boucle

Condition: est indiquée à chaque itération de la boucle. La boucle exécute l'instruction à plusieurs reprises, jusqu'à ce que cette condition renvoie false.

Incrémenter / décrémenter: s'exécute après chaque itération de la boucle.

L'exemple suivant imprime les chiffres de 1 à 5.

```
for(int x = 1; x <=5; x++) {
 System.out.println(x);
}

/* Outputs
1
2
3
4
5
*/
```

Cela initialise x à la valeur 1 et imprime à plusieurs reprises la valeur de x, jusqu'à ce que la condition x <= 5 devient fausse. À chaque itération, l'instruction x ++ est exécutée, incrémentant x de un.

**for Loops**

Vous pouvez avoir tout type de condition et tout type d'instructions d'incrémentation dans la boucle for.

L'exemple ci-dessous imprime uniquement les valeurs paires comprises entre 0 et 10:

```
for(int x=0; x<=10; x=x+2) {
 System.out.println(x);
}
/*
0
2
4
6
8
10
*/
```

Une boucle for est préférable lorsque les nombres de début et de fin sont connus.

### H - Boucle DO... while

Une boucle do ... while est similaire à une boucle while, sauf qu'une boucle do ... while est garantie pour s'exécuter au moins une fois.

Exemple:

```
int x = 1;
```

```
do {
 System.out.println(x);
 x++;
} while(x < 5);

/*
1
2
3
4
*/
```

Notez que la condition apparaît à la fin de la boucle, donc les instructions de la boucle s'exécutent une fois avant d'être testées.
Même avec une condition fausse, le code s'exécutera une fois. Exemple:

```
int x = 1;
do {
 System.out.println(x);
 x++;
} while(x < 0);

//Outputs 1
```

Notez que dans les boucles do ... while, le while n'est que la condition et n'a pas de corps lui-même.

**Instructions de contrôle de boucle**
Les instructions break et continue modifient le flux d'exécution de la boucle.
L'instruction break termine la boucle et transfère l'exécution à

l'instruction immédiatement après la boucle.

Exemple:

```
int x = 1;

while(x > 0) {
System.out.println(x);
 if(x == 4) {
   break;
 }
 x++;
}

/* Outputs
1
2
3
4
*/
```

L'instruction **continue** oblige la boucle à ignorer le reste de son corps, puis à retester immédiatement sa condition avant de la réitérer. En d'autres termes, il fait passer la boucle à sa prochaine itération.

Exemple:

```
for(int x=10; x<=40; x=x+10) {
 if(x == 30) {
   continue;
 }
 System.out.println(x);
}
```

```
/* Outputs
 10
 20
 40
*/
```

Comme vous pouvez le voir, le code ci-dessus ignore la valeur de 30, comme indiqué par l'instruction continue.

# III. Module 3: Tableaux

**A - Tableaux :**

Un tableau est une collection de variables du même type. Lorsque vous devez stocker une liste de valeurs, telles que des nombres, vous pouvez les stocker dans un tableau, au lieu de déclarer des variables distinctes pour chaque nombre.

Pour déclarer un tableau, vous devez définir le type des éléments entre crochets.
Par exemple, pour déclarer un tableau d'entiers:

```
int[ ] arr;
```

Le nom du tableau est **arr**. Le type d'éléments qu'il contiendra est **int**.

Maintenant, vous devez définir la capacité du tableau ou le nombre d'éléments qu'il contiendra. Pour ce faire, utilisez le mot-clé **new**.

```
int[ ] arr = new int[5];
```

Le code ci-dessus déclare un tableau de 5 entiers.
Dans un tableau, les éléments sont ordonnés et chacun a une position spécifique et constante, qui est appelée un index.

Pour référencer des éléments dans un tableau, tapez le nom du tableau suivi de la position d'index dans une paire de crochets.
Exemple:

```
arr[2] = 42;
```

Cela affecte une valeur de 42 à l'élément avec 2 comme index.
Notez que les éléments du tableau sont identifiés par des numéros d'index de base zéro, ce qui signifie que **l'index du premier élément est 0** plutôt qu'un. Ainsi, l'indice maximum du tableau int [5] est 4.

**Initialisation des tableaux**
Java fournit un raccourci pour instancier des tableaux de types et de chaînes primitifs.
Si vous savez déjà quelles valeurs insérer dans le tableau, vous pouvez utiliser un littéral de tableau.
Exemple d'un littéral de tableau:

```
String[ ] myNames = { "A", "B", "C", "D"};
System.out.println(myNames[2]);

// Outputs "C"
```

Placez les valeurs dans une liste séparée par des virgules, placée entre accolades.
Le code ci-dessus initialise automatiquement un tableau contenant 4 éléments et stocke les valeurs fournies.

Parfois, vous pouvez voir les crochets placés après le nom du tableau, ce qui fonctionne également, mais la méthode préférée consiste à placer les crochets après le type de données du tableau

### B - Additionner des éléments dans des tableaux

**La taille d'un tableau (Array length) :**

Vous pouvez accéder à la longueur d'un tableau (le nombre d'éléments qu'il stocke) via sa propriété length.

Exemple:

```
int[ ] intArr = new int[5];
System.out.println(intArr.length);

//Outputs 5
```

Maintenant que nous savons comment définir et obtenir les éléments d'un tableau, nous pouvons *calculer la somme de tous les éléments d'un tableau à l'aide de boucles*.

**La boucle for** est la boucle la plus utilisée lorsque vous travaillez avec des tableaux, car nous pouvons utiliser la longueur du tableau pour déterminer combien de fois exécuter la boucle.

```
int [ ] myArr = {6, 42, 3, 7};
int sum=0;
for(int x=0; x < myArr.length; x++) {
  sum += myArr[x];
}
System.out.println(sum);
```

```
// 58
```

Dans le code ci-dessus, nous avons déclaré une somme variable pour stocker le résultat et lui avons attribué 0.
Ensuite, nous avons utilisé une boucle for pour parcourir le tableau et ajouter la valeur de chaque élément à la variable.

La condition de la boucle for est x < myArr.length, car l'index du dernier élément est **myArr.length-1**.

### C - Boucle For améliorer

La boucle for améliorée (parfois appelée boucle "pour chaque") est utilisée pour parcourir les éléments dans les tableaux.
Les avantages sont qu'il élimine la possibilité de bogues et rend le code plus facile à lire.
Exemple:

```
int[ ] primes = {2, 3, 5, 7};

for (int t: primes) {
  System.out.println(t);
}

/*
2
3
5
7
*/
```

La boucle for améliorée déclare une variable d'un type

compatible avec les éléments du tableau auquel vous accédez. La variable sera disponible dans le bloc for et sa valeur sera la même que l'élément de tableau actuel.
Ainsi, à chaque itération de la boucle, la variable t sera égale à l'élément correspondant dans le tableau.
**NB :** Remarquez les deux points après la variable dans la syntaxe.

### D - Tableaux Multidimensionnels

Les tableaux multidimensionnels sont des tableaux qui contiennent d'autres tableaux. Le tableau bidimensionnel est le tableau multidimensionnel le plus élémentaire.
Pour créer des tableaux multidimensionnels, placez chaque tableau dans son propre ensemble de crochets. Exemple d'un tableau à deux dimensions:

```
int[ ][ ] sample = { {1, 2, 3}, {4, 5, 6} };
```

Cela déclare un tableau avec deux tableaux comme éléments.
Pour accéder à un élément du tableau à deux dimensions, fournissez deux index, un pour le tableau et un autre pour l'élément à l'intérieur de ce tableau.
L'exemple suivant accède au premier élément du deuxième tableau d'échantillons.

```
int x = sample[1][0];
System.out.println(x);

// Outputs 4
```

Les deux index du tableau sont appelés index de ligne et index de colonne.

Vous pouvez obtenir et définir les éléments d'un tableau multidimensionnel en utilisant la même paire de crochets.
Exemple:

```
int[ ][ ] myArr = { {1, 2, 3}, {4}, {5, 6, 7} };
myArr[0][2] = 42;
int x = myArr[1][0]; // 4
```

Le tableau bidimensionnel ci-dessus contient trois tableaux. Le premier tableau a trois éléments, le second a un seul élément et le dernier à trois éléments.

En Java, vous n'êtes pas limité à des tableaux à deux dimensions seulement. Les tableaux peuvent être imbriqués dans des tableaux à autant de niveaux que votre programme en a besoin. Tout ce dont vous avez besoin pour déclarer un tableau de plus de deux dimensions est d'ajouter autant d'ensembles de parenthèses vides que nécessaire. Cependant, ceux-ci sont plus difficiles à maintenir.
N'oubliez pas que tous les membres du tableau doivent être du même type.

# IV. Module 4: Classes et Objets

### A - Programmation Orienté Objet

Java utilise la programmation orientée objet (POO), un style de programmation qui vise à rapprocher la réflexion sur la programmation de la réflexion sur le monde réel.
Dans la POO, chaque objet est une unité indépendante avec une identité unique, tout comme les objets du monde réel.

Une pomme est un objet; tout comme une tasse. Chacun a son identité unique. Il est possible d'avoir deux tasses qui semblent identiques, mais ce sont toujours des objets distincts et uniques.

Les objets ont également des caractéristiques qui sont utilisées pour les décrire.
Par exemple, une voiture peut être rouge ou bleue, une tasse peut être pleine ou vide, etc. Ces caractéristiques sont également appelées attributs. Un attribut décrit l'état actuel d'un objet.
Dans le monde réel, chaque objet se comporte à sa manière. La voiture bouge, le téléphone sonne, etc.
Il en va de même pour les objets: le comportement est spécifique au type d'objet.

En résumé, dans la programmation orientée objet, chaque objet a trois dimensions: identité, attributs et comportement.
Les attributs décrivent l'état actuel de l'objet et ce que l'objet est capable de faire est démontré par le comportement de l'objet.

**Des classes**
Une classe décrit ce que sera l'objet, mais est distincte de l'objet lui-même.
En d'autres termes, les classes peuvent être décrites comme des plans, des descriptions ou des définitions pour un objet. Vous pouvez utiliser la même classe comme modèle pour créer plusieurs objets. La première étape consiste à définir la classe, qui devient alors un modèle de création d'objet.

Chaque classe a un nom et chacune est utilisée pour définir des attributs et un comportement.
Quelques exemples d'attributs et de comportement:


En d'autres termes, un objet est une instance d'une classe.

### B - Méthodes

Les méthodes définissent le comportement. Une méthode est un ensemble d'instructions regroupées pour effectuer une opération. System.out.println () est un exemple de méthode.
Vous pouvez définir vos propres méthodes pour effectuer les tâches souhaitées.

Prenons le code suivant:

```
class MyClass {

 static void sayHello() {
   System.out.println("Hello World!");
 }

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

```
}
// Outputs "Hello World!"
```

Le code ci-dessus déclare une méthode appelée "sayHello", qui imprime un texte, puis est appelée dans main.
*Pour appeler une méthode, tapez son nom, puis suivez le nom avec un jeu de parenthèses.*

**Appeler une Méthode**
Vous pouvez appeler une méthode autant de fois que nécessaire.
Lorsqu'une méthode s'exécute, le code saute à l'endroit où la méthode est définie, exécute le code à l'intérieur de celle-ci, puis revient en arrière et passe à la ligne suivante.
**Exemple:**

```
class MyClass {

 static void sayHello() {
   System.out.println("Hello World!");
 }

 public static void main(String[ ] args) {
   sayHello();
   sayHello();
   sayHello();
 }
}

// Hello World!
// Hello World!
// Hello World!
```

Dans des cas comme celui ci-dessus, où la même chose se répète encore et encore, vous pouvez obtenir le même résultat en utilisant des boucles (**While** ou **For**).

**Paramètres de méthode**

Vous pouvez également créer une méthode qui prend des données, appelées paramètres, avec elle lorsque vous l'appelez. Écrivez les paramètres entre parenthèses de la méthode.

Par exemple, nous pouvons modifier notre méthode sayHello () pour prendre et sortir un paramètre String.

```
class MyClass {

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

 public static void main(String[ ] args) {
   sayHello("David");
   sayHello("Amy");
 }

}
// Hello David
// Hello Amy
```

La méthode ci-dessus prend une chaîne appelée nom comme paramètre, qui est utilisée dans le corps de la méthode. Ensuite, lors de l'appel de la méthode, nous passons la valeur du paramètre entre parenthèses.

Les méthodes peuvent prendre plusieurs paramètres séparés

par des virgules.

Les avantages de l'utilisation de méthodes au lieu d'instructions simples sont les suivants:
- **réutilisation du code:** vous pouvez écrire une méthode une fois et l'utiliser plusieurs fois, sans avoir à réécrire le code à chaque fois.
- **paramètres:** en fonction des paramètres transmis, les méthodes peuvent effectuer diverses actions.

### C - Type de retour d'une Méthode

Le mot clé return peut être utilisé dans les méthodes pour renvoyer une valeur.
Par exemple, nous pourrions définir une méthode nommée sum qui renvoie la somme de ses deux paramètres.

```
static int sum(int val1, int val2) {
  return val1 + val2;
}
```

Notez que dans la définition de méthode, nous avons défini le type de retour avant de définir le nom de la méthode. Pour notre méthode sum, elle est int, car elle prend deux paramètres de type int et retourne leur somme, qui est aussi un int.
Le mot-clé **statique** sera discuté dans une prochaine leçon.

Maintenant, nous pouvons utiliser la méthode dans notre main.

```
class MyClass {

static int sum(int val1, int val2) {
```

```
 return val1 + val2;
}

 public static void main(String[ ] args) {
   int x = sum(2, 5);
   System.out.println(x);
 }
}
// Outputs "7"
```

Comme la méthode renvoie une valeur, nous pouvons l'affecter à une variable.

Lorsque vous n'avez pas besoin de renvoyer de valeur de votre méthode, utilisez le mot clé **void**.
Remarquez *le mot clé void* dans la définition de la méthode main - cela signifie que main ne renvoie rien.

**Le type de retour**

Jetez un œil au même code de notre leçon précédente avec des commentaires explicatifs, afin que vous puissiez mieux comprendre comment cela fonctionne:

```
// returns an int value 5
static int returnFive() {
 return 5;
}

// has a parameter
static void sayHelloTo(String name) {
 System.out.println("Hello " + name);
}
```

```
// simply prints"Hello World!"
static void sayHello() {
 System.out.println("Hello World!");
}
```

Ayant acquis une connaissance des types et paramètres de retour de méthode, examinons à nouveau la définition de la méthode principale.

```
public static void main(String[ ] args)
```

Cette définition indique que la méthode principale prend un tableau de chaînes comme paramètres et ne renvoie pas de valeur.

Créons une méthode qui prend deux paramètres de type int et renvoie le plus grand, puis appelons-le en principal:

```
public static void main(String[ ] args) {
 int res = max(7, 42);
 System.out.println(res); //42
}

static int max(int a, int b) {
 if(a > b) {
   return a;
 }
 else {
   return b;
 }
}
```

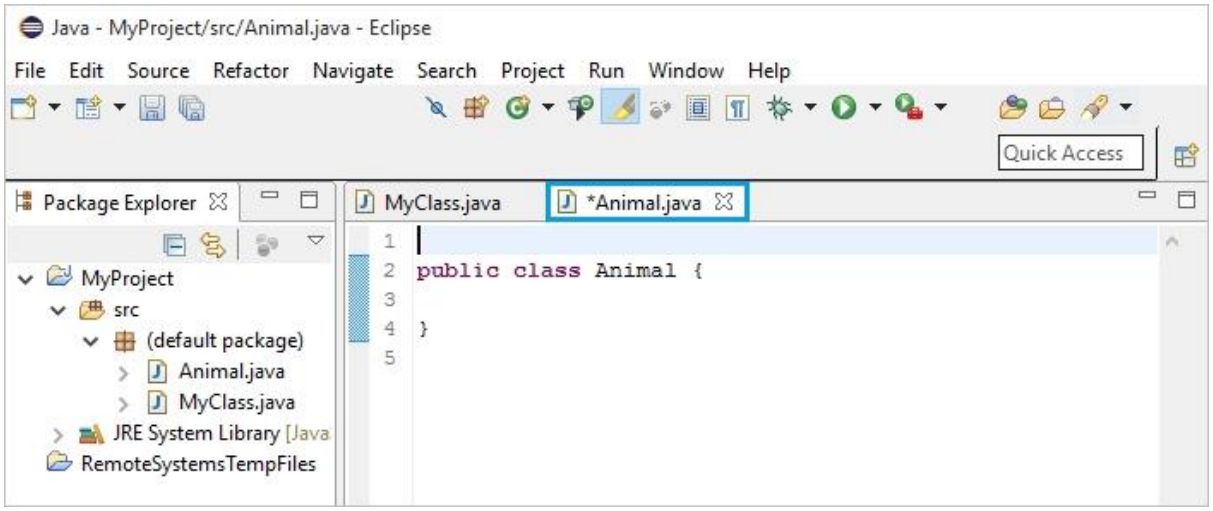
Une méthode peut avoir un type de paramètre (ou des

paramètres) et renvoyer un autre type différent. Par exemple, il peut prendre deux doubles et retourner un int.

## D - Création de Classes et d'Objets

### Création de classes

Pour créer vos propres objets personnalisés, vous devez d'abord créer les classes correspondantes. Pour ce faire, cliquez avec le bouton droit sur le dossier src dans Eclipse et sélectionnez Créer-> Nouveau-> Classe. Donnez un nom à votre classe et cliquez sur Terminer pour ajouter la nouvelle classe à votre projet




Comme vous pouvez le voir, Eclipse a déjà ajouté le code initial de la classe.

Permet maintenant de créer une méthode simple dans notre nouvelle classe.

### Animal.java

```
public class Animal {
 void bark() {
   System.out.println("Woof-Woof");
 }
}
```

Nous avons déclaré une méthode bark() dans notre classe Animal.
Maintenant, pour utiliser la classe et ses méthodes, nous devons déclarer un objet de cette classe.
Dirigeons-nous vers notre principal et créons un nouvel objet de notre classe.

**MyClass.java**

```
class MyClass {
 public static void main(String[ ] args) {
   Animal dog = new Animal();
   dog.bark();
 }
}
// Outputs "Woof-Woof"
```

Maintenant, le chien est un objet de type Animal. Ainsi, nous pouvons appeler sa méthode bark(), en utilisant le nom de l'objet et un point.
La notation par points est utilisée pour accéder aux attributs et méthodes de l'objet.

### E - Les attributs d'une classe

#### Définition des attributs

Une classe a des attributs et des méthodes. Les attributs sont essentiellement des variables au sein d'une classe.
Créons une classe appelée Vehicle, avec ses attributs et méthodes correspondants.

Une classe a des attributs et des méthodes. Les attributs sont essentiellement des variables au sein d'une classe.
Créons une classe appelée Vehicle, avec ses attributs et méthodes correspondants.

```
public class Vehicle {
 int maxSpeed;
 int roues;
 String color;
 double fuelCapacity;

 void klaxon() {
   System.out.println("Beep!");
 }
}
```

**maxSpeed**, **roues**, **couleur** et **fuelCapacity** sont les attributs de notre classe Vehicle, et **klaxon()** est la seule **méthode**.

**Création d'objets**

Ensuite, nous pouvons créer plusieurs objets de notre classe Vehicle et utiliser la syntaxe à points pour accéder à leurs attributs et méthodes.

```
class MyClass {
  public static void main(String[ ] args) {
    Vehicle v1 = new Vehicle();
    Vehicle v2 = new Vehicle();
    v1.color = "red";
    v2.klaxon();
  }
}
```

### F - Modificateurs d'accès

Voyons maintenant le mot-clé **public** devant la méthode principale.

```
public static void main(String[ ] args)
```

**public** est un modificateur d'accès, ce qui signifie qu'il est utilisé pour définir le niveau d'accès. Vous pouvez utiliser des modificateurs d'accès pour les classes, les attributs et les méthodes.

Pour les classes, les modificateurs disponibles sont publics ou par défaut (laissés en blanc), comme décrit ci-dessous:

**public:** la classe est accessible par toute autre classe.
**par défaut**: la classe n'est accessible que par les classes du même package.

Les choix suivants sont disponibles pour les attributs et les méthodes:
**par défaut:** une variable ou une méthode déclarée sans modificateur de contrôle d'accès est disponible pour toute autre classe du même package.
public: accessible depuis toute autre classe.

**protected:** fournit le même accès que le modificateur d'accès par défaut, avec l'ajout que les sous-classes peuvent accéder aux méthodes et variables protégées de la superclasse (les sous-classes et les superclasses sont traitées dans les leçons à venir).

**privé:** accessible uniquement dans la classe déclarée elle-même.

Exemple:

```
public class Vehicle {
 private int maxSpeed;
 private int wheels;
 private String color;
 private double fuelCapacity;

 public void horn() {
   System.out.println("Beep!");
 }
}
```

Il est recommandé de conserver les variables d'une classe privées. Les variables sont accessibles et modifiées à l'aide de Getters et Setters.
Appuyez sur Continuer pour en savoir plus sur les Getters et Setters.

**Getters & Setters:**
Les Getters et Setters sont utilisés pour protéger efficacement vos données, en particulier lors de la création de classes. Pour chaque variable, la méthode get renvoie sa valeur, tandis que la méthode set définit la valeur.

Les getters commencent par get, suivi du nom de la variable, avec la première lettre du nom de la variable en majuscule.
Les setters commencent par set, suivi du nom de la variable,

avec la première lettre du nom de la variable en majuscule.

Exemple :

```
public class Vehicle {
 private String color;

 // Getter
 public String getColor() {
   return color;
 }

// Setter
 public void setColor(String c) {
   this.color = c;
 }
}
```

La méthode **getter** renvoie la valeur de l'attribut.
La méthode **setter** prend un paramètre et l'affecte à l'attribut.
Le **mot-clé this** est utilisé pour faire référence à l'objet courant. Fondamentalement, this.color est l'attribut de couleur de l'objet actuel.

Une fois que notre getter et setter ont été définis, nous pouvons les utiliser dans notre principal **main**:

```
public static void main(String[ ] args) {
 Vehicle v1 = new Vehicle();
 v1.setColor("Red");
 System.out.println(v1.getColor());
}
```

```
//Outputs "Red"
```

Les getters et setters nous permettent d'avoir le contrôle sur les valeurs. Vous pouvez, par exemple, valider la valeur donnée dans le setter avant de définir réellement la valeur.

Les getters et setters sont des éléments fondamentaux de l'encapsulation, qui seront traités dans le prochain module.

**Constructeurs :**
Les constructeurs sont des méthodes spéciales invoquées lorsqu'un objet est créé et sont utilisées pour les initialiser.

Un constructeur peut être utilisé pour fournir des valeurs initiales pour les attributs d'objet.

- Un nom de *constructeur* doit être identique à son nom de classe.
- Un *constructeur* ne doit pas avoir de type de retour explicite.

**Exemple de constructeur**:

```
public class Vehicle {
 private String color;
 Vehicle() {
    color = "Red";
 }
}
```

La méthode Vehicle() est le constructeur de notre classe, donc chaque fois qu'un objet de cette classe est créé, l'attribut color

sera défini sur "Red".
Un constructeur peut également prendre des paramètres pour initialiser des attributs.

```
public class Vehicle {
 private String color;
 Vehicle(String c) {
   color = c;
 }
}
```

Vous pouvez considérer les constructeurs comme des méthodes qui configurent votre classe par défaut, vous n'avez donc pas besoin de répéter le même code à chaque fois.

**Utilisation du constructeur:**
Le constructeur est appelé lorsque vous créez un objet à l'aide du nouveau mot clé.
**Exemple**:

```
public class MyClass {
 public static void main(String[ ] args) {
   Vehicle v = new Vehicle("Blue");
 }
}
```